多上下文 TLS（mcTLS）：在 TLS 中启用安全的网络内功能

会议论文·2015 年 8 月

DOI：10.1145 / 2785956.2787482

9 位作者，其中包括：

马特奥·瓦维罗（Matteo Varvello）

查看个人资料

杰里米·布莱克本

宾汉姆顿大学

103 出版物 **1,871 个**引用

查看个人资料

迭戈·R·洛佩兹

西班牙电信 I + D

查看个人资料

康斯坦蒂娜·帕帕吉安娜基（Konstantina Papagiannaki）

西班牙电信 I + D

143 个出版物 **8541 个** CITATIONS

多上下文 TLS（mcTLS）：
在 TLS 中启用安全的网络内功能

David Naylor *，Kyle Schomp *，Matteo Varvello *，Ilias Leontiadis *，Jeremy Blackburn *，
Diego Lopez *，Konstantina Papagiannaki *，
Pablo Rodriguez Rodriguez *和 Peter Steenkiste *

摘要

现在，很大一部分 Internet 通信已加密，HTTPS 可能是 HTTP / 2 中的默认设置。但是，传输层安全性（TLS）是 Internet 中用于加密的标准协议，它假定所有功能都驻留在端点上，因此无法使用可优化网络资源使用，改善用户体验并保护客户端和客户端的网络内服务。服务器免受安全威胁。如今，通过黑客攻击可以将网络内功能重新引入 TLS 会话，这通常会削弱整体安全性。

在本文中，我们介绍了多上下文 TLS（mcTLS），它扩展了 TLS 以支持中间盒。mcTLS 通过允许端点和内容提供商在安全的端到端会话中明确引入中间盒，同时控制他们可以读取或写入数据的哪些部分，打破了当前的“全有或全无”安全模型。

我们在受控和“实时”实验中评估了 mcTLS 实现的原型，表明其好处是以最小的开销为代价的。更重要的是，我们表明 mcTLS 可以增量部署，并且只需对客户端，服务器和中间盒软件进行少量更改。

CCS 概念

·**安全和隐私安全协议；·网络**
中间箱/网络设备；*会话协议；*

关键字词

TLS; SSL; HTTPS

SIGCOMM '15，2015 年 8 月 17 日至 21 日，英国伦敦
ISBN 978-1-4503-3542-3 / 15/08 ... 15.00 美元
DOI：http：//dx.doi.org/10.1145/2785956.2787482

1. 引言

Internet 服务的个性化增强以及对 Internet 上用户隐私的关注日益增加，导致许多服务（例如 Facebook，Twitter 和 Google）仅通过 HTTPS 提供访问。HTTPS 当前占所有 Internet 流量的很大一部分（在[26]中为 40％，估计每 6 个月以 40％的速度增长[27]）。作为 HTTPS 基础的传输层安全性（TLS）已成为 Web 端对端加密的标准，因为它可确保（i）实体身份验证，（ii）数据保密以及（iii）数据完整性和身份验证。而且，它很可能是 HTTP / 2 的默认传输协议。

这对于保护隐私是个好消息。但是，TLS 做一个基本假设：所有功能必须驻留在端点上。实际上，Internet 会话沿路径增加了功能单元，提供了入侵检测，缓存，父母过滤，内容优化（例如，压缩，代码转换）或符合企业环境中的公司惯例等服务。这些功能单元（通常称为*中间盒*）为用户，内容提供商和网络运营商提供了许多好处，正如它们在当今 Internet 中的广泛部署所证明的那样。

尽管有争论认为这些服务可以（并且应该）在端点上实现，但我们认为这通常不是最佳的，甚至是不可能的。首先，尽管可能不需要网络内实现，但它可能比基于端点的实现（例如，成千上万的用户共享 ISP 缓存）在*本质上更有效*。其次，可能有实际的原因为什么需要这种解决方案驻留在网络中（例如，病毒扫描程序可以始终是最新的，并可以立即保护所有客户端）。第三，某些功能（例如，入侵检测或基于内容的路由决策）在没有网络范围内的可见性的情况下根本无法实现。最后网络内服务可以导致最终用户增加竞争，创新和选择。

最近，爱立信和 AT＆T [20]以及 Google [28]的一项工业努力试图提供一种解决方案。

将加密与当今网络内服务的丰富性结合在一起的解决方案。但是，正如我们将要讨论的那样，仍缺乏一个好的解决方案，并且该话题在 GSMA *ENCRY* 和 IETF *httpbis 中*仍然活跃。

在本文中，我们介绍了 mcTLS，它是一种基于 TLS 的协议，它允许端点明确安全地包含具有完全可见性和控制力的网络内功能。mcTLS：（i）向端点提供明确的知识并控制*哪些*功能元素是会话的一部分，（ii）允许用户和内容提供者*动态*选择哪些部分内容暴露于网络内服务（例如，HTTP 标头与内容）；（iii）保护数据的真实性和完整性，同时仍通过分隔读写权限来允许选定的网络内服务进行修改，并且（iv）可增量部署。

我们将 mcTLS 实施为 OpenSSL 库的简单扩展。我们的评估表明，在很多情况下，mcTLS 对加载前 500 个 Alexa 站点并将 mcTLS 集成到应用程序中的页面加载时间或数据开销的影响可忽略不计。

我们的贡献如下：（i）TLS 的实用扩展，它以所需的最小访问级别将受信任的 innetwork 元素明确引入安全会话中；（ii）在受控和实时环境中测试的 mcTLS 原型实现（我们的实现是在线提供[1]），（iii）我们展示了一种高效的细粒度访问控制机制，成本低廉；（iv）使用 mcTLS 解决具体相关用例的策略，其中许多可以立即使应用受益轻松使用 mcTLS 的最基本配置。

快取		Ø	• •
压缩			• •
负载均衡器		Ø	
	入侵检测系统	OO	ØØ
家长过滤器		Ø	
追踪器拦截器		•	•
包步行者			Ø
广域网优化器		OO	ØØ

（•=读/写； o =只读）

表 1：应用层中间箱示例以及它们对 HTTP 所需的权限的示例。没有中间盒需要对所有数据的读/写访问权限。

2. 中间框和加密

显然，人们对用户隐私的兴趣与日俱增，并且在 Internet 中广泛使用网络内处理。在本节中，我们描述了中间盒，以及为什么随着 Internet 向无处不在的加密的发展而保持中间盒的好处。然后，我们解释了为什么使用 TLS 很难做到这一点。

2.1 中间盒

*中间盒*是在网络内部运行的服务，逻辑上位于通信会话的端点之间。一个*客户端*（如 Web 浏览器）可以连接到*服务器*通过一个或多个中间件（例如，网络服务器）的附加价值超越了基本的数据传输。客户端和服务器是*端点*; *用户*拥有/管理客户端，*内容提供者*拥有/管理服务器。所有各方之间的整个交流都是一个*环节*; *连接* 链接会话中的各个跃点（例如，客户端和中间盒之间的 TCP 连接）。

我们的重点是应用级中间盒，也称为*代理*或*路径内服务*，我们松散地将它们可以很好地用作访问应用程序数据的中间盒，例如入侵检测系统（IDS），内容过滤器和缓存/压缩代理（请参见表 1）。

中间盒有时被视为不受欢迎的。原因之一是隐私问题，我们稍后会解决。另一个是它们违反了最初的 Internet 体系结构，该体系结构将所有功能（传输和向上传输）都放置在端点上，这种设计是基于端到端原理[31]的。但是，Internet 发生了巨大变化：连接和内容服务都得到了商业支持，安全是主要问题，性能期望值更高，技术更加复杂，并且用户通常无权对其进行管理。结果，在何处放置功能的决定不仅取决于技术问题，而且越来越多的“在网络内部”是一个好的解决方案。

中间盒非常有用：在网络中提供处理和存储已被证明是帮助用户，内容提供商和网络运营商的有效方法。例如，诸如缓存，压缩，预取，数据包步调和重新格式化之类的技术可改善用户的加载时间[38、18]，减少运营商和用户的数据使用量[3、26、13、37、29]并降低能耗客户的消费[26、12、29、14]。中间盒还可以添加端点未提供的功能，例如企业中的病毒扫描程序或儿童内容过滤器。

