Ryoan：用于秘密数据的不可信计算的分布式沙箱

德克萨斯大学奥斯汀分校的 TYLER HUNT 和 ZHITING ZHU
徐元忠，Google
德州大学奥斯汀分校的 SIMON PETER 和 EMMETT WITCHEL

诸如税务准备或基因组筛选之类的现代数据处理服务的用户被迫以他们希望保密的数据信任他们。当数据所有者不信任的服务处理秘密数据时，Ryoan[1]会对其进行保护。由于用户无法控制服务提供商或计算平台，因此很难在分布式环境中实现此目标。限制代码以防止其泄露机密是非常困难的，但是 Ryoan 受益于新硬件和面向请求的数据模型。

Ryoan 提供了一个分布式沙箱，利用硬件安全区域（例如，英特尔的软件防护扩展（SGX）[40]）来保护沙箱实例免受潜在恶意计算平台的攻击。受保护的沙箱实例限制了不受信任的数据处理模块，以防止用户输入数据泄漏。Ryoan 专为面向请求的数据模型而设计，其中受限模块仅处理输入一次，并且不保留输入状态。我们介绍 Ryoan 的设计和原型实现，并针对一系列具有挑战性的问题（包括电子邮件过滤，运行状况分析，图像处理和机器翻译）进行评估。

CCS 概念：•**安全和隐私** **系统安全**； **操作系统安全**；

其他关键词和短语：英特尔 SGX，飞地，沙盒，不受信任的操作系统，私有计算

ACM 参考格式：
泰勒 亨特（Tyler Hunt），朱志婷，徐元忠，西蒙 彼得（Simon Peter）和艾米特 维奇（Emmett Witchel）。2018 年。Ryoan：用于秘密数据的不可信计算的分布式沙箱。*ACM Trans*。*计算 Syst*。35，4，Article 13（2018 年 12 月），32 页。
https://doi.org/10.1145/3231594

1 引言

数据处理服务在 Internet 上广泛可用。单个用户可以方便地访问它们以执行任务，包括图像编辑（Pixlr），税务准备（TurboTax），数据分析（SAS OnDemand ）甚至个人健康分析（23andMe）。但是，用户对此输入

[1]该工作首次以相同的标题发布在第十二届 USENIX 操作系统设计和实现研讨会（OSDI'16）[38]上。在此进行了扩展，并进行了其他讨论和实验。

我们感谢 NSF 资助的 CNS-1228843，CCF-1333594 和 CNS-1618563 以及 Google 的研究奖。作者的地址：德克萨斯大学奥斯汀分校计算机科学系，T。Hunt，Z。Zhu，S。Peter 和 E. Witchel，2317 Speedway，Stop D9500 Austin，TX 78712；Y. Xu，Google，1600 Amphitheatre Parkway Mountain View，CA 94043，美国。

0734-2071 / 2018 / 12-ART13 $ 15.00 https://doi.org/10.1145/3231594

服务通常是敏感的，例如税务文件和健康数据，这给用户带来了困境。为了利用这些服务的便利性和专业知识，他（她）必须向他们披露敏感数据，从而可能使他们向其他各方披露数据。如果他或她想对她的数据保密，那么他或她要么必须放弃使用服务，要么希望它们可以被信任-他们的服务软件不会（有意或无意地）泄漏数据，而他们的管理员也不会在数据驻留在服务器计算机上时读取数据。

为用户提供数据处理服务的公司通常希望将部分计算外包给第三方云服务，这种做法称为“软件即服务（SaaS）”。例如，23andMe 可能选择使用由 Amazon 托管的通用机器学习服务。SaaS 鼓励将问题分解为专业的部分，服务提供商可以代表用户进行组装，例如，将 23andMe 的健康专业知识与亚马逊的机器学习专业知识和强大的云基础架构相结合。但是，23andMe 现在发现自己是 Amazon 机器学习服务的用户，并且面临着自己的困境—为了使用 Amazon 的机器学习服务，它必须披露健康数据和各种疾病之间的专有关联。在这些情况下，

我们建议龙安，[2] 分布式沙箱，使用户的数据保持秘密数据处理服务，而不信任的软件堆栈，开发者，或者这些服务的管理员。首先，它提供沙箱实例以限制单个数据处理模块并防止它们泄漏数据。其次，它使用受信任的硬件允许远程用户验证沙盒实例的完整性并保护其执行。第三，Ryoan 可以配置为允许受限代码模块以受控方式进行通信，从而在互不信任的各方之间实现灵活的委派。Ryoan 使用户确信服务已保护了她的秘密。

Ryoan 的一项关键启用技术是硬件保护区执行（例如，英特尔的软件保护扩展（SGX）[40]），这是一种新的硬件原语，它使用受信任的硬件来保护用户级计算免受潜在的恶意特权软件的侵害。处理器硬件将未加密的数据保留在芯片上，但是当数据移入 RAM 时会对其进行加密。系统管理程序和操作系统保留了管理内存的能力（例如，将内存页移至辅助存储），但是数据的保密性受到保护，因为特权软件只能看到数据的加密版本，并且完整性通过冲突得以保留-抵抗哈希。避风港[7]和 SCONE [5]是使用安全区保护用户的计算不受潜在恶意系统软件攻击的系统示例。它们包括一个库操作系统，以增加向后兼容性。

Ryoan 面临的问题超出了诸如 Haven [7]之类的飞地保护计算所面临的问题。安全区旨在保护用户信任的应用程序，并且不会与基础结构发生冲突（尽管它可能会无意间通过侧通道泄漏数据；请参见第 2.3 节）。在 Ryoan 的威胁模型中，应用程序和基础结构都不在用户的控制之下，他们可能试图通过*秘密渠道*合谋窃取用户的秘密，*甚至*如果应用程序本身使用安全区保护与提供程序的基础结构隔离开来。Ryoan 的目标是防止此类秘密通道并阻止不可信的应用程序有意和秘密地使用用户数据来调制事件（例如系统调用参数或 I / O 流量统计信息），这些事件对于基础结构是可见的。

Ryoan 限制了构成不受信任应用程序的不受信任模块。限制不受信任的代码[51]是一个长期存在的问题，在技术上仍然具有挑战性。Ryoan 通过利用硬件支持的飞地防护和

[2] Ryoan 是一个沙箱，其名称的灵感来自著名的干旱景观禅宗花园，该花园激发了沉思（Ryoan-ji）。

假设一个面向请求的数据模型。受限模块仅处理一次输入，并且在接收到输入后无法读取或写入持久性存储。该模型将 Ryoan 的适用性限制在面向请求的服务器应用程序中，但是这种服务器是将可扩展服务带给大量用户的最常用方法。

Ryoan 使用受信任沙箱的多个实例来限制应用程序。在龙安原型中使用的可信沙箱是基于本机客户端（氯化钠）[78，91]，国家的最先进的用户级沙箱（它可以建成为一个独立的二进制，独立于浏览器）。NaCl 使用基于编译器的技术来限制不受信任的代码，而不是依赖地址空间分隔，而地址空间分隔是与 SGX 安全区兼容所必需的属性。Ryoan 沙箱通过控制显式 I / O 通道以及秘密通道（例如系统调用跟踪和数据大小）来保护秘密。

Ryoan 原型使用 SGX 提供硬件区域。每个 SGX 飞地均包含一个沙箱实例，该沙箱实例可加载并执行不受信任的模块。沙箱实例彼此通信以形成一个分布式沙箱，该沙箱可为所有参与方（用户和不同的服务提供者）实施强有力的隐私保证。Ryoan 提供了由用户和服务提供商定义的污点标签（类似于来自 DIFC 的保密标签[68]），这使他们可以确保 Ryoan 限制处理其机密的任何模块。

Ryoan 的安全目标很简单：防止泄露机密数据。但是，如果没有集中的可信任平台，限制用户无法控制的服务将是一个挑战。我们做出以下贡献：

- 一种新的执行模型，该模型允许互不信任的各方在不受信任的基础结构上以分布式方式处理敏感数据。
- 分布式沙箱原型的设计和实现，该沙箱限制了不受信任的代码模块（可能在不同的机器上），并实施了 I / O 策略，以防止机密信息泄露。
- 几个实际应用场景的案例研究，以说明它们如何从 Ryoan 的保密保证中受益，包括图像处理系统，电子邮件垃圾邮件/病毒过滤器，个人健康分析工具和机器翻译器。
- 通过测量其每个构建块的执行开销来评估我们的原型的性能特征：SGX 区域，限制区和检查点/回滚。评估基于 SGX 硬件和仿真。

2 背景和威胁模型

我们假设处理器具有受硬件保护的区域，例如，英特尔支持 SGX 的 Skylake（或更高版本）架构。SGX 提供代码和初始数据的加密哈希（称为度量），从而使运行在受保护区域中的程序可以验证代码和数据完整性，并允许其访问由主机软件不知道且无法找到的密钥加密的私有数据出去。受保护的飞地的内存具有由硬件保证的私密性和完整性。硬件在芯片外移动时对内存内容进行加密和哈希处理，以保护内容免受其他用户以及平台特权软件（操作系统和系统管理程序）的侵害。飞地中的代码可以操纵用户机密，而不必担心将其泄露给底层执行平台。

SGX 的安全保证非常适合 Ryoan 的基于 NaCl 的分布式沙箱实例。每个沙箱实例都会限制其加载的代码，以确保代码不会通过存储，网络或基础平台提供的其他渠道泄漏机密。沙盒实例使用安全的 TLS 连接相互通信。通过收集 SGX 测量值并通过提供

通过信任的初始化代码，Ryoan 可以向用户证明其处理拓扑已正确设置。

2.1 威胁模型

我们考虑涉及数据处理服务的多个相互不信任的各方。服务的用户不信任服务提供者以保持数据秘密如果服务提供者将部分计算外包给其他服务提供者，则它将成为这些服务提供者的用户，并且也不信任它们提供保密性。每个服务提供商都可以在其计算平台上部署其软件，或使用互不信任所有服务提供商的第三方云平台。我们假设用户和提供者信任他们的代码和平台，但不信任彼此的代码或平台。每个人都必须信任 Ryoan 和 SGX。

服务提供者可能与其计算平台提供者相同，并且两者可能合谋从其输入数据中窃取秘密。除了直接传递数据之外，不受信任的代码还可以通过*软件接口*（例如系统调用序列和自变量）使用秘密通道，以将位从用户输入传递到平台。

服务的用户不信任计算平台中任何特权级别的软件。例如，攻击者可能是计算机的所有者和操作员他或她可能是好奇的，甚至是恶意的管理员；他或她可能是控制了操作系统和/或管理程序的入侵者；他或她可能拥有一台虚拟机，该虚拟机与被攻击的虚拟机在同一地点；他甚至可以成为不受信任的应用程序或 OS 的开发人员，并编写代码直接记录用户输入。

Ryoan 不会采取任何措施来防止各方故意（或通过错误）泄露自己的秘密。该模型适用于服务提供商在其自己的计算平台上部署代码的情况（有关更多讨论，请参见第 3.3 节）。当在其他平台提供商上执行时，Ryoan 提供针对恶意 OS 的保护，例如，系统调用验证以防止 Iago 攻击[14]（类似于 Haven [7]，Inktag [37]，Sego [50]和 SCONE [5]和 Graphene- SGX [85] ）和加密以保护数据保密。正交技术[13 ， 20 ， 45 ， 75 ， 95]可以被用来减轻软件错误非故意秘密输入数据泄漏到计算的输出。同样我们假设计算平台提供商负责保护其自身的机密（例如，管理员的密码）。

拒绝服务不在我们威胁模型的范围之内。不受信任的应用程序可以拒绝运行，或者底层不受信任的操作系统可以拒绝安排我们的代码。

尽管我们基于系统调用之类的软件接口来考虑隐蔽通道，但在本文中，我们不会基于*硬件限制*（第 2.3 节） 或*执行时间来*考虑旁通道或隐蔽通道。不受信任的安全区可以通过调整其缓存访问，页面访问，执行时间等来泄漏位。这样的信道本身技术上是困难的，并且通常需要专用系统到地址充分[22 28 ， 47 ， 58 ， 97] 。许多公认为安全的系统设计因素出基于硬件的限制或执行时间，至少在一定程度上侧/隐蔽通道[7 ， 61 ， 73 ， 86 ， 93]，因为这样做能够使设计的进步和构建安全系统。尽管我们并不声称要阻止执行时间通道，但 Ryoan 确实将此通道的使用限制为每个请求一次（第 3.1.1 节）。

2.2 我 NTEL 软件狗扩展

新的英特尔处理器中提供了 SGX，它允许进程将其部分地址空间与特权软件隔离开。具有 SGX 的计算机上的进程可以构造一个安全*区*，该安全*区*是一个地址区域，其内容受到保护（不受加密和散列处理），使其免受该安全区外部的所有软件的侵害。因此，加载到安全区中的代码和数据可以在

机密数据，而不必担心无意间泄露给平台。硬件提供这些保证[40]。

SGX 提供飞地身份证明。就我们的目的而言，将一个安全区标识视为安全区初始状态的哈希即可即有效内存内容，权限和安全区中的相对位置。我们对硬件的信任扩展到这些身份。特别是，我们假设在标准密码学假设下无法模拟飞地的初始状态。Ryoan 使用 SGX 证明所有飞地都具有相同的初始状态，因此具有相同的身份。在将敏感数据传递给 Ryoan 之前，用户将向 SGX 请求证明，并确认该身份是 Ryoan 身份。

知道安全区的初始状态可确保不模拟沙箱实例，因为一旦执行开始，SGX 便会保证安全区内存的完整性。平台无法更改安全区内存（包括可执行指令）。SGX 也执行良好定义的执行的入口点，以防止回流到 libc 的[12 ，25]由平台式控制流程操作（在飞地返回到 libc 的攻击仍然是可能的）。

安全区代码可以访问地址空间中不属于另一个安全区的任何部分。但是，安全区代码无法访问所有 x86 功能。所有安全区代码均无特权（环 3），任何会提高其特权的指令都将导致错误。

2.3 ^ h ardware 安全限制

现代英特尔处理器中存在一些已知的安全限制。我们认为这些局限性（以及将来发现的任何其他局限性）必须与 Ryoan 独立解决，我们希望能够解决。这些限制中的每一个都会损害 Ryoan 的安全目标。构建 Ryoan 之类的原型系统的部分目的是确定其安全保证如何依赖于它们依赖的硬件的安全保证，从而为解决基于硬件的限制提供动力。

*2.3.1 SGX 页面错误。*在当前硬件上，特权软件可以操纵安全区的页表，以观察其代码和数据的页粒度跟踪。已经证明了毁灭性攻击，其中应用程序级信息用于从这些粗略地址（例如，文档中的单词和图像）重新创建细粒度的秘密[89]。有在检测或使用其它处理器的功能防止这些攻击，例如，事务存储器活跃的研究[16 ，80]。如果 SGX 飞地维修了自己的页面错误，则该泄漏通道将消失。

*2.3.2 地址总线监视。*尽管 SGX 对 RAM 中的数据进行加密，但是如果攻击者通过嗅探器或经过修改的 RAM 芯片监视地址总线，则它将形成高速缓存行粒度侧或隐蔽通道。如果不进行新的体系结构更改，Ryoan 无法阻止此类攻击。

