InviCloak: An End-to-End Approach to Privacy and Performance 翻译

在当今的网络生态系统中，使用内容交付网络（CDN）的网站与CDN共享其传输层安全（TLS）私钥或会话密钥。在本文中，我们介绍了InviCloak的设计和实现，该系统可以在不改变TLS或升级CDN的情况下保护用户和网站私人通信的保密性和完整性。InviCloak建立了一个轻量级但安全实用的密钥分发机制，使用现有的DNS基础设施来分发与网站域名相关的新公钥。网络客户端和网站可以使用新的密钥对在TLS内建立一个加密通道。InviCloak适应了当前的网络生态系统。网站可以在没有客户参与的情况下单方面部署InviCloak，以防止CDN内部的被动攻击者窃听其通信。如果客户端也安装了InviCloak的浏览器扩展，那么在CDN内的主动攻击者存在的情况下，客户端和网站可以实现端到端的保密和不受干扰的通信。我们的评估显示，InviCloak将现实网页的中位页面加载时间（PLTs）从2.0s增加到2.1s，这比最先进的基于TEE的解决方案的中位PLTs（2.8s）小。

内容交付网络（CDN）在网络生态系统中发挥着重要作用。它们不仅加快了网络内容的传播速度，而且还保护网站免受各种攻击。例如，Akamai[68]、Cloudflare[18]和CloudFront[16]等CDN提供分布式拒绝服务（DDoS）攻击缓解和恶意内容清除服务[22, 48]。
不幸的是，随着越来越多的网站迁移到100%的HTTPS[8]，第三方CDN引入了不理想的安全后果。一个网站与CDN共享其传输层安全（TLS）证书的私钥，以充分利用CDN的性能和安全优势，这是一种常见的做法。Cangialosi等人在2016年进行的一项测量显示，76.5%的组织与第三方托管提供商共享其私钥，对于流行的网站，他们主要与CDN提供商共享其密钥[34]。

CCS '22，2022年11月7-11日，美国加利福尼亚州洛杉矶市

2022. acm ISBN 978-1-4503-9450-5/22/11…$15.00 https://doi.org/10.1145/3548606.3559336

这种密钥共享的做法打破了TLS/HTTPS提供的端到端安全保障，带来了潜在的安全风险。作为一个组织，CDN可能遭受内部攻击或可利用的安全漏洞。由于CDN通常为许多网站提供服务，一个被破坏的CDN可能会泄露许多网络服务的私人凭证，成为一个中心故障点。作为一个例子，网络缓存欺骗攻击[46, 64]利用了CDN的配置漏洞。研究人员表明，攻击者可以欺骗CDN进行缓存并暴露个人身份信息（PII），如姓名和电话号码[64]。
目前这种做法的安全风险已经促使了多种解决方案。每个解决方案都在可采用性、安全性和性能之间做出了权衡。Cloudflare的无密钥SSL[80]和证书委托[62]不会将网站的TLS私钥暴露给CDN，但仍然允许CDN拥有TLS会话密钥--这使得CDN内部的攻击者可以继续观察和修改TLS会话中的内容。其他的解决方案，如mcTLS和maTLS[33, 59, 67]修改了TLS协议，在TLS握手中包括中间盒。尽管在技术上是合理的，但这些解决方案需要客户、中间盒和网站的协调努力来升级他们的TLS实现。
另外，一个网站可以为两个不同的域名（如site.com和site-cdn.com）获得两个TLS证书：一个用于自己托管的隐私敏感内容，另一个用于CDN托管的内容，类似于[47]。网站与它的CDN共享CDN相关的TLS证书的私钥，并保持另一个私钥。我们把这个建议称为双域解决方案。双域解决方案的一个主要缺点是，当网站使用CDN分发其基本的HTML文件时，它不能防止主动攻击，而这是用户访问网页时下载的第一个文件。出于性能方面的考虑，

网站希望通过CDN86发布其基础HTML文件，但CDN中的主动攻击者可以修改该文件并劫持随后的私有TLS会话。此外，双域解决方案可以防止CDN缓存任何私有内容，即使是加密形式的内容，因为网站会通过单独的TLS会话发送。这种设计减少了CDN的性能优势。
在不同的方向上，研究人员提出使用可信执行环境（TEEs）[29, 38]来防止不信任的CDN代码访问共享的TLS私钥[27, 85]或TLS会话密钥[51]。这些解决方案提供了理想的安全保证，而不需要在用户端做任何部署。然而，它们对CDN来说面临着部署和性能的挑战。由于飞地内昂贵的系统调用，目前的TEE硬件可能会使CDN边缘服务器的吞吐量降低2到4倍[51]。此外，这些解决方案要求CDN升级其基础设施的硬件。根据我们的分析，升级CDN的基础设施以支持基于TEE的解决方案的财务成本可能超过1亿美元（§6.6）。此外，目前基于TEE的解决方案的前景还不清楚，因为英特尔已经宣布，英特尔TEE，即软件防护扩展（SGX）[38]在第12代英特尔CPU[10]中被废弃。