网络中可能会更好： 首先，诸如缓存和数据包步调之类的功能在网络中本质上更有效[13、12、14]。其次，客户端实现可能会出现问题，因为客户端可能不受信任，或者其软件，URL 黑名单，病毒签名等可能已过时（例如，只有三分之一的 Android 用户运行最新版本的 OS 或更高版本）一半是超过两年的过期日期[2]）。最终，用户对中间盒的信任可能比对应用程序的信任更大。例如，应用程序可能会意外地将个人信息泄漏到服务器[36]，因此用户可能希望中间盒充当看门狗。

它们被广泛使用：在 2012 年对网络运营商的调查中，各种规模的网络报告的中间盒数量大约与 L3 路由器一样多[33]。特别是对于 Web 代理，Netalyzer 会话中有 14%显示出代理的证据[35]，美国所有四大移动运营商都使用代理-与前 100 个 Alexa 网站的连接都被代理，T-Mobile 上的 YouTube 除外[38]此外，Internet 中的所有参与者都使用中间盒。它们广泛部署在客户端网络（例如企业防火墙，蜂窝网络）中，并且在使用带有 TLS 中间盒的三个 IETF RFC 中，两个由运营商[20、18]领导，一个由内容提供商[28]领导。有了这项投资，中间盒不太可能消失，因此我们需要一种干净，安全的方式来将它们包括在加密会话中。

互联网是一个市场驱动的生态系统：互联网不是集中管理的垄断，而是一个市场驱动的生态系统，许多参与者都可以做出独立的决策。例如，尽管服务器可以压缩数据，但许多服务器只能选择性地压缩数据[29]。类似地，内容提供者可能决定不支持设备特定的内容格式，而是依赖于可由客户端或内容提供者选择的第三方提供者。对于诸如内容过滤之类的功能客户端可能会决定为拥有自己选择的单个中间盒提供程序而付费，而不是依赖于各个内容提供程序。企业网络可以类似地决定外包功能[33]。从根本上说，中间盒为演员提供了更多选择，从而引发了竞争和创新。

底线很简单：就像端到端加密一样，中间盒是 Internet 不可或缺的有用部分，它们将继续存在。

2.2 中间盒*和*TLS

考虑到这些趋势，自然需要两全其美。在讨论如何将中间盒和加密技术今天一起使用之前，让我们仔细研究一下传输层安全性（TLS）[11]，这是安全网络通信的标准协议。

TLS 给我们带来了什么？TLS 包含两个协议：用于会话建立的*握手协议*和用于数据交换的*记录协议*，它们共同实现三个安全属性：

*(1) 实体身份验证：*在握手期间，客户端通过验证有效证书链接服务器的域名和公共密钥来对服务器进行身份验证。客户端也可以使用证书向服务器进行身份验证，但这很少使用客户端身份验证通常在应用程序层中进行，例如使用密码。

*(2) 数据保密：*端点在握手期间建立一个对称的*会话密钥*，记录协议使用该*密钥*来对*记录*（应用程序数据块）进行加密/解密。

*(3) 数据完整性和认证：*会话密钥还用于为每条记录生成消息认证码（MAC）；有效的 MAC 表示（1）来自其他端点的数据（真实性）和（2）数据在飞行中未更改（完整性）。

如何将中间盒添加到 TLS 会话中？简而言之：您不会。通过设计，TLS 支持恰好在两方之间进行安全通信。尽管如此，中间盒还是经常插入 TLS 会话中，但这必须对 TLS 透明地进行。考虑一个企业网络，该网络想在所有员工会话中插入病毒扫描程序。常见的解决方案是在客户端上安装自定义根证书。然后，中间盒可以为其本身创建一个证书，该证书据称来自预期的服务器，并使用自定义根证书对其进行签名。客户端连接到中间盒后，中间盒连接到服务器，并将数据从一个连接传递到另一个。我们将其称为 *Split TLS*），它引起了几个问题：

*(i) 没有用于验证中间盒的机制。*更糟糕的是，中间盒对服务器是完全透明的，尽管用户可以检查证书链来检查谁签署了证书，但很少有人这样做或了解它们之间的区别。而且，即使他们这样做，他们也没有有关中间盒执行什么功能的信息。

*(ii) 客户没有超过第一跳的担保。*加密与中间盒的连接后，客户端无法验证是否正在使用从中间盒到服务器的 TLS，是否存在其他中间盒，或者（取决于所使用的应用程序级别的身份验证）会话的端点是否是预期的服务器。用户需要完全信任他没有选择甚至可能不知道存在的中间盒。

*(iii) 中间盒具有对数据流的完全读/写访问权限。*尽管许多中间盒只需要对数据流进行选择性访问，但中间盒可以读取和修改通过 TLS 会话发送和接收的任何数据（表 1）。

考虑到这些问题，用户关注（透明）中间盒就不足为奇了。甚至有人可能会争辩说，将 TLS 与中间盒一起使用比不完全使用 TLS 更为糟糕[7]，因为客户端和服务器对它们具有安全会话的错觉是幻想，而某些预期的安全属性实际上并不成立。在下一节中，我们提出一种基于 TLS 的协议，该协议明确支持中间盒并解决上述问题。

3. 多上下文 TLS（mcTLS）

本节介绍了多上下文 TLS（mcTLS）的设计，该设计通过安全引入受信任的中间盒的能力来增强 TLS。在两个端点都同意的情况下，中间盒必须由客户端或服务器显式插入，因此可以信任中间盒。我们首先总结我们的设计要求，然后介绍关键思想，最后描述协议的关键组件。可以在线获得有关 mcTLS 的更详细说明[1]。

3.1 协议要求

首先，我们要求 mcTLS 维护 TLS 提供的属性（扩展到适用于中间盒）：

R1：实体身份验证端点应该能够相互和所有中间盒进行身份验证。与今天的 TLS 使用类似，我们期望客户端将对会话中的所有实体进行身份验证，但是服务器可能不希望这样做（例如，以减少开销）。

R2：数据保密性只有端点和受信任的中间盒才能读取或写入数据。

R3：数据完整性和身份验证会话的所有成员都必须能够检测到未经授权的第三方进行的动态修改，并且端点必须能够检查数据是否由另一个端点发起（相对于已由受信任者修改过中间盒）。

其次，中间盒的引入带来了两个全新的要求：

R4：显式控制和可见性该协议必须确保在两个端点的同意下将受信任的中间盒添加到会话中。端点必须始终能够查看会话中的所有受信任的中间盒。

R5：最低特权按照最低特权[32]的原则，应该为中间盒提供执行其工作所需的最低访问级别[24，19]。中间盒只能访问与其功能相关的数据流部分；如果该功能不需要修改数据，则访问应为只读。

最后，我们的协议应该满足所有五个需求而没有大量开销，例如，在延迟，数据使用，计算，连接状态，用户或管理员的负担等方面。

3.2 威胁模型

面对在计算上受限的网络攻击者，在会话的任何阶段都可以拦截，更改，丢弃或插入数据包的情况下，成功协商的 mcTLS 会话可以满足上述要求。像 TLS 一样，mcTLS 不会阻止拒绝服务。

我们假设 mcTLS 会话中的所有参与者都正确执行了协议并且不共享带外信息。例如，客户端可以与未经服务器批准的中间盒共享密钥，或者中间盒可以串通以升级其权限。我们不考虑此类攻击，因为没有协议（包括 TLS）可以阻止一方带外共享会话密钥。此外，这种攻击是不太可能的，因为至少有两个合作方需要运行恶意的 mcTLS 实现。主流的浏览器和 Web 服务器（尤其是开放源代码的浏览器）不太可能这样做，因为它们几乎肯定会被捕获。使用 HTTP 的移动应用需要自己实现 HTTP，而不是使用平台的（诚实的）HTTP 库。

最后，即使各方都很诚实，向会话中添加更多实体也必定会增加攻击面：任何人的错误或配置错误都可能损害会话。这种风险是问题固有的，不是任何特定的解决方案。

3.3 设计概述

为了满足五个设计要求，我们向 TLS 添加了两个关键功能：

(1) 加密上下文（*R2，R3，R5*）在 TLS 中，只有两个参与方，因此限制一个参与方对部分数据的访问没有任何意义。但是，对于受信任的中间盒，端点可能希望将中间盒的访问权限限制为仅数据的一部分，或者将其授予只读访问权限。为了使之成为可能，mcTLS 将*加密上下文*（或多个*上下文*）的概念引入了 TLS。加密上下文只是一组对称的加密和消息认证代码（MAC）密钥，用于控制谁可以读写在该上下文中发送的数据（§3.4）。应用程序可以将每个上下文与一个目的（对于 mcTLS 本身是不透明的）相关联，并可以访问每个中间盒的权限。例如，Web 浏览器/服务器可以将一个上下文用于 HTTP 标头，将另一个上下文用于内容。在第 4.2 节中，我们描述了几种使用上下文的策略。

