然后还有更多：
多于两个方的安全通信

戴维·奈洛（David Naylor）*†，李察（Richard Li）*克里斯托斯·甘坎迪斯（Christos Gkantsidis）*，托马斯·卡拉吉尼斯（Thomas Karagiannis）*，彼得·斯汀基斯特（Peter Steenkiste）*

摘要

今天的互联网通信通常涉及中介*中期 dleboxes* 像高速缓存，压缩代理，或病毒扫描。不幸的是，随着加密变得越来越普遍，这些中间盒变得盲目，我们失去了它们的安全性，功能和性能优势。尽管行业和学术界都做出了初步努力，但我们仍不确定如何将中间盒集成到安全会话中，甚至还不清楚如何 在这种多实体环境中*定义*“安全”。

在本文中，我们首先描述了安全多实体通信协议的设计空间，重点介绍了互不兼容的属性之间的权衡。然后，我们以现有协议*无法*满足的实际需求为目标，例如*将中间盒外包到不受信任的基础架构中，并支持传统客户端*。我们提出了一个安全定义，并提出了提供此协议的中间盒 *TLS（mbTLS）*（部分使用 Intel SGX 来保护中间盒不受不可信硬件的攻击）。我们证明了 mbTLS 可以在今天部署并且几乎没有开销，并且我们描述了构建简单的 mbTLS HTTP 代理的经验。

CCS 概念

•安全性和隐私性安全协议；*可信计算*；**•网络会话协议**；**中间箱/网络电器**;

关键词

TLS，中间盒，可信计算，SGX

ACM 参考格式：

David Naylor，Richard Li，Christos Gkantsidis，Thomas Karagiannis 和 Peter Steenkiste。2017 年。还有更多：超过两个缔约方的安全通信。在*诉讼 ofCoNEXT '17：新兴的网络试验和技术，仁川，韩国，12 月 12 日至 15 日，2017 年（CoNEXT '17），第 13 届国际大会* 13 页。DOI：10.1145 / 3143361.3143383

1 引言

互联网通信不再是两个端点在愚蠢的数据包转发核心上交换消息。我们的数据经常由中介中间*盒*（例如缓存，压缩代理，入侵检测系统或病毒扫描程序）处理。例如，美国所有四个主要的移动运营商都使用 HTTP 代理[55]，而典型的企业网络具有与路由器和交换机一样多的中间盒[47]。随着在线加密的使用增加（截至 2014 年，几乎所有 Web 流中的一半使用 HTTPS [40]），这些中间盒变得盲目，无法再执行其工作，这促使学术界和行业界都在考虑以下问题：*我们如何将中间盒集成到安全的通信会话中？*

因为 TLS（Internet 中使用的标准安全通信协议）是专为两方设计的，所以当前的做法是将会话“拆分”为两个单独的 TLS 连接：中间盒将服务器模拟为客户端，并打开与之的第二个连接。服务器。这样做会极大地削弱安全性，部分原因是客户端无法显式认证中间盒或确保中间盒将正确认证服务器[23]。

这些弱点凸显了以下事实：虽然在两方情况下 TLS 的属性已得到很好的理解，但尚不清楚如何在多方情况下定义“安全”。最近的工作提出了新的协议以及新的安全性定义。例如，多上下文 TLS（mcTLS）[41] 允许端点限制中间盒可以读取或写入数据流的哪些部分，而 BlindBox [48] 允许中间盒直接对加密数据进行操作。但是，这些主要是点解决方案，尽管在某些情况下很有用，但仍未满足一些实际需求。

在本文中，我们集中于三个尚未解决的实际要求。首先，在中间盒外包第三方云供应商越来越多的兴趣[2，11，31，47]或向 ISP [5，9，12]。此设置提出了一个新的挑战：中间盒软件的所有者和运行它的硬件的所有者是不同的。如果基础架构不受信任，则现有协议（如“split TLS”和 mcTLS）无法提供 TLS 如今提供的标准安全属性，因为（1）会话数据和密钥在内存中可见，并且（2）端点无法告知基础架构提供商是否运行了预期的代码。其次，为了切实可行地进行部署，任何新协议都应与 TLS 反向兼容（即，升级后的端点应能够在与旧版 TLS 端点的会话中包括中间盒）。第三，在许多情况下，实时发现中间箱的能力是一项实际要求。例如，如果服务提供商将代理放置在边缘 ISP 中，

David 和 Richard 在 Microsoft Research 实习期间完成的工作。

CoNEXT '17，韩国仁川
©2017 ACM。978-1-4503-5422-6 / 17/12 ... 15.00 美元
DOI：10.1145 / 3143361.3143383

在本文中，我们做出了两个主要贡献。首先，我们仔细阐明**安全多实体通信协议**的**设计空间**（并将其用于放置先前的工作）。我们描述了这样一个协议可能具有的不同属性，并指出了为什么无法同时实现某些组合的原因，这表明社区需要仔细选择要支持的属性集，或者针对不同的用例开发不同的协议。其次，我们介绍 ***Middlebox TLS（mbTLS）***，*这*是一种用于安全多实体通信的协议，可以满足上述需求：

*(1) mbTLS 可立即部署。*与 mcTLS 或 BlindBox 不同，它可与旧版 TLS 端点互操作并提供带内中间盒发现。

*(2) mbTLS 保护来自第三方基础结构提供程序的会话数据。*mbTLS 杠杆可信计算技术，如英特尔 SGX [15，26，37]，以从第三方基础设施的中间件执行环境隔离。

*(3) mbTLS 提供了多方设置独有的其他重要安全属性。*例如，mbTLS 保证数据按端点指定的顺序访问中间盒，防止攻击者在转发数据之前了解中间盒是否修改了一段数据，并向端点保证中间盒正在运行什么代码。

我们使用 OpenSSL 和 Intel SGX SDK 实施了 mbTLS，并评估了其可部署性和性能，显示了（1）mbTLS 可立即部署，（2）mbTLS 减少了中间盒上的 CPU 负载并增加了服务器上的合理开销，以及（3）运行 SGX 飞地内部的数据不会降低吞吐量。

我们希望 mbTLS 代表着弥合端到端安全与中间盒不会消失的现实之间的重大而实际的一步。

2 多方通讯

如今，大多数网络通信会话都涉及更多的参与者，而不仅仅是客户端和服务器。总的来说，这些额外的实体属于以下三类之一：

网络层*中间***盒***（例如，防火墙，NAT，第 3 层负载平衡器）*。这些数据包一个接一个地处理，不需要重建或访问应用层数据。

应用程序层中间盒*（例如，病毒扫描程序，IDS，父母过滤器，缓存，压缩代理，应用程序层负载平衡器）*。这些*确实*需要访问应用程序层数据。

应用层代表*（例如 CDN）*。与*在通信时*充当客户端和服务器之间的中介的中间盒相反，我们使用术语"*委托*"来指代在会话期间充当服务器角色的中介（尽管就现实世界中的关系而言，它们更自然地被视为中介）。内容交付网络（CDN）是一个很好的例子；客户直接与 CDN 服务器对话；原始服务器可能根本不参与。

当我们走向加密普遍存在的互联网时，很明显，我们没有足够的协议来进行安全的多实体通信，我们也不知道到底是什么应该提供的属性。在两方的情况下，我们很了解我们想要什么安全属性以及如何实现它们；我们已经使用 TLS 多年了。但是，在多方情况下，仍然存在两个未解决的关键问题：*（1）对于涉及三个或更多方的会话，应保留哪些安全属性？*和*（2）什么是强制执行这些属性的最佳机制？*

对于这三个中介类别，这些问题的答案将有所不同。在本文中，我们专注于*针对**应用层中间盒的**安全多实体通信*。即使在仅应用程序层中间盒中，安全需求也可能存在差异-例如，入侵检测系统和压缩代理的行为非常不同，管理员指定的病毒扫描程序和选择加入的压缩服务之间的信任关系也有所不同-这表明可能没有是一个单一的 hts-all 解决方案。我们回答这些问题的第一步是阐明设计空间。

2.1 设计空间

TLS 安全属性。TLS 当前在两方情况下提供以下属性。显然，我们也希望在多方案例中使用这些属性，但是事实证明，有多种方法可以将两方定义扩展到多方案例。

*数据保密*和*数据认证。*仅端点可以读取和写入（创建，修改，删除，重放或重新排序）会话数据。此外，使用现代化的密码套件，通信是*前向机密的*（长期私钥的妥协并不能帮助攻击者访问先前会话的数据）。为了将这些属性扩展到两方之外，出现了以下问题。

*数据访问的粒度。*是/否 RW / RO /无功能。加密货币
中间盒是否具有对会话数据的完全访问权限，或者它们具有某种程度的部分访问权限？这可能意味着它们可以读取/写入某些字节，但不能读取/写入其他字节（例如，HTTP 标头，但不能读取 HTTP 正文），如 mcTLS [41]一样，或者它们可以对加密数据执行一组有限的操作（例如，搜索模式）），如 BlindBox [48]所示。

*"聚会"的定义。*机器程序
将参与方添加到会话后，对计算机有物理访问权或仅对中间盒服务软件具有访问权限的任何人都可以访问会话数据吗？当中间盒外包给第三方硬件（例如，云提供商或 ISP）时，这一区别就变得很重要。