*2.3.3 处理器监视。*处理器监视单元（PMU）为片上事件提供广泛的性能计数器信息。如果 PMU 更新了针对飞地保护执行中发生的事件的信息，则操作系统可以将该信息用作秘密通道，以通过不受信任的代码来学习机密，从而可以调制其行为，例如，增加某些事件计数。

根据对 Skylake 处理器的测量，在飞地执行过程中某些监视功能（例如，基于事件的精确采样）被关闭，但是启用了非核心计数器（例如，最后一级的高速缓存未命中）[21]。基于分支历史的有效攻击已被证明[52]。目前尚不知道基于处理器监视的其他攻击的有效性如何。

*2.3.4 缓存时序。*驻留在同一内核上的两个进程可以使用缓存定时来获取有关彼此的细粒度信息。例如，张等人。演示了（在类似 Amazon EC2 的平台上）从无冲突的 VM 中提取 ElGamal 密钥的方法[99]。当流程可以共存时，问题就更严重了。其他人已经证明了使用缓存行为高带宽隐蔽通道[87 ，88]。有解决硬件缓存定时攻击的硬件建议[60]。

*2.3.5 基于瞬时指令的攻击。*崩溃[57]和幽灵[48]是分别利用乱序执行和推测执行来执行瞬时指令（由于异常或错误的分支预测而已执行但未被淘汰的指令）的攻击。这些瞬态指令会影响微体系结构的最新处理器，从而创建隐蔽通道。两种攻击都使用处理器缓存进行了演示。Meltdown 特定于 Intel 处理器，并允许用户级程序读取任意内核内存。Spectre 适用于更广泛的处理器（包括 Intel，AMD 和 ARM），但更难以利用。它允许攻击者读取属于在单独地址空间中运行的受害者的内存。

Ryoan 使用 Native Client [78]沙盒化不受信任的代码。Native Client 的沙盒机制可防止沙盒代码引发 Meltdown 攻击。崩溃需要攻击者为内核地址发出内存操作，然后将其用于推测执行；本机客户端将不受信任的代码可以引用的内存地址限制为从不与内核地址重叠的 4GB 范围。

Native Client 的沙箱机制也使挂载 Spectre 攻击更加困难。幽灵式攻击依赖于攻击者训练硬件分支预测器的能力。攻击者训练分支预测程序在执行受害者期间做出错误的预测。Native Client 强制限制代码独立于位置，并将间接分支的目标限制为 4GB 范围内的对齐块。这些措施降低了攻击者操纵分支预测器的自由度，从而缩小了攻击面。

已经有成功的幽灵攻击违反新交所隔离，允许非飞地代码改为飞地内存[15 ，71]。Ryoan 不会缓解此类违规行为，前提是这些安全漏洞将在以后的硬件迭代中得到修复。

2.4 SGX 特有的限制

这些限制存在于 SGX 隐含的威胁模型中，在该模型中，特权软件不受信任，但可以控制诸如持久性存储和网络之类的资源。Ryoan 通过设计消除了这些限制（第 3.6 节）。

*2.4.1 回滚。*平台可以回滚任何持久状态。SGX 可以使用硬件生成的密钥对数据进行哈希处理和加密，以进行存储。但是，它不能保证新鲜度。这种缺陷迫使飞地设计者不得不依靠硬件计数器[82]它具有局限性和性能问题，或者依靠利用其他机器的基于软件的策略 [63]。

*2.4.2 飞地不可区分。*虽然 SGX 使飞地能够向外部方证明其完整性，但没有什么可以阻止平台实例化飞地的多个副本。如果没有唯一标识同一安全区的不同实例的机制，恶意平台可能会通过恶意重定向远程用户视为单个安全区的一组安全区之间的通信，使远程用户混淆特定安全区的状态。临时会话密钥（由 TLS 使用）之类的机制使受害者能够检测到此类攻击。

2.5 应用限制

Ryoan 强制应用程序采用面向请求的数据模型。该数据模型足以对大多数唯一输入进行批处理。有些应用程序行为未映射

完全或完全不依赖 Ryoan 的数据模型。以下是 Ryoan 不支持的应用程序行为类别。

*2.5.1 储存。*Ryoan 不适合存放；它旨在保护敏感输入上的计算。一旦 Ryoan 模块看到了用户数据 Ryoan 就会阻止该模块写入持久性存储。

*2.5.2 网络元数据。*Ryoan 不会采取任何措施来保护网络连接元数据，例如用户的 IP 地址或数据包的长度。Ryoan 保护用户数据，但不保护连接元数据（尽管存在确实保护连接元数据的系统，例如 Tor [26] 从服务器隐藏了客户端的网络地址）。

*2.5.3 对相同/相似的输入数据进行重复计算。*Ryoan 无法消除所有计时通道，而是通过其面向请求的数据模型来减轻其影响。对于重复处理相同或非常相似输入的服务，Ryoan 可能会泄漏太多机密信息。例如，某些在线照片服务旨在使用户重复阅读和编辑照片。Ryoan 不太适合这些服务，因为有了足够的重复输入，不受信任的模块将能够提取输入数据。

*2.5.4 多用户计算。*如果单个请求包含来自多个互不信任的用户的机密，则 Ryoan 无法隔离这些机密。Ryoan 以请求的粒度跟踪数据流，即使数据来自不同的，互不信任的用户，应用程序也可以自由地在单个请求中混合数据。

2.6 机客户端

Google Native Client（NaCl）是一个沙箱，用于使用软件故障隔离来运行 x86 / x86-64 本机代码（NaCl 模块）。NaCl 由验证程序和服务运行时组成。验证程序反汇编二进制文件并验证反汇编指令的执行安全性，以确保不受信任的模块不会脱离 NaCl 的 SFI 沙箱。

NaCl 代表已加载的应用程序执行系统调用。系统将应用程序中的控制权转移到 NaCl 运行时中，以确定适当的操作。Ryoan 不允许应用程序使用其系统调用将信息传递给底层操作系统。例如，如果 Ryoan 将读取的系统调用从应用程序直接传递到平台，则应用程序可以使用调用的大小和数量来编码有关其正在处理的机密数据的信息。我们将在第 3.5 节中讨论 Ryoan 提供的禁闭的细节。

3 设计

Ryoan 是一个分布式沙箱，它执行对秘密数据进行操作的通信不可信模块的有向非循环图（DAG）。Ryoan 的主要工作是防止模块在系统范围之外（包括外部主机和平台的特权软件）通信任何秘密数据。

一个模块由代码，初始化的数据和动态分配的内存的最大大小组成。为了向后兼容，Ryoan 模块支持为 libc 编写的程序，其中可能包括完全编译的语言和在 libc 之上构建的运行时。Ryoan 模块可以是 Linux 程序，也可以包含库操作系统 [7]。SGX 禁止在安全区域执行环 0，因此 Ryoan 无法直接支持操作系统或虚拟机管理程序。

不信任特权软件（即操作系统和管理程序）而封闭模块是 Ryoan 的主要技术挑战。在最坏的情况下，模块和特权软件可能会合谋窃取机密。共谋的可能性迫使良安考虑任何行为

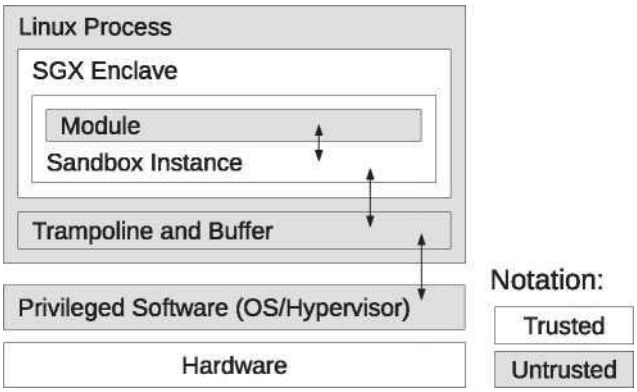

图 1.组成 Ryoan 部署的几个沙盒实例之一。特权软件包括操作系统和可选的管理程序。

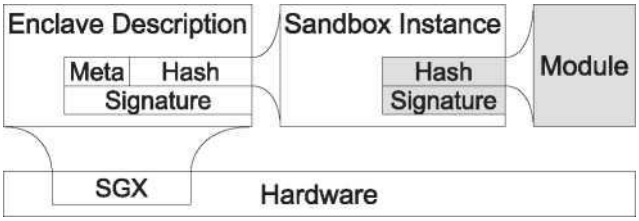

图 2. Ryoan 信任链。SGX 硬件使用预期的 SGX 配置（元）证明有效的沙箱实例正在执行（哈希）。沙盒实例可确保期望的二进制文件已加载有来自软件提供商的签名哈希（灰色）。

对特权软件可见（此后从*外部可见的行为*），成为泄漏机密的潜在渠道。

图 1 显示了分布式沙箱的单个实例。委托人（例如，提供软件即服务的公司）可以贡献 Ryoan 加载和限制的模块，从而使该模块能够安全地处理秘密数据。我们将把提供模块的任何原理称为*模块提供者*。NaCl 沙箱使用加载时代码验证器来确保模块不会通过访问其地址范围之外的内存或在没有 Ryoan 干预的情况下进行系统调用来违反沙箱。

Ryoan 通过在 SGX 提供的受硬件保护的飞地中执行来确保其保密性和完整性。硬件证明了 Ryoan 的初始状态，因此，硬件成为 Ryoan 信任链的基础（图 2 ）。SGX 为用户生成了不可伪造的远程证明，说明沙盒实例正在平台的某个区域中执行。用户可以建立一个加密通道，他或她知道该通道在该沙盒实例中终止。SGX 保证安全区的加密保密性和完整性，防止特权软件对其进行操作。

主安全区会创建所有沙箱实例，并且它们会根据用户的指定在它们之间建立受密码保护的通信通道一旦建立了分布式拓扑，主机就将拓扑中每个节点的证明转发给用户，该用户验证配置是否符合其规范。然后，用户输入他或她的秘密数据。Ryoan 提供了简单的标签来保护 DAG 中的模块添加的机密数据（第 3.3 节）。Ryoan 的所有沙箱实例都一起构成了一个分布式沙箱，可保护秘密输入数据不被对其上运行的不受信任的模块泄漏。

表 1. Ryoan 对不可信模块施加的属性，
实施它们的技术以及 Ryoan 对它们施加原因的原因

模块属性	强制执行	原因
OS 无法访问模块内存（第 2.2 节）。	新交所	安全
初始模块代码和数据已验证（第 2.2 节）。	新交所	安全
只能寻址模块存储器（第 2.6 节）。	氯化钠	安全
Ryoan 拦截系统调用（第 2.6 节，第 3.1 节）。	氯化钠	安全
无法修改 SGX 状态（第 3.2.3 节）。	氯化钠	安全
用户定义拓扑（第 3.2 节）。	龙安	安全
标签跟踪的数据流（第 3.3 节）。	龙安	安全
在请求之间清除内存（第 3.1 节）。	龙安	安全
模块定义了初始化状态（第 3.4 节）。	龙安	表现
无限制的初始化（第 3.2.3 节）。	龙安	兼容性
内存中的 POSIX API（第 3.5 节）。	龙安	兼容性

Ryoan 通过将外部可见行为与机密数据的内容脱钩来防止模块泄漏敏感数据。模块响应输入数据所做的任何事情都有可能成为传送该数据的辅助通道的危险。因此，Ryoan 使得模块的外部可见行为独立于输入数据。SGX 硬件将外部可见行为限制为显式存储到不受保护的内存和使用系统服务（系统调用）。NaCl 工具链和运行时消除了不受保护的存储。

Ryoan 通过在 NaCl 中提供其功能来消除大多数系统调用。例如，Ryoan 通过管理 SGX 飞地内的固定大小的内存池来提供 mmap 功能。但是，不受信任的模块必须读取输入并写入输出，因此 Ryoan 提供了一个受限制的 I / O 模型，可以防止数据泄漏（例如，输出大小是输入大小的固定函数）。表 1 汇总了 Ryoan 施加在模块上的属性，以实现可观察到的行为与机密输入数据的安全去耦。

图 4 显示了来自用户 Alice 的 Ryoan 处理输入的示例，该用户的敏感数据由 23andMe 和 Amazon 处理。每个沙盒实例都在相同或不同计算机上的某个区域中执行。主机可能由 23andMe，Amazon 或第三方提供。在任何情况下，Ryoan 都不会泄漏用户的机密，还可以防止泄漏 23andMe 和亚马逊使用的任何商业机密。

*3.0.1 Ryoan 数据审核跟踪。*当数据遍历模块的 DAG 时，Ryoan 会跟踪哪些模块处理每项用户工作。用户可以使用每个工作单元的审核跟踪，作为 DAG 输出的一部分。尽管 Ryoan 无法验证模块是否正在执行其预期或要求的功能，但是审核跟踪仍然有用。例如，给定的数据可能已经由已知有故障的模块版本处理过。用户是否需要审核跟踪以及用于什么目的取决于应用程序和用户。

3.1 Restricte d I / OM Odel 等

在大多数情况下，Ryoan 禁止访问或替换系统服务，以消除模块控制的外部可见行为。但是，必须允许 I / O，并且不能由不直接控制设备的 Ryoan 代替。相反，Ryoan 在模块上强制使用受限制的 I / O 模型。I / O 模型可确保数据流始终独立于输入数据的内容；一旦模块读取了输入数据，Ryoan 就不会在不受信任的模块的控制下响应于活动而移动数据。有时将此安全属性称为数据可忽略[70]。

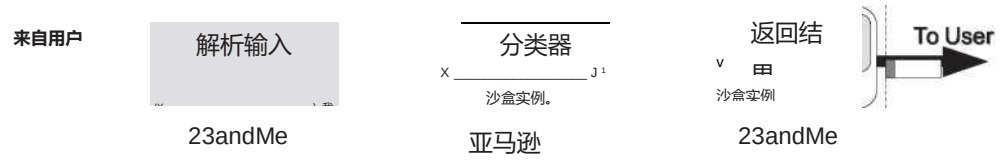

图 3. Ryoan 的分布式沙箱。委托人（在本例中为平台提供者 23andMe 和 Amazon）提供的模块仅限于安全地处理用户数据。

Ryoan 要求模块必须以请求为导向：输入可以是任意大小，但是每个输入都是应用程序定义的“工作单元”。例如，工作单元可以是分类垃圾邮件时的电子邮件，也可以是扫描病毒时的完整文件。每个模块都有一个处理单个工作单元的机会。生成输出后，必须销毁该模块（或将其重置，请参见第 3.4 节），以防止其将一个用户的机密发送给另一用户，或使用将来的请求的处理时间泄漏有关过去请求的信息（请参阅第 3.1.1 节）进行全面讨论）。

工作单位可以是任意大小，但是 Ryoan 通过将模块输出大小设置为输入大小的固定的，应用程序定义的函数，来确保数据流模式不会泄漏输入数据中的秘密。Ryoan 使用以下规则保护通信：（1）在模块开始处理之前，每个沙箱实例都会从每个与输入连接的沙箱实例中读取其整个输入。（2）输出大小是输入大小的固定函数，是 DAG 的一部分。沙箱实例将所有输出填充或截断为由函数和输入大小确定的确切长度。（3）每个沙盒实例的输出完成后，模块都会通知每个沙盒实例，并将模块的输出写入所有与输出连接的沙盒实例。沙盒实例将模块输出封装在一条消息中，该消息包含元数据，该元数据描述了消息的哪一部分是模块输出以及哪一部分是填充（如果有）。解释元数据，并在将数据公开到其模块之前，由下一个沙箱实例删除任何填充。这些规则已足够，因为它们可确保输出流量独立于输入数据（尽管有可能的替代方法，例如，每个请求都可以指定其输出大小）。