现有的每一个建议都有自己的安全、性能和部署成本的权衡。在这项工作中，我们旨在探索一种具有不同成本和效益权衡的解决方案，供市场选择。该解决方案，InviCloak，采取了一种端到端的方法。它适应了当前的密钥共享做法，并将内容服务授权与保密性分开。一个网站使用共享的TLS密钥来授权CDN提供其非隐私敏感的内容。然后，它使用一对新的私人/公共密钥，不与CDN共享，以保护隐私敏感的内容。InviCloak可以防止主动攻击，并且不增加发送到源服务器的流量。
InviCloak面临的一个主要设计挑战是如何平衡其带来的安全利益和其部署及性能成本。为了解决这一挑战，我们设计InviCloak使用现有的DNSSEC[31]和DNS-over-HTTPS[52]（DoH）基础设施来分发网站的新公钥，从而避免了网站为新域名获取新的TLS证书。为了便于部署，InviCloak在网络客户端和网站源服务器之间的现有TLS会话中嵌入了一条端对端加密隧道，以传输私人数据，如用户的登录密码。这种设计避免了修改TLS、CDN、网络服务器或网络资源的需要。
因此，InviCloak与现有的建议相比有几个部署优势。它不改变底层TLS协议，对CDN完全透明。CDN不需要升级他们的基础设施。此外，它不修改现有的网络服务器实现，网络开发人员不需要改变现有的网络资源或管理新的域名和证书。网站可以单方面将InviCloak部署为一个JavaScript库，以击败被动窃听者，而无需任何用户端操作。如果用户安装了InviCloak的浏览器扩展，她可以检测并防止主动攻击者篡改或窃听她与网站的私人通信。
由于性能对网络应用至关重要，InviCloak的加密中加密设计由于其开销很大，很容易被人否定。我们实施了一个原型，对InviCloak的操作进行了微观基准测试，并测量了它对网页加载时间（PLTs）的影响。在我们的测试平台上，InviCloak为中位PLTs引入了不到100ms的延迟。这一开销比最先进的基于TEE的解决方案[51]低了约700毫秒。如果开销成为一个问题，我们可以修改浏览器的实现，以消除内层加密，代价是增加部署障碍。
这项工作有以下主要贡献。

- InviCloak的设计，在保持CDN作为网站的DDoS屏蔽功能的同时，保护用户的私人数据不被破坏。
- 一个可在当前网络生态系统中立即部署的原型实现。我们的评估表明，它为网络内容分发引入了可接受的开销。
- 我们分析了InviCloak的部署工作，并将其与相关工作进行比较。我们表明，InviCloak的部署不需要修改CDN、TLS、操作系统或网络服务器。它既不需要新的域名，也不需要新的TLS证书。
  伦理问题。这项工作没有引起任何道德问题。

我们对Alexa排名前100的网站进行了测量研究[15]，以了解使用第三方CDN的网站如何保护其隐私敏感数据，如用户登录密码。
我们参考了现有研究中的方法[34, 54, 57, 61]来发现网站的CDN使用情况，并确定网站所属的组织。如果一个网站的组织机构与CDN提供商不一样，我们就得出结论，该网站使用了第三方CDN。结果显示，百强网站中有67家使用了第三方CDN，其中54家使用CDN来传输主页的基本HTML文件。对于所有这54个网站，即使它们采用了第1节所述的双域解决方案，CDN内的主动攻击者也可以修改这些网页。
我们进一步检查了这些网站的登录程序，以调查其敏感数据的传输策略。在67个支持CDN的网站中，有7个网站不使用密码登录，有25个网站在登录程序中暴露了其登录服务器的IP地址，而有35个网站通过CDN发送用户的密码。这一结果表明，并非所有的网站都信任他们所使用的CDN，这一点从25家绕过CDN进行用户登录的网站可以看出。然而，暴露网站登录服务器的IP地址使网站容易受到DDoS攻击。对于那些将用户密码暴露给CDN的网站，他们有可能将用户的敏感和隐私数据泄露给CDN。
上述观察促使我们设计一个概念上简单且安全的解决方案。