*实体认证。*每个端点可以通过验证它们是否拥有由 CA 认证的属于该实体的私钥来验证对方是否由预期实体操作。为了将此财产扩展到两方之外，出现了以下问题。

*"身份"的定义。*所有者代码
当会话中的一方验证另一方的"身份"时，它在检查什么？机器是否属于预期实体（例如，这是 YouTube 服务器）？机器是否正在运行预期的软件并已正确配置（例如，仅启用了强 TLS 密码套件的 Apache v2.4.25）？两个都？

其他安全属性。在多方情况下，会出现许多新的安全属性。

路径完整性。 是的没有
协议是否强制数据必须以固定顺序遍历中间盒（并且不能跳过中间盒）路径顺序会影响安全性，尤其是在中间盒执行过滤/清理功能时。

数据更改保密性， 无值+大小
对手能否通过观察中间盒前后的数据来了解有关交流的任何信息？协议不能提供任何保护（对手随时知道中间箱进行更改），*值*保护（中间箱更改消息时只要大小保持不变，对手就不会学习）或*值+大小*保护（对手不会了解任何变化）。

授权。 0 个端点 1 个端点两个端点+ mboxes
F .... ■■■HF ... --HF ..... »■■■■ ..... 一世
谁可以在会话中添加中间盒（并确定其具有哪些权限）？是否需要使两个端点都知道，以便它们在不批准的情况下可以终止会话？只有一个？中间盒应该知道其他中间盒吗？

其他属性。最后，还有许多非安全属性会影响协议的可部署性和可用性。

旧版端点。 都升级 1 旧版都升级
两个端点都需要升级到新协议，还是一个或两个都是旧式 TLS 端点？

带内发现， 是是+ 1 个 RTT 否
该协议是否允许端点即时发现路径上的中间盒？如果是这样，添加发现的中间盒是否会增加握手时间？

计算。 任意限制
协议是否限制了中间盒可以执行哪些类型的作业（即，任意计算与有限的一组操作（如模式匹配））？

2.2 设计权衡

接下来，我们在设计空间的背景下研究现有方法，重点介绍它们引入的机制如何与上述属性交互。通常情况下，提供了一个特定的属性（表示像的机构这沿着一个维度）往往消除沿另一选项（表示像这样）。

TLS 拦截与自定义根证书[23，27，42]是当今的标准方法。首先，管理员为客户提供自定义的根证书（这在公司网络等托管环境中很容易）。然后，当客户端打开新连接时，中间盒将拦截该中间连接，通过为该域构造证书来模拟目标服务器，并打开与该服务器的第二个连接。虽然这意味着两个客户端都可以是旧版 TLS 客户端*[Legacy：* 两个旧版*]*，但是这也使客户端无法对服务器进行身份验证*[Authentication：* ~~own e r~~ *]*-他们必须信任中间盒才能这样做（这种信任在实践中经常放错位置[23]）。

多上下文 TLS（mcTLS）[41]提供*访问*控制端点，可以限制数据中间盒的哪些部分可以访问，以及访问是读/写还是只读[数据*访问：* RW / RO /无]。它通过使用不同的密钥加密数据的不同部分，并仅为中间盒提供某些密钥来实现此目的。这要求两个端点运行 mcTLS，排除传统的端点，因为传统的 TLS 端点只知道做什么用一个键做*[遗产：* ~~1 升 Ë 掌一立方码都升 é 掌一立方码~~ *]*。此外，每个端点都会为这些密钥中的每个密钥生成一部分密钥材料，从而确保只有在两个端点都同意*[授权：* 两个端点*]*的情况下，中间盒才能获得访问权限。这也阻止了传统支持。

BlindBox [48]提供*可搜索的*加密-模式匹配中间盒（如入侵检测系统）可以直接对加密数据进行操作[数据访问：func。加密]。但是可搜索的加密仅适用于模式匹配。它不能支持其他中间件，像压缩代理，执行任意的计算*[计算：* ~~一个 rbitr 一个 RY~~ *]*。它也要求两个端点使用 BlindBox 的加密方案*[遗产：* ~~1 升 Ë 克一个 CY 两个升 Ë 克一个 CY~~ *]*。

中间盒 TLS（mbTLS）*（本文）*。我们也很快会在 mbTLS 中看到这一点：例如，mbTLS 对会话中的每个“跳”使用不同的对称密钥，从而允许 mbTLS 提供路径完整性*[路径完整性：* 是*]*，但无法支持*两个*旧式端点*[遗留：* ~~两个升 é 克一个 CY~~ *]*。

收获是：***没有一种适合所有人的解决方案，可以与应用程序层中间盒进行安全通信***。这里的每个协议为了提供其他协议都放弃了期望的属性。不同的属性，因此不同的协议，将最适合不同的用例。例如，mcTLS 是只读中间盒的理想选择，因为它的访问控制机制提供了密码保证，该中间盒不会修改数据。对于 IDSes 这样的模式匹配中间盒，与 mbTLS 或 mcTLS 相比，BlindBox 提供了更好的隐私保证。不幸的是，mcTLS 和 BlindBox 使用难以部署的机制来实现这些属性。在本文中，我们的目标是优先考虑可部署性和可操作性的协议。

3 MIDDLEBOX TLS

§2.2 中介绍的解决方案 可满足特殊需求，但无法满足一些常见需求，因此使其更难以部署，并且一开始就降低了这样做的动力。在本节中，我们提出***中间件 TLS***，或***mbTLS***，对于包括应用层安全中间件通信会话的协议。我们在 §2.2 中看到，很难构建一个融合了 §2.1 中所有良好功能的超级协议。相反，我们针对以下三种常见的实际需求：

*(1) 立即可部署性：*首先，mbTLS 需要**与旧式端点互操作**。BlindBox [48] 和 mcTLS [41]要求两个端点都进行升级。其他协议要求至少所述客户端被升级[33，34，36]，这意味着服务器可以不包括结合遗留客户端的会话中间件。然而，实际上，等待互联网中的每个客户端都升级是不可行的，特别是因为多达 10％的 HTTPS 连接*已经* 被拦截[23]。二，**带内中间盒发现**对于我们针对的用例进行实际部署非常重要。例如，假设服务提供商放置代理

在边缘 ISP 中。使用 DNS 将客户端定向到其本地代理（1）是不必要的配置负担，而（2）则很脆弱，因为客户端可以使用非本地 DNS 解析器，例如 OpenDNS。另一个示例是访客网络，在该网络中，管理员无法可行地配置所有加入的客户端设备（这些用户也不希望它们加入）。

*(2) 保护外包*中间盒：在第三方环境中部署中间盒的兴趣日益浓厚。这采用两种形式之一。首先，网络的功能可以被外包给云提供[R[1]，专门操作中间件，从学习操作专门盒和利用规模经济以降低成本释放网络管理员[2，11，47]。第二，在客户机的 ISP 部署中间件可以帮助降低等待时间或带宽成本[5，9，12]。（例如，Google 的边缘网络使用客户端 ISP 中的节点来代理连接[5]。）在两种情况下，*网络功能的逻辑所有者与运行该功能的硬件的运营商是不同的。*由于可能不信任中间盒基础结构，因此 mbTLS 除了传统的网络攻击者之外，还必须**保护会话数据不受中间盒基础结构的**攻击。

(3) 中间盒*问责制：如果*端点能够保证其行为，则让中间盒访问它们的数据可能会更舒适。mcTLS 和 BlindBox 在一定程度上提供了此功能，但是 BlindBox 仅支持模式匹配，并且在 mcTLS 中，一旦对中间盒授予了数据访问权限，它就可以做任何想要的事情。第一步，mbTLS 允许端点**验证中间盒代码身份**；将来，结合程序分析，可以为中间盒行为提供保证。

3.1 威胁模型

派对。为了捕获这些要求所隐含的威胁，我们确定了六个不同的参与方，并将每个参与方标记为“可信”或“不可信”，其中可信表示已授权读取和写入会话数据。

*客户端（C）[受信任]：*用户，他们的机器以及他们运行的软件（例如，网络浏览器）。我们假定计算机上运行的任何其他软件都是受信任的（即其行为超出范围）。

*服务提供商（S）[可信任]：*提供在线服务的公司，其服务器及其运行的软件（例如 Web 服务器）。我们不考虑由 S 服务器上运行的其他软件或恶意员工发起的攻击。

*第三方（TP）[不受信任]：*有权访问网络流量的其他任何人，例如 ISP 或咖啡馆 Wi-Fi 嗅探器。

中间盒软件（*MS*）*[受信任]：*处理会话数据的中间盒软件。

中间盒服务*提供商（MSP）[可信任]：*提供中间盒服务的实体。

*中间盒基础结构提供程序（MIP）[不受信任]：*提供运行 *MS* 的硬件的实体。

该 *MIP* 可能是 *MSP* 本身或第三方，如边缘 ISP 或专用云中间件服务，在这种情况下，我们认为这家公司，其员工，其硬件，并在其计算机上运行的其他任何软件都*不会*信任。例如，假设 Google 使用在 Amazon EC2 上运行的 Apache httpd 实现了 Flywheel 代理[14]。在这种情况下，MS = Apache httpd，*MSP* = Google 和 MIP = Amazon。通过区分 MS 和 MIP，我们可以要求 mbTLS 来允许 MS（而不是 MIP）访问会话数据*[Party：* program]