考虑图 3 中的场景。每个输入来自用户。用户可以选择泄漏输入的大小，或者他或她可以通过填充输入来隐藏大小。该应用程序的描述指定（1）将 23andMe 的第一个模块的输出填充到 23andMe 定义的固定大小，该大小可以容纳最大的用户输入；（2）Amazon Machine Learning 的分类器模块的输出填充到一个固定大小。大小以编码分类结果，以及（3）23andMe 第二个模块对用户的响应也被填充为固定大小，可以容纳最大可能的结果。每个沙箱实例必须在执行其模块之前接收工作单元的完整输入。

Ryoan 确保输出大小是输入的固定函数，因此如果输出不够大，这是模块的错误。Ryoan 将截断太大的输出和填充的太小输出。但是，模块作者应该能够描述给定输入请求大小下的最大可能输出。例如，垃圾邮件检测器的输出将是输入邮件消息（刚刚复制）的大小加上足以容纳电子邮件垃圾邮件等级的恒定大小。对于许多任务，很容易根据输入来限制输出的大小。例如，直接将机器学习模型的大小与已知任务的已知拓扑的已知拓扑绑定在一起，并将其从一种人类语言翻译成另一种人类语言，这很容易。

*3.1.1 处理时间通道。*尽管 Ryoan 仔细控制 I/O，但执行计算所需的时间由模块控制。模块可以使用此事实，通过基于输入数据中的秘密来更改生成输出所花费的时间长度，来构造定时通道。Ryoan 采取以下步骤来限制通过处理时间通道的泄漏。

- *在输入数据时拍摄一张照片。*Ryoan 允许每个模块有一次机会处理其输入数据，而没有机会将状态从一个输入传递到下一个输入。一站式策略可限制数据泄漏。Ryoan 通过（1）要求数据处理拓扑为 DAG 来避免周期，来强制执行一站式策略；（2）禁止访问通过处理不同工作单元而修改的任何状态；（3）通过以下方式阻止输入重放攻击：如果有任何连接断开，请重新初始化所有安全连接。安全的通信协议包含针对重放攻击的保护措施[92]，因此重新初始化损坏的链接会阻止输入重播。请注意，OS 可以暂停或停止 SGX 安全区的执行，但无法回滚其状态[40]这意味着无法回退安全连接的状态。Ryoan 本身使用可通过处理器的 RDRAND 指令获得的高质量随机性来建立不依赖于操作系统的安全连接。
- *随机性。*用户可以指定受限模块是否需要访问随机性。如果用户允许，则模块可以通过处理器访问随机性，例如英特尔的 RDRAND 指令。Ryoan 不允许受限模块从操作系统中获取随机性访问随机性意味着恶意模块可能泄漏来自输入的随机位，例如，随机选择一个输入位并使用其处理时间泄漏它。如果用户重复输入数据，则具有随机性的恶意模块最终可能会在其处理时间通道上泄漏整个输入，即使对于每个输入工作单元它只泄漏一次。使用固定的处理时间可以消除此通道。

  某些自然类型的输入数据可以充当随机性的来源。Ryoan 假定重复输入是逐位相同的。如果计算的输入包含不断变化的元数据（例如，请求的嵌入时间戳），则受限模块可以使用这些变化的位来播种伪随机数生成器，以从语义相同的输入中泄漏出多个位。就像用户必须注意防止泄漏其输入数据的大小一样，他们也必须注意避免将语义相同的输入编码为不同的位表示形式

以下是其他可以减轻泄漏的设计选择。它们没有在原型中实现。

- *固定的处理时间。*可以通过强制固定的处理时间来消除定时通道，处理时间的长度是在模块看到任何数据之前确定的。操作系统无法直接确定模块何时完成。因此，Ryoan 运行时可以通过忙等待来填充执行时间。但是，在没有操作系统协作的情况下控制其时间安排是一个挑战。对于具有广泛变化的运行时的计算，固定的处理时间可能会非常昂贵，因为所有运行时都会被填充为最坏的情况。但是，对于具有高度可预测的运行时（例如，评估某些机器学习模型（如决策树））或光吞吐量要求低的计算，固定的处理时间可能非常适度。定时执行不会泄漏信息，尽管我们推迟到将来构建支持它的沙盒实例的工作。执行时间也可以是输入长度的固定函数，以增加灵活性而不损失安全性。
- *量化的处理时间。*通过减少潜在处理时间的粒度来减轻处理时间通道。执行被填充到固定数目的量化的，预先定义的值的 [6 ， 83 ， 96 ， 97]。因为 Ryoan 仅允许模块查看敏感数据

一次，单个模块只能泄漏与不同执行持续时间数量的对数成正比的位数（例如，如果代码在八个不同的静态确定间隔之一之后终止，则泄漏三位数）。

3.2 安全初始化

Ryoan 的安全初始化可确保通过特定应用程序中指定拓扑中的真实沙箱实例正确加载模块。*DAG 规范*描述了 Ryoan 应用程序，该规范指定了模块的连接方式（出于安全考虑，请始终使用 DAG，请参阅第 3.1.1 节）。用户可以定义 DAG 规范，也可以明确批准它。

*3.2.1 初始化应用程序。*DAG 规范首先传递给引导隔离区，我们称之为*主控区，*包含 Ryoan 提供的标准，受信任的初始化代码。主服务器要求平台实例化包含针对规范中列出的模块的沙箱实例的安全区。这些飞地可以托管在不同的计算机上。主机使用 SGX 本地或远程证明来验证每个沙箱实例的有效性，然后将其邻居在 DAG 规范中的位置通知沙箱实例。沙箱实例使用适当的不受信任的通信介质（例如，网络或本地进程间通信）作为传输，通过与邻居的密钥交换来建立受密码保护的通信通道。

用户可以通过证明来验证母版的有效性，并询问是否已初始化所需的拓扑。如果是这样，则用户将使用 DAG 的进入和退出沙箱实例建立安全通道，然后开始数据处理。

主机很方便，但对我们的设计不是必不可少的。我们可以改为将 DAG 规范附加到每个用户请求，并让每个沙箱实例在发送输出之前根据该规范验证其邻居的身份。

*3.2.2 Ryoan 身份和模块身份。*SGX 使用处理器硬件证明了沙箱实例，而沙箱实例又利用软件加密法证明了模块的初始状态（图 2 ）。SGX 支持两种形式的身份，一种基于安全区初始状态的哈希（MRENCLAVE），另一种基于公钥，产品标识符和安全版本号（MRSIGNER）。SGX 可以使用任何一种身份验证 Ryoan；我们的原型使用 MRENCLAVE。Ryoan 可以为不受信任的模块支持两种身份的软件类似物；该 protot YP 由公钥迹象发电子邮件识别模块。

*3.2.3 模块初始化。*沙盒实例首先要验证为其提供的要加载的模块与 DAG 规范匹配。成功验证后沙箱实例将通过加载和验证其模块来继续。成功验证的模块允许初始化。初始化时，该模块不受限制可以完全访问香草 NaCl 公开的系统服务。无限制的初始化使模块的创建更加有效，并且使移植更加容易，因为初始化代码可以保持不变。初始化完成后，模块会通过调用 wait_for_work 向 Ryoan 发出信号，由 Ryoan 执行的例程。模块初始化后，将处理一个请求，生成其输出，然后销毁或重置该模块以防止机密数据的积累。

Ryoan 模块验证通过对正在加载的代码施加一组约束来确保模块安全执行。Ryoan 使用 NaCl 的加载时间代码验证器来确保模块的代码遵循严格的格式。NaCl 的代码格式旨在进行有效验证和有效沙盒处理，从而限制控制流目标并将代码与数据完全分开。内存访问被限制为保留在模块占用的地址空间内包括执行提取。的 NaCl 的详细保证是作为现有工作[78 ， 91]和

Ryoan 不会更改 NaCl 沙箱的基本保证。Ryoan 增加了以下限制：模块可能不包含任何 SGX 指令，并且控制流仅限于初始模块代码，即 Ryoan 不允许动态代码生成。

*3.2.4 沙箱实例迁移。*为了平衡服务器利用率，Ryoan 可能会定期重新配置数据处理 DAG 的部署由于 Ryoan 旨在一次处理机密数据，因此它不会维护或迁移持久状态。但是，模块可能会维护持久性数据，例如用于初始化的数据库。Ryoan 不保证模块的持久状态。当决定依赖持久性数据时（例如，使用加密和 MAC 提供回滚保护），模块提供者应考虑与数据提供者和交付系统的信任关系。如果模块存储持久性数据，则在迁移后重新初始化数据时，例如，通过将其存储在新节点上可访问的分布式数据存储中，服务提供商将使该数据对模块可用。

Ryoan 原型仅支持通过关闭处理 DAG 并在一组新节点上重新创建 DAG 而实现的最粗粒度的迁移。如果迁移成为更频繁的操作，则可以通过与主安全区域配合来对其进行优化，例如，仅迁移 DAG 中的某些节点。

3.3 P rotecting 模块提供者的秘密

Ryoan 使用安全标签来防止模块提供者的机密流回用户。从概念上讲，*标签是一组标签，* 其中每个标签都是从庞大的宇宙中提取的不透明标识符，用于标识主体，并指出该主体的秘密。Ryoan 使用公钥作为标签。Ryoan 会将用户的标签分配给用户提供的任何数据。模块二进制文件已签名；加载的模块的标记是正确验证其二进制文件上的签名的公钥。模块提供者可以选择使用不同的密钥对来签名其模块二进制文件，从而实现特权分离。

Ryoan 改编了以前的基于标签的系统，以使多个互不信任的模块可以协同处理敏感数据。龙安标签类似于在 DIFC 系统标签[49 ， 61 ， 68 ， 73 ， 73 ， 86 ， 93]，但简单得多。Ryoan 标签仅用于推断数据的保密性（不完整性），并且具有粗糙的纹理。它们将应用于整个模块及其生成的数据。也可以将 Ryoan 对标签的使用视为污点追踪[18] 在飞地级别的粒度，带有每个主要类别的污点。污渍以工作粒度（在工作单位由应用程序定义）的情况下附加到数据。

*3.3.1 标签操作规则。*每个模块都创建有一个空标签，并能够添加或删除与其主体对应的单个标签-每个模块都可以解密其自身的机密。当模块读取带有非空标签的数据时（例如，从用户或另一个模块的输出中读取），该模块的标签将替换为数据标签和模块的旧标签的并集。Ryoan 用模块的标签标记模块的输出数据。

在图 4 中，来自 Alice 的输入被标记为她的标签，并且第一个 23andMe 模块添加了 23andMe 标签，以确保其秘密在将它们交给 Amazon 的机器学习模块后不会流回用户。由于用户可以控制拓扑，因此此控制至关重要。第二个 23andMe 模块从其输出的数据标签中删除其标签。从某种意义上说，23andMe 的公钥创建了一个组，并且它的两个模块都是该组的成员（由 Ryoan 验证），因为两者均已使用该密钥签名。当 Ryoan 通过受保护且经过身份验证的连接与用户通信时，可以相信 Ryoan 会删除该用户的标签。

*3.3.2 无限制标签。*如果模块的标签不包含其他主体的标签，则该模块不受限制。这样的标签称为*非限制标签。*具有非限制标签的模块可以执行任何文件系统操作，网络通信或地址空间

图4.沙箱实例管理数据和模块上的标签。用户的标签将传播到所有模块，从而在接收到输入后限制它们；例如，当23andMe 的标签外包给 Amazon Machine Learning 时会保留，以防止 23andMe 泄露机密。

Ryoan 和 NaCl 允许修改。例如，它可以通过从网络或文件系统读取内容来自由初始化其状态。Ryoan 允许不受限制地访问外部资源，因为主体的自有标签意味着该模块可能仅从其自身看到了秘密。在 Ryoan 的威胁模型中，每个负责人都信任自己的模块，不要泄露其机密（第 2.1 节），并验证从不可靠来源收到的任何数据。

在许多系统 DIFC [49 ， 61 ， 73 ， 86 ， 93]，主体是独立的应用程序代码，例如，多个用户（主体）使用相同的维基Web应用程序，并且用户不信任应用程序。Ryoan 允许应用程序所有者（模块提供者）成为信任自己的代码的主体，这不同于标准的 DIFC 模型。尽管模块提供者的代码可能存在导致其在其输出中释放其自身秘密的错误，但对于 Ryoan 而言，这不在威胁模型之内，可以使用正交技术加以缓解（第 2.1 节）。 。当数据由不受主体控制的模块处理时，Ryoan 将保护该主体的数据。

模块提供者可以将其模块和机密数据托管在自己的计算机上，以保护它们。但是，如果选择使用不信任的第三方计算平台，则其包含非限制标签的模块需要加密以保护该平台的持久机密。Ryoan 使用 SGX 密封功能代表模块存储秘密数据。密封提供了只有在同一处理器上执行的具有相同标识的安全区域才能访问的加密密钥。对于 Ryoan，所有飞地都包含沙盒实例，并且具有相同的标识。模块要安全保留的所有数据都将传递给 Ryoan，Ryoan 将添加自己的元数据，包括请求模块的公钥。Ryoan 密封数据和元数据并将结果写入文件。

*3.3.3 限制标签。*当模块的标签包含其他主体的标签时（由于从用户或另一个模块的输出中接收了机密），它受到 Ryoan 的限制。这样的标签称为*限制标签。*限定标签表示该模块可能已经看到其他主体的秘密；Ryoan 必须防止模块泄漏这些秘密。

Ryoan 防止带有限制标签的模块保留数据。其结果是，龙安的标签系统是远远超过 DIFC 系统更简单的[49 ， 68 ， 73 ， 86 ， 93]。受限模块已经从其他主体中看到了秘密数据，因此允许它们持久存储违反了 Ryoan 的“一次性”面向请求的数据模型-模块仅处理一次请求。

3.4 优化模块复位

限制模块所需的限制会增加执行时间和内存空间开销。在本节中，我们讨论了减轻这些开销的策略。

*3.4.1 基于检查点的安全区重置。*创建和初始化模块通常比处理单个请求需要更多的 CPU 时间（有关测量，请参见第 6 节）。例如，

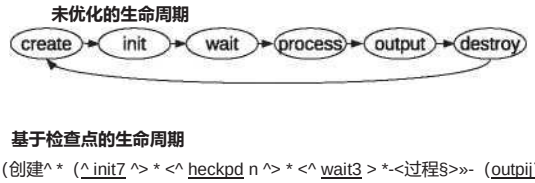

图 5.沙盒实例生命周期：未优化与基于检查点。

加载必要进行病毒扫描的数据需要 24 秒，震级大于订单~0.124s 需要处理一个单一的电子邮件。Ryoan 使用基于检查点的安全区重置来有效管理模块生命周期。

创建和初始化受硬件保护的安全区域的速度很慢（例如，对于一个小型安全区域，我们测量了 30 毫秒）。使问题更加复杂的是，应用程序通常不会基于不经常执行的初始化序列来优化自己的初始化序列。但是，Ryoan 不允许将任何数据从一个输入工作单元结转到下一个，因此每个输入都要求计算从相同的非秘密状态开始，从而使初始化成为瓶颈。