(2) 贡献上下文密钥（*R1，R4*）如果客户端和服务器选择了中间盒的证书（R1），则它们分别与每个中间盒执行密钥交换。接下来，每个端点生成每个上下文密钥的一半，并向每个中间盒发送它有权访问的上下文的半密钥，并使用上面导出的对称密钥对其进行加密。中间盒只有在接收到两个密钥的同时才获得对上下文的访问，以确保客户端和服务器都知道每个中间盒并同意其访问权限（R4）。服务器可以根据需要放弃此控件，以避免额外的计算（第 3.6 节）。

3.4 mcTLS 记录协议

TLS 记录协议从高层（例如，应用程序）获取数据，将其分为“可管理”的块，可选地进行压缩，加密，然后对每个块进行 MAC 保护，最后发送这些块。尽管每个 mcTLS 记录仅包含与单个上下文关联的数据，但 mcTLS 的工作方式几乎相同。我们将一个字节的上下文 ID 添加到 TLS 记录格式。记录序号在上下文中是全局的，以确保客户端和服务器上所有应用程序数据的正确排序，并防止对手删除未检测到的整个记录。TLS 支持的任何标准加密和 MAC 算法均可用于保护 mcTLS 中的记录。（因此，诸如加密顺序和 MAC 之类的细节取决于密码套件； mcTLS 在此不做任何更改。）

在[24，25]的基础上，mcTLS 通过控制为哪个中间盒指定了哪些上下文密钥来管理对每个上下文的访问。对于每个上下文，有四个相关方，以特权的降序排列：*端点*（客户端和服务器），*作者*（具有对上下文的写访问权限的中间框），*读者*（具有对上下文的只读访问权限的中间框）和*第三个各方*（中间箱的空白术语，在传输过程中无法访问上下文，攻击者和位翻转）。作者的更改是*合法的更改*，而读者和第三方的更改是*非法的更改*。mcTLS 实现以下三个访问控制属性：

(1) 端点只能将对上下文的读取访问限制为作者和读者。
(2) 端点可以检测到合法和非法的修改。
(3) 作家可以发现非法修改。

控制读取访问权限每个上下文都有自己的加密密钥（称为 $K_{读取器}$，如下所述）。拥有此密钥构成读取访问，因此 mcTLS 可以通过保留该上下文的密钥来防止中间盒读取上下文。

控制写访问权限通过限制谁可以生成有效的 MAC，可以**控制写访问**权限。mcTLS 对 MAC 采用以下“端点写入器-读取器”方法。每个 mcTLS 记录都包含三个密钥 MAC，分别由密钥 $K_{端点}$（由端点共享），$K_{写程序}$（由端点和写程序共享）和 $K_{读者}$（由端点，写者和读者共享）生成。每个上下文都有其自己的 $K_{作者}$和 $K_{读者}$，但该会话只有一个 $K_{端点}$，因为这些端点可以访问所有上下文。

生成 MAC

- 当一个**端点**组装的记录，它包括三个互助，一个与每个键。
- 当**写入者**修改记录时，它将与 K 个**写入者**和 $K_{读取者}$一起生成新的 MAC，并仅转发原始的 $K_{端点}$MAC。
- 当一个**读者**转发的记录，它的叶子全部三个互助修改。

检查 MAC

- 当**端点**接收到一条记录时，它将检查 K 个写入者MAC 以确认未进行任何非法修改，并检查 $K_{端点}$MAC 以找出是否进行了任何合法修改（如果应用程序在乎）。
- 当**写入者**收到记录时，它将检查 K 个**写入者**MAC，以确认没有进行任何非法修改。
- 当一个**阅读器**接收到一个记录，它使用 $k_{意图ders}$MAC 检查，如果任何*第三方 MODI-fications* 已经作出。

请注意，使用终结点写入器-读取器 MAC 方案，*读取器无法检测到其他读取器进行的非法更改*。问题在于，实体无法使用共享密钥来以相同特权级别管理其他实体。因为所有阅读器共享 $K_{阅读器}$（以便他们可以检测到第三方修改），所以所有阅读器还能够生成有效的 $K_{阅读器}$MAC。仅当上下文有两个或多个读取器时才是问题，并且通常来说，未检测到读取器修改的读取器应该不成问题（读取器修改仍可在下一个写入器或端点处检测到）。但是，如果需要，有两种方法可以解决此问题：（a）读取器和写入器/端点共享成对的对称密钥；编写者/端点为*每个*读取器计算并附加一个 MAC 或者（b）端点和编写者附加数字签名而不是 MAC；与 $K_{作者的}$MAC 不同，读者可以验证这些签名。好处似乎不足以证明（a）或（b）的额外开销是合理的，但是它们可以实现为握手期间协商的可选模式。

3.5 mcTLS 握手协议

mcTLS 握手与 TLS 握手非常相似。为了便于说明，我们在此处进行了两个简化：首先，尽管 mcTLS 握手的原理适用于 TLS 中可用的许多密码套件，但是我们使用带有 RSA 签名密钥的临时 Diffie-Hellman 描述了握手，因为它很容易说明和说明。在实践中很常见。其次，我们用一个中间盒描述握手，但是将其扩展到多个中间盒很简单。表 2 定义了我们在本文中使用的表示法。

握手的目的是：

- 允许端点在密码套件，一组加密上下文，中间盒列表以及这些中间盒的权限上达成一致。
- 允许端点彼此和所有中间盒（如果他们选择的话）进行身份验证。
- 在*端点*之间建立共享的对称密钥 K 端点。

符号	意义
DH ê，DH Ē	实体 E 的临时 Diffie-Hellman 公钥/私钥对
DHCombine （，	结合 Diffie-Hellman 公钥和私钥以生成共享机密
PK ê，PK Ē	实体 E 的长期签名公钥/私钥对（例如 RSA）
登录 PR .（•）	使用 E 的私钥签名
小号 Ë	仅实体 *E* 知道的秘密
PS E 1 - E 2	实体 E ₁和 E ₂共享的秘密前
小号	实体 E ₁和 E ₂之间共享的秘密
PRF 秒（•）	TLS RFC [11]中定义的用秘密 *S* 键控的伪随机函数
ᴷ E1-E2	E ₁和 E ₂共享的对称密钥
ķ ᵉ	实体 *E* 生成的关键材料
Enc ᴷ（•）	使用密钥 *K 进行*对称加密
MAC ᴋ（•）	带有密钥 *K 的*消息身份验证代码
AuthEnc ᴷ（•）	使用密钥 *K 进行*身份验证的加密
高（•）	加密哈希
II	级联

表 2：本文使用的符号。

•为所有编写者之间的每个上下文建立一个共享的对称密钥 *K 编写者*，并为所有读者之间的每个上下文建立一个共享的对称密钥 *K 读者*[1]。

握手下面，我们解释 mcTLS 握手的步骤（如图 1 所示），重点介绍与 TLS 的区别。请注意，它具有与 TLS 相同的 2-RTT“形状”。

设置：每一方生成一个（公共）随机值和一个临时 Diffie-Hellman 密钥对（中间盒生成两个密钥对，一个用于客户端，一个用于服务器）。每个端点还生成一个秘密值。

。2 **客户端**问候：与 TLS 一样，mcTLS 会话以包含随机值的 ClientHello 消息开始。在 mcTLS 中，客户问候携带 MiddleboxListExtension，其中包含：（1）中间件的列表，我们将讨论建立这个名单摆在首位的会话而包括在§ 6.1 和（2）加密上下文的列表，它们的目的（对应用程序有意义的字符串）以及每个上下文的中间箱访问权限。客户端打开一个与中间盒的 TCP 连接，并发送 ClientHello。中间盒将打开与服务器的 TCP 连接，并转发 ClientHello。