对手能力。我们假设一个活跃的全球对手可以观察和控制系统的任何不受信任的部分。在网络中，对手可以观察，修改或丢弃任何数据包并注入新数据包。在中间盒基础结构上，对手可以完全访问所有硬件（例如，它可以读取和操纵内存）和软件（例如，它可以执行任意代码，包括特权代码，例如恶意 OS）。这包括修改或替换要由 MIP 执行的由 MSP 发送的 MS 代码的能力。我们假设对手在计算上受到限制（即，不能破坏标准的加密原语），并且不能损害可信的计算硬件（例如，启用了 Intel SGX 的 CPU）。旁道攻击（例如，基于流量或缓存访问模式），中间盒软件中的可利用漏洞，

3.2 mbTLS 属性

根据上面的设计要求，我们通过以下四个安全属性为 mbTLS 定义“安全”。

P1 数据保密。| **P1A** 对手必须不能读取会话数据。[访问：是/否] | **P1B** 通信应该是*前向机密的*（长期私钥的泄露不会帮助攻击者访问先前会话的数据）。| **P1C** 对手应该从观察密文中学到更多的知识，就像每个“跃点”都是其自己的独立 TLS 连接一样。*[更改机密：*价值]

P2 数据认证。对手必须不能修改，删除或注入会话数据。这包括重放或重新排序数据。更正式地讲，如果会话由节点 N_1，...，N_m 的有序集合组成，则 N_i 接收的任何数据都必须是 N_{i-1} 发送的数据的前缀。

P3 实体认证。端点必须能够验证它们是否在谈论“正确的事情”。这包含两个属性。| **P3A** 每个端点可以验证另一个端点是否由预期实体运行，并且每个 MS 是否由预期 *MSP 运行*（例如，此代理由 AT&T 运行）。（这要求被验证的实体具有证书。）*[标识：*所有者] | **P3B** 每个端点可以验证另一个端点和每个 MS 是否正在运行预期的软件并且已正确配置（例如，仅启用了强 TLS 密码套件的 Apache v2.4.25）。[2↑]（这需要验证实体以支持安全的执行环境，请参见下文。）*[Identity：* code]

P4 路径完整性。每个端点为其中间盒确定顺序。任何其他实体（包括中间盒或其他端点）都不可能导致中间盒以不同的顺序处理会话数据（包括跳过

[1] 这一趋势是由成熟的技术用在商品硬件（NFV）上运行的应用程序中间盒鼓励[24，25，35，46]，包括商业产品[3，7，8]。

[2] 请注意，这不能保证该软件的行为或缺陷。为此，代码身份验证必须与软件分析/验证相结合，这不在本文的讨论范围之内。

图 1：幼稚的方法。端到端建立 TLS 会话，并将会话密钥通过辅助 TLS 会话传递给中间盒。

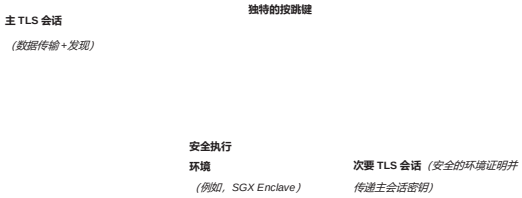


图 2：mbTLS 方法。 为每个“跃点”生成唯一的密钥，并在安全的执行环境中运行中间盒。

中间盒）。更正式地，如果会话由一组有序的节点中的 $\tilde{N}_{\dot{y}}, \ldots, \tilde{N}_{\divideontimes}$，然后由接收记录 $\tilde{N}_{\text{我}}$要么已经被 N 处理 I_Y 或起源被 N I_Y 。*[路径完整性：*是]

除了这些安全属性外，我们还具有以下与性能和可部署性相关的目标：

IP5 | 旧版互操作性。mbTLS 应该与旧版 TLS 端点（客户端或服务器）一起使用，只要其中一个端点已升级即可。*[旧版端点：* 1_。旧版]

IP6 1 带内发现。mbTLS 应该在会话设置过程中发现路径上的中间盒。[发现：是]

IP7 | 最小的开销。mbTLS 应该引入尽可能少的开销（与 TLS 相比）。重要的是，mbTLS 不应将任何往返行程添加到 TLS 握手中。

3.3 设计概述

由于 TLS 已经提供了我们想要的许多属性，因此一种简单的方法是：在客户端和服务器之间建立常规的 TLS 会话，然后将会话密钥通过单独的辅助 TLS 会话传递给中间盒（图 1）[43]。这提供了我们想要的许多属性：数据经过加密和完整性保护，以防止第三方更改（部分 **I P1A |**）。如果使用前向安全密码套件，则通信是前向秘密**| P1B |，**端点可以使用证书|验证彼此的身份**。 P3A |**，并且可以感知中间箱的端点可以添加中间箱，而无需其他端点的支持**| P5 |**。

但是，以这种方式使用 TLS 不足以构成我们的威胁模型，原因有以下三个：（1）由于它是专为两方设计的，因此它没有提供路径完整性的机制**。 P4** 1；（2）在会话中的每个“跃点”上都使用相同的密钥进行加密，从而使对手可以轻松比较进入和离开中间盒的记录以查看它们是否发生了更改**。 P1C |**；（3）基础结构提供程序可以访问内存中的会话数据**| P1A |**，访问中的关键材料造 MAC **| P2** 1，并且可能运行 *MSP* 提供的软件以外的软件**| P3B |。**此外，TLS 不提供发现机制**| P6** 1。

我们通过引入以下功能（图 2）来解决这些不足，并将结果称为 ***Middlebox TLS（mbTLS）***。

- **带内中间盒发现。**只要端点之一支持 mbTLS，中间盒就可以在主 TLS 握手|端点声明中声明自己并加入会话（具有端点批准）**。 P6** 1。
- **安全执行环境。**中间件可以在一个安全执行环境可以选择运行，像一个英特尔 SGX 飞地（见下文），[3]，其提供存储器加密，保护会话数据和密钥从不可信的 *MIP* **[P1A |[P2]**和远程证明，允许端点验证 MS **|** 的软件身份。**P3B |。**端点也可以在安全环境中运行以提供**| P3B |**。
- **唯一的逐跳键。**每个“跳”使用其自己的对称密钥来保护会话数据。这样可以防止对手将记录传递到乱序的中间盒**[P4** 1]，并且无法确定中间盒何时转发数据而不更改它**。 P1C |**。

旁白：可信计算和 SGX。mbTLS 的一些功能依赖于可信计算技术，如英特尔的软件狗扩展（SGX）[15，26，37] 。尤其是，mbTLS 使用 SGX 提供的两个功能-安全执行环境和远程证明-尽管提供这些功能的任何受信任的计算技术（例如 Microsoft 的虚拟安全模式（VSM）[21]）也可以使用。（其他技术，例如 ARM TrustZone [1]，提供相似的功能，但提供的安全保证略有不同。）如果您熟悉 SGX，请跳至 §3.4。

*安全执行环境。*SGX 允许应用程序在称为安全*区*的安全环境中运行代码。飞地是受保护的内存区域；在将*缓存*行移至 DRAM 之前，它们已被 CPU 加密并进行完整性保护。只要未在物理上破坏 CPU，恶意硬件或特权软件就无法访问或修改安全区域内存。

*远程认证。*SGX 可以为运行在飞地中的代码提供由 CPU 签名的特殊消息（称为*证明*），该消息向远程方证明有关代码确实在真正的 Intel CPU 的飞地中运行。该证明包括安全区代码和数据页的初始状态的加密哈希（因此，远程验证程序可以看到期望的代码正在运行），以及安全区应用程序提供的自定义数据（我们将其用于将验证与 TLS 集成在一起）握手）。

3.4 mbTLS 协议

在较高级别上，端点执行标准的 TLS 握手，建立*主要的 TLS 会话*，该*会话*最终将用于数据传输。每个端点向会话添加零个或多个中间箱，我们将它们称为*客户端*和*服务器端*中间箱，可以先验地知道，也可以在握手过程中发现**。 P6** 1。每个端点都不知道对方的中间盒（或者是否知道）

[3]随着越来越多的中间盒被设计为可在商用 CPU 上运行，SGX 等功能很快将变得司空见惯，这将变得越来越实用。

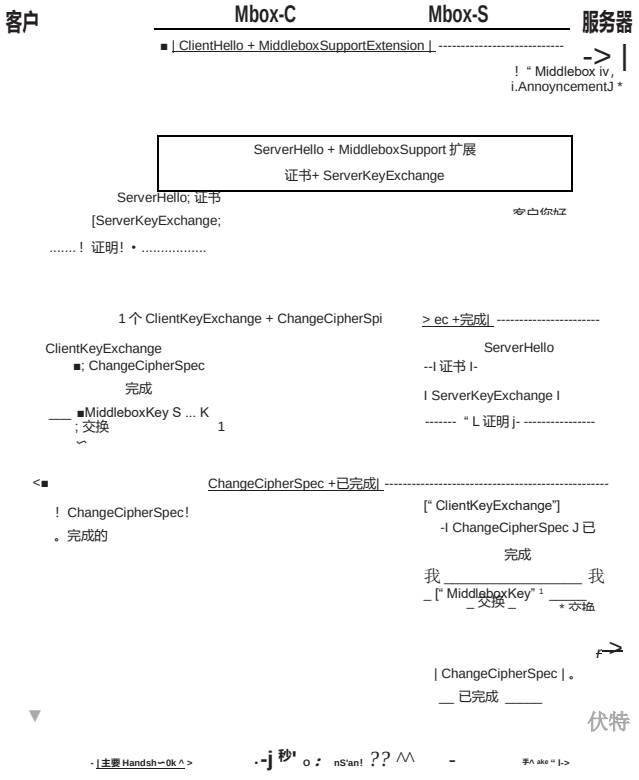


图 3：mbTLS 握手。请注意，它包含多个交错的标准 TLS 握手，以及一些其他消息（阴影）。

根本不存在），这意味着 mbTLS 端点可以与旧版 TLS 端点进行互操作 | **P5**。端点与它们的每个中间盒同时建立*辅助 TLS 会话*。端点具有通向中间盒的安全通道后（如果中间盒支持，则可以包括验证中间盒软件是否在安全执行环境中运行），它将向中间盒发送加入主会话所需的密钥材料。