2.2 CDN服务模型

为了清楚起见，我们描述了一个网站和其CDN服务提供商之间的服务模型。我们使用这个模型来设计InviCloak。具体来说，我们将网站提供的内容分为CDN可见的内容和私人内容。我们认为私人内容是属于一个网站的注册用户的内容，只应由经过认证的用户访问。例如，一个用户必须登录以检查她的银行账户余额。如果一些内容被CDN缓存，或者对用户来说不是私有的，我们认为它是CDN可见的。例如，由CDN服务器缓存的静态内容或付费墙后面的内容，如订阅服务中的视频，是CDN可见的。
由于隐私问题，网站不会与CDN分享其私人数据库。它使用CDN来缓存和提供其CDN可见的内容，但用户将直接向/从网站的源服务器发送/检索私人内容[26]。此外，我们假设网站最好在CDN上缓存一些私人内容，如私人用户的照片，以提高性能。因此，InviCloak的设计也支持这样的用例，同时对CDN的私人内容保持秘密。我们注意到，InviCloak并没有改变现有的CDN服务模式，因此它不会增加发送到网站的流量。
发送到网站源服务器的流量。不提供任何私人内容的网站不在本工作范围内，不需要部署InviCloak。
在本文中，我们把网站托管的服务器称为 "原点服务器"，它是网站所有内容的初始来源。术语 "用户 "总是表示一个网站的用户，而不是CDN。当我们使用术语 "客户端 "时，它指的是用户使用的终端（浏览器或计算机）。最后，"攻击者 "是指任何恶意的实体。

我们假设，并非所有的CDN客户都完全信任他们的CDN。这些客户希望从CDN的缓存和DDoS保护服务中获益，而不把私人内容暴露给CDN。我们假设一个被攻击的CDN可能会发起两种类型的攻击，具有不同的风险因素。
被动攻击。一个CDN可能有一个软件或配置漏洞，导致非故意的信息泄漏，如网络缓存欺骗攻击[46, 64]。或者CDN内部好奇的窃听者可能会在CDN的边缘服务器解密其收到的TLS会话数据后记录明文数据。我们将这种类型的漏洞建模为被动攻击，即窃听网络客户端和网站起源服务器之间的私人通信。
主动攻击。一个CDN可能有一个受到威胁的内部人员，可以访问其客户网站的TLS私钥，或者CDN内部有一个软件漏洞，允许攻击者注入恶意代码。我们将这种威胁建模为主动攻击，可以窃听、篡改或泄露它收到的任何信息。

2.4 信任假设

我们做了以下信任假设。

DNS和CDN之间没有串通的攻击者。我们假设对手没有强大到可以同时渗透到网站的CDN提供商和DNS提供商。换句话说，即使网站的DNS提供商和CDN提供商中存在攻击者，我们假设这两个对手是相互独立的，不能相互勾结。

InviCloak无法击败同时入侵网站DNS和CDN提供商的强大攻击者，因为其设计使用DNS来分发新的公钥。因此，如果一个网站服务担心DNS和CDN攻击者的串通，它可以选择将其CDN提供商与DNS提供商分开。在实践中，我们调查的所有主要CDN，包括Akamai、CloudFront、谷歌云、Azure和Fastly，都允许用户使用单独的DNS提供商。Cloudflare默认要求其客户使用其DNS服务来托管客户的域名，但它也允许客户切换到其他DNS提供商[36]。

我们对Alexa前10K网站进行了测量研究，以验证这一假设。我们首先使用现有的方法[34, 54, 57, 61]来发现网站的第三方CDN使用情况。此外，我们使用dig[37]来获取网站中每个支持CDN的域名的名称服务器，并从WHOIS服务[40]中获取域名注册商。一个网站可能拥有多个域名。如果一个域名的名称服务器不属于该网站的CDN提供商，我们认为该网站将DNS提供商与CDN提供商分开。此外，如果域名注册商不是CDN提供商，我们认为该网站将域名注册商与CDN提供商分开。

在前10K网站中，有4867个网站使用了第三方CDN提供商。我们发现，4867个网站中的2765个（57%）已经将其DNS提供商与CDN提供商分开。此外，在那些没有分离的2102个（43%）网站中，有1668个网站将其域名注册商与CDN分离。这1668个网站可以通过将他们的DNS提供商转移到他们的域名注册商，将他们的DNS提供商与他们的CDN分离，而不需要经济成本。

引导性安全。我们假设用户或网站可以安全地获得他们的操作系统、浏览器和InviCloak相关的模块或扩展，而不涉及被破坏的CDN。

可信计算基地（TCB）。我们认为浏览器、网站、操作系统以及客户端和源服务器的硬件的实现是我们的TCB。具体来说，我们信任部署InviCloak的网站的实现，包括其网页（HTML、JavaScript、CSS）和它使用的所有其他系统组件，如数据库和服务后端。诚然，攻击者可能会利用网站或客户端浏览器的漏洞发起攻击，如SQL注入[50]和跨站脚本（XSS）[87]。我们认为防止这种攻击不在本工作的范围之内。