证书和公钥交换：与 TLS 中一样，服务器以一系列消息作为响应

[1]尽管为简单起见，我们将每个描述为一个密钥，但 *K 个读取器* 和 *K 个端点*实际上每个都是四个密钥（就像 TLS 中的“会话密钥”一样）：每个方向上的数据的加密密钥和每个方向上的数据的 MAC 密钥。同样，*K 个作者*实际上是每个方向的一个 MAC 密钥。

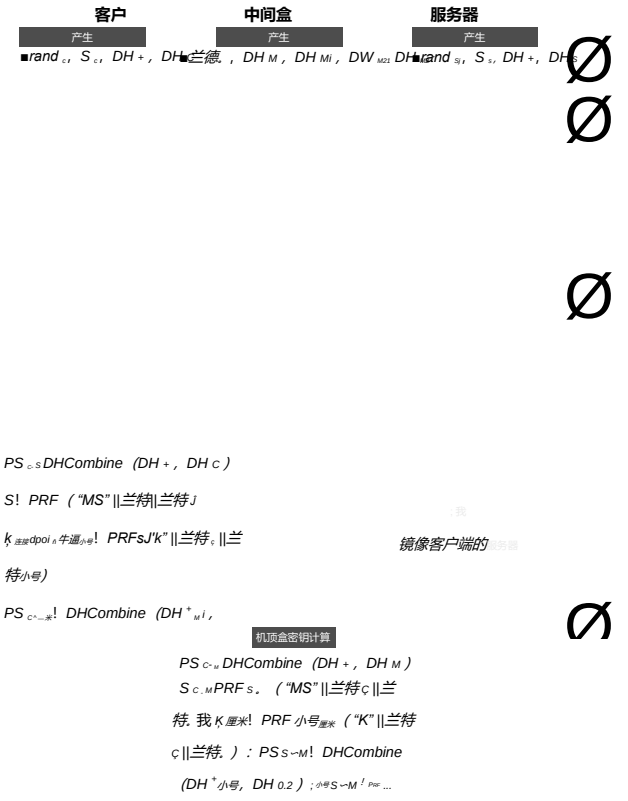

图 1：mcTLS 握手。

包含一个随机值，其证书以及由证书中的密钥签名的临时公共密钥。中间盒的作用相同：它将其随机值，证书和临时公共密钥发送到客户端和服务器。客户端发送一个临时的公共密钥，中间盒将其保存并转发给服务器。中间盒将其消息背负在 ServerKeyExchange 和 ClientKeyExchange 消息上（由虚线箭头指示）。临时密钥提供前向保密性；中间盒为客户端和服务器使用不同的密钥对，以防止小型子组攻击[22]。

杂凑	12 + 6N	0	12 + 6N	10 + 5N	0	10 + 5N	10	20	10
秘密比赛	N + 1	2 个	*N* + 1	N + 1	1 个	1 个	1 个	2 个	1 个
关键一代	4K + N + 1	（k < 2K）+2	4K + N + 1	2K + N + 1	1 个	1 个	1 个	2 个	1 个
非对称验证	N + 1	*n* < 1	n < N	N + 1	n < 1	0	1 个	1 个	0
符号加密	N + 2	0	N + 2	N + 2	0	1 个	1 个	2 个	1 个
符号解密	2 个	2 个	2 个	1 个	1 个	2 个	1 个	2 个	1 个

（*N* =中间盒的数量；*K* =上下文的数量）

表 3：握手期间，客户端，中间盒和服务器执行的加密操作。假定没有 TLS 扩展，DHE-RSA 密码套件，并且客户端未使用证书进行身份验证。

Q **共享密钥计算：**该**客户端**计算两个秘密（S ç-中号和*小号ç -小号*使用从服务器和中间盒，它用来产生具有所述中间共享的对称密钥的贡献）（*K* Ç-中号）和服务器（*K 个端点*）。客户端还产生"部分密钥，"*ķ W一riters* 和 *ķ* Çeaders，为每个上下文，使用已知不仅对本身是个秘密。

所述**服务器**遵循相同的程序，在客户端，从而导致 *ķ 端点，Ķ 小号-米，ķ 作家，*和 *ķ 小号 eaders*。服务器可以选择通过要求客户端生成并分发完整的上下文密钥来避免这种开销（第 3.6 节）。

当**中间盒**接收到 ClientKeyExchange 时，它分别使用客户端和服务器的临时公共密钥来计算 *K C-M* 和 *K S-M*。稍后它将使用这些密钥来解密来自客户端和服务器的上下文密钥材料。

0 **上下文密钥交换：**接下来，对于每个上下文，端点将部分上下文密钥发送到中间盒 *(*如果具有读取访问权限，则将 *ĸ Ceaders* 和 *Meaders* 发送给中间盒；如果具有写入访问权限，则将 *K C riters* 和 *K W riters* 发送给中间盒）。这些消息在 *K C-M* 和 *K S-M* 下以加密和身份验证的方式发送，确保部分上下文密钥的保密性和完整性。中间盒将每个消息转发到相对的端点，以便可以将其包含在握手消息的哈希中，该哈希在握手结束时进行了验证。端点还将所有部分上下文密钥发送到在 *K 个端点*下加密的相对*端点*。中间盒转发此消息（但无法读取）。

© **上下文密钥计算：**客户端指示应通过发送 ChangeCipherSpec 消息来使用握手中协商的密码。收到 ChangeCipherSpec 消息会提示所有各方使用与部分上下文密钥的级联键入密钥的 *PRF*（- ）来生成上下文密钥。这种"部分密钥"方法有两个目的：（1）提供贡献密钥协议（两个端点都提供随机性）；（2）确保只有在客户端和服务器都同意的情况下，中间盒才能访问上下文。

Q 已**完成：**mcTLS 握手以完成消息的交换结束。与 TLS 中一样，"完成"消息包含所有握手消息（包括定向到中间盒的握手消息）的级联的哈希：*PRF 的。s（finished_la，bel，H（消息））*。验证此消息可确保两个端点都观察到相同序列的相同握手消息，即，在运行中未修改任何消息。

详细信息 mcTLS 和 TLS 握手之间有一些细微的差异。我们在此简要介绍这些更改，并争论为什么它们是安全的；有关更详细的安全性分析，请参见[1]。

- 为简单起见，中间盒无法协商会话参数（例如，密码套件或上下文数量）。如果需要，可以在以后的工作中考虑更复杂的协商协议。
- 服务器的上下文密钥材料不包含在客户端的"完成"消息中，因为这将需要额外的 RTT。但是，此密钥材料是经过加密和 MAC 保护的，因此对手无法学习或修改它。
- 客户端无法解密服务器发送中间盒的上下文密钥材料反之亦然。这将需要在两个端点和每个中间盒之间建立一个三向对称密钥。由于中间盒需要来自*两个*端点的密钥材料，因此一个恶意端点无法单方面增加中间盒的权限。
- 中间盒无法验证 Finished 消息中的握手哈希，因为它不知道 *K 个端点*。为了避免开销，我们不包括每个中间盒已完成的消息。这意味着中间盒可能会观察到错误的握手消息序列。但是，这最多是拒绝服务攻击。例如，即使中间盒无法检测到密码套件降级攻击，端点也将检测到它并终止会话。此外，上下文密钥材料在每个端点与中间盒共享的密钥下以加密和 MAC 保护的方式发送，因此，只要至少一个端点验证中间盒的证书，对手就无法学习或修改上下文密钥。