数据传输。我们的实现为每个跃点创建一个单独的 TCP 连接。这不是绝对必要的-更改数据流的中间盒可以相应地调整 TCP 序列号-但是，由于对加密数据进行操作的中间盒必须在解密和对其进行操作之前收集完整的 TLS 记录，因此在每个跃点上进行可靠的传输是有意义的。

控制消息。mbTLS 在同一 TCP 连接上多路复用主要和辅助 TLS 会话。与打开辅助 TCP 连接相比，这减少了开销 **P7，方法**是：（1）减少中间盒和端点上的 TCP 状态，（2）将所有握手消息保留在同一路径上，以及（3）防止客户端中间盒发现添加回合旅行。我们引入了新的 TLS 记录类型（Encapsulated ），以在中间盒及其端点之间包装辅助 TLS 记录。这些记录由一个外部 TLS 记录标头，一个字节的*子通道 ID* 和封装的记录组成。有关 mbTLS 消息格式的详细信息，请参阅附录 A。

中间盒发现。中间盒可以通过四种方式成为会话的一部分：客户端预配置，客户端发现，服务器端预配置和服务器端发现。

*客户端中间盒。*如果客户端事先了解中间盒（例如，通过用户配置），则它将直接打开与中间盒的 TCP 连接。然后，它发送主要的 ClientHello，它包括一个新的 MiddleboxSupport TLS 扩展（图 3 中的顶行），其中包括（其中包括）所有已知先验中间盒的列表。中间盒然后打开到下一跳的 TCP 连接，并转发 ClientHello 。另外，可以在握手过程中发现默认路由路径上的中间盒**[P6**。该 MiddleboxSupport 扩展向这些中间盒发出信号，表明客户端支持 mbTLS；他们乐观地拆分了 TCP 连接，并在看到扩展名后加入了握手，如下所述。

在任何一种情况下，当看到扩展名时，中间盒（1）将 ClientHello 向前转发给服务器，而（2）准备与客户端启动其自身的辅助握手。在此辅助握手中，中间盒充当服务器的角色。原始的主 ClientHello 充当了辅助握手的 ClientHello 的双重职责，允许中间盒在将主 ServerHello 向客户端转发后立即插入自己的 ServerHello，而不必等待来回的新的辅助 ClientHello **P7**。客户端收到辅助 ServerHello 时，如果中间盒不是由客户端预先配置的，则客户端 mbTLS 库将请求应用程序批准，这可能会将此决定推迟给用户。

可能有多个客户端中间盒。辅助握手消息在封装记录中发送，每个中间盒都有其自己的子通道 ID。中间盒等待，直到他们看到主要的 ServerHello，对其进行缓冲，为自己分配下一个可用的子通道 ID，使用该 ID 将其自己的辅助 ServerHello 注入到数据流中，最后转发主要 ServerHello 。此过程可确保每个中间盒在协调最少的情况下获得唯一的子通道 ID。

*服务器端中间盒。*服务器端中间盒也可以预先配置或发现。在预配置的情况下，服务器必须以某种方式安排握手遍历（可能是路径外的）中间盒。这可以通过更改服务器的 DNS 条目以解析为中间盒或通过询问客户端（如果它支持 mbTLS）来在 MiddleboxSupport 扩展中包括中间盒（例如，通过在新的 DNS 记录类型中列出中间盒）来完成。

在发现情况下，与客户端不同，服务器不使用 MiddleboxSupport 扩展来宣布 mbTLS 支持，原因有两个：首先，TLS 规范禁止服务器在 ServerHello 中包含客户端未包括在 ClientHello 中 的扩展[22]]；如果客户端也不支持 mbTLS，则依赖于服务器的 MiddleboxSupport 扩展将失败。其次，即使允许这样做，如果服务器端中间盒在服务器 ServerHello 之后等待宣布其存在，则中间盒-服务器握手将在主握手之后完成，从而将整个握手延长到两个以上的 RTT（相对于| **P7**|）。

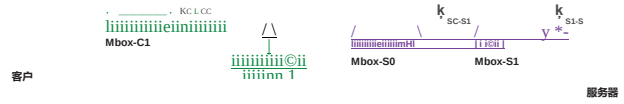


图 4：唯一的逐跳密钥。每个跃点使用不同的密钥对数据进行加密和 MAC 保护-客户端为客户端跃点生成密钥 *(*K_{c-c1}和K_{c1-co}），服务器为服务器端跃点生成密钥 *(*$K_{so.s1}$和K_{s1-s}），并且主会话密钥（K_{c-s}）桥接双方。

相反，服务器端中间盒会*在*知道服务器是否支持 mbTLS *之前*以新的 MiddleboxAnnouncement 消息乐观地声明自己。如果不是，则根据其 TLS 实施，它将忽略 MiddleboxAnnouncement，并且握手将在没有 Middlebox 的情况下继续进行，否则握手将失败。（在任何一种情况下，中间盒都将缓存此信息，而不会再次向该服务器宣布自身。）如果握手失败，则客户端将需要重试。客户端软件可能会将其解释为这意味着服务器正在运行过期的 TLS 堆栈，并使用较旧版本的 TLS 重试，这有潜在的危险。我们证实，Chrome 和 Firefox *不会*表现出这种行为；Chrome 将尝试第二次 TLS 1.2 握手，而 Firefox 将完全不重试（这意味着用户将需要手动单击“刷新”）。此外，在实践中，我们希望服务器端中间盒和服务器通常处于相同的管理控制之下，在这种情况下，中间盒知道服务器支持 mbTLS 并期待公告。

安全环境认证。我们已将 TLS 1.2 握手扩展为可选地包括远程证明（除了标准证书检查之外）。此扩展独立于 mbTLS，并且可以在标准的客户端/服务器握手中使用，以防任一端点要验证另一端点在安全环境中运行。但是，在 mbTLS 的上下文中，端点通常将使用此功能来验证（1）具有外包中间盒的辅助 TLS 会话是否在安全执行环境中终止。**P1A 1 1 P2**（2）这些中间盒以预期的配置运行预期的软件|**P3B**|。目的是说服端点*仅在安全区域中运行的中间盒应用程序知道正在建立的 TLS 会话密钥。*主要思想如下：由于证明包括代码的身份，并且我们假设代码（应用程序+ mbTLS 库）已经过检查并且是受信任的，因此，如果代码断言它为此握手生成了密钥材料，并且没有导出它，则端点可以信任此断言。

挑战在于识别“此握手”—端点如何确保证明是最新的，而不是对手从其他握手中重播旧的证明？这意味着，除了代码标识之外，证明还必须包括某种*握手标识符*以证明新鲜度。为此，我们使用到目前为止交换的所有握手消息的哈希值，就像 TLS 已经完成身份验证完成消息中的握手一样。由于这包括客户端和服务器的随机数，因此每次握手的证明都是唯一的，并且对手无法强迫他们重复。

唯一的逐跳键。在 mbTLS 握手结束时，会话如图 4 所示。在将辅助握手及其中间盒隐藏起来之后，每个端点都会为连接侧的每个跃点生成一个对称密钥（例如，客户端生成K_{c-c1}和K_{c1-co}）-这样可以防止对手导致记录跳过中间框或无序遍历中间框|**P4**|并且还防止窃听者检测中间盒是否修改了记录|**P1C**|。端点将这些密钥分发到 MiddleboxKeyExchange 记录中的中间盒，这些记录通过辅助 TLS 连接作为（加密的）数据发送。（就像辅助握手消息一样，辅助数据记录在同一 TCP 流中的封装记录中发送。）此时，辅助 TLS 连接不再使用。在主握手期间建立的会话密钥K_{c-s}充当客户端和服务器端中间盒之间（或中间盒和旧式端点|**P5 之间**）的“桥梁”。

3.5 讨论

会话恢复。mbTLS 支持基于 ID 和基于票证的会话恢复。每个子握手（主要握手和次要握手）都替换为标准的缩写握手；唯一的次要区别是，中间盒的会话票证应包含主要端到端会话的会话密钥（除了次要端点-中间盒会话的密钥）。不需要新的证明，因为只有安全区才知道解密会话票证所需的密钥。客户端存储服务器和每个客户端中间盒的会话 ID 或票证。服务器可以缓存服务器端中间盒的会话 ID/票证，也可以要求客户端对其进行缓存并在其 ClientHello 中返回它们。

假设客户端在第一时间发送应用程序数据，则数据有可能在 ChangeCipherSpec 和 Finished 从服务器*之前*到达服务器端中间盒。这与 TLS False Start [32]中看到的情况相同。中间盒可以选择等待服务器的 Finished，从而将握手略微延迟到超过 1 个 RTT，或者如果将密码套件列入白名单以与 False Start 一起使用，则继续处理数据。