密码学的坚硬程度。我们假设攻击者无法克服密码学算法的硬度。例如，他们不能在没有加密密钥的情况下解密密码文本，他们不能在没有私人密钥的情况下伪造数字签名。

我们的设计目标是多方面的，包括隐私和保密性、可用性、低部署成本和性能。这些目标使我们的工作有别于相关工作。

隐私和保密性。我们最重要的目标是隐私和保密性。我们的目标是保护用户和网站之间传输的私人内容不被泄露给第三方CDN。我们的设计应能抵御来自受损的CDN的主动和被动攻击。

可用性。InviCloak的一个重要设计目标是符合当前网络的可用性模式。我们要求网站不需要获得一个新的域名，也不需要与它的CDN供应商协商新的服务类型。此外，网站不应改变显示给用户的域名，否则可能会造成品牌名称的混淆和钓鱼网站的漏洞。

低部署成本。我们假设，如果一个安全功能在财务上是昂贵的，并且需要多个利益相关者的协调升级，那么在实践中就很难部署该功能。因此，我们的目标是实现部署InviCloak的低财务成本。除了财务成本，我们的目标是为网络开发人员和用户提供最低的操作成本。例如，网络开发者不需要修改他们的应用逻辑或现有的网络资源。网站不需要撤销它与CDN共享的证书，也不需要申请新的证书，因为商业证书的撤销和申请可能很耗时[62]。我们的目标还有一个端到端的设计，网站和用户可以在没有CDN支持的情况下进行部署。

图1：该图显示了InviCloak在主动攻击者情况下的高级设计。它有三个组成部分：一个客户端代理，一个服务器代理，和一个完整性验证器。客户端代理和服务器代理对客户端和网站源服务器之间的通信进行加密和解密，而完整性验证器对CDN返回的HTML和JavaScript文件的完整性进行验证。客户端代理是一个服务工作器，它与普通的JavaScript代码不同（§ 3.3）。在被动攻击者的情况下，完整性验证器是不必要的。
性能。我们的解决方案应该保留CDN的性能优势，如低延迟和高吞吐量，以及其安全优势，如DDoS缓解。

我们面临的主要设计挑战是如何在不牺牲可用性、可部署性或性能的情况下实现InviCloak的安全目标。为了应对这一挑战，我们做出了一些设计决定。
首先，我们使用基于DNS的命名实体认证（DNS-based Authentication of Named Entities
(DANE) [42]来分发网站的新公钥，以简化密钥管理。其次，我们将InviCloak的主要客户端组件设计成一个JavaScript库，这样网站就可以通过其登陆页面分发，而无需让用户安装。最后，我们使用反向服务器代理来提供InviCloak的服务器端功能，因此不需要修改网络服务器。
图1描述了InviCloak的整体架构、其关键组件和它们的位置。InviCloak引入了三个新的组件：一个在网络浏览器中运行的客户端代理，一个作为浏览器扩展引入的完整性验证器，以及一个在网站源服务器前作为反向代理运行的服务器代理。完整性验证器适用于所有支持InviCloak的网站，而客户端代理是一个JavaScript库，包括每个网站的特定配置。一个网站在其登陆页面中嵌入客户端代理，而完整性验证器将通过浏览者认可的机制（如扩展市场）分发。我们注意到，在被动攻击者的情况下，完整性验证器是不必要的。InviCloak使用DNS基础设施来分发网站的公钥，以便在浏览器和网站的源服务器之间建立一个加密通道。它的设计对CDN来说是完全透明的。
我们用一个例子来说明它是如何在高层次上工作的。在这个例子中，一个叫Alice的用户访问她的银行bank.com。
我们假设她是一个有安全意识的用户，并在她的网络浏览器中安装了InviCloak的完整性验证器。首先，Alice的浏览器从bank.com的CDN提供商那里获取着陆页
bank.com/login.html。这个页面将自动下载bank.com的客户端代理代码
（JavaScript）。该下载将触发完整性验证器向启用DNSSEC的解析器发送DNS-over-HTTPS查询，以获得bank.com的公钥。该公钥将被缓存在扩展的存储中，以便重复使用。完整性验证器验证客户端代理并将其安装在浏览器中。
在Alice填写了她的银行账户的用户名和密码后，她点击了提交按钮。这个动作触发了客户端代理对请求进行加密，因为提交的URL被列为包含私人信息，需要在客户端代理的配置文件中进行加密。由于客户端代理和完整性验证器作为独立的进程运行，客户端代理启动自己的DNS查询以获得bank.com的公钥。它将公钥缓存在浏览器的缓存存储中，以便重复使用。然后，客户端代理调用一个与服务器代理的密钥交换过程，并生成一个对称会话密钥来加密包含Alice的登录凭证的请求。
加密后的请求被转发到bank.com的CDN，然后再转发到其源服务器。由于请求在客户端代理和服务器代理之间进行了端对端加密，CDN内部的任何路径对手都无法窥视并获得Alice的登录凭证。同样地，服务器代理对bank.com对Alice的回应中的私人内容进行加密。
接下来，我们将描述InviCloak的每个组件，以及它们如何相互作用。
3.2 密钥分发和管理