Ryoan 提供了一个检查点服务，该服务使应用程序可以回滚到未污染但已初始化的内存状态（图 5）在我们的原型中，此状态是在 wait_for_work 的第一次调用时。Ryoan 不允许看到秘密输入的区域被检查点，因为它的数据模型是面向请求的：模块不能依靠过去请求中的数据进行操作。对发现秘密数据的模块进行检查点检查（可能）会为该模块在单个请求的工作单元上提供多个执行机会。

通过检查点还原，Ryoan 可以节省拆除和重建 SGX 飞地的成本，并可以节省执行应用程序初始化代码的成本。Ryoan 检查点仅使用一次，但在处理每个请求后将恢复。因此，Ryoan 会制作模块可写状态的完整副本，并仅跟踪页面被修改（在处理过程中避免存储副本）。仅还原在输入处理期间修改的页面的内容（第 4.6 节）。SGX 为安全区代码提供了一种方法来验证页面权限并可靠地通知内存错误，这对于跟踪写入哪些页面是必不可少的。

*3.4.2 重置之前的批处理请求。*用户可能希望通过允许模块在重置之前处理多个输入工作单元来提高效率。在单个请求中批处理多个输入是否构成威胁取决于用户和应用程序。但是，如果一个模块可以为同一数据源处理多个单元，则它可以在多个 waitprocess-output 周期内累积秘密。可以在更长的时间段内访问更多机密数据，加剧了泄漏缓慢（例如，定时通道泄漏）的问题。例如，通过使用多个等待过程输出周期中的处理时间通道，允许未经处理即可处理多封电子邮件的电子邮件过滤模块可能泄漏一封电子邮件中包含的多个密码位。

3.5 良安的密闭环境

任何带有限制标签的模块都在 Ryoan 的限制环境中执行。Ryoan 的密闭环境旨在防止信息泄漏，同时减少移植工作。为了允许在 Ryoan 中使用为通用计算环境开发的代码，受信任的 Ryoan 运行时可以提供向后兼容的服务。当模块接收到请求中包含的机密数据时，它将进入受限环境，并失去通过任何系统调用与不受信任的 OS 通信的能力。因此，Ryoan 提供了足以使大多数传统代码执行其功能而无需修改的系统 API。Ryoan 提供以下服务：

- 最重要的服务是*内存中的虚拟文件系统*。 首先，Ryoan 允许用户将文件预加载到模块内存中，并且必须在限制模块之前确定预加载文件的列表，例如，可以在 DAG 规范中列出这些文件，或者在初始化期间由模块请求。Ryoan 提供了 POSIX 兼容的 API，以访问即使在模块受限后仍可用的预加载文件。其次，一个受限的模块可以创建临时文件和目录（Ryoan 将该文件和目录保存在安全区域内存中）。销毁或重置模块后，所有临时文件和目录都会销毁，并且对预加载文件的所有更改都将还原。
- mmap 调用对于满足动态内存分配至关重要，因此 Ryoan 通过从预分配的内存区域返回地址来支持匿名内存映射。必须在限制模块之前确定区域的最大大小。

Ryoan 的受限环境足以完成许多数据处理任务。例如，流行的病毒扫描工具 ClamAV 在初始化过程中会加载整个病毒数据库。扫描输入（例如 PDF 文件）时，它将创建临时文件来存储从 PDF 中提取的对象。Ryoan 的内存中文件系统满足了这些要求。

但是，如果应用程序需要在处理数据时无法容纳在内存中的大型数据库，则 Ryoan 无法将其作为单个模块支持。一种解决方法是对数据库进行分区，并使用多个模块来加载不同的分区并执行任务的不同部分（如果对特定应用程序可行）。

任何允许访问持久性文件（与 Ryoan 的内存中文件相对）的设计替代方案，都必须处理允许操作系统查看文件读取而创建的秘密通道，这可能是基于不可信模块中的计算而发生的。Ryoan 通过仅从内存执行来消除此通道。所有 Ryoan 模块都必须在整个生命周期中都适合内存，因为 Ryoan 进行的任何“交换”都会在模块和操作系统之间创建隐蔽通道。基于不经意 RAM 文件访问技术（ORAM [59 ， 76]）可以隐藏数据访问模式，但我们认为太高了，龙安性能和资源成本。

3.6 P 从特权软件 rotecting 龙安

沙箱实例需要不受信任的操作系统（可能还包括管理程序）提供的服务。沙盒实例必须检查来自不受信任的操作系统的结果，以确保它不会被误导。这些检查中的大多数都可以透明地插入 libc （与操作系统通信的最低级别的软件）。Ryoan-libc 是 Ryoan 替代 libc，它管理系统调用参数并检查其返回值。Ryoan 沙箱代码通过标准 libc 函数（例如，系统调用的包装器（例如 read ））调用 Ryoan-libc 。SCONE [5]和石墨烯-SGX [85]，还可以通过修改 libc 提供保护。

3.6.1 Iago 攻击。Ryoan-libc 通过将状态保留在安全区内存中并仔细检查系统调用的结果来防范所有已知的 Iago 攻击[14]，例如，确保从 mmap 返回的地址不与先前分配的内存（如堆栈）重叠。对于 Linux，可以通过例如维护安全区内存中的信号量计数以及在安全区内外复制 futex [30]内存来保护系统调用接口。龙安共享用于这种检查与不信任操作系统的所有系统的需求[17 ， 37 ， 50] ，尽管有些检查比系统调用低[7]。

3.6.2 控制安全区的地址空间。SGX 提供用户对内存映射的控制，包括权限。Ryoan-libc 维护的数据结构等效于内核的虚拟内存区域（VMA）列表。它知道每个映射区域及其权限。地图

在请求时（即作为 mmap 调用的一部分），而不是在页面错误时，请求会被 Ryoan-libc 急切地满足并验证。

SGX 规定了验证飞地映射的非常特定的过程。典型的新映射如下进行：（1）不可信代码通过 Ryoan-libc 进行的系统调用将新的所需映射通知内核，（2）OS 选择新的安全区页面框架来满足该映射并修改页面表以具有请求的许可权将帧映射到请求的虚拟地址，（3）恢复不受信任的用户代码并将控制权交给安全区代码，（4）安全区代码通过在每个新页面上调用 SGX 指令 EACCEPT 来验证映射是否按预期完成。该 EACCEPT 指令接受虚拟地址和保护位，并验证当前地址空间是否将该页面映射到有效的，受 SGX 保护的 4KB 物理帧。添加到安全区的新页面始终以读取和写入权限开始，并且其内容由硬件清零。

如果用户需要除读取和写入权限之外的其他权限，则 SGX 提供 EMODPE 指令使它们更具宽松性，而 EMODPR 指令使它们具有较低的宽松性。 EMODPE 仅适用于安全区代码，而 EMODPR 仅适用于特权软件（安全区外部的环 0）。如果某个安全区需要较少的宽松页面访问权限，则它必须向特权软件发出信号以请求该限制，但可以通过另一次使用 EACCEPT 指令来验证该操作是否正确完成。

Ryoan-libc 通过代表用户执行 SGX 所需的工作来模拟 mmap 行为。例如，如果用户希望新页面具有特定内容（例如，他或她私下映射了文件）并且是只读的，则 Ryoan-libc 将文件的请求部分复制到安全区域内存中，并确保这些页面具有返回之前的只读权限。

3.6.3 回滚。特权软件可以回滚任何持久状态。Ryoan 不依赖于持久状态，因此可以通过设计防止回滚攻击。Ryoan 还提供了允许模块提供程序避免依赖持久状态的机制。Ryoan 的初始化仅取决于其初始的内存状态，该状态受硬件保护和证明。所有其他状态均来自硬件随机性，或由沙箱提供程序在运行时安全地提供。

模块可能会使用持久状态，例如在初始化期间看到任何用户提供的机密之前。在图 3 中， Amazon 的机器学习分类器可以加载存储在本地文件系统中 Ryoan 管理的每个应用程序目录中的预先计算的参数。模块提供者应根据与平台提供者（和任何信息提供者）的信任关系，采用适当的加密，散列和回滚保护。

对模块的持久状态保护是模块提供者的责任，就像模块功能/正确性是模块提供者的责任一样。Ryoan 保证，一旦模块看到用户数据，就不会泄漏该数据，但不保证模块根据规范进行操作，例如，模块正确识别垃圾邮件。

3.6.4 飞地不可区分。虽然 SGX 使飞地能够向外部方证明其完整性，但没有什么可以阻止平台实例化飞地的多个副本。Ryoan 通过使用永不持久的密钥在不同安全区域之间以及安全区域与用户之间建立安全通道来防止平台利用此事实，因为密钥永远不会保留，因此必须与每个新安全区域重新协商

4 实施

沙箱实例原型基于 NaCl 版本 2d5bba1，最后一次上游提交于 2016 年 1 月 19 日。我们利用 NaCl 现有的沙箱保证来控制模块对平台的访问。NaCl 确保沙箱中的模块不直接访问 OS 服务。通过引入 Ryoan-libc 层，我们将 NaCl 移植到 SGX 中。氯化钠

取决于 libc 与平台的接口。在检查允许系统调用之后，Ryoan-libc 代表沙箱实例进行系统调用。我们修改了 eglibc 的动态链接器，以支持将 Ryoan 加载到安全区中，但是必须静态链接所有模块。我们将 Ryoan-libc 基于与我们的 NaCl 版本兼容的 eglibc 2.19。

4.1 当前硬件的约束

Ryoan 依赖于 SGX 硬件版本 2 中的功能，而当前仅提供版本 1。版本 2 增加了动态修改安全区的功能，即用新的内存扩展执行中的安全区并更改现有安全区内存的保护。Ryoan 依靠更改内存保护来实现有效的检查点恢复。此外，我们第一代支持 SGX 的计算机只为 SGX 提供了有限的物理内存（我们的计算机上为 128MB）。

4.2 R yoan-libc

Ryoan-libc 管理与不受信任的操作系统的交互。操作系统无法读取安全区内存，因此 Ryoan-libc 将系统调用参数编组到进程的不可信内存中并复制回结果。从插入的 libc 是对于不信任操作系统应用中常见的[17 ， 37 ， 50]，而黑文保护一个较小的系统接口[7]。

*4.2.1 故障处理。*信号允许用户级代码被系统中断。当操作系统不受信任时，大多数信号的来源都是不可靠的，但是 SGX 允许我们获得有关内存故障的可靠信息；这使 Ryoan-libc 可以通过常规的信号处理程序注册界面将此信息公开给沙箱实例。Ryoan-libc 信号支持当前仅限于内存故障信号（SIGSEGV）。

在发生任何故障，异常或中断后，操作系统将控制权返回到进程中包含的不受信任的蹦床代码。在发生内存故障的情况下，我们的蹦床代码进入了安全区，可以从 SGX 读取有关故障的可靠信息，并采取必要的措施来修复内存故障，而不是简单地将安全区恢复到已暂停的位置（在正常情况下）。故障（例如，更改权限）。处理完故障后，安全区域退出，然后我们的蹦床按照导致内存故障的指令恢复安全区域。我们无法从操作系统保护蹦床代码，但只能使用 EENTER 指令进入安全区，这会将控制权转移到我们的故障检查入口点，或使用 ERESUME 指令恢复安全区，该指令将重新执行该指令。错了

4.3 模块地址空间

x86-64 NaCl 为 NaCl 模块分配了一个 84GB 的区域，该模块具有 4GB 的模块地址空间，其上方和下方分别是 40GB 的无法访问的保护页面，但是当前的 SGX 硬件仅允许具有 64GB 虚拟地址空间的飞地。幸运的是，原始的 x86-64 NaCl 设计[78] 高估了允许将来更改体系结构所需的保护页数。详细分析[32]表示，可以通过保持上部保护区域不变但将下部区域从 40GB 减少到 4GB 来保持安全。因此，沙盒实例需要 48GB 的虚拟地址空间，以适合当前的 SGX 硬件。

4.4 I / O 控制

沙箱实例在受限时控制其模块对文件和请求（工作单元）缓冲区的访问，从而防止模块通过直接 syscall 泄漏数据。

*4.4.1 内存中虚拟文件系统。*受限的模块无法访问文件系统，但是 Ryoan 为内存中的虚拟文件（包括预加载的文件和临时文件）实现了 POSIX 兼容的 API。内存文件由一组 4KB 块支持，该块由两级树结构（类似于页表）索引。随着文件的增长，文件的块将按需分配。内存中文件的最大大小为 1GB。内存目录由哈希表支持，我们使用引用计数来跟踪文件的生存期。此虚拟文件系统支持标准 API，包括 open，close，read，write，stat，lseek，取消链接，mkdir ，rmdir 和 getdents 。当模块写入预加载的文件时，沙箱实例将保留原始文件块。重置模块后，预加载的文件将还原为原始版本，并删除临时文件。

*4.4.2 输入/输出缓冲区。*对于每个工作单元，模块调用 wait_for_work（由 Ryoan 实现的系统服务），并且沙箱实例将其所有输入从所有输入通道读取到内存缓冲区中，然后返回模块。处理完工作单元后模块的输出将写入缓冲区，并在下一个 wait_for_work 调用中，沙箱实例在将输出填充或截断为使用输入大小的固定函数计算出的大小后，将缓冲区刷新到输出通道 DAG 规范。该模块使用虚拟文件系统中实现的 API 通过文件描述符访问这些缓冲区，就像使用常规管道或套接字一样。

4.5 飞地之间的关键建立

沙箱实例使用 libsodium [55] 库提供的经过身份验证的加密算法（AES-GCM [65]）实现受保护的通道。加密密钥在运行时使用 Diffie-Hellman 密钥交换达成协议。SGX 允许安全区代码将关键参数嵌入证明中，从而加速了安全区之间的 Diffie-Hellman 密钥交换[41]。在我们的硬件上（第 6 节），SGX 密钥交换花费 1.78ms，而 OpenSSL 花费 1.90ms。密钥交换需要随机性，Ryoan 使用 x86 指令 RDRAND 来获取它。

4.6 检查点限制代码

Ryoan 使用页面权限限制和故障信息来检测模块写入。回想一下，SGX 提供了可靠的内存页面权限和有关内存故障的信息；Ryoan 不信任该操作系统（第 4.2 节）。整个模块在受限状态下都受到操作系统的写保护。Ryoan 验证使用 EACCEPT 进行了保护。在模块写入时，沙盒实例会捕获权限错误并记录有问题的页面的地址，然后再更改权限以允许写入并恢复模块。但是，更新页表中的权限需要 ring-0 特权。沙盒实例的信号处理程序首先在安全区外部执行并进行 mprotect 系统调用以更改页面权限，以避免额外的飞地退出。一旦该过程完成，它将进入安全区模式，以使用 EMODPE 更新 SGX 页面权限（并执行上述记账）。完成后，处理程序将返回并恢复正常执行。