TLS 1.3。与 TLS 1.2 及更早版本相比，TLS 1.3 [44]显着改变了 TLS 握手，将其从两次往返缩短为一次。通过较小的修改，mbTLS 的握手可以适应 TLS 1.3。有一个警告：当存在客户端中间箱时，在与服务器 Finished 相同的飞行中，服务器发送的数据可能会延迟，在最坏的情况下，可能会延迟一次往返。但是，在大多数情况下，客户端首先发送数据。

相信。端点如何决定信任中间盒？尽管这与 mbTLS 的设计正交，但我们简要描述了我们预期的三种情况。首先，用户可以注册服务（例如，来自其 ISP 的父母过滤），并明确配置其浏览器以信任该服务。其次，客户端软件可能已预先配置为信任来自已知集合的中间盒，并在发现一个中间盒时提示用户（例如，Chrome 可以为 Flywheel 做到这一点）。菲娜升 LY，服务提供商可能依赖 mbTLS 发现自己的中间件的最接近的实例; 服务器需要中间盒，并使用证书和证明对其进行验证。

4 安全分析

4.1 核心安全属性

现在，我们从 §3.2 重新讨论每个安全性属性，争论为什么 mbTLS 提供它们。（表 1 汇总。）

P1 | 数据保密。

P1A 解密会话数据需要访问图 4 所示的对称密钥之一。桥密钥 $K_{Q_\$}$是在端到端客户端-服务器 TLS 握手过程中建立的，端点在该握手过程中相互验证对方的证书。接下来，此密钥和其余会话密钥（例如，$K_{C\text{-}Q1}$，$K_{C\text{-}Q0}$等）通过各个辅助 TLS 连接传输到中间盒；重要的是，由于这些辅助连接在 SGX 飞地内部终止（远程证明向端点证明是这种情况），因此 *MIP* 无法访问内存中辅助会话的密钥，因此只有 *MS*（而非 MIP）才能学习主会话密钥。

P1B 桥密钥（$K_{Q\text{-}S}$）是（标准）主 TLS 握手的结果，因此，如果主握手是前向安全的，则 $K_{Q\text{-}S}$ *也是*。另一会话密钥（例如，$K_{Q\text{-}Q0}$，$K_{Q0\text{-}Q1}$，等）生成每个会话新鲜并且被发送到中间盒上（标准）次级 TLS 连接。因此，如果这些辅助握手是前向安全的，则非桥接会话密钥也是如此。

P1C 由于每个跃点都使用其自己的独立加密和 MAC 密钥，因此在握手之后，每个跃点都可以像其自己的 TLS 连接一样有效地运行。特别是，这阻止了对手了解中间盒是否修改了记录（尽管它仍然可以看到每个记录的大小和时间，包括中间盒是增加还是减小了数据大小）。

P2]数据身份验证。每个记录都带有*消息认证码（MAC），这*是一个使用会话密钥生成的小标签，用于标识一条数据。如果 MAC 与数据不匹配，则可以检测到未经授权的更改。由于只有端点和每个 MS 都知道会话密钥（请参阅| **P1A**），因此只有这些实体才能修改或创建记录。

P3]实体身份验证。

P3A 首先，客户端和服务器可以在一次握手中要求对方的证书（尽管通常客户端身份验证发生在应用程序层）。证书将服务器的公钥绑定到其身份，并且该公钥在主握手中用于协商共享桥密钥，因此，在成功握手之后，客户端将确保使用该桥密钥加密的任何数据只能被解密。由预期的服务提供商（或它选择信任的中间盒）提供。其次，端点还可能需要来自中间箱的证书。由于相应的私钥存储在中间盒的区域中，MIP 无法访问它们（并且远程证明证明是这种情况），因此端点确信该会话正在与预期的提供和配置的软件进行通信。*MSP*。

P3B 由于我们的威胁模型假定 SP 及其在服务器上运行的所有软件都是受信任的，并且在 P3A 中，我们验证了服务器拥有 SP 的私钥，因此客户端相信该计算机已正确配置了预期的应用程序软件。

如果中间盒是在内部操作的，则同样的逻辑也适用于中间盒。对于外包的中间盒，端点可以通过远程证明直接验证 MS 代码和配置。

P4 路径完整性。这是因为 mbTLS 对每个跃点都使用了一个新密钥。假设对手从图 4 中的 C1 - C0 链接中嗅探一条记录，然后尝试将其插入 S0 - S1 链接中（从而跳过中间框 C0 和 S0）。记录将被加密并以 $K_{Q1\text{-}Q0}$进行 MAC 加密，但是 C1 希望数据以 $K_{S1\text{-}S0}$ *进行保护*，因此 MAC 检查将失败并且记录将被丢弃。（请注意，*端点*可以在路径部分的任意位置注入，删除或修改数据，因为它知道其一侧的所有会话密钥。我们将在下面讨论潜在的安全隐患。）

4.2 其他安全属性

端点隔离。*端点只能验证自己的中间盒，而不能验证其他端点添加的中间盒*。实际上，端点可能甚至不知道对方的中间盒。这是根据密钥生成和分发的方式进行的。仅当证书中的公共密钥用于密钥交换时，检查证书或证明才有意义（然后，您相信只有与该公共密钥关联的实体才能解密使用新对称密钥发送的内容）。由于端点不与另一侧的中间盒进行 KE 因此即使交换了证书/证明，它们也无法相互进行身份验证。这个限制似乎是合理的。因为这些端点大概相互信任，或者它们最初不会进行通信，所以自然可以信任另一个端点来正确地验证添加到会话中的任何中间盒。

路径灵活性。*不可能交错客户端和服务器中间盒*。为了支持这一点，端点将需要协调以将密钥生成/分发到路径的交错部分。这意味着（1）端点的额外工作，以及（2）端点需要了解彼此的（某些）中间盒。这也意味着一个端点可以在另一个端点的中间盒*之后*修改/注入流量，如果这些中间盒中的一个执行过滤或清理操作，则可能是一个安全问题。

中间盒状态中毒。*将 mbTLS 与保留**全局状态的客户端**中间盒一起使用是不安全的***。由于端点知道会话一侧的每个跃点的密钥，因此恶意客户端可以在不知道其中间盒的情况下读取和/或修改任何跃点上的数据。当中间盒在多个客户端之间共享状态时，这是一个问题，就像 Web 缓存一样。有权访问缓存与服务器之间的链接的客户端可以（1）请求一个页面，（2）丢弃服务器的响应，并且（3）注入自己的响应，从而使其他客户端的缓存中毒。

一种可能的解决方案是更改握手协议，以便中间盒与其邻居建立密钥，而不是端点生成和分配会话密钥。这意味着各方仅知道与其相邻的跃点的密钥。缺点是客户端失去了直接认证服务器的能力。相反，客户端必须信任其中间盒以对服务器进行身份验证。

担保财产	威胁	防御（TLS）	防御（mbTLS）
数据保密	**[P1A]**通过 *TP* 或 *MIP* 在线读取的数据	加密	加密*
	MIP 在 *MS* 应用程序存储器中读取的 **P1A** 数据	-	安全执行环境
	P1B 长期密钥泄漏后 TP 解密了旧数据	临时密钥交换	临时密钥交换
	P1C TP 比较进入和离开 MS 的记录以查看其是否被修改	-	独特的按跳键
资料验证	**P2** 记录是在网上删除，注入或修改的	媒体访问控制	MAC *
	P2]通过 *MIP* 在 RAM 中删除，注入或修改数据	-	安全执行环境
实体认证	**P3A** *C* 用错误的软件在 S 上建立密钥	证书	证书
	P3A C 在非 *S* 的人的硬件上通过软件建立密钥	证书	证书
	P3A C 或 S 使用 *MSP* 以外的人员操作的 MS 软件来建立密钥	-	证书*
	P3B C 或 S 用错误的 MS 软件建立密钥	-	远程认证
路径完整性	**P4** 记录以错误的顺序传递到中间盒	-	独特的按跳键

表 1：威胁与防御。mbTLS 如何防御对我们核心安全属性的具体威胁。为了进行比较，我们在适用的地方包括了 TLS。星号表示防御也依赖于安全环境来保护会话密钥。

绕过"过滤器"中间盒。事实似乎是端点知道他们身边的所有会话密钥都会引发另一种攻击：如果中间盒执行某种过滤功能（例如，病毒扫描程序，父母过滤器或数据泄露检测器），则表明端点具有键，以*在*过滤输入数据*之前*访问传入数据，或在以后注入出站数据。但是，如果端点能够"在过滤器的另一侧"读取或写入数据（即，从中间盒物理地将数据包从网络中检索/注入到网络中，或从中间盒中引入到网络中），则过滤器从一开始就无效。

5 评估

我们评估了 mbTLS 的四个关键方面。首先，我们的安全性分析认为 mbTLS 是**安全的**（第 4 节）。其次，通过一系列实际实验，我们证明 mbTLS 可**立即部署**（第 5.1 节）。第三，我们显示 mbTLS 对希望部署它的服务器施加了**合理的 CPU 开销**（并减少了中间盒的**开销**）（第 5.2 节）。最后，我们表明 **SGX 应用程序可以支持网络 I / O 繁重的工作负载**（第 5.3 节）。