3.6 减少服务器开销

关于部署 TLS 的一个关注点（尽管正在减少[17，4]）是握手对计算的要求很高，从而限制了每秒钟服务器可以处理的新连接数。我们不想在 mcTLS 中使此问题变得更糟，而避免这种情况的一种方法是将某些功能设为可选。例如，类似于 TLS，在会话中对实体的身份验证是可选的-在某些情况下，服务器可能不在乎中间盒是谁。mcTLS 中服务器的另一个负担是生成和加密部分上下文密钥以分发到中间盒。可以选择将其移至客户端，而不必在客户端和服务器之间拆分此工作：上下文密钥是由端点共享的主密钥生成的，客户端对它们进行加密并将其分发到中间盒（“客户端密钥分发模式”）。这减少了服务器负载，但是它的缺点是未强制执行有关中间盒权限的协议。（请注意，这不会牺牲两个端点都具有随机性的意义，因此不会牺牲贡献密钥一致性。客户端从它与服务器共享的秘密中生成上下文密钥；如果客户端/服务器密钥交换是贡献性的，则上下文密钥会继承此好处。选择握手模式留给内容提供者，他们可以单独决定如何进行这种控制性能的权衡；服务器在 ServerHello 中向客户端指示其选择。但是它的缺点是未强制执行有关中间盒权限的协议。（请注意，这不会牺牲两个端点都具有随机性的意义，因此不会牺牲贡献密钥一致性。客户端从它与服务器共享的秘密中生成上下文密钥；如果客户端/服务器密钥交换是贡献性的，则上下文密钥会继承此好处。）选择握手模式留给内容提供者，他们可以单独决定如何进行这种控制性能的权衡；服务器在 ServerHello 中向客户端指示其选择。但是它的缺点是未强制执行有关中间盒权限的协议。（请注意，这不会牺牲两个端点都具有随机性的意义，因此不会牺牲贡献密钥一致性。客户端从它与服务器共享的秘密中生成上下文密钥；如果客户端/服务器密钥交换是贡献性的，则上下文密钥会继承此好处。）选择握手模式留给内容提供者，他们可以单独决定如何进行这种控制性能的权衡；服务器在 ServerHello 中向客户端指示其选择。如果客户端/服务器密钥交换是有贡献的，则上下文密钥将继承此好处。）选择握手模式留给内容提供者，内容提供者可以单独决定如何进行此控制性能的权衡。服务器在 ServerHello 中向客户端指示其选择。如果客户端/服务器密钥交换是有贡献的，则上下文密钥将继承此好处。）选择握手模式留给内容提供者，内容提供者可以单独决定如何进行此控制性能的权衡；服务器在 ServerHello 中向客户端指示其选择。

表 3 比较了 mcTLS 和 2.2 节中描述的分割 TLS 方法执行的加密操作的数量。我们显示了不带和带客户端上下文密钥分发的 mcTLS 编号。如果我们考虑一个带有单个中间盒（$N = 1$）的简单示例，则使用客户端密钥分发模式的额外服务器负载将限于少量的轻量级操作（哈希和 Sym 解密）。

4. 使用 mcTLS

4.1 使用上下文

正如 HTTP 的架构师必须定义其在 TLS 上的运行方式[30]一样，协议设计者也需要标准化其应用程序如何使用 mcTLS。从应用程序开发人员的角度来看，mcTLS 带来的最

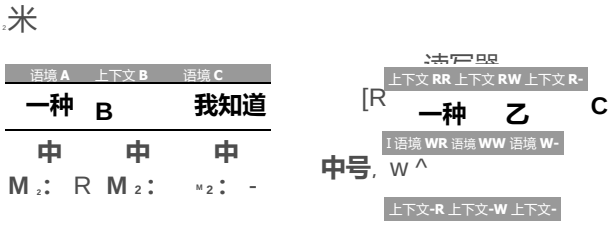

图 2：使用加密上下文的策略：按节上下文（左）和按权限上下文（右）。

可以为每种权限组合创建一个上下文，并在发送文档的每一部分时使用适当的上下文（右图 2）。

哪种模型合适取决于使用情况：在每部分上下文模型中，n 个部分表示 n 个上下文。在许可的上下文模型中，m 个中间框表示 3^m 个上下文。在实践中，我们期望这些数字中的至少一个很小，因为会话中的数据通常没有很不同的敏感度级别，并且由于大多数中间盒都需要类似的权限（表 1）。例如，对于 HTTP，我们想象四个上下文就足够了：请求标头，请求正文，响应标头和响应正文。（尽管您可以想象极端的情况，其中每个 HTTP 标头都有其自己的访问控制设置。）

最后，以上示例使用静态上下文分配，但是也可以动态选择上下文。一个应用程序可以创建两个上下文，一个可以被中间箱读取，而另一个则不能，然后在它们之间切换以即时启用或禁用中间箱访问（例如，响应于特定的用户代理而启用压缩）。

4.2 用例

数据压缩代理许多用户（尤其是在移动设备上）使用 Chrome 的数据压缩代理[3]之类的代理来重新缩放/重新编码图像，以减少其数据使用量。但是，Google 的代理当前忽略 HTTPS 流。使用 mcTLS，用户可以指示其浏览器为压缩代理提供对 HTTP 响应的写访问权限。进一步，浏览器和 Web 服务器可以协调以使用两个上下文进行响应：一个用于代理可以访问的图像，另一个用于代理不能访问的 HTML，CSS 和脚本。上下文分配甚至可以动态更改：如果移动用户连接到 Wi-Fi 页面中加载，由于不再需要压缩，因此图像也可能会在无访问上下文中传输。

父母过滤图书馆和学校，有时甚至是整个国家[8]，通常都使用过滤器来阻止年龄不适当的内容。此类过滤器通常取决于查看完整的 URL（Internet Watch Foundation 黑名单中只有 5％的条目）

是整个（子）域[26]）。借助 mcTLS，IT 员工可以将其计算机配置为允许其过滤器对 HTTP 请求标头进行只读访问，并且可以将连接到网络的用户拥有的设备配置为通过 DHCP 动态地执行此操作。筛选器将丢弃不兼容的连接。

公司防火墙大多数公司通过入侵检测系统（IDS）/防火墙/病毒扫描程序将所有网络流量汇集在一起。当前，这些设备要么忽略加密的流量，要么在员工的设备上安装根证书，从而透明地允许他们访问所有“安全”会话。借助 mcTLS，管理员可以将设备配置为授予 IDS（用户现在可以看到）只读访问权限。安全设备不再需要模拟终端服务器，并且用户不再习惯于安装根证书。

网上银行尽管我们设计了 mcTLS，以使用户能够控制其会话但是在某些情况下，内容提供商确实比用户更了解，并且应该能够对中间盒说“不”。一个典型的例子是网上银行：银行有责任保护粗心或非技术用户避免与第三方共享其财务信息。通过简单地不向中间盒提供其上下文密钥的一半，服务器可以轻松地防止这种情况的发生，而不管客户端分配的访问级别如何。

HTTP / 2 流 HTTP / 2 的功能之一是在单个传输连接上复用多个流。mcTLS 使浏览器可以轻松地为每个流设置不同的访问控制。

5. 评估

mcTLS 的细粒度访问控制要求生成和分配额外的密钥，计算额外的 MAC，并且可能发送比 TLS 更大数量的较小记录。在本节中，我们将评估此开销。

实验设置我们通过修改 TLSv1.2 的 OpenSSL [2]实现构建了 mcTLS 原型。该原型支持 mcTLS 默认模式的所有功能，如第 3.4 节和第 3.5 节中所述。我们使用 DHE-RSA-AES128-SHA256 密码套件，尽管没有什么阻止 mcTLS 与任何标准密钥交换，加密或 MAC 算法一起使用。此外，尽管应该使用从端点和中间盒之间的 DHE 密钥交换生成的密钥对 MiddleboxKeyMaterial 消息进行加密，但是为了简化实现，我们使用 RSA 公钥加密。结果，*向前保密*目前不受我们的实施支持。mcTLS 需要对 API 进行一些补充，例如，定义上下文及其读取/写入权限，但是它们类似于当前的 OpenSSL API。

接下来，我们编写了一个简单的 HTTP 客户端，服务器和代理，并支持四种操作模式：

(1) mcTLS：使用 mcTLS 传输的数据。
(2) SplitTLS：在跃点之间拆分 TLS 连接；中间盒解密并重新加密数据。
(3) E2E-TLS：单个端到端 TLS 连接；中间盒盲目转发加密数据。
(4) NoEncrypt：不加密；通过 TCP 进行明文传输和转发的数据。

我们检测了 mcTLS 库和我们的应用程序，以测量握手持续时间，文件传输时间，数据量开销和每秒连接数。

我们在两种环境中进行测试。（1）*受控*：客户端，中间盒和服务器都在一台计算机上运行。我们使用 tc 控制带宽（除非另有说明，否则应从 SpeedTest.net 样本的中位数中选择 10 Mbps）和延迟。 （2）*广域*：我们负责客户，分别位于西班牙，爱尔兰和加利福尼亚的 EC2 实例上的中间盒和服务器。客户端通过光纤或 3G 连接。除非另有说明，否则在任一环境中进行的实验均由 50 次运行组成，我们将其报告为平均值；误差棒表示一个标准偏差。中间盒被赋予对每个上下文的完全读/写访问权限，因为这是 mcTLS 性能最差的情况。