所有已写入的页面均恢复为初始值，并再次变为不可写状态以重置安全区。在我们的原型中，在首次限制不受信任的模块之前，沙盒实例通过复制模块的完整可写内存状态来创建检查点。这种初始化时复制策略可优化以下情况：创建沙箱实例一次，然后将其用于许多请求并进行重置。如果初始化时复制的成本太高，则 Ryoan 可以通过对每个请求执行写入时复制来逐步创建检查点，从而逐渐累积和保留在任何执行期间修改的任何页面的未修改版本。

在我们的原型，检查点取模块时阻塞 wait_for_work ，并恢复在发生下一次在模块块 wait_for_work 。这使模块编写者更加清楚

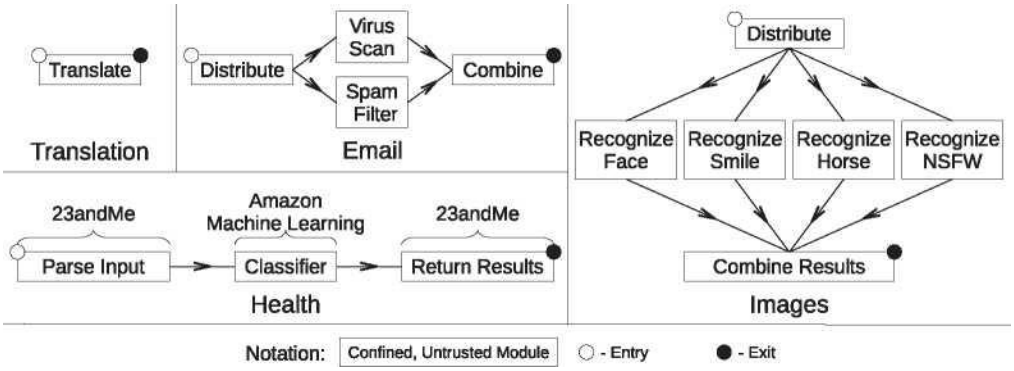

图 6. Ryoan 示例应用程序的拓扑。图中的节点是沙箱实例，尽管我们通过其不受信任的模块来识别它们。用户使用受信任的 Ryoan 代码为源节点和宿节点建立安全通道，以分别提供输入和输出。

关于什么状态的语义不会在调用之间持久存在，并允许沙箱实例清除保存在寄存器中的所有机密。

恢复检查点确实会引起其他页面错误，这些页面错误可以用作泄漏数据的通道。我们发现这些额外的错误是可以接受的，因为即使模块正常的页面访问也是 SGX 不会关闭的模块和 OS 之间的通道[89]页面错误将继续泄漏有关安全区执行的信息，直到下一代的硬件安全区可以为自己的页面错误提供服务（第 2.3 节），或者 SGX 提供了另一个硬件修复程序。为了使 Ryoan 在当前 SGX 硬件上的执行更加安全，我们可以保存/恢复模块的所有可写区域，而不是使用写保护来跟踪单个页面。此策略效率较低，但不会泄漏其他每页信息。

5 使用案例

本节介绍了四种方案，其中 Ryoan 为处理敏感数据提供了以前无法达到的安全级别。对于所有示例，沙箱实例可以在同一平台或不同平台上执行，例如，整个计算可能在第三方云平台（例如 Google Compute Engine）上执行，或者提供程序的模块可能在其自己的服务器上执行。Ryoan 的安全保证适用于所有情况。

5.1 电子邮件处理

公司可以使用 Ryoan 外包电子邮件过滤和扫描功能，同时保持电子邮件文本的机密性。我们考虑使用流行的传统应用程序 DSPAM 3.10.2 和 ClamAV 0.98.7 进行垃圾邮件过滤和病毒扫描。

此服务的计算 DAG 包含四个沙箱实例，每个实例都限制一个数据处理模块（请参见图 6 ）。电子邮件通过安全通道到达入口区域，该区域将电子邮件文本和附件分别分发到包含 DSPAM 和 ClamAV 的区域。病毒扫描和垃圾邮件过滤的结果将发送到最终的后处理区域，该区域将通过安全通道构建对用户的响应。

5.2 个人健康分析

考虑一家公司（例如 23andMe），该公司基于各种健康数据为用户提供自定义的健康报告。23andMe 接受用户的遗传数据，病史和身体活动

日志作为输入，从这些数据中提取重要的健康特征，并预测某些疾病的可能性[1]。由于遗传和健康信息极为敏感，因此用户可能对公司保留其数据不满意。为了鼓励使用该服务，23andMe 可以将其与 Ryoan 一起部署，以确保用户处理数据的代码不会保留或泄露其机密。

23andMe 拥有有关疾病与健康特征之间关联的研究结果，但它可能希望在云中使用第三方机器学习服务（例如，Amazon Machine Learning [3]）以训练其模型并生成预测。23andMe 的商业秘密是如何将用户的复杂，多模式健康数据映射到机器学习功能上。Amazon Machine Learning 提供了一种基于未标记的功能和查询该模型的软件（分类器）来训练模型的方法。在通过这种方式训练模型之后，23andMe 希望保持对分类器的输入的秘密，这些参与者有能力将输入映射回秘密的健康数据：服务的用户。Ryoan 使 23andMe 可以将机器学习任务外包给 Amazon，同时保护其从用户数据到健康功能的专有转换。

用户和 23andMe 的保密性都受到图 4 和 6 所示的 DAG 的保护。23andMe 会编译一个训练数据集，并将其传输到 Amazon 以构建模型。Amazon 提供了分类器，用于查询作为 Ryoan 模块建模的模型。用户在请求中提供其遗传信息，病史和活动日志。收到用户的请求后，23andMe 的第一个模块构造健康特征的布尔向量并将其转发到 Amazon 的模块。亚马逊的模块根据模型生成预测，并将结果转发到 23andMe 的第二个区域，然后将结果转发回用户。

用户的标签会保留在整个流水线中，以便所有飞地在收到用户的输入时都受到限制，并防止泄漏有关输入的信息。此外，23andMe 将其标签保留在发送给亚马逊的请求中，以使 Amazon 不会将有关 23andMe 健康功能的数据泄露给其他方（尤其是用户），因为他们无法删除 23andMe 的标签以将数据释放出 Ryoan 的范围。亚马逊的模块将分类结果传递给 23andMe 拥有的另一个模块，该模块在删除 23andMe 标签并将结果返回给用户之前，验证其专有转换是否未被泄漏。

真正的遗传预测模型是专有的，我们不知道，并且不在本文的讨论范围之内。我们的工作量使用常识和最佳实践。我们培养了支持向量机（SVM），并选择 20 充分研究的疾病和它们的顶部 500 相关的基因，根据 DisGeNet 提供的数据库[33]。使用基于该数据库的综合数据来训练 SVM 模型。我们的原型使用随机梯度下降作为训练算法[10]，该算法允许对现有模型进行增量更新。

5.3 图像处理

图像分类即服务是新兴的领域，可以从 Ryoan 的安全保证中受益（例如 Clarifai [19]或 IBM 的 Visual Recognition 服务[39]）。我们设想了一个场景，其中用户希望根据其专业知识提供不同的图像分类服务。例如，一种服务可能用于准确识别成人内容[62]，而另一种服务可以很好地识别和分割马匹。图 6 中的图像处理 DAG 显示了一个示例，其中图像过滤服务将不同的子任务外包给不同的提供者，然后合并结果。用户的标签将传播到所有处理区域，从而使 Ryoan 限制其执行。我们的原型使用 OpenCV 3.1.0 来实现所有这些检测任务，并且每个检测任务都加载一个专门用于检测任务的模型，这将代表公司的竞争优势。

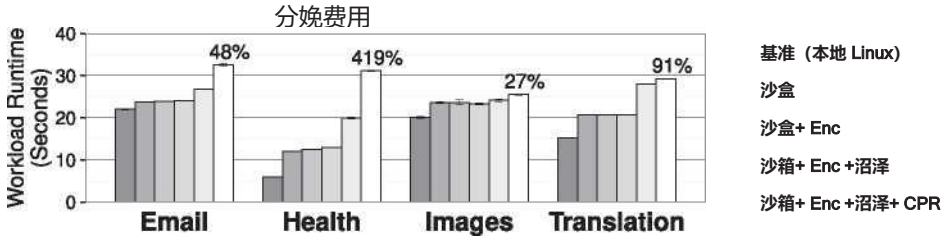

图 7.枚举了 Ryoan 开销的应用程序的运行时。每个条形代表以 95％置信区间标注的 5 个试验的平均值。Ryoan 柱显示相对于本机而言的减速百分比。（Enc：加密； Marsh：系统调用封送； CPR：检查点还原； Ryoan：Sandbox + Enc + Marsh + CPR + SGX）。

5.4 翻译

一家公司使用 Ryoan 提供机器翻译服务，同时将上载的文本保密。用户将文本上载到翻译区域并返回翻译后的文本。我们的原型使用 Moses [67]，这是一种统计机器翻译系统。我们使用针对机器翻译的 2013 年研讨会发布的新闻评论数据集训练基于短语的法语到英语模型 [2]。

6 评估

我们通过结合实际硬件和仿真来评估上一节中描述的用例的执行情况，来量化 Ryoan 及其组件的时间和空间成本。

所有基准测试都是在配备 Intel Core i5-6200U 2.3GHz 处理器（带有 Skylake 微体系结构和 SGX 版本 1）和 4GB RAM 的 Dell Inspiron 7359 笔记本电脑上进行的。我们使用笔记本电脑，因为它包含我们可以购买的第一个支持 SGX 的处理器。但是，我们使用更新的 Intel E3-1270 验证了测量结果（请参阅下面的 SGX 开销分析）。我们使用英特尔的 SGX Linux 驱动程序[43]和 SDK [42]来衡量 SGX 指令的成本。

为了测试我们的实现并克服硬件的限制，我们基于 QEMU [74]（完全仿真模式）构建了 SGX 仿真器，并增加了 SGX 版本 2 指令。为了模拟 SGX V2 的性能，我们根据当前 SGX 硬件的测量结果插入延迟，根据英特尔的 SGX 规范刷新 TLB，并根据 V1 指令的性能估算 V2 指令的开销。我们无法使用 OpenSGX [44]，因为它缺少 64 位信号。我们的仿真器可以运行完整的软件堆栈，包括可识别 SGX 的 Linux 内核。

6.1 了解工作负载性能

图 7 显示了 Ryoan 的各种间接费用来源的细分。基准是运行为本地 Linux 环境构建的应用程序，然后添加沙箱，加密，系统调用封送，检查点还原和 SGX（其中 SGX 开销是模拟和测量的混合，请参见下面的讨论）。表 2 列出了每个工作负载的输入，以及 DAG 中每个模块的详细度量以及重要事件的计数（工作负载在第 5 节中进行了说明）。

6.1.1 输入。工作负载输入被设计为切合实际。电子邮件正文来自垃圾邮件培训集[24]。电子邮件附件是随机附在 30％的电子邮件上的一组 PDF（该数字来自对公司电子邮件特征的研究[34]）。图像是照片，计算机生成的图案和徽标的组合。基因数据是基于 DisGeNet 合成的[33]。翻译文本来自新闻评论数据集[2]。

数字信号处理	19.6	273.5	45.3 兆字节	11.15	22.10	1.29m	1.81m	6k	
ClamAV	21.1	403.9	83.3 兆字节	24.96	29.17	24.7 万	423k	7k	
分类器	19.3	19.4	36KB	0.58	18.23	1.84m	35.9 万	151k	
返回	18.0	18.1	16KB	0.59	6.77	668K	162k	3k	
认出	26.6	27.1	83.2 兆字节	0.63	24.79	88k	17.4 万	6k	
结合	18.0	18.1							

表 2.每个应用程序的输入

应用	输入
电子邮件健康图	250 封电子邮件，其中 30％带有 103KB-12MB 附件 20,000 个 1.4KB 来自不同用户的布尔向量 12 张图像，大小 17KB-613KB 30 个简短的段落，大小为 25-300B，总计 4.1KB

表 3。 对于每个工作负载，报告每个模块一次执行期间的重要事件计数

翻译	25.3	386.9	29.1 兆字节	2.34	26.65	303k	248k	8kk	367	统 通话次数 PF 整数
£ 负荷模型	19.3	19.4	28KB	0.58	12.52	82k	28 万	56k		数数 数数
S 分发	18.0	18.1	11.6 兆字节	0.59	1.32	47k	60k	473		

加载大小：执行前已加载模块的大小；初始化大小：初始化后的模块大小；初始化时间：模块初始化时间。CPU 时间：安全区的处理时间（秒），CPR 大小：在检查点还原时复制/清零的数据，系统。调用：系统调用，PF：页面错误，Int：中断。“图像：识别”报告所有四个图像识别区域的最大值。

*6.1.2 禁闭费用。*在图 7 中， 沙盒和 Sandbox + Enc 开销对于限制是必要的，并且在所有工作负载中，加密不会增加大量开销。对于 Genes 而言，限制开销很高（100％），因为它运行一个非常简单的 SVM 分类器，并且实际数据处理时间很小，这放大了 Ryoan 数据缓冲/填充的影响，并且是最坏的情况。对于图像，工作量涉及使用 OpenCV 进行大量计算，并且限制开销为 18％。

*6.1.3 Checkpoint 还原开销。*表 3 中的 CPR Size 列显示了检查点还原时复制/清零的内存量。图 7（Sandbox + Enc + Marsh 与 Sandbox + Enc + Marsh + CPR 列之间的差异）表明，检查点还原对 Genes 的性能影响很大（55％），因为它的单位工作量最轻（«1ms），并且页面错误处理的相对成本很高；相反，其上的图像的影响仅为 3％，其具有最重的每单位工作负载（∽2 秒）。

*6.1.4 SGX 开销。*在受 SGX 保护的飞地中执行代码会产生一些开销。我们通过使用延迟对 SGX 指令的性能进行建模并在所有安全区出口刷新 TLB 来模拟 SGX 硬件开销（我们无法直接衡量我们硬件上的执行，因为它缺少 SGX 版本 2 功能（第 4.1 节））。除了显式的 EEXIT 指令外，我们还对因异常和中断之类的事件导致的安全区出口进行建模（表 3）。对于每个 EENTER/EEXIT 对，我们测量的硬件延迟为 3.9 ^ s，对于每个 ERESUME/ Async-Exit 对，测量为 3.14 ^ s。

我们还评估了最新，功能更强大的 Intel Xeon E3-1270 v6 3.80GHz 处理器上的 SGX 指令成本。在 Xeon 处理器上，EENTER / EEXIT 对的成本为 2.34 ^ s，而 ERESUME /

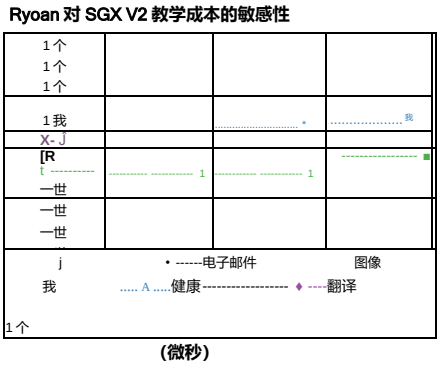

图 8. Ryoan 应用程序工作负载对模拟指令成本的敏感性。垂直虚线表示用于计算图 7 中的 Ryoan 条的延迟（0.39 *g* s）。

异步出口对的成本为 1.85 ^ s。该处理器的时钟速率比笔记本电脑快约 65％，并且 SGX 成本已降低了约一倍。

第 2 版指令 EACCEPT，EMODPE，EMODPR 比 EENTER 和 EEXIT 更简单，因此我们以一对 EENTER / EEXIT 对 的十分之一建模它们的成本。图 8 探索了改变此成本对工作负载的运行时间的影响。例如，如果版本 2 指令的成本与 EENTER / EEXIT 对的成本一样高（3.9 ^ s），那么我们的电子邮件，运行状况，图像和翻译工作负载的运行时间将分别增加 25％，14％，7 ％和 4％。