原型实现。我们使用 Windows 的 Intel SGX SDK（v1.7）在 OpenSSL（v1.1.1-dev）中实现了 mbTLS。我们的原型目前支持使用 DHE 或 ECDHE 进行密钥交换和 AES256-GCM 的所有密码套件用于批量加密（这些不是协议的基本限制，任何 TLS 密码套件都可以在 mbTLS 中使用）我们还提供了一个支持库，用于在 SGX 飞地中运行 mbTLS 实现。我们的支持库直接在安全区中实现 8 个 libc 函数（其中 3 个仅用于调试，可以在生产版本中删除），并退出安全区中的另外 7 个 libc 函数（其中 4 个用于调试）。以下实验中的中间盒是执行 HTTP 标头插入的简单 HTTP 代理。

测试台。我们的本地测试平台包括四台运行 Windows Server 2016 的服务器，以及支持 SGX 的 Intel Core i7 6700 处理器和支持 SGX 的主板。这些通过 Mellanox ConnectX-3 40Gbps NIC 连接到本地 Arista 7050X 交换机。

网络类型	# 个网站	网络类型	# 个网站
企业	6	*大学*	11
住宅	34	*上市*	1 个
移动的	2 个	*代管*	56
托管服务	35	*数据中心*	19
未分类	77	**全部的**	241

表 2：握手可行性。我们从中执行 mbTLS 握手到测试服务器的不同站点的数量（按网络类型划分）。所有握手均成功。

5.1 可部署性

我们通过实际部署测试了两件事：（1）公共 Internet 中的防火墙或流量规范器是否会阻止 mbTLS 连接？（2）mbTLS 可以与旧式端点互操作吗？

握手可行性。由于 mbTLS 引入了新的 TLS 扩展（MiddleboxSupport ）和记录类型（Encapsulated 和 MiddleboxAnnouncement），我们确认现有的过滤器（例如防火墙，流量标准化器或 IDS）不会丢下我们的握手。为此，我们从位于世界各地的各个网络中的客户端连接到运行在 Azure 中的中间盒和服务器。中间盒配置为客户端中间盒，因此新记录类型遍历客户端和数据中心之间的网络。为了测试一组不同的客户端网络，我们做两件事。首先，我们使用位于 214 个 AS 中 46 个国家/地区的 550 个出口节点通过 Tor 来运行我们的客户。使用 whois 数据，我们按类型对网络进行了分类。我们选择不使用诸如 PlanetLab（其节点主要位于大学网络中）之类的平台，或诸如 Azure 或 EC2 之类的公共云，因为这些环境更加均一，并且通常不会被严格过滤。第二，为了填补 Tor 实验中的空白，我们手动连接了不同类型的网络（即公共，移动和数据中心网络）。桌子图 2 显示了我们从其发起握手的不同客户端网络的细分。所有握手均成功。

旧版互操作性。为了证明 mbTLS 可以与传统的端点进行通信，我们使用的版本卷曲的[4] 修改

图 5：握手 CPU 微基准测试。每个栏显示执行一次握手所花费的时间（不包括等待网络 I / O）。每个小节是 1000 次试验的平均值；误差线显示平均值的 95％置信区间。

图 6：mbTLS 与 TLS 延迟。是时候通过一个中间盒跨数据中心之间的各种路径来获取一个小对象了。

使用 mbTLS 通过 Azure 中运行的 SOCKS HTTP 代理为支持 HTTPS 的前 500 个 Alexa 网站下载根 HTML 文档。Alexa 排名前 500 名的站点中只有 385 个站点支持 HTTPS。我们成功连接了其中的 308 个。在失败的 77 个证书中，有 19 个证书无效或已过期。另外 40 个不支持 AES256-GCM，这是我们的原型当前支持的唯一加密算法（请注意，这是我们的原型的限制，而不是协议的限制）。由于重定向我们的 SOCKS 实现未正确处理，还有 13 个失败。其余 5 个失败原因不明。

5.2 性能开销

mbTLS 不会修改 TLS 记录层，因此不会影响数据传输性能。另一方面它*确实*改变了握手，因此我们（1）研究执行握手的计算开销，并且（2）凭经验验证 mbTLS 不会增加会话建立延迟。

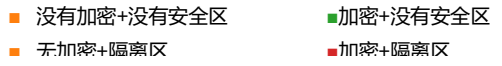
握手 CPU Microbenchmarks。为了了解 mbTLS 对 CPU 负载的影响，我们测量了客户端，中间盒和服务器执行 TLS 和 mbTLS 握手所花费的时间。CPU 开销对于中间盒和服务器特别重要，它们需要同时服务多个连接。如图 5 所示，在没有中间盒的情况下，TLS 和 mbTLS 的时间接近（我们怀疑稍有不同是我们的实现效率低下，而不是协议效率低下）其次，对于中间盒而言，mbTLS 握手*更便宜*而不是拆分 TLS，因为中间盒仅执行一次 TLS 握手，而不执行两次。最后，服务器的负载不受客户端中间盒的影响，并且随服务器中间盒的数量线性增加。但是请注意，每个服务器端中间盒仅增加了原始无中间盒握手时间的 20％左右。这是因为，对于每个中间盒，服务器都会执行一次附加的*客户端* TLS 握手，这比服务器握手便宜。（密钥交换是 ECDHE-RSA；结果与 DHE-RSA 相似。机器规格：4 GHz 的 Intel i7-6700K CPU； 16 GB RAM Windows10。）

握手延迟。为了确认我们的握手协议在实践中不会夸大会话设置（它不应该，因为它保持与 TLS 相同的四次飞行“形状”），我们在 Microsoft Azure 中跨数据中心执行了几次握手。我们在四个区域（澳大利亚，美国西部，美国东部和英国）部署 VM，并测试它们之间的客户端-中间盒-服务器路径的所有排列（在同一 DC 中没有两个 VM）。在每个测试中，我们比较使用 mbTLS 和 TLS 提取小对象的时间。对于常规 TLS，中间盒仅中继数据包，即，它不执行拆分 TLS，这是比较 mbTLS 的最坏情况，因为中间盒不执行任何握手操作。图 6 总结结果，分为握手时间和数据传输时间。每个小节是 100 次试验的平均值；误差线显示 95％的置信区间。我们观察到 mbTLS 将握手延迟平均提高了 0.7％（在最坏的情况下为 1.2％，在 800 毫秒中增加了 10 毫秒）。这很可能是由于中间盒上的握手计算所致。

5.3 SGX 中的网络 I / O

最后，我们从飞地调查网络 I / O 性能。SGX 对安全区代码可以执行的操作施加了限制。由于仅信任 CPU，因此默认情况下不允许与外界进行交互（值得注意的是，由于操作系统不受信任，因此不允许系统调用）。当安全区线程需要进行系统调用时，有两种高级策略：（1）将参数复制到不受保护的内存中，退出安全区，执行调用，重新进入安全区，然后将结果复制回飞地内存（此边界会导致性能下降）；或（2）将请求放置在共享队列中，安全区外部的另一个线程执行该调用，将结果通过响应队列传递回安全区。借用 SCONE 的术语[16]，它们分别是*同步*和*异步*系统调用。

在重复的 pwrite () 的微基准测试中，SCONE 发现，对于较小的缓冲区大小，异步调用可以快一个数量级。但是，这里我们特别关注 send () 和 recv ()，因此我们进行了一个小实验来测试飞地对吞吐量的影响。我们在实验室中将四台计算机配置为一个中间盒，一台服务器和两个客户端。客户端发送随机字节流，以加密块的形式发送，大小各不相同。中间盒配置有以下四种行为之一：它要么简单地将（加密的）数据转发到服务器，要么在转发之前对其进行解密和重新加密，然后执行此操作



缓冲区大小（字节）

图 7：SGX（非）开销。带有/不带有加密以及带有/不带有 SGX 的中间盒吞吐量。置信区间在平均值的 1-5％范围内，小缓冲区的不同方案之间的差异在统计上不显着。

在飞地的内部或外部。我们添加来自多个客户端线程的连接，直到中间盒 CPU 饱和。

图 7 显示，隔离区并*没有*对产量产生显着影响，这表明像异步系统调用优化是没有必要与 I / O 繁重的工作量的应用程序。我们怀疑这是由于 I / O 中断率高所致；中断处理产生的开销淹没了跨越安全区边界的开销。即使开发人员使用异步系统调用，在线程将永久驻留在安全区的印象下，只要该内核中断，该线程仍将离开安全区。虽然将中断固定到另一个内核以避免中断安全区可能有所帮助，但是您却要付出将接收到的数据从中断处理内核传输到安全区内核的成本。图 7 还显示了中间盒解密/加密大约 7 Gbps 的数据平稳段时的吞吐量。这是 CPU 开销的又一个来源，可以帮助克服飞地过渡带来的任何性能损失。

6 相关工作

中间盒作为端点。在 §2.2 中，我们描述了使用自定义根 CA 证书拦截 TLS 连接的当前做法。其他建议包括爱立信和 AT&T 于 2014 年提出的 IETF 草案[34]，该草案将允许代理拦截 TLS 握手并返回证明自身身份的证书；如果客户端选择，它可以与代理完成握手，并依靠代理打开自己的与服务器的 TLS 连接。与此相关的思科建议[36] 通过引入 ProxyInfoExtension 建立在此基础上代理将使用它来传递有关代理服务器连接上使用的证书和密码套件的客户端信息。最后，至少一个 ISP 随同自定义浏览器一起提供了内置于[33]的代理证书。不幸的是，这些方法不允许客户端对服务器进行身份验证或验证中间盒与服务器之间使用的密码套件。