5.1 时间开销

握手时间图 3（左）显示了随着上下文数量的增加，到达第一个字节的时间。有一个中间盒，每个链接有 20 毫秒的延迟（总 RTT 为 80 毫秒）。NoEncrypt 用作基线，到第一个字节的时间为 160 毫秒，即 2 个 RTT。多达 9 个上下文，mcTLS，E2E-TLS 和 SplitTLS 各自占用 4 个 RTT。在 10 个上下文中，mcTLS 跳到 5 个 RTT，在 14 个跳到 7 个。

罪魁祸首是 TCP 的 Nagle 算法，该算法会延迟数据传输，直到准备好发送完整的 MSS。在 10 个上下文中，从代理到服务器的握手消息超过 1 个 MSS，Nagle 保留了额外的字节，直到第一个 MSS 被确认为止。在 14 个上下文中，来自客户端（+1 RTT）和服务器（+1 RTT）的中间盒密钥材料发生了相同的事情。禁用 Nagle 算法（在实践中并不罕见）[23]解决了这个问题。我们也尝试了不带 Nagle 的 E2E-TLS，SplitTLS 和 NoEncrypt，但是由于它们的消息从未超过 1 个 MSS，因此它们的性能并没有提高。

到第一个字节的时间与中间盒的数量成线性比例，因为在我们的实验中添加中间盒还增加了 20 ms 的链接（右图 3）。延迟增加以及分发的额外关键材料加剧了 Nagle 所引起的问题；再次禁用它会使 mcTLS 性能与 E2E-TLS 和 SplitTLS 保持一致。最后，如果中间盒直接位于数据路径上（这经常发生），那么唯一的额外开销就是处理时间，这是最短的时间。***要点：****mcTLS 的握手时间不比 SplitTLS 或 E2E-TLS 的握手长。*

[2] 2014 年 10 月起的 OpenSSL v1.0.1j

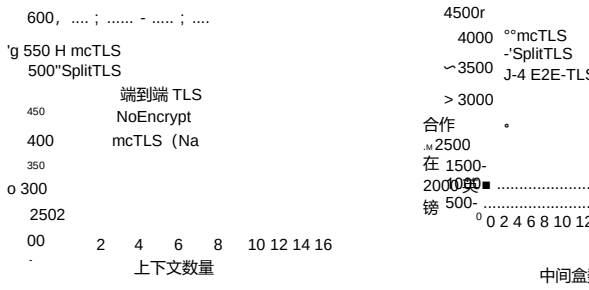

图 3：第一个字节的时间与＃个上下文（左）和＃个中间框（右）的关系。

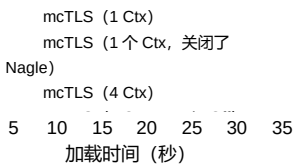

图 4：不同数量的 mcTLS 上下文的页面加载时间。

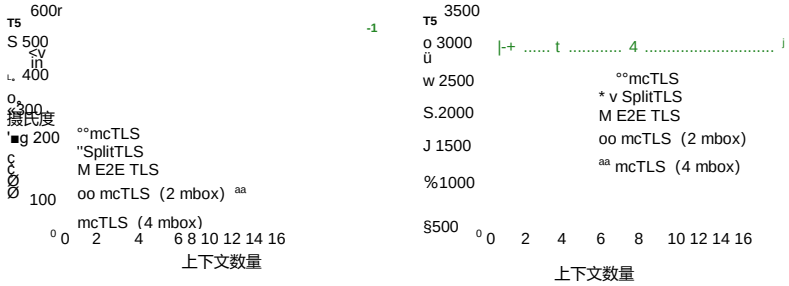

图 5：在服务器（左）和中间盒（右）处承受可持续的负载。

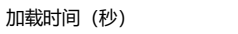

图 6：页面加载时间。

文件传输时间接下来，我们通过通过单个中间盒传输文件来探索每个完整协议的时序行为。要选择实际的文件大小，我们加载了前 500 个 Alexa 页面，并选择了第 10、50 和 99％个对象大小（分别为 0.5 kB，4.9 kB 和 185 kB）。我们还考虑了较大的下载量（10MB）（例如，较大的 zip 文件或视频块）。

图 7 中的前四个条形图组显示了以 1 Mbps 的速度增加文件大小时的下载时间。每个小节代表 10 次重复。如预期的那样，握手开销在较小的文件（<5 kB）中占主导；所有协议的使用加密需要额外的∽相比 NoEncrypt 17 毫秒。mcTLS 可与 E2E-TLS 和 SplitTLS 相提并论。当以不同的链接速率或在广域（最后四个条形组）下载文件时，我们看到相同的行为。握手和数据传输控制着下载时间；协议特定的处理没有什么区别。

要点：*无论链接类型，带宽或文件大小如何，mcTLS 传输时间都不会明显高于 SplitTLS 或 E2E-TLS。*

页面加载时间为了了解上述微基准如何转化为实际性能，我们检查了网页加载时间。尽管我们尚未将功能完善的 Web 浏览器移植到 mcTLS，但我们可以如下估算简单客户端中的整个页面加载量。首先，我们在 Chrome 中加载所有支持 HTTPS 的 Alexa 前 500 个页面。对于每个页面，我们提取一个列表，列出加载的对象，它们的大小以及是否现有的连接被重新使用以获取每个连接（我们无法确定使用了*哪个*连接，因此我们将对象分配给随机选择的现有连接）。接下来，我们的客户端通过从服务器请求适当大小的虚拟对象来“回放”页面加载。我们进行简化的假设，即每个对象仅依赖于相同连接中加载的先前对象（这可能会引入错误的依赖关系，而忽略真实的依赖关系）。

首先，我们比较三种 mcTLS 策略：1-Context （所有数据在一个上下文中），4-Context（请求标头，请求正文，响应标头，响应正文）和 ContextPerHeader （每个 HTTP 标头一个上下文，一个用于请求正文，一个为响应正文）。图 4 显示了每种策略的页面加载时间的 CDF。该图显示了每种策略的相似性能，表明 mcTLS 对将数据放入上下文的方式并不太敏感。

接下来，我们将 mcTLS 与 SplitTLS，E2E-TLS 和 NoEncrypt 进行比较（图 6）。我们将 4 上下文策略用于 mcTLS，因为我们认为它将是最常见的策略。SplitTLS，E2E-TLS 和 NoEncrypt 的性能相同，而 mcTLS 增加了半秒或更长时间。再次要怪 Nagle：在多个上下文中发送数据会导致对 TCP 的背对背 send（）调用。第一条记录将立即发送，但随后的记录将保留，因为它们比 MSS 小并且正在传输未确认的数据。在关闭 Nagle 的情况下重复实验，可以缩小差距。

要点： *mcTLS 对现实世界的网页加载时间没有影响。*

图 7：各种链接速度和文件大小配置的文件下载时间。

图 8：握手大小。

5.2 数据量开销

mcTLS 的大部分数据开销来自握手，并且随着中间盒和上下文（由于证书和上下文密钥分发）的数量而增加。图 8 显示了不同数量的上下文和中间盒的握手总大小。对于具有一个上下文且没有中间盒的基本配置，mcTLS 握手为 2.1 kB 而 SplitTLS 和 E2E-TLS 为 1.6 kB。请注意，握手大小与文件大小无关。

接下来，每个记录都携带 MAC（在 mcTLS 中为三个，在 TLS 中为一个）。它们的影响取决于应用程序的发送模式-较小的记录意味着较大的开销。对于第 5.1 节中的网络浏览实验，与 NoEncrypt 相比，SplitTLS 的 MAC 开销中位数为 0.6%；如预期的那样，mcTLS 将其增加了两倍，达到 2.4%。

最后，填充和标头开销可以忽略不计。***要点：*** *除了最初的握手开销（对于短连接以外的所有开销都可以忽略不计）之外，与 SplitTLS 或 E2E-TLS 相比，mcTLS 为 Web 浏览引入了不到 2%的额外开销。*