一个网站的新公钥是InviCloak设计中的信任根。InviCloak使用现有的DNS基础设施来分发新的公钥。一个网站按照DANE[53]的规范，将其新的公钥存储在TLSA记录中。由于网站不会经常改变其公钥，它可以为TLSA记录设置一个较长的生存时间（TTL）值，例如，以小时为单位，以减少DNS的查询负荷。
一个网站应该使用DNSSEC[31]来保护TLSA记录的完整性。如果没有DNSSEC，路径上的主动对手可能会篡改分布在TLSA记录中的公钥，损害InviCloak的安全性。根据2019年的一项测量研究[76]，在20个最常用的域名注册商中，有15个支持DNSSEC，所有顶级域名如.com和.org都支持DNSSEC[74]。因此，渴望保护其用户隐私的网站可以找到一个合适的域名注册商或支持DNSSEC的DNS提供商。

尽管越来越多的域名注册商启用了DNSSEC[76]，但客户的默认DNS解析器可能没有启用DNSSEC。因此，在InviCloak中，客户端代理和完整性验证器将在DNS-over-HTTPS（DoH）[52]中向支持DNSSEC的解析器发送公钥查询，如图1所示。目前，所有主流浏览器都可以发起DoH请求，因为它们本质上是HTTPS请求。一个网站可以在其客户端代理中指定一个支持DNSSEC的可信DNS解析器。InviCloak参考了TLS加密客户端你好（ECH），通过DNS分发公钥[73]。InviCloak通过采用DoH来优化ECH，以确保即使在客户端的默认解析器不支持DNSSEC的情况下也能使用支持DNSSEC的解析器。
我们注意到，这种密钥分配机制不会严重增加DNS基础设施的负载，也不会引入DDoS漏洞，因为对公钥的请求是一次性的。InviCloak在浏览器中缓存了一个网站的公钥，在用户清除缓存之前不会请求公钥。
InviCloak在TLSA记录中存储公开密钥而不是证书，有两个原因。首先，证书验证需要一个客户端代理来访问客户端操作系统或浏览器的可信根证书。目前，浏览器中的JavaScript代码不能访问这些证书。支持这种设计需要对浏览器的实现进行修改。其次，一个网站可以独立有效地发布或撤销公开密钥。它不需要为新的证书申请或潜在的撤销与证书颁发机构（CA）沟通，这可能是昂贵和耗时的[62]。我们的设计简化了网站的密钥管理。
有两种可供选择的密钥分发方式。一种是将网站的公钥附在交付给客户的HTML文件上。然而，这种方法只在有被动对手的情况下有效，因为CDN中的主动对手可能会替换HTML文件中的公钥。另一种方法是客户端绕过CDN，直接从源服务器获取公钥，但这将使源服务器受到DDoS攻击。

在InviCloak的设计中，客户端代理是一个在网络浏览器中运行的JavaScript库。希望部署InviCloak的网络服务通过登陆页面向其用户分发该代理。
(例如，login.html或主页）。我们设计的客户端代理将配置文件作为输入，这样每个网站只需要配置文件来部署InviCloak，不需要开发新的JavaScript代码。客户端代理负责：1）查询DNS以下载网络服务的新公钥；2）与源服务器建立加密通道；3）加密用户发送给服务器的私人请求；以及4）解密服务器包含私人内容的响应。
一个网站在客户端代理中定制一个配置文件（configure.js），以包括私人内容的URL和握手API。客户端代理使用握手URL与服务器代理建立一个加密通道。一个网站可以使用正则表达式来指定私人请求的URL集，以减少配置开销和配置文件的大小。附录B中显示了configure.js的一个例子。我们在第6.5节评估了配置工作。
InviCloak的客户端代理与通常的JavaScript代码的主要区别在于，客户端代理是一个服务工作器[78]，它在浏览器的后台运行，并在同一域的不同网页上工作。一个CDN可见的登陆页面通过在HTML中向浏览器注册来安装代理，而浏览器随后会对其进行缓存。当浏览器导航到同一域的私有网页时，客户端代理不需要再次安装。