每个与检查点相关的页面错误都需要一个 EMODPE 来扩展页面权限。在检查点之后还原的每个页面都需要一个 EMODPE 和一个 EACCEPT 。不幸的是，当修改安全区页面状态时，SGX 的版本 2 还强加了额外的同步（通过 ETRACK 的扩展行为）[66]。鉴于我们的应用程序每个安全区只有一个线程，因此我们认为这些性能对这些工作负载的影响可以忽略不计。SGX 执行还需要系统调用封送处理，以将系统调用参数和结果复制到不受信任的内存中，或从中复制，但我们认为封送处理的开销可以忽略不计。所有结果如图所示 7 。

*6.1.5 检查点还原与初始化。*在我们所有情况下，创建一个安全区并加载模块所需的时间少于 0.5s 但表 3 显示 DSPAM 和 ClamAV 的应用程序级初始化时间超过 20s，因为它们需要加载和解析数据库。结果，对于这种工作负载，最好使用 Ryoan 基于检查点的重置，而不是为每个工作单元重新初始化模块。飞地的构造在重新初始化上增加了额外的开销。即使创建小型飞地（例如 298KB）也要付出 30 毫秒的罚款。相比之下，Ryoan 的基于检查点的重置效率更高，单位成本低于 10ms。

6.2 SGX 加密开销

安全区内存在离开处理器时都会被加密。这意味着当处理器从 RAM 读取内存时，需要执行其他操作写入时加密，读取时解密。这些额外的操作增加了最后一级缓存（LLC）丢失的延迟。我们的性能模型中没有与加密相关的性能损失；在这里，我们探讨其成本。

*6.2.1 LLC Microbenchmark 小姐。*为了衡量 SGX 的内存控制器开销，我们使用了一个微基准测试该基准测试对每个高速缓存未命中（读或写）执行固定数量的指令。作为正常流程的一部分，我们执行相同的微基准测试，并与在 SGX 保护的飞地中的执行进行比较。通过在 SGX 飞地中针对不同数量的退休指令（以及针对 LLC 读取或写入未命中）运行微基准所导致的速度下降是

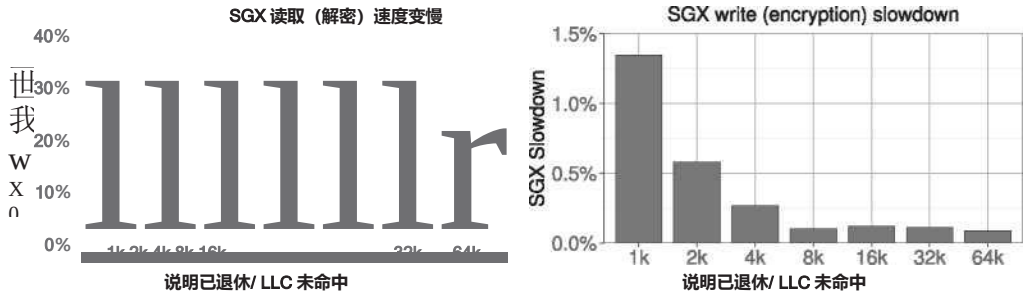

图 9.减速观察到在 SGX 飞地内运行我们的 cache-miss 微基准，而不是在没有 SGX 的情况下运行相同的代码。

表 4. RLL 基准上每个 LLC 小姐的说明

电子邮件	ClamAV	1,260 5,090	32.0%
健康	分类器	14,310 24,650	3.2%
图片	认出	32,760 9,000	1.4%
翻译	（一个模块）	12,560 34,510	3.5%

内存控制器 SGX 减慢是针对 SGX 硬件上等效缺失模式的微基准测得的减慢。

如图 9 所示。 当计算不访问内存（每个高速缓存未命中我们有大量指令）时，安全区代码的性能与非屏蔽执行非常相似。微基准测试不会进行系统调用，并且通过确保在测量开始之前触摸了所有隔离区内存，可以消除页面错误。因此，只有飞地出口是由于中断造成的，它们对总时间的影响微不足道。

由于处理器无需等待内存控制器对数据进行加密，因此写未命中的代价很低。每 1,000 条指令的最后一级写高速缓存未命中，将导致大约 1.4%的执行时间开销。读未命中可能会导致严重的延迟，而具有高读未命中率的程序将在某个区域中缓慢运行。每 1,000 条指令读一次未命中会导致 38.1%的性能开销，一旦每 4,000 条指令发生读未命中，则性能开销将降至 10.1%。处理器通常需要读取的数据才能执行任何有用的工作，因此将停滞等待数据被解密。

为了了解由于 SGX 加密开销而使 Ryoan 基准测试降低的速度，我们测量了每 LLC 丢失所淘汰的指令数量，如表 4 所示。我们专注于支配基准性能的飞地。

“内存控制器 SGX 速度下降”列根据工作负荷 LLC 速率和 LLC 遗漏微基准的测量结果报告了预计的飞地速度下降。电子邮件显示出最大的影响，预计速度会降低 32%，远高于其他基准测试。其他应用程序为每个最后一级的未命中执行大量指令，从而使我们对 SGX 加密开销的估计少于 5%。

7 相关工作

Ryoan 与屏蔽来自不受信任平台的秘密数据的计算（第 7.1 节），防止泄露给未授权方（第 7.2 节）以及通过定时和终止渠道控制泄漏（第 7.3 节）关系最密切。

7.1 施 elding 系统。

屏蔽系统旨在保护在不受信任的环境中处理的机密数据。Ryoan 防御不受信任的环境，但同时也限制了应用程序，因此无需信任它即可维护数据保密性。

*7.1.1 软件屏蔽。*软件屏蔽使用管理程序或编译器来保留在不受信任的平台上执行的计算的隐私性和完整性。掩盖[17]，InkTag [37]，和 SEGO [50] 使用可信任管理程序，以保护受信任应用程序从不可信的操作系统。InkTag 和 Sego 允许受信任的应用程序在管理程序的帮助下验证不受信任的操作系统服务（例如，文件系统）。Virtual Ghost [23] 使用受信任的编译器而非管理程序进行保护。

*7.1.2 硬件屏蔽。*硬件屏蔽使用硬件原语（例如 SGX）来保护计算不受平台软件的侵害。Haven [7] Scone [5]和 Graphene-SGX [85]允许受信任的程序及其库操作系统在 SGX 安全区中执行，以保护其免受主机软件的攻击。VC3 [77] 使用 SGX 保护受信任的 MapReduce。Ryoan 防御平台软件环境，但也限制了应用程序，因此无需信任它即可维护数据保密性。

ARM TrustZone [56] 是另一种可商用的硬件原语，可保护计算不受平台软件的侵害。TrustZone 提供了一个单一的“安全世界”，允许代码以多个特权级别执行。相反，SGX 提供了无限数量的安全区，所有这些安全区都在用户级别执行。TrustZone 当前不加密内存，因此它对物理攻击的抵抗力较小，但是 TrustZone 可以在安全的环境中将页面错误传递给特权代码，从而消除了受控通道攻击[89]。科莫多[27]使用经过正式验证的软件在 TrustZone 之上提供安全区域抽象。要用 TrustZone 替换 SGX，Ryoan 将需要像 Komodo 这样的管理层。

由于 TPM 要求的执行环境受到限制，因此尝试使用后期启动和 TPM 来确保用户安全（例如 Flicker [64]）的可用性很差。后期启动代码无法访问操作系统，并且必须管理裸机。飞地中执行的代码要比后期发布中实际执行的代码复杂。

Ironclad [36] 通过必须包含在每个受信任的二进制文件中的（小型）经过验证的系统堆栈来解决后期启动环境的局限性。Ironclad 不向后兼容，要求用户编写经过验证的代码，这给程序员带来了负担

MiniBox [54] 使用 TPM 和本机客户端来保护应用程序和操作系统之间的相互保护。与 Ryoan 不同 MiniBox 严格使用 Native Client 来保护操作系统及其安全的管理程序，而不是防止应用程序泄漏敏感数据。

对于所有基于 TPM 的系统，计算数据都在内存总线上可见，主机平台的不道德管理员可以在其中窃取它。SGX 安全区数据在通过内存总线之前已加密，从而保留了安全区的保密性。

*7.1.3 加密屏蔽。*同态加密[11 ， 31]允许直接在具有很强的安全保证加密的数据不可信代码来计算。不幸的是，通用同态加密的实际实现不可用，并且当前的开销是令人望而却步的。

属性保留加密（例如，保持顺序的加密[8]）可保护一些计算的保密性[69]，和一些系统使用这些原语[9 ， 72 ， 79]。但是，这些系统的安全保证较弱[35]，仅适用于有限的场景，或者具有显着的性能开销。相比之下，Ryoan 的限制不需要特定于应用程序的领域知识。

7.2 分散信息流控制

分散式信息流控制（DIFC）允许不受信任的应用程序访问机密数据，但可以防止它们将数据泄露给未授权方。然而，大多数系统 DIFC 要求所有的受信任的代码被部署在可信的特权参考监视器下一个集中的平台或管理域[4 ， 49 ， 61 ， 73 ， 86 ， 93]；在浏览器（COWL [81]）和移动设备（Maxoid [90]）中也已经实现了类似的实施。DStar [94] 和 Fabric 是两个例外 [61]，没有集中的参考监视器。但是，尽管 DStar 或 Fabric 用户不需要信任系统中涉及的所有计算机，但他或她必须信任在其上处理其数据的计算机，这意味着正确的参考监视器（操作系统或运行时支持 DIFC）必须正确安装在计算机上，并且该计算机的管理员不使用 root 特权来窃取机密数据。在 Ryoan 中不需要这种信任。

系统，其轨道信息向下流到硬件门级[53 ， 83 ， 84 ， 98]形成用于强信息流的保证，并且通过龙安忽略关闭正时和高速缓存信道的基础。但是，此类硬件不可用，并且按设计不包括新交所提供的隐私和完整性保证。

7.3 钛明和终止

定时和终止信道研究了以前的工作[29 ， 46]中的信息流控制的上下文中。在 Ryoan 中，每个工作单元必须终止一个模块，并且处理时间通道每个单元只能使用一次；不同的单元不会由于模块复位而产生干扰。

8 结论

Ryoan 允许用户使用他们不信任的软件处理数据，在他们无法安全控制的平台上执行，从而使用户，数据处理服务和计算平台受益。为此，Ryoan 通过受信任的沙箱（由 Google 的 NaCl 提供）来限制不受信任的应用程序代码，该沙箱本身通过受硬件安全区域保护的执行（由 Intel 的 SGX 提供）进行了防篡改。Ryoan 还定义并执行了一个执行模型，该模型允许互不信任的软件节点交换数据而不会向彼此或平台提供者透露秘密。我们通过实际应用的各种案例研究来实现和评估 Ryoan 原型。我们基于真实的 SGX 硬件和模拟进行的评估显示，Ryoan 的开销取决于工作量，

致谢

我们感谢 Mark Silberstein，Christopher J. Rossbach，Bennet Yee 和 Petros Maniatis，匿名审稿人以及 Jeff Chase 对这项工作的早期修订发表的评论。我们还要感谢 Shane Williams 提供了有关邮件服务器的背景知识，并感谢 Robert McInvale 探索 SGX 加密开销的工作。

参考

[1] 23andMe 。 2016. 23andMe 比较了家族病史和基因测试对复杂疾病风险的预测。于 2016 年 9 月从 http://mediacenter.23andme.com/blog/23andme-compares-family-history-and-genetic-tests- 预测复杂疾病风险/ 。

[2] ACL。2013。共享的任务：机器翻译。于 2018 年 8 月 23 日从 http://www.statmt.org/wmt13/检索 translation-task.html 。

[3] 亚马逊。2016。亚马逊机器学习。检索 2018 年 8 月 23 日从 https://aws.amazon.com/machine- 学习/ 。

[4] Owen Arden，Michael D. George，Jed Liu，K。Vikram，Aslan Askarov 和 Andrew C. Myers。2012 年。通过信息流控制安全地共享移动代码。在 *2012 IEEE 安全与隐私研讨会（ SP'12 ）的会议记录中*。IEEE 计算机协会，华盛顿特区，191-205。D0I：https：//doi.org/10.1109/SP.2012.22

[5] Sergei Arnautov，Bohdan Trach，Franz Gregor，Thomas Knauth，Andre Martin，Christian Priebe，Joshua Lind，Divya Muthukumaran，Dan O'Keeffe，Mark L. Stillwell，David Goltzsche，David Eyers，Rudiger Kapitza，Peter Pietzuch 和 Christof Fetzer。2016 年。SCONE：使用 intel SGX 保护 linux 容器。在*第十二届 USENIX 操作系统设计和实现研讨会（OSDI'16 ）的会议记录中*。 USENIX 协会，Berkele Y， CA，689-703。http：// dl。 acm.org/citation.cfm?id=3026877.3026930

[6] Aslan Askarov，Danfeng Zhang 和 Andrew C. Myers。2010。时序通道的预测性黑匣子缓解。在*第 17 届 ACM 计算机和通信安全会议（ CCS'10 ）中*。ACM，纽约，纽约，297-307。DOI：https：//doi.org/10.1145/1866307.1866341

[7] Andrew Baumann，Marcus Peinado 和 Galen Hunt。2015 年。利用避风港屏蔽不受信任的云中的应用程序。*ACM Trans。计算 Syst*。33、3，第 8 条（2015 年 8 月），共 26 页。DOI：https：//doi.org/10.1145/2799647

[8] 亚历山德拉 波迪列娃（Alexandra Boldyreva），内森 谢内特（Nathan Chenette），李英豪（Younho Lee）和亚当 奥尼尔（Adam O'Neill）。2009 年保序对称 EN-CR YP 重刑。在*密码学进展：2009 年EUROCRYPT，* 安托万 乔（主编）。柏林，施普林格，224-241。DOI：https：// doi.org/10.1007/978-3-642-22792-9_29

[9] Raphael Bost，Raluca Ada Popa，Stephen Tu 和 Shafi Goldwasser。2015 年学习机分类过 EN-CR YP 特德数据。在*《2015 年网络和分布式系统安全性研讨会（NDSS'15 ）》中*。互联网协会。

[10] Leon Bottou。2016。随机梯度 SVM。于 2018 年 8 月 23 日从 http://leon.bottou.org/projects/ sgd＃stochastic_gradient_svm 检索。

[11] Zvika Brakerski 和 Vinod Vaikuntanathan。2011。来自 ring-LWE 的完全同态加密和密钥相关消息的安全性。在*密码学进展中：CRYPTO 2011*，Phillip Rogawa 等人（编辑）。柏林，史普林格，505-524。DOI：https：//doi.org/10.1007/978-3-642-01001-9_13