5.3 CPU 开销

图 5（左）显示了服务器每秒可以维持的连接数（仅握手）。我们看到用于分发中间盒密钥材料的额外非对称加密造成了巨大的损失。mcTLS 服务器处理的连接数比 SplitTLS 或 E2E-TLS; 随着上下文数量的减少，该数量减少了 35%，因此服务器必须加密的部分上下文密钥的数量增加了。我们注意到两件事：（1）我们打算在将来的工作中进行的密钥分发优化可以缩小这一差距，并且（2）如果客户端处理密钥生成/分发，服务器可以收回这种损失的性能（第 3.6 节）。

中间盒的结果更有趣（右图 5）。首先，由于 E2E-TLS 不参与 TLS 握手，因此其性能明显优于 mcTLS 和 SplitTLS（请注意 Y 比例的变化）。其次，mcTLS 的性能*优于* SplitTLS，因为在 SplitTLS 中，代理必须参与*两次* TLS 握手。这些结果表明，在核心网络中使用中间盒不仅可行，而且很实用。

要点：*与 TLS 相比，mcTLS 服务器每秒可减少 23%-35%的连接，但 mcTLS 中间盒可以为 fTLS -75%以上。*

5.4 部署

为了开始理解可部署性，我们构建了 Ruby SSL 库的扩展，使用少于 250 行的 C 代码添加了对 mcTLS 的支持。然后，使用该扩展程序，我们构建了一个 17 行的 Ruby Web 客户端，该客户端具有与基于 C / OpenSSL 的评估客户端相同的功能。为了使扩展更像 Ruby，还需要做更多的工作，但使用像 RubyMotion [3] 这样对开发人员友好的工具轻松编写支持 mcTLS 的移动应用程序的潜力是有希望的。

我们还修改了 OpenSSL s_time 基准测试工具以支持 mcTLS。同样，只需进行最小的更改：添加了少于 30 行的 C 代码，略微更改了大约 10 行。这意味着要获得 mcTLS 的全部好处，需要相对较少的开发人员精力。

虽然支持细粒度的访问控制需要将数据分配给上下文并为这些上下文设置中间盒权限的工作量最小，但是只要获得 HTTP 客户端库和服务器的支持，mcTLS 的许多好处就可以立即获得。例如，默认情况下，HTTP 库可以使用 4-Context 策略，不需要其他程序

[3] http://www.rubymotion.com/

应用程序开发人员的建议或努力。最后，我们注意到，如果无法协商 mcTLS 连接，则客户端和服务器很容易退回到常规 TLS。***总结：*** *将应用程序或库升级到 mcTLS 似乎很简单容易。*

6. 讨论

6.1 Middlebox 发现

mcTLS 假设客户端在初始化握手之前已拥有一个中间盒列表，该列表包含在 ClientHello 中。建立此列表在很大程度上与 mcTLS 本身正交。可以使用许多现有机制，具体取决于*谁*试图向会话添加中间盒。例如：

- **用户**或**系统管理员**可以直接配置客户端（应用程序或操作系统）（例如，用户可能将其浏览器指向 Google 的 SPDY 代理）。如果用户对例如“附近”的数据压缩代理而不是特定的代理感兴趣，则客户端可以使用 mDNS [10]或 DNS-SD [9]发现可用的代理。
- **内容提供商**可以指定在使用 DNS 与其服务器的任何连接中使用的中间盒。
- **网络运营商**可以使用 DHCP 或 PDP / PDN 将任何所需的中间盒（例如，病毒扫描程序）通知客户端。

如果像这样的先验机制不够灵活，则可以扩展握手以允许（例如，路径上的中间盒）在会话建立过程中插入自身（当然要经过端点的后续批准）。成本和收益尚不清楚。我们为将来的工作留出了更复杂的会话协商的细节。

6.2 用户界面

在没有合适的界面来控制用户的情况下，将中间盒添加到安全的通信会话中的技术解决方案几乎没有意义。这种接口的主要挑战是：

- *向用户指示会话“安全”。”*由于 TLS 和 mcTLS 的语义不同，因此重复使用众所周知的锁定图标会产生误导。
- *与可以做什么的用户进行沟通。*哪些中间盒可以读取用户的数据？哪个可以修改呢？他们做了哪些修改？谁拥有中间盒？谁将它们添加到会话中，为什么？
- *允许用户设置访问控制。*中间盒可以看到哪些会话？在这些会话中，它可以读取或写入哪些字段？困难在于使这样的控件简单且可扩展。例如，要求用户为他们访问的每个域设置中间盒权限是不切实际的。

在 mcTLS 之上设计令人满意的接口本身就是一个项目，我们不能在这里开始伸张正义。

	R1	R2	R3	R4	R5
mcTLS	•	•		•••	
（1）海关证明					
（2）代理证书标志	ø			ø	
（3）会话密钥带外		•	•	ø	
（4）自定义浏览器					
（5）代理服务器扩展	ø	ø		oo	

（•=完全符合；o=部分符合）

表 4：mcTLS 和竞争性提案的设计原则合规性。

7. 相关工作

最近，特别是在工业上，已经有很多兴趣将中介程序包含在加密会话中。首先，我们在 TLS 的背景下描述了五个建议。如表 4 所示，它们都不符合我们的全部五个要求。然后，我们讨论完全替代 TLS 的替代方案。

(1) 自定义根证书第 2.2 节描述了网络管理员在客户端上安装自定义根证书的常用技术。

*讨论：*此技术不能满足我们的任何要求。首先，服务器，在许多情况下是客户端，都不知道中间盒（R4）的存在，因此它显然无法对其进行身份验证（R1）。其次，中间盒对会话（R5）中的所有数据具有完全的读取和写入访问权限。最后由于客户端在第一跳之后没有控制权，因此无法保证数据（R2，R3）或服务器（R1）的保密性，完整性或真实性。

(2) “我是代理人”证书标志爱立信和 AT&T 于 2014 年提出的 IETF 草案建议使用 X.509 扩展密钥用法扩展来表明证书属于代理人[20]。在 TLS 握手期间接收到此类证书后，用户代理将省略域名检查（大概具有用户许可），并与代理建立 TLS 会话，这反过来又将打开与服务器的连接。根据用户首选项，用户代理可能仅接受某些会话的代理证书。

*讨论：*在这种情况下，使客户端明确知道中间盒的存在，因此它可以对其进行身份验证（R1），并可以基于每个连接（R4 控制其使用。客户端仍然无法验证服务器，并且服务器不知道中间盒。R2，R3 和 R5 保持未寻址状态。

(3) 通过会话密钥带外通过 Google 的另一个 IETF 草案假设客户端与代理保持持久的 TLS 连接，并在该连接上多路复用多个会话（这与 Google 的数据压缩代理的运行方式非常相似）。与服务器建立端到端 TLS 连接（代理盲目转发）后，客户端在传输之前将会话密钥传递给代理

在新连接上发送数据[28]。再次，用户代理可以基于用户偏好在每个连接的基础上选择性地授予代理访问权限。

*讨论：*与（1）相比，此解决方案的另一个好处是，客户端对中间盒和服务器都进行了身份验证（R1），并且知道会话是端到端加密的（R2）。R3，R4 和 R5 仍然部分或完全未寻址。

(4) 交付自定义浏览器第四个选项是修改浏览器本身，以接受来自某些受信任代理的证书。这是 Viasat 为其 Exede 卫星互联网客户所采用的方法[18]，认为缓存和预取对高延迟链接至关重要。

*讨论：*此解决方案本质上与（1）相同，因此它也不能满足所有要求。另外，其缺点是定制浏览器可能无法快速更新，开发和维护成本高昂，并且可能给用户带来不便。

(5) 代理服务器扩展迄今为止，最有前途的方法是 Cisco 的 TLS 代理服务器扩展[21]。代理从客户端接收 ClientHello，与服务器建立 TLS 连接，并将服务器的证书和有关为代理服务器连接协商的密码套件的信息包含在 ProxyInfoExtension 中，该信息附加在返回给客户端的 ServerHello 中。然后，客户端可以同时检查代理服务器和服务器的证书。

*讨论：*客户端必须*完全*信任中间盒，以提供有关服务器证书和密码套件的诚实信息，因此此解决方案仅部分满足 R1，R2 和 R3。代理不一定对服务器可见，因此仅部分 R4。最后，代理具有对所有数据（R5）的读/写访问权限。