3.4 服务器代理
我们把服务器代理设计成网络服务器的附加模块，如NGINX[71]。一个网站不需要修改其网络服务器的实现来部署InviCloak。网站在服务器代理的配置文件中列出私人内容的URLs
(nginx.conf)中列出私有内容的URL，与配置客户端代理的方式类似。附录B显示了nginx.conf的一个例子。服务器代理负责：1）与客户端代理建立加密通道；2）解密客户端代理加密的请求；3）加密返回给客户端的私有内容。
3.5 建立加密通道
现在我们描述一下客户端代理如何与服务器代理建立一个加密通道。客户端代理将首先发送一个DoH查询，以获得存储网站新公钥的TLSA记录。然后，它将与服务器代理建立一个秘密会话密钥，用于加密和解密。我们的密钥交换协议是基于TLS 1.3 [72]，因为TLS的安全性已经被研究人员仔细检查过。我们在附录A中描述该协议。
会话密钥的重复使用。我们设计的会话密钥可以在多个请求/回复中重复使用，因此会话密钥的设置在网络会话中是一次性的成本。根据TLS 1.3 [72]，InviCloak将在每次密钥交换后随机化一个32字节的会话ID并存储相应的会话密钥。会话ID被返回给客户端。当客户端代理向服务器代理发送加密的私人内容时，它将包括明文的会话ID。服务器代理通过会话ID检索会话密钥，如果没有找到相应的会话密钥，它将拒绝该请求。
按需与异步的密钥交换。密钥交换过程可以按需进行，也可以异步进行。每种方法都有其优点和缺点。对于按需密钥交换，客户端代理在网络会话中发送第一个私人请求时触发密钥交换。对于异步密钥交换，客户端代理在用户加载分发客户端代理的登陆页面后立即开始密钥交换。例如，密钥交换可能发生在用户访问网站的登录页面时。当用户输入她的登录信息时，会话密钥已经可以用来加密她的密码了。相反，在按需方法中，密钥交换在用户点击密码提交按钮时发生。

按需方法的优点是在用户没有登录的情况下不会浪费网站的资源来进行密钥交换；缺点是在第一次加密请求的加载延迟上增加了一个往返时间（RTT）。相比之下，异步方法缩短了用户首次请求私人内容时的页面加载时间（PLT）。
我们把它作为一个配置选项，让网站在按需或异步密钥交换之间进行选择。我们注意到，即使采用按需方式，密钥设置也是整个InviCloak会话的一次性成本。
3.6 使用加密通道
会话密钥设置完成后，客户端和服务器代理可以使用它来加密和解密浏览器和网站源服务器之间的私人通信。这些操作描述如下。
加密一个HTTP请求。当浏览器发起一个URL在客户端代理的配置文件列表中的请求时，客户端代理会生成一个包括会话ID、序列号和请求内容的消息。会话ID唯一标识它与服务器代理共享的InviCloak会话，而序列号唯一标识该会话中的请求，以防止重放攻击（稍后描述）。客户端代理使用与服务器代理共享的会话密钥对序列号和请求内容进行加密，使用TLS 1.3批准的密码组。然后，它将加密的请求发送到CDN，CDN将把它转发到源服务器。InviCloak不改变TLS，因此底层TLS连接不知道客户端代理添加的额外加密，并将加密请求视为普通的HTTP请求。
我们目前的设计只对HTTP请求的主体进行加密，因为大多数头文件不包含私人内容。Cookie头可能包含重要的认证信息，我们在第3.9节描述了Cookie的加密和管理。目前的设计可以扩展到加密由客户代理的配置文件指定的其他请求头。
客户端会话状态。客户端代理维护的会话状态包括会话ID、最新的未完成序列号，以及它与服务器代理共享的会话密钥。客户端代理将会话状态存储在浏览器的缓存中。因此，一

个InviCloak会话可能在多个网络会话中有效。对于同一服务器的私人请求，客户端代理可以重复使用会话密钥。网络开发者可以为会话配置一个过期时间。
处理加密的HTTP请求。当服务器代理收到一个加密的请求时，它使用会话ID和加密的序列号来检测和防止重放攻击。服务器代理维护一个最近收到的序列号的滑动窗口，任何超出窗口的请求或序列号重复的请求都会被服务器代理丢弃。对于一个有效的请求，服务器代理对其进行解密，剥离由客户代理添加的附加字段，并将解密后的请求转发到网站的源服务器。
InviCloak和TCP的滑动窗口之间的区别是，InviCloak在接受其之前的请求之前接受一个失序的请求，因为浏览器会并行地启动HTTP请求。
加密HTTP响应。当服务器代理从源服务器收到一个私有URL的HTTP响应时，它将把相应的序列号附加到响应体的开头，然后用会话密钥对响应体进行加密。服务器代理还对Set-Cookie头中指定的某些Cookie进行加密，以避免Cookie泄漏，如第3.9节所述。
解密一个HTTP响应。当客户端代理收到服务器代理的加密响应时，它将使用会话密钥对其进行解密。响应将包括原始序列号以防止响应重放攻击。如果客户端代理发现序列号与它发出的序列号相匹配，它就会将响应返回给网络浏览器；否则，它将丢弃该响应。