[12] ErikBuchanan，RyanRoemer，Hovav Shacham 和 Stefan Savage。2008 年当好说明变坏：泛化面向返回 p 在 AGC 到 RISC。在*第 15 届 ACM 计算机和通信安全会议（CCS'08）的会议记录中*。 ACM，纽约，纽约，27-38。DOI：https：//doi.org/10.1145/1455770.1455776

[13] 克里斯蒂安 卡达尔（Cristian Cadar），丹尼尔 邓巴（Daniel Dunbar）和道森 英格勒（Dawson Engler）。2008. KLEE：非辅助和自动生成的高覆盖测试复杂系统 p rograms。在 *第八届 USENIX 操作系统设计和实现研讨会（OSDI'08 ）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，209-224。

[14] Stephen Checkoway 和 Hovav Shacham。2013 年。Iago 攻击：为什么系统调用 API 是不良的，不可信任的 RPC 接口。在*诉讼Ø ℉ 的体系结构支持第 18 届国际会议的编程语言和操作系统（ASPLOS'13 ）*。ACM，纽约州纽约市，253-264。DOI：https：//doi.org/10.1145/2451116.2451145

[15] 国信摹陈，三川辰，元晓，张 Yinqian，张志强林，十 H.睐。2018. SgxPectre 攻击：通过推测性执行泄漏飞地秘密。arxiv：1802.09085 取自 https://arxiv.org/abs/1802.09085

[16] 陈三川，张晓宽，Michael K. Reiter 和张银千。2017 年。使用 DeJa Vu 在屏蔽执行中检测特权侧通道攻击。在*诉讼Ø ℉ 计算机与通信安全（ASIA CCS'17 ）2017 年 ACM 亚洲会议*。ACM，纽约州纽约市，7-18。DOI：https：//doi.org/10.1145/3052973.3053007

[17] 陈晓欣，Tal Garfinkel，E。Christopher Lewis，Pratap Subrahmanyam，Carl A. Waldspurger，Dan Boneh，Jeffrey Dwoskin 和 Dan RK Ports 2008 年。《阴影》：一种基于虚拟化的方法来对商品操作系统中的保护进行改造。在*第 13 届国际会议上亲架构支持论文集 gramming 语言和操作系统（ASPLOS '08 ）*。ACM，纽约，纽约，2-13。DOI：https：//doi.org/10.1145/ 1346281.1346284

[18] 吉姆 凋，本 普法夫，塔尔 加芬克尔，凯文 克里斯托弗和孟德尔 罗森布拉姆。2004 年。通过整个系统仿真了解数据寿命。在*第 13 届 USENIX 安全研讨会论文集（USENIX Security'04 ）中*。USENIX 协会，加利福尼亚州伯克利，321-336。

[19] Clarifai。2016 年。克拉里菲。于 2018 年 8 月 23 日从 https://www.clarifai.com 检索

[20] Bart Coppens，Ingrid Verbauwhede，Koen De Bosschere 和 Bjorn De Sutter。2009 年。针对现代 x86 处理器上基于定时的边信道攻击的实用缓解措施。在 *2009 IEEE 安全和隐私研讨会（SP'09 ）的会议记录中*。IEEE 计算机协会，华盛顿特区，45-60。DOI：https：//doi.org/10.1109/SP.2009.19

[21] Victor Costan 和 Srinivas Devadas。2016。*英特尔 SGX 解释*。https://eprint.iacr.org/2016/086.pdf

[22] Victor Costan，Ilia Lebedev 和 Srinivas Devadas。2016. Sanctum：最小的硬件扩展，实现了强大的软件隔离。在*诉讼Ø ℉ 25 Usenix 安全专题讨论会（USENIX Security'16 ）*。USENIX 协会，加利福尼亚州伯克利，857-874。

[23] John Criswell，Nathan Dautenhahn 和 Vikram Adve。2014 年。虚拟幽灵：保护应用程序免受恶意操作系统的攻击。在*第 19 届国际编程语言和操作系统的体系结构支持国际会议论文集（ASPLOS'14 ）中*。纽约州纽约市 ACM，81-96。DOI：https：//doi.org/10.1145/2541940.2541986

[24] CSMINING 集团。2016。CSMINING 组：垃圾电子邮件数据集。2016 年 4 月从 https://csmining.org/ index 检索。p hp / spam- email-datasets- .html 。

[25] 太阳能设计师。1997 年。“返回 libc”攻击。*Bugtraq*。

[26] Roger Dingledine，Nick Mathewson 和 Paul Syverson。2004. Tor：第二代洋葱路由器。在*第 13 届 USENIX 安全研讨会论文集（USENIX Security'04）中*。USENIX 协会，加利福尼亚州伯克利，21-21。

[27] Andrew Ferraiuolo，AndrewBaumann，Chris Hawblitzel 和 Bryan Parno。2017 年。科莫多：使用验证从软件中解开安全区域硬件。在*诉讼Ø ˚F 第 26 届研讨会作业系统原理（SOSP'17）*。ACM，纽约，纽约，287-305。DOI：https：//doi.org/10.1145/3132747.3132782

[28] Andrew Ferraiuolo，王瑶，张丹凤，Andrew C. Myers 和 G. Edward Suh。2016 年。晶格优先级调度：共享内存控制器的低开销定时通道保护。在 *2016 年 IEEE 在-论文集 ternationalSymposium 高性能 ComputerArchitecture（HPCA'16）*。382-393。DOI：https：//doi.org/10.1109/ HPCA.2016.7446080

[29] 布莱恩 福特。2012 年。利用定时信息流控制来堵塞侧通道泄漏。在*第四届 USENIX CloudComputing 热门话题研讨会（HotCloud'12）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，24-24。

[30] Hubertus Franke，Rusty Russell 和 Matthew Kirkwood。2002 年。Fuss，futexes 和 furwocks：Linux 中的快速用户级锁定。在*诉讼Ø ˚F 2002 年渥太华 Linux 的研讨会*。479-495。

[31] Craig Gentry。2009。*完全同态加密方案*。博士 论文。斯坦福大学，加利福尼亚州斯坦福市。

[32] Google。2016。NaClSFI for x86-64 的实施和安全性。于 2016 年 9 月从 https：// groups 检索。 google.com/forum/#!topic/native-client-discuss/C-wXFdR2lf8 。

[33] GRIB / IMIM / UPF 综合生物医学信息学小组。2016。DisGeNET 数据库。从 2016 年 2 月从 http://www.disgenet.org/ds/DisGeNET/files/current/DisGeNET_2016.db.gz 检索。

[34] 拉迪卡蒂小组。2009。RadicatiGroup，Inc：电子邮件统计报告 2009-2013（摘要）。从 2018 年 8 月 23 日从 http://www.radicati.com/wp/wp-content/uploads/2009/05/email-stats-report-exec-summary.pdf 检索。

[35] Paul Grubbs，Thomas Ristenpart 和 Vitaly Shmatikov。2017。为什么您的加密数据库不安全。在*诉讼Ø ˚F 在操作系统热门话题第 16 届研讨会（HotOS'17）*。ACM，纽约，纽约，162-168。DOI：https：//doi.org/10.1145/3102980.3103007

[36] 克里斯 霍布利策尔（Chris Hawblitzel），乔恩 豪威尔（Jon Howell），雅各布 R 洛奇（Jacob R. Lorch），阿俊 纳拉扬（Bryan Parno），张丹峰和布莱恩 吉尔（Brian Zill）。2014. Ironclad pp s：通过自动化的全系统验证进行端到端的安全性。在*第 1 届 USENIX 操作系统设计和实现研讨会（OSDI'14）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，165-181。

[37] Owen S. Hofmann，Sangman Kim，Alan M. Dunn，Michael Z. Lee 和 Emmett Witchel。2013。InkTag：在不受信任的操作系统上保护应用程序。在*对建筑苏第 18 届国际会议论文集第 ORT 编程语言和操作系统（ASPLOS'13）*。ACM，纽约，纽约，265-278。DOI：https：//doi.org/10.1145/2451116.2451146

[38] 泰勒 亨特（Tyler Hunt），朱志婷，徐元忠，西蒙 彼得和埃米特 维切尔（Emmett Witchel）。2016 年。Ryoan：用于秘密数据的不可信计算的分布式沙箱。在*诉讼Ø ˚F 上的操作系统设计与实现（OSDI'16）第 12 届 USENIX 研讨会*。USENIX 协会，加利福尼亚州伯克利，533-549。

[39] IBM。2016 年 IBM 视觉 RECO 摹 nition 服务。检索 2018 年 8 月 23 日从 http://www.ibm.com/smarterplanet/us/ EN / ibmwatson / developercloud /视觉 recognition.html 。

[40] 英特尔。2014 年英特尔软件保护扩展编程与摹参考。从 2018 年 8 月 23 日从 https：// software.intel.com/sites/default/files/managed/48/88/329298-002.pdf 检索。

[41] 英特尔。2015 年。《Intel Software Guard 扩展评估 SDK 用户指南：Diffie-Hellman 密钥交换》。从 https://software.intel.com/sites/products/sgx-sdk-users-guide-windows/Default.htm 检索于 2018 年 8 月 23 日。

[42] 英特尔。2016。用于 Linux 操作系统的英特尔软件保护扩展：linux-sgx。https://github.com/01org/linux-sgx 。（提交：d686fb0）。

[43] 英特尔。2016。用于 Linux OS 的英特尔软件保护扩展：linux-sgx-driver。https://github.com/01org/ linux-sgx-driver 。（提交：0fb8995）。

[44] Prerit Jain，Soham Desai，Kim Seongmin，Shiming-Wei，JaeHyuk Lee，Changho Choi，Youjung Shin，Taesoo Kim，Brent Byunghoon Kang 和 Dongdong Su。2016。OpenSGX：SGX 研究的开放平台。在*《2016 年网络和分布式系统安全性研讨会》（NDSS'16）中*。互联网协会。

[45] Min Gyung Kang，Stephen McCamant，Pongsin Poosankam 和 Dawn Song。2011 年。DTA ++：具有目标控制流传播的动态污点分析。在 *2011 年网络和分布式系统安全研讨会（NDSS'11）的会议记录中*。互联网协会。

[46] Vineeth Kashyap，Ben Wiedermann 和 Ben Hardekopf。2011。对时间和终止敏感的安全信息流：探索一种新方法。在 *2011 IEEE 安全与隐私研讨会（SP'11）的会议记录中*。IEEE 计算机协会，华盛顿特区，413-428。DOI：https：//doi.org/10.1109/SP.2011.19

[47] Taesoo Kim，Marcus Peinado 和 Gloria Mainar-Ruiz。2012 年。STEALTHMEM：针对云中基于缓存的侧通道攻击的系统级保护。在*第 21 届 USENIX 安全研讨会论文集（USENIX Security'12）中*。加利福尼亚州伯克利的 USENIX 协会。

[48] Paul Kocher，Daniel Genkin，Daniel Gruss，Werner Haas，Mike Hamburg，Moritz Lipp，Stefan Mangard，Thomas Prescher， Michael Schwarz 和 Yuval Yarom。2018 年。“幽灵攻击”：利用投机执行。arxiv：1801.01203。取自 https://arxiv.org/abs/1801.01203

[49] Maxwell Krohn，Alexander Yip，Micah Brodsky，Natan Cliffer，M。Frans Kaashoek，Eddie Kohler 和 Robert Morris。2007。用于标准 OS 抽象的信息流控制。在 *论文集 ø ˚F 二十一 ACM SIGOPS 研讨会操作 SystemsPrinciples（SOSP'07）*。美国纽约州纽约市 ACM，321-334。DOI：https：//doi.org/10.1145/1294261.1294293

[50] 权荣珍，艾伦 邓恩，迈克尔 李，欧文 霍夫曼，徐元忠和艾米特 维奇尔。2016 年。Sego：普及的可信元数据，用于有效验证不受信任的系统服务。在 *对建筑苏第 21 届国际会议论文集第 ortforProgramming 语言和操作系统（ASPLOS'16）*。ACM，纽约，纽约，277-290。DOI：https：//doi.org/10.1145/2872362.2872372

[51] 巴特勒 W·兰普森。1973 年。关于禁闭问题的说明。*社区 ACM* 16，10 （1973 年 10 月）， 613-615。DOI：https：//doi.org/10.1145/362375.362389

[52] 李尚镐，施明伟，普拉松 格拉，金泰秀，金孝 oon 和马库斯 佩纳多。2017。用分支阴影推断 SGX 飞地内部的细粒度控制流。在 *诉讼 Ø ˚F 第 26 届 USENIX 安全研讨会（USENIX Security'17）*。USENIX 协会，加利福尼亚州伯克利，557-574。

[53] Xun Li，Vineeth Kashyap，Jason K. Oberg，Mohit Tiwari，Vasanth Ram Rajarathinam，Ryan Kastner，Timothy Sherwood，Ben Hardekopf 和 Frederic T. Chong。2014 年。Sapper：一种用于硬件级安全策略实施的语言。在 *诉讼 Ø ˚F 的体系结构支持第 19 届国际会议的编程语言和操作系统（ASPLOS'14）*。纽约，纽约，ACM，97-112。DOI：https：//doi.org/10.1145/2541940.2541947

[54] 李艳琳，乔纳森 麦库恩，詹姆斯 纽索斯，阿德里安 佩里格，布兰登 贝克和威尔 德鲁里。2014 年 MINIBOX：一种双 WA y 沙箱用于 x86 本地代码。在 *2014 年 USENIX 年度技术会议（USENIX ATC'14）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，409-420。http://dl.acm.org/citation.cfm?id=2643634.2643676

[55] 钠。2016 年。libsodium：一个现代且易于使用的加密库。从 https：// github 检索 2016 年 9 月。 com / j edisct1 / libsodium 。

[56] ARM Limited。2009。使用 TrustZone 技术构建安全系统。参考 PRD29-GENC-009492C。

[57] 莫里茨 利普（Moritz Lipp），迈克尔 施瓦兹（Michael Schwarz），丹尼尔 格鲁斯（Daniel Gruss），托马斯 普雷歇尔（Thomas Prescher），维尔纳 哈斯（Werner Haas），斯特凡 曼加德，保罗 科彻（Paul Kocher），德肯尼尔 根金（Dkaniel Genkin），尤瓦 亚罗姆（Yuval Yarom）和迈克 汉堡（Mike Hamburg）。2018.崩溃。arxiv：1801.01207。取自 https://arxiv.org/abs/ 1801.01207

[58] 刘安忆，陈吉姆和哈里 韦克斯勒。2013。网络虚拟环境中的实时秘密定时信道检测。在 *在进展 d igital 取证 IX*，吉尔伯特 Peterson 和 Sujeet Shenoi（编辑）。柏林，史普林格，273-288。DOI：https：//doi.org/10.1007/978-3-642-41148-9_19

[59] Chang Liu，Austin Harris，Martin Maas，Michael Hicks，Mohit Tiwari 和 Elaine Shi。2015 年。GhostRider：用于内存跟踪遗忘计算的硬件软件系统。在 *对建筑苏第 20 届国际会议论文集第 ORT 编程语言和操作系统（ASPLOS'15）*。ACM，纽约，纽约，87-101。DOI：https：//doi.org/10.1145/2694344.2694385