其他方法基于 TLS 的技术的一种替代方法是对 IPsec 的扩展它允许在安全性关联的两个端点之间对有效负载的某些部分进行加密/身份验证，而其余部分则一目了然[16]。作者针对这种架构，旨在为无线移动用户安全地启用基于中介的服务。该解决方案使中间盒的数据完全未加密（R2）；R1 和 R3 也被违反。此外，这种方法不允许显式控制流向不同实体（R4）的数据。

Tcpcrypt [6，5]是用于建立端到端加密会话的替代方案。与 TLS 相比，它减少了服务器的开销，将身份验证留给了应用程序，可以嵌入到 TCP 握手中，并使用会话 ID 明确标识会话的端点。与 TLS 相似，tcpcrypt 仅支持两个端点之间的通信，但是我们认为加密上下文和贡献上下文密钥的概念也可以应用到它。但是，由于 mcTLS 如果增加了握手大小，则可能无法再将整个握手嵌入到 TCP 握手中。

传输层协议（例如 TLS 或 mcTLS）的替代方法是在网络层支持受信任的中介。面向委托的体系结构（DOA）[34]和命名数据网络（NDN）[15]通过它们自己的安全性机制和属性来完成此任务。由于它的广泛使用，我们选择修改 TLS，使其成为立即试验和增量部署的理想工具。

8. 结论

Internet 服务对 TLS 的越来越多的使用提供了隐私和安全性，但同时也导致功能的丧失，这些功能通常是由提供安全性，压缩，缓存或内容/网络资源优化的无形中间盒提供的。要找到一种既可以带来这些好处又又能保持客户端，内容提供商和网络运营商对安全性期望的解决方案，并非易事。mcTLS 通过扩展 TLS 来做到这一点，TLS 已经承载了很大一部分 HTTP 流量。mcTLS 专注于透明性和控制：（1）在客户端和服务器的同意下引入受信任的中间盒；（2）在每个会话的基础上；（3）具有明确的访问权限（读/写）；以及（4）数据流的特定部分。

我们表明，构建这样的协议不仅可行，而且还会在延迟，加载时间和数据开销方面引入有限的开销。更重要的是，mcTLS 可以增量部署，并且只需要对客户端和服务器软件进行少量修改即可支持大多数预期的用例。通过使用 mcTLS，在用户和内容提供商的明确同意下，安全的通信会话可以恢复失去的效率。

致谢

非常感谢审稿人的评论，也感谢我们的牧羊人莎朗·戈德堡（Sharon Goldberg）的超越。这项研究部分由 NSF 资助，编号为 CNS-1345305，由美国国防部，空军科学研究所，国防科学与工程研究生（NDSEG）研究金 32 CFR 168a，以及欧盟根据 FP7 资助协议 n 资助。。318627（集成项目“ mPlane”）

9 M参考

[1] http://mctls.org。

[2] 资讯主页-Android 开发人员。https：// developer。安卓。com / about / dashboards / index.html。访问时间：2014 年 12 月。

[3] 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，2015 年 5 月。USENIX 协会。

[4] D.海狸。HTTP2 表达兴趣。http：//lists.w3。org / Archives / Public / ietf- http- wg / 2012JulSep / 0251.html，2012 年 7 月。

[5] A. Bittau，D。Boneh，M。Hamburg 等。tcp 流的加密保护（tcpcrypt），2014 年 2 月。
[6] A. Bittau，M。Hamburg，M。Handley，D。Mazieres 和 D. Boneh。普遍存在的传输级加密的情况。USENIX Security'10，美国加利福尼亚州伯克利，2010 年。USENIX 协会。
[7] I.布朗。主动网络中的端到端安全性。在*伦敦大学学院博士论文，* 2001。
[8] D. Cameron。互联网和色情制品：总理呼吁采取行动 https://www.gov.uk/government/speeches/the- internet-and-pornography-prime-部长-呼吁行动。访问时间：2015 年 1 月。
[9] S. Cheshire 和 M. Krochmal。基于 DNS 的服务发现。RFC 6763（建议标准），2013 年 2 月。
[10] S. Cheshire 和 M. Krochmal。组播 DNS。RFC 6762（建议标准），2013 年 2 月。
[11] 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。
[12] FR Dogar，P。Steenkiste 和 K. Papagiannaki。Catnap：利用高带宽无线接口为移动设备节省能源。MobiSys '10，美国纽约，纽约，2010 年。ACM。
[13] J. Erman，A。Gerber，M。Hajiaghayi 等。要缓存还是不缓存：3g 的情况。*Internet Computing，IEEE，* 15（2）：27-34，2011 年 3 月。
[14] M. Hoque，M。Siekkinen 和 JK Nurminen。基于代理的流量整形技术在移动音频流传输中的能效。CCNC，第 891-895 页。IEEE，2011 年。
[15] V. Jacobson，DK Smetters，JD Thornton 等。网络命名的内容。CoNEXT '09，第 1 至 12 页，美国纽约，纽约，2009 年。ACM。
[16] S. Kasera，S。Mizikovsky，GS Sundaram 和 TYC Woo。为无线移动用户安全地启用基于中介的服务和性能增强。在*无线安全研讨会上，2003 年，* 2003 年。
[17] A. Langley，N。Modadugu 和 W.-T。昌 超频 SSL。https：//www.imperialviolet。org / 2010/06/25 / overclocking-ssl.html，2010 年 6 月。
[18] 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。
[19] V. Liu，S。Han，A。Krishnamurthy 和 T.安德森。基于最小特权原则的 Internet 体系结构。技术报告 UW-CSE-12-09-05，华盛顿大学，2012 年 9 月。
[20] S. Loreto，J。Mattsson，R。Skog 等。HTTP / 2.0 中的显式受信任代理。互联网草案草稿-loreto-httpbis-trusted-proxy20-01，IETF 秘书处，2014 年 2 月。
[21] 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。
[22] A. Menezes 和 B. Ustaoglu。关于在 diffie-hellman 密钥协商协议中重用临时密钥。*国际应用密码学杂志，* 2（2）：154-158，2010。
[23] JC Mogul 和 G. Minshall。重新考虑 tcp nagle 算法。*SIGCOMM CCR*，2001 年 1 月。
[24] C. Muthukrishnan，V。Paxson，M。Allman 和 A. Akella。使用强类型网络来设计 tussle。Hotnets-IX，第 9：1-9：6 页，美国纽约，纽约，2010 年。ACM。
[25] D. Naor，A。Shenhav 和 A. Wool。旨在无需公钥操作即可保护不受信任的存储。StorageSS '05，第 51-56 页，美国纽约，纽约，2005 年。ACM。
[26] 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。
[27] C. Nikolouzakis。在发生爱德华 斯诺登（nward）泄漏事件后，加密流量增长了 40％。http ： // www 。 sinefa.com/blog/encrypted-traffic-gross-edward-snowden-nsa-leak。访问时间：2015 年 1 月。
[28] 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。
[29] 钱谦，S。森和 O. Spatscheck。表征移动 Web 浏览的资源使用情况。MobiSys '14，第 218-231 页，美国纽约，纽约，2014 年。ACM。
[30] E. Rescorla。TLS 上的 HTTP。RFC 2818（信息性），2000 年 5 月。
[31] JH Saltzer，DP Reed 和 DD Clark。系统设计中的端到端参数。*ACM Trans。计算 Syst。*，1984 年 11 月 2（4）：277-288。
[32] JH Saltzer 和 MD Schroeder。计算机系统中信息的保护。*IEEE 会议论文集，* 63（9）：1278-1308，1975。
[33] 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。
[34] 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。
[35] N. Weaver，C。Kreibich，M。Dam 和 V. Paxson。这是网络代理。在“*被动和主动测量”中，*第 183-192 页。施普林格，2014 年。
[36] 维斯涅夫斯基（C. Wisniewski）。路径和时髦 iPhone 应用程序泄漏敏感数据，而无需通知。https://nakedsecurity.sophos.com/2012/02/08/苹果-移动应用-路径和-行家和-泄漏敏感数据-wi 访问时间：2015 年 5 月。
[37] S. Woo，E。Jeong，S。Park 等。现代蜂窝回程网络中缓存策略的比较。MobiSys '13，第 319-332 页，美国纽约，纽约，2013 年。ACM。
[38] X. Xu，Y。Jiang，T。Flach 等人。研究蜂窝网络中的透明 Web 代理。PAM '15。