客户端代理和服务器代理足以击败居住在CDN内的被动对手，因为私人通信将受到InviCloak加密通道的保护。然而，居住在CDN中的主动对手可能会注入恶意代码并劫持加密通道，从而获得对私人内容的访问。
InviCloak使用完整性验证来抵御这样的主动对手。一个网站将使用对应于其DNS TLSA记录中分发的公钥的私钥来签署CDN可见对象，如客户端代理、HTML文件和JavaScript代码。我们把这些对象称为CDN可见的可执行对象，其中也包括客户端代理的代码。其他CDN可见对象，如CSS和图片，都不是可执行对象，修改它们不能破坏InviCloak的保护。网站不需要签署它们来减少计算成本。网站可以在CDN缓存之前使用任何签名工具（如OpenSSL[82]）离线签署CDN可见的可执行对象。
我们在客户端引入一个完整性验证器来验证签名。与客户端代理不同，完整性验证器是对现有浏览器的一个扩展。它是一个独立于网络服务的组件，可以在网络会话之外安全地获得（例如，通过浏览器中的扩展市场），正如我们在第2.4节中假设的那样。
当一个请求被发送出去时，完整性验证器同时通过DoH获取服务器的新公钥（如果没有缓存），它通过TLSA记录的存在确定InviCloak对网站的启用。当浏览器收到来自CDN的响应时，完整性验证器将拦截该响应，并检查响应体是否是CDN可见的可执行对象。如果是的话，它将使用服务器的新公钥验证该对象的签名。如果其中一个网络对象验证失败，该扩展将阻止加载页面，并发送一个弹出窗口提醒用户。验证器还可以提示用户向中央存储库报告该事件。这个存储库可以由完整性验证器的分销商或任何对汇总用户报告感兴趣的第三方维护，以共同检测正在发生的安全威胁。完整性验证器不验证来自服务器代理的加密响应。相反，客户端代理解密并验证此类消息的完整性。
保护网络资源完整性的另一种方法是使用现有的网络技术，即子资源完整性（SRI）[28]或签名的HTTP交换（SXG）[89]。然而，SRI和SXG都将子资源的完整性附加到基础HTML文件的信任上，它们并不能确保这些HTML文件的完整性。此外，它们需要对HTML文件进行大量的修改，而InviCloak的设计是通过尽量减少对现有网络资源的修改来简化部署。
3.8 完整性验证器的部分部署
在InviCloak的设计中，网站可以在没有用户参与的情况下部署服务器代理和客户端代理，以击败被动对手，但用户需要自己安装完整性验证器，以击败主动对手。
然而，部分部署完整性验证器可以阻止主动攻击。这是因为对手无法判断哪个用户安装了

完整性验证器。如果它发起主动攻击，如代码注入攻击，它有可能被那些安装了完整性验证器的用户发现。诚然，对手可以尝试进行有针对性的攻击，例如，为那些没有安装完整性验证器的用户修改客户端代理代码。了解客户端信息的典型方法是使用User-Agent头，但InviCloak可以被配置为在客户端代理中屏蔽该头，以防止此类信息泄露。我们承认有些浏览器目前不支持浏览器扩展，但总的来说，InviCloak为有安全意识的用户和网站提供了一个选择，即通过使用支持浏览器扩展的浏览器来保护他们的敏感数据不受潜在的主动攻击者的影响。

一个网站可以在用户成功认证后向其发出一个cookie。我们把这种cookie称为用户认证cookie。当用户的HTTP请求出现该cookie时，网站可能会返回仅对认证用户可用的私人内容。目前，这种cookie在HTTP请求/响应头中对CDN是可见的。CDN内部的对手可能会拦截这种cookie，并试图使用它来访问用户的私人内容。

InviCloak通过用客户端和服务器代理之间建立的会话密钥对用户的认证cookie进行加密来防止这种泄漏。网站将在其服务器代理的配置文件中指定哪些cookie对CDN是私有的（称为私有cookie）。服务器代理将在HTTP响应的Set-Cookie标头中加密私人cookies，并在HTTP请求的Cookie标头中解密这些cookies。客户端代理不需要对cookie进行解密，因此浏览器将存储加密的cookie，并在需要时将其附加到请求头中。

加密将用户认证cookie与建立会话密钥的InviCloak会话绑定在一起。当一个路径上的对手复制一个加密的用户认证cookie时，它不能冒充用户，因为它没有相应的会话密钥来生成一个有效的私人请求给服务器。它也不能解密一个加密的响应。因此，对抗者无法获得用户的私人内容。

由于InviCloak的会话状态存储在浏览器缓存中，只要用户不清除缓存，并且InviCloak会话没有过期，InviCloak就可以解密现有的密码-cookies。因此，当用户重新访问一个网页时，她不需要重新登录。

我们注意到，网站可能与CDN共享cookies。例如，在用户登录付费墙并获得付费墙后的内容后，网站可能会共享一个付费墙cookie。这种cookie与认证cookie是分开的，被称为签名cookie[12, 25]。签名cookie应配置为CDN可见，InviCloak不会对其进行加密，也不会影响CDN和网站之间现有的cookie共享做法。