[60] 刘芳菲，吴浩和露比 李。2015 年。随机映射能否保护来自边信道攻击的指令缓存？在 *诉讼第四届国税发 Worksho p 在硬件 andArchitectural Supportfor SecurityandPrivacy （HASP'15 ）*。ACM，纽约，纽约，第 4 条，第 8 页。DOI：https：//doi.org/10.1145/2768566.2768570

[61] 刘杰（Jed Liu），迈克尔 乔治（Michael D. George），维克拉姆（K. Vikram），辛奇，卢卡斯 韦（Lucas Waye）和安德鲁 迈尔（Andrew C. Myers）。2009 年。Fabric：安全的分布式计算和存储平台。在 *诉讼 ACM 国税发 22 号 SIGOPS 学术研讨会操作系摹系统原理（SOSP'09）*。美国纽约州纽约市 ACM，321-334。DOI：https：//doi.org/10.1145/1629575.1629606

[62] Jay Mahadeokar 和 Gerry Pesavento。2016 年。开放采购用于检测 NSFW 图像的深度学习解决方案。于 2018 年 8 月 24 日从 https://yahooeng.tumblr.com/post/151148689421/open-sourcing-a-检索 的深度学习解决方案。

[63] Sinisa Matetic，Mansoor Ahmed，Kari Kostiainen，Aritra Dhar，David Sommer，Arthur Gervais，Ari Juels 和 Srdjan Capkun。2017 年。ROTE：回滚保护，可信任执行。在 *第 26 届 USENIX 安全研讨会（USENIX Security'17）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，1289-1306。

[64] 乔纳森 麦昆（Jonathan M. McCune），布莱恩 帕尔诺（Bryan J. Parno），阿德里安 佩里格（Adrian Perrig），迈克尔 雷特（Michael K. Reiter）和伊佐佐木宏（Hiroshi Isozaki）。2008 年。《忽悠》：实现 Tcb 最小化的执行基础架构。在 *第三届 ACM SIGOPS / EuroSys 欧洲计算机系统会议 2008 （Eurosys'08 ）的会议记录中*。纽约州纽约市 ACM，电话 315-328。DOI：https：//doi.org/10.1145/1352592.1352625

[65] David A. McGrew 和 John Viega。2005 年。伽罗瓦 / 计数器操作模式（GCM）。于 2016 年 9 月从 http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf 检索。

[66] 弗兰克 麦金（Frank McKeen），伊利亚 亚历山德罗维奇（Ilya Alexandrovich），伊泰 阿纳提（Ittai Anati），德罗 卡斯皮（Dor Caspi），西蒙 约翰逊（Simon Johnson），丽贝卡 莱斯利 赫德（Rebekah Leslie-Hurd）和卡洛斯 罗萨斯（Carlos Rozas）。2016 年。英特尔软件保护扩展（Intel SGX）支持在安全区域内进行动态内存管理。在 *诉讼硬件国税发与建筑 Supportfor 安全和隐私 2016（HASP'16）*。ACM，纽约，纽约，第 10 条，第 9 页。DOI：https：//doi.org/10.1145/2948618.2954331

[67] 摩西。2016 年。摩西统计机器学习翻译系统。于 2018 年 8 月 23 日从 http：// www 检索。 statmt.org/moses/ 。

[68] Andrew C. Myers 和 Barbara Liskov。1997.信息流控制的分散模型。在 *诉讼 Ø ˚F 16 日 ACM 研讨会作业系统原理（SOSP'97）*。ACM，纽约，纽约，129-142。DOI：https：//doi.org/10.1145/268998.266669

[69] 穆罕默德 纳维德（Muhammad Naveed），塞尼 卡马拉（Seny Kamara）和查尔斯 Ⅴ 赖特（Charles V. Wright）。2015 年。对保留属性的加密数据库的推理攻击。在 *22Nd ACM SIGSAC 计算机和通信安全会议（CCS'15）的会议记录中*。ACM，纽约，NY，644-655。DOI：https：//doi.org/10.1145/2810103.2813651

[70] 奥尔加 奥里缅科（Olga Ohrimenko），费利克斯 舒斯特（Felix Schuster），塞德里克 富纳特（Cedric Fournet），阿斯塔 梅塔（Aastha Mehta），塞巴斯蒂安 诺沃辛（Sebastian Nowozin），卡皮尔 瓦斯瓦尼（Kapil Vaswani）和曼努埃尔 科斯塔（Manuel Costa）。2016 年不经意多部分 Ÿ 机器学习上的受信任的处理器。在 *第 25 届 USENIXSecurity Symposium（USENIX Security'16）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，电话 619-636。

[71] Dan O'Keeffe，Divya Muthukumaran，Pierre-Louis Aublin，Florian Kelbert，Christian Priebe ，J osh Lind，Huanzhou Zhu 和 Peter Pietzuch 2018 年.spectre-attack-sgx。从 https://github.com/lsds/spectre-attack-sgx 检索。

[72] Raluca Ada Popa，Catherine MS Redfield，Nickolai Zeldovich 和 Hari Balakrishnan。2011. CryptDB：保护机密性与加密 QUER yp rocessing 在 *第 23 届 ACM 操作系统原理研讨会（SOSP'11）的会议记录中*。ACM， 纽约州纽约市， 85-100 。 DOI： https：//doi.org/10.1145/2043556.2043566

[73] 唐纳德 Ɛ 波特，迈克尔 Đ 邦德，英德拉吉 罗伊，凯瑟琳 麦金利和艾米特 维奇。2014。使用层流的实用细粒度信息流控制。*ACM Trans。程序。郎 Syst*。37，1，Article 4（2014 年 11 月），共 51 页。DOI：https：//doi.org/10.1145/2638548

[74] 问：动车组。2016 年。QEMU：开源处理器仿真器。于 2018 年 8 月 23 日从 http://wiki.qemu.org/Main_ Page 检索。

[75] Ashay Rane，Calvin Lin 和 Mohit Tiwari。2015 年。浣熊：通过混淆执行来关闭数字侧渠道。在 *第 24 届 USENIX Secury y Symposium（USENIX Security'15）的会议记录中*。USENIX 协会，伯克利，CA，431-446。http://dl.acm.org/citation.cfm?id=2831143.2831171

[76] 任玲，克里斯托弗 弗莱彻，阿尔伯特 权，埃米尔 斯特法诺夫，史莱恩，马丁 迪克和马丁 戴维达斯。2015 年。常量计数：对遗忘的 RAM 的实际改进。在 *诉讼 Ø ˚F 第 24 届 USENIX 安全研讨会（USENIX Security'15）*。USENIX 协会，加利福尼亚州伯克利，415-430。

[77] 费利克斯 舒斯特（Felix Schuster），哥斯达黎加（Manuel Costa），塞德里克 富纳（Cedric Fournet），克里斯托斯 甘坎迪斯（Christos Gkantsidis），马库斯 佩纳多（Marcus Peinado），格洛里亚 梅纳尔 鲁伊斯（Gloria Mainar-Ruiz）和马克 鲁西诺维奇（Mark Russinovich）。2015 年。VC3：使用 SGX 在云中进行可信赖的数据分析。在 *2015 IEEE 安全与隐私研讨会（SP'15 ）的会议记录中*。IEEE 计算机协会，华盛顿特区，38 -54。DOI：https：//doi.org/10.1109/SP.2015。10

[78] David Sehr，Robert Muth，Cliff Biffle，Victor Khimenko，Egor Pasko，Karl Schimpf，Bennet Yee 和 Brad Chen。2010 年 Adaptin 摹软件故障隔离当代 CPU 架构。在 *第 19 届 USENIX 安全研讨会论文集（USENIX Security'10）中*。USENIX 协会，加利福尼亚州伯克利，1-1。

[79] 贾斯汀 雪莉（Justine Sherry），张兰（Chang Lan），拉露卡 阿达 波帕（Raluca Ada Popa）和西尔维亚 拉特纳萨米（Sylvia Ratnasamy）。2015 年 BlindBox：在 ENCR 深度包检测 YP 特德流量。在 *2015 年 ACM 数据通信特别兴趣小组会议（SIG-COMM'15）的会议记录中*。纽约州纽约市 ACM，电话：213-226。DOI：https：//doi.org/10.1145/2785956.2787502

[80] 施明伟，李尚浩，金泰秀和马库斯 佩纳多。2017 T-SGX：针对飞地根除控制信道攻击 p rograms。在 *2017 年网络和分布式系统安全研讨会（NDSS'17）的会议记录中*。互联网协会。

[81] Deian Stefan，Edward Z. Yang，Petr Marchenko，Alejandro Russo，Dave Herman，Brad Karp 和 David Mazieres。2014 年保护管理克用户通过与 COWL confiningjavascript。在 *第 11 届 USENIX 操作系统设计和实现研讨会（OSDI'14）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，131-146。

[82] Raoul Strackx 和 Frank Piessens。2016。Ariadne：状态连续性的最小方法。在 *第 25 届 USENIX 安全研讨会（USENIX Security'16）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，875-892。

[83] Mohit Tiwari，Xun Li，Hassan MG Wassel，Frederic T. Chong 和 Timothy Sherwood。2009 年。执行租约：一种硬件支持的机制，用于实施强互不干扰。在 *第 42 届年度 IEEE / ACM 国际微体系结构研讨会（MICRO 42）的会议记录中*。ACM，纽约，NY，493-504。DOI：https : //doi.org/10 。1145 / 1669112.1669174

[84] Mohit Tiwari，Jason K. Oberg，Xun Li，Jonathan Valamehr，Timothy Levin，Ben Hardekopf，Ryan Kastner，Frederic T. Chong 和 Timothy Sherwood。2011 年。设计具有严格且可证明的信息流安全性的可用微内核，处理器和 I / O 系统。在 *第 38 届年度国际计算机体系结构研讨会（ISCA'11）的会议记录中*。ACM，纽约，纽约，189-200。DOI：https：//doi.org/10.1145/2000064.2000087

[85] 蔡家 Che，唐纳德 Ɛ 波特和蒙娜 维伊。2017 年。Graphene-SGX：实用的库 OS，适用于 SGX 上未经修改的应用程序。在 *2017 年 USENIX 年度技术会议（USENIX ATC'17）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，645-658。

[86] 史蒂夫 Vandebogart，佩特罗斯 Efstathopoulos，埃迪-科勒，麦克斯韦·克罗恩，克里夫·弗雷，大卫·齐格勒，弗兰斯·卡肖克，罗伯特莫里斯，和戴维·马齐尔斯。2007。石棉操作系统中的标签和事件过程。*ACM Trans*。*计算Syst*。25、4，第 11 条（2007 年 12 月）。DOI：https：//doi.org/10.1145/1314299.1314302

[87] 吴振宇，张旭和王海宁。2015。在超空间低语：云内部的高带宽和可靠的隐蔽通道攻击。*IEEE / ACM Trans*。*网络*23,2（2015 年 4 月），603-614。DOI：https：//doi.org/10.1109/TNET。2014.2304439

[88] 徐云晶，迈克尔·贝利，法纳姆·贾哈尼安，考斯图·乔希，马蒂·希尔顿和理查德·施利希廷。2011。探索虚拟化环境中的 L2 缓存隐蔽通道。在*诉讼国税发 3rdACM Worksho p 关于云计算安全研讨会（CCSW'11）*。ACM，纽约州纽约市，29-40。DOI：https：//doi.org/10.1145/2046660.2046670

[89] 徐元忠，崔卫东和马库斯·佩纳多。2015。受控通道攻击：不可信操作系统的确定性副通道。在*2015 年IEEE 安全与特权研讨会（SP'15）的会议记录中*。IEEE 计算机协会，华盛顿特区，640-656。DOI：https：//doi.org/10.1109/SP.2015.45

[90] 徐元忠和埃米特·维奇尔。2015 年。Maxoid：使用自定义状态视图透明地限制移动应用程序。在*第十届欧洲计算机系统会议（EuroSys'15）的会议记录中*。ACM，纽约，纽约，第 26 条，第 16 页。DOI：https：//doi.org/10.1145/2741948.2741966

[91] Bennet Yee，David Sehr，Gregory Dardyk，J。Bradley Chen，Robert Muth，Tavis Ormandy，Shiki Okasaka，Neha Narula 和 Nicholas Fullagar。2009。本机客户端：用于便携式，不受信任的 x86 本机代码的沙箱。在 *2009 IEEE 安全与特权研讨会（SP'09）的会议记录中*。IEEE 计算机协会，华盛顿特区，79-93。DOI：https：//doi.org/10.1109/SP.2009.25

[92] Tatu Ylonen 和 Chris Lonvick。2008。RFC 5246：传输层安全（TLS）协议：版本 1.2。从 2018 年 8 月 23 日从 https://tools.ietf.org/html/rfc5246 检索。

[93] Nickolai Zeldovich，Silas Boyd-Wickizer，Eddie Kohler 和 David Mazieres。2006 年。在 Histar 中明确显示信息流。在*诉讼 7 日国税发研讨会操作系统的设计与实现（OSDI'06）*。USENIX 协会，加利福尼亚州伯克利，263-278。

[94] Nickolai Zeldovich，Silas Boyd-Wickizer 和 David Mazieres。2008。通过信息流控制保护分布式系统。在*第五届 USENIX 网络系统设计与实现研讨会（NSDI'08）的会议记录中*。USENIX 协会，加利福尼亚州伯克利，293-308。

[95] 张超，陶炜，陈兆丰，段蕾，拉斯洛·塞克斯，斯蒂芬·麦卡曼特，黎明之歌和邹维。2013。实用的控制流完整性和二进制可执行文件的随机化。在*2013 IEEE 安全与隐私研讨会（SP'13）的会议记录中*。IEEE 计算机协会，华盛顿特区，559-573。DOI：https：//doi.org/10 1109 / SP.2013.44

[96] Zhang Danfeng，Aslan Askarov 和 Andrew C. Myers。2011。交互式系统中时序通道的预测性缓解。在*第 18 届 ACM 计算机和通信安全会议（CCS'11）的会议记录中*。ACM，纽约，纽约，563-574。DOI：https：//doi.org/10.1145/2046707.2046772

[97] 张丹凤，阿斯兰·阿斯卡罗夫和安德鲁·迈尔斯。2012。基于语言的控制和缓解计时渠道。在*诉讼 Ø ℉ 对编程语言设计与实现（PLDI'12）第 33 届 ACM SIGPLAN 会议*。ACM，纽约，纽约，电话：99-110。DOI：https：//doi.org/10.1145/2254064.2254078

[98] 张丹凤，王瑶，G。Edward Suh 和 Andrew C. Myers。2015。一种对时序敏感的信息流安全性的硬件设计语言。在*诉讼 20 国税发国际会议上进行编程与架构支持摹语言和操作系统（ASPLOS'15）*。ACM，纽约，纽约，503-516。DOI：https：//doi.org/10。1145 / 2694344.2694372

[99] Zhang Yinqian，Ari Juels，Michael K. Reiter 和 Thomas Ristenpart。2012 年。跨 VM 辅助通道及其用于提取私钥的功能。在 *2012 年 ACM 计算机和通信安全会议（CCS'12）的会议记录中*。ACM，纽约，纽约，305-316。DOI：https：//doi.org/10.1145/2382196.2382230

2018 年 2 月收到; 2018 年 6 月接受