中间盒作为中间盒。 与将单独的 TLS 连接粘合在一起的上述方法相比几种技术保持某种形式的端到端会话。Google 的 IETF 草案让客户端直接连接到服务器，并将会话密钥通过单独的 TLS 连接带外传递给代理带外[43]。CloudFlare 的无密钥 SSL 对服务器端委托做了很多相同的事情[20，52]这两种技术都将数据公开给中间盒基础结构，并且无法提供路径完整性（在每个跃点上使用相同的会话密钥）。

我们讨论 mcTLS [41] 和 BlindBox [48] 在 §2.2。

最后，在 L3 和 L4 标头字段（例如 NAT，防火墙和负载平衡器）上运行的网络层中间盒的安全问题与 mbTLS 地址正交。像踏上系统[31] SplitBox [17] ，以及其他[29，38，49]允许网络管理员外包网络层中间件到云提供商或 ISP 没有透露关于他们的网络的私人信息。

网络体系结构。DOA [53]为通过一个或多个中间设备路由数据包提供网络支持，但其本身并不能提供我们所有的安全属性。ICING [39]是一种用于增强路径完整性的机制，它比我们的机制更通用。我们的路径完整性机制通过利用 mbTLS 各方已经共享密钥并且每个 mbTLS 记录已经受 MAC 保护的事实来进行优化。

新交所。在[30]中简要讨论了使用 SGX 保护外包中间盒，但没有具体协议或实现。S-NFV [50] 概述了用于实现受 SGX 保护的中间盒应用程序的框架。PRI [45]详细介绍了基于 SGX 的 IDS 的设计。两者都集中在中间盒应用程序体系结构上，而不是在通信会话中首先包含中间盒的协议。也有一些 TLS 实现设计为工作在 SGX 飞地[6，10，18，54]。这些库可移植未修改的 TLS。我们进一步走了一步，将 TLS 握手扩展到包括远程证明，从而允许一方验证 TLS 会话是否在安全区域内终止。最后，还有一个工作的关于如何建立 SGX 保护系统（或端口现有的）一个快速发展的体[13，16，19，28，51] 。这些都与这项工作正交，可以与 mbTLS 配合使用以构建受 SGX 保护的中间盒。

7 结论

在本文中，我们介绍了用于安全多实体通信的协议 Middlebox TLS（即 mbTLS）。与以前的用于将中间盒集成到安全会话中的解决方案不同，mbTLS（1）可与旧版 TLS 端点互操作，而（2）可以使用英特尔 SGX 等受信任的计算技术来保护会话数据不受不可信的中间盒基础架构的影响。我们的原型实现显示，mbTLS 确实可以与未经修改的真实 Web 服务器通信，并产生合理的开销。最后，我们讨论了用于多实体通信的安全属性的空间，以及折衷的协议设计者必须在其中进行权衡。

致谢非常感谢 Cedric Fournet，Antoine Delignat-Lavaud，Felix Schuster Manuel Costa 和 Istvan Haller 的时间，反馈和专业知识。我们也感谢审稿人和我们的牧者的有益意见和建议。这项工作部分由 NSF 资助，编号为 CNS-1345305。

MBTLS 协议详细信息

本附录描述了在我们的 mbTLS 实现中使用的消息格式和协议常量。它遵循 RFC 5246（TLS 1.2）[22]中规定的格式约定。

A.1 记录协议

mbTLS 添加了三种新的记录类型（粗体）：

```
枚举{
    change_cipher_spec（20），
    警报（21），握手（22），application_data（23），mbtls_encapsulated（30），
    mbtls_key_material（31），mbtls_middlebox_announcement（32），
    （255）
} 内容类型;
```

封装的。所述 MBTLSEncapsulated 记录用于携带二次握手消息。

```
结构{
        uint8 subchannelID;
        不透明记录[TLSPlaintext.length-1];
} EncapsulatedRecord;
```

在这里，记录是另一个完整的 TLS 记录。因为内部记录是外部记录的有效负载，限制为 2 个 ^{14} 字节，并且由于子通道 ID 使用 1 个字节，所以内部记录的有效负载被限制为 2 个 $^{14-1}$ 字节。 MBTLS 封装的记录只能在握手或重新协商期间发送。

关键材料。该 MBTLSKeyMaterial 记录所使用的端点对称密钥材料发给他们的中间件。

```
结构{
        uint32 key_len;
        uint32 iv_len;
        不透明的 clientWriteKey [key_len];
        不透明的 clientWriteIV [iv_len];
        不透明的 clientReadKey [key_len];
        opaque clientReadIV [iv_len];
        opaque serverWriteKey [key_len];
        opaque serverWriteIV [iv_len];
        opaque serverReadKey [key_len];
        opaque serverReadIV [iv_len];
} MBTLSGCMKeyMaterial;

结构{
        版本 client_server_version;
        不透明的 client_to_server_sequence [8];
        不透明的 server_to_client_sequence [8];
        CipherSuite cipher_suite; / * 2 个字节* / select（cipher_suite）{
            情况 TLS_RSA_WITH_AES_256_GCM_SHA384：
                MBTLSGCMKeyMaterial;
        }
} MBTLSKeyMaterial;
```

所述 MBTLSKeyMaterial 始终发送消息封装在子信道（即，在一个 MBTLSEncapsulated 记录）。它包含客户端和服务器之间协商的 TLS 版本，客户端到服务器数据的序列号（从客户端的角度看写序列和从服务器的读取序列）以及服务器到客户端数据的序列号。它还包含依赖于密码套件的格式的密钥和 IV 资料。

中间盒公告。该 MBTLSMiddlebox-公告消息由中间件提醒服务器他们的存在。

```
结构{
} MBTLSMiddleboxAnnouncement;
```

该 MBTLSMiddleboxAnnouncement 消息总是在发送 MBTLSEncapsulated 记录该消息为空，仅用于警告服务器中间盒的存在。中间盒从不将此消息发送给客户端。

A.2 握手协议

mbTLS 添加了一个握手协议消息（粗体）：

```
枚举{
        hello_request（0），CLIENT_HELLO（1），server_hello（2），证书（11），
        server_key_exchange（12），certificate_request（13），server_hello_done
        （14），certificate_verify（15），客户端密钥（16），sgx_attestation
        （17），已完成（20），（255）
} HandshakeType;
```

SGX 证明。该 SGXAttestation 可以握手服务器中任选地使用握手消息以发送所述客户端的认证 SGX（报价）。此功能与 mbTLS 的其余部分无关。sgx_quote 遵循 Intel 的 sgx_quote_t 格式。

```
结构{
        sgx 不透明引号<0..2⌢14-1>;
} SGXAttestation;
```

中间盒支持扩展。mbTLS 还添加了一个 TLS 扩展名 MiddleboxSupportExtension：

```
结构{
        uint8 numHellos;
        uint16 helloLengths [numHellos]; 不透明的
        clientHellos [numHellos]; uint8 numMboxes;
        不透明的中间盒[numMboxes];
} MiddleboxSupportExtension;
```

该 MiddleboxSupportExtension 由 TLS 客户端在发送的 ClientHello 消息。它指示客户端支持 mbTLS，邀请路径中间盒向客户端宣告自己。该扩展包含一个或多个“乐观” ClientHello，中间盒可以使用 ServerHello 响应，以及客户端先验已知的中间盒列表。

参考

[1] ARM TrustZone。 https://www.arm.com/products/security-on-arm/trustzone。访问：2017 年 1 月。

[2] Aryaka。http://www.aryaka.com。访问：2017 年 1 月。

[3] Brocade 网络功能虚拟化。 http://www.brocade.com/cn/products-services/software-networking/network-functions-virtualization 。 html。 访问：2017 年 1 月。

[4] 卷曲。https://curl.haxx.se。

[5] Google Edge 网络。 https://peering.google.com。访问：2017 年 1 月。

[6] 英特尔官方 SGX OpenSSL 库。 https://software.intel.com/zh-cn/sgx-sdk/ 下载。 访问时间：2017 年 6 月。

[7] 瞻博网络技术转型架构。https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000633-en.pdf 访问：2017 年 1 月。

[8] 功能虚拟化（Dell）。 http://www.dell.com/zh-cn/work/learn/ tme-telecommunications- solutions-telecom- nfv。访问：2017 年 1 月。

[9] 西班牙电信 NFV 参考实验室。 http://www.tid.es/long-term-innovation/network-innovation/telefonica-nfv-reference-lab 。访问：2017 年 1 月。

[10] SGX 的 TLS：mbedtls 的端口。https://github.com/bl4ck5un/mbedtls-SGX。访问时间：2017 年 6 月。

[11] Zscaler。https://www.zscaler.com。访问：2017 年 1 月。

[12] AT＆TDomain 2.0 愿景白皮书。 https://www.att.com/Common/about_us/ pdf / AT＆T%20Domain 202.0%20Vision%20White%20Paper.pdf， 2013 年。访问时间：2017 年 1 月。

[13] Graphene-SGX：针对 SGX 上未经修改的应用程序的实用库 OS。在 *2017 年 USENIX 年度技术会议（USENIX ATC 17）上*，加利福尼亚州圣克拉拉，2017 年。USENIX 协会。

[14] V. Agababov，M。Buettner，V。Chudnovsky 等。飞轮：Google 的移动网络数据压缩代理。 NSDI '15，第 367-380 页，加利福尼亚州奥克兰，5 月 2015. 。USENIX 协会。

[15] I. Anati，S。Gueron，S。Johnson 和 V. Scarlata。基于 cpu 的认证和密封的创新技术。在 *HASP '13*，2013 年第 13 卷中。

[16] S. Arnautov，B。Trach，F。Gregor 等。SCONE：使用 Intel SGX 保护 Linux 容器。在 *OSDI '16*，第 689-703 页，乔治亚州，2016 年。USENIX 协会。

[17] HJ Asghar，L。Melis，C。Soldani 等。Splitbox：实现高效的专用网络功能虚拟化。在“ *2016 年关于*中间盒*和网络功能虚拟化中的热门话题的研讨会”的论文集中*， HotMIddlebox '16，第 7-13 页，纽约，纽约，美国，2016 年。ACM。

[18] P.-L. Aublin，F.Kelbert，D.O'Keeffe 等。TaLoS：SGX Enclaves 内部的安全且透明的 TLS 终止。

[19] A. Baumann，M。Peinado 和 G. Hunt。使用 Haven 从不受信任的云中屏蔽应用程序。在 *OSDI '14 中*。USENIX-高级计算系统协会，2014 年 10 月。

[20] K. Bhargavan，I。Boureanu，P.-A。Fouque，C.Onete 和 B.Richard。通过 tls 进行内容交付：无密钥 ssl 的加密分析。在 2017 年 *第二届 IEEE 欧洲安全性和隐私性研讨会论文集中*。

[21] A. de Zylva。Windows 10 设备防护和凭据防护神秘化。 https://blogs.technet.microsoft.com/ash/2016/03/02/Windows-10-设备防护和凭据防护已解密/。 访问：2017 年 1 月。

[22] T Dierks 和 E. Rescorla。传输层安全性（TLS）协议版本 1.2。RFC 5246（建议标准），2008 年 8 月。由 RFC 5746、5878、6176 更新。

[23] Z. Durumeric，Z。Ma，D。Springall 等。HTTPS 拦截的安全性影响。在 *NDSS '17 中*，2017。

[24] SK Fayazbakhsh，L。Chiang，V。Sekar，M。Yu 和 JC Mogul。在存在使用流标签的动态中间盒操作的情况下实施网络范围的策略。在 *NSDI 14 中*，第 543-546 页，华盛顿州西雅图，2014 年。USENIX 协会。

[25] A. Gember-Jacobson，R。Viswanathan，C。Prakash 等。Opennf：启用网络功能控制方面的创新 SIGCOMM '14，第 163-174 页，美国纽约，2014 年。ACM。

[26] M. Hoekstra，R。Lal，P。Pappachan，V。Phegade 和 J. Del Cuvillo。使用创新的说明来创建值得信赖的软件解决方案。在 *HASP '13*，第 11 页，2013 年。

[27] LS Huang，A。Rice，E。Ellingsen 和 C.Jackson。在野外分析伪造的 ssl 证书。在 *IEEE 安全与隐私专题研讨会*，SP '14，第 83-97 页，美国华盛顿特区，2014 年。IEEE 计算机协会。

[28] T. Hunt，Z。Zhu，Y。Xu，S。Peter 和 E. Witchel。Ryoan：用于秘密数据的不可信计算的分布式沙箱。在 *第 12 届 USENIX 操作系统设计和实现专题讨论会（OSDI 16）*，第 533-549 页，乔治亚州，2016 年。USENIX 协会。

[29] AR Khakpour 和 AX Liu。迈向基于云的防火墙的第一步。在 *2012 年 IEEE 第 31 届可靠分布式系统研讨会上*，第 41-50 页，2012 年 10 月。

[30] S. Kim，Y。Shin，J。Ha，T。Kim 和 D. Han。向网络应用程序利用商品可信赖的执行环境的第一步。在 *HotNets-XIV 中*，第 7：1-7：7 页，美国纽约，纽约，2015 年。ACM。

[31] C. Lan，J。Sherry，RA Popa，S。Ratnasamy 和 Z. Liu。Embark：将中间盒安全地外包到云中。在 *NSDI '16*，第 255-273 页，加利福尼亚州圣克拉拉，2016 年 3 月。USENIX 协会。

[32] A. Langley，N。Modadugu 和 B. Moeller。传输层安全性（TLS）错误启动。RFC 7918（信息性），2016 年 8 月。

[33] P. Lepeska。受信任的代理和比特成本。http://www.ietf.org/proceedings/ 90 / slides / slides-90-httpbis-6.pdf， 2014 年 7 月。

[34] S. Loreto，J。Mattsson，R。Skog 等。HTTP / 2.0 中的显式受信任代理。InternetDraft 草案-loreto-httpbis-trusted-proxy20-01，IETF 秘书处，2014 年 2 月。

[35] J. Martins，M。Ahmed，C。Raiciu 等。ClickOS 和网络功能虚拟化的艺术。在 *NSDI '14*，第 459-473 页，华盛顿州西雅图，2014 年。USENIX 协会。

[36] D. McGrew，D。Wing，Y。Nir 和 P. Gladstone。TLS 代理服务器扩展。互联网草案草稿 mcgrew-tls-proxy-server-01，IETF 秘书处，2012 年 7 月。

[37] F. McKeen，I。Alexandrovich，A。Berenzon 等。创新的指令和软件模型可独立执行。在 *HASP '13*，第 10 页，2013 年。

[38] L. Melis，HJ Asghar，E。De Cristofaro 和 MA Kaafar。私有处理外包网络功能：可行性和构建。在 *2016 年 ACM 软件定义网络安全性国际研讨会的论文集中，＆#38; 网络功能虚拟化，* SDN-NFV 安全性'16，第 39-44 页，美国纽约，纽约，2016 年。ACM。

[39] J. Naous，M。Walfish，A。Nicolosi 等。使用结冰验证和加强网络路径。CoNEXT '11，第 30：1-30：12 页，美国纽约，2011 年。ACM。

[40] D. Naylor，A。Finamore，I。Leontiadis 等。HTTPS 中“ S”的成本。CoNEXT '14，第 133-140 页美国纽约，2014 年。ACM。

[41] D. Naylor，K。Schomp，M。Varvello 等。多上下文 TLS（mcTLS）：在 TLS 中启用安全的网络内功能。在 *2015 年ACM 数据通信特别兴趣小组会议上，* SIGCOMM '15，第 199-212 页，纽约，纽约，美国，2015 年。ACM。

[42] M. O'Neill，S。Ruoti，K。Seamons 和 D. Zappala。Tls 代理：敌还是友？在 *IMC '16 上*，第 551-557 页，美国纽约，纽约，2016 年。ACM。

[43] R. Peon。HTTP / 2.0 的显式代理。互联网草案 rpeon-httpbis-exproxy-00，IETF 秘书处，2012 年 6 月。

[44] E. Rescorla。传输层安全性（TLS）协议版本 1.3。InternetDraft 草案-ietf-tls-tls13-18，IETF 秘书处，2016 年 10 月。

[45] L. Schiff 和 S。施密德。PRI：加密网络流量的隐私保护检查。在 *安全和隐私研讨会（SPW）中，2016 IEEE*，第 296-303 页。IEEE，2016 年。

[46] V. Sekar，N。Egi，S。Ratnasamy，MK Reiter 和 G. Shi。整合的中间盒体系结构的设计和实现 NSDI'12，美国加州伯克利，2012 年。USENIX 协会。

[47] J. Sherry，S。Hasan，C。Scott 等。使中间盒成为其他人的问题：网络处理作为云服务。 SIGCOMM '12，第 13-24 页，美国纽约，纽约，2012 年。ACM。

[48] J. Sherry，C。Lan，RA Popa 和 S. Ratnasamy。Blindbox：对加密流量进行深度数据包检查。在 *SIGCOMM '15*，第 213-226 页，美国纽约，纽约，2015 年。ACM。

[49] 史 J.张玉和钟 S.。隐私保护网络功能外包。*CoRR*，abs / 1502.00389，2015 年。

[50] M.-W. Shih，M。Kumar，T。Kim 和 A. Gavrilovska。S-nfv：使用 sgx 保护 nfv 状态。在 *2016 年 ACM 软件定义网络安全性国际研讨会的论文集中，＆#38; 网络功能虚拟化，* SDN-NFV 安全性'16，第 45-48 页，美国纽约，纽约，2016 年。ACM。

[51] S. Shinde，DL Tien，S。Tople 和 P. Saxena。Panoply：具有 sgx 飞地的低 tcb linux 应用程序。 *2017 年 2 月 26 日至 3 月 1 日，在第 24 届年度网络和分布式系统安全研讨会上，NDSS 2017，美国加利福尼亚州圣地亚哥，* 2017。

[52] 沙利文（N. Sullivan）。无密钥 SSL：“坚韧不拔的技术细节”。 https：//博客。cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/。访问时间：2016 年 9 月。

[53] M. Walfish，J。Stribling，M。Krohn 等。中间盒不再被视为有害。OSDI'04，美国加州伯克利，2004 年。USENIX 协会。

[54] wolfSSL。带 Intel SGX 的 wolfSSL。 https://software.intel.com/zh-cn/sgx-sdk/下载。访问时间：2017 年 6 月。

[55] X. Xu，Y。Jiang，T Flach 等人。研究蜂窝网络中的透明 Web 代理。PAM '15。