4 安全分析

在本节中，我们分析了InviCloak的安全属性。我们的分析表明，InviCloak可以在CDN或DNS提供商被破坏的情况下保持保密性和完整性。

中间人攻击。CDN内部的攻击者在客户端和源服务器之间的通信路径上，可能试图窃听和篡改其传输的信息。然而，InviCloak使客户端和源服务器能够建立一个秘密会话密钥来加密他们的私人内容。服务器用不与CDN共享的私钥签署其密钥交换信息，因此CDN无法破坏会话密钥的设置过程。因此，即使一个路径上的对手可以访问一个网站的TLS私钥，它也不能窃听或篡改客户端和源服务器交换的私人内容。

代码注入攻击。在InviCloak的设计中，客户端代理是一个JavaScript库，所以客户端可以从CDN下载。CDN内部的攻击者可能会试图注入或修改客户端代理中的代码，以阻挠InviCloak的会话密钥设置。InviCloak使用完整性验证器（§3.7）来防止此类攻击。

重放攻击。CDN内部的攻击者可能会重放它在客户端和源服务器之间收到的信息。我们通过在客户端的请求中包括一个会话ID和一个加密的序列号来防止这些攻击（§ 3.6）。服务器代理可以使用请求ID来检测重放的客户端请求。

前向保密性。InviCloak提供前向保密性，因为它使用TLS 1.3[72]中的Diffie-Hellman密钥交换协议来加密私人内容。该协议中的长期秘密是网站的私钥。即使这个私钥或未来的会

话密钥被破坏，攻击者也无法解密过去会话中发送的信息。
冒充用户。与TLS[72]一样，InviCloak的密钥交换协议不对用户进行认证。恶意的对手可能试图冒充用户来访问私人内容，但我们假设网站会使用额外的认证机制，如登录密码来保护私人内容。因此，尽管对手可以像任何客户端一样与原服务器建立会话密钥，但它不能作为用户进行认证。此外，我们在用户认证后使用会话密钥对用户认证cookie进行加密。正如第3.9节所述，这种设计将用户的认证cookie与InviCloak会话密钥绑定。因此，路径上的攻击者不能冒充用户来访问由会话密钥保护的用户的私人内容。
DDoS攻击。InviCloak不会改变目前的CDN服务模式，因此原点服务器可以享受同样的DDoS攻击。

应用层攻击。CDN可以作为Web应用防火墙（WAF），过滤包含应用层攻击有效载荷的客户端请求[48]，包括SQL注入[50]、跨站脚本（XSS）[87]和应用层DDoS攻击请求[41, 88]。通过我们的解决方案，客户端包含私人内容的请求会被加密。尽管CDN可以继续过滤未加密的请求，但它不能再过滤加密的请求。
一个网站可以自己采用WAF来抵御这种攻击。有一个开源的WAF，ModSecurity[21]，可用于两个常用的网络服务器。Apache[17]和NGINX[71]。此外，针对常见攻击的WAF规则是公开的[23]。开发人员可以通过简单的单线配置来维护NGINX和ModSecurity的WAF。此外，由CDN提供的一般防御可能是无效的，因为应用层的攻击，如算法复杂的DDoS攻击[39, 41]可能是针对特定站点的。一个网站需要部署一个特定站点的WAF以获得有效的攻击保护。鉴于InviCloak对隐私的保护，网站可能认为部署现场WAF并在解密后过滤恶意请求是可以接受的权衡。
一个受损的DNS供应商。InviCloak使用DNSSEC和DoH来保证密钥分发的安全。当一个网站的DNS提供商被破坏时，该网站的公钥可能被篡改。在这种情况下，只要对手没有获得网站的TLS私钥或与网站CDN提供商共享的会话密钥，就不能成功地冒充网站完成客户端代理和服务器代理之间的TLS交换或会话密钥交换。因此，对抗者仍然无法获得用户的私人内容。
一个强大的攻击者如果同时破坏了一个网站的DNS提供商和CDN提供商，就会破坏InviCloak的安全性。从技术上讲，通过分发新的TLS证书而不是网站TLSA记录中的公钥，并修改浏览器实现以验证新的证书，是可以战胜这种威胁的。然而，我们认为这种威胁的风险很低，并选择了一种不需要修改浏览器的设计。
5 实施
我们使用JavaScript和Service Worker API（SW）[78]实现了一个原型的客户端代理。同样的实现适用于支持SW的浏览器，包括Firefox、Chrome、Safari和Edge。自2018年以来，SW已经在主流浏览器（Firefox、Chrome、Safari和Edge）中启用[30]。在浏览器过时且不支持SW的情况下，出于安全考虑，服务器代理将拒绝没有SW加密的请求，并返回一个响应以提示用户升级她的浏览器。
我们使用浏览器提供的Web Cryptography API[84]进行加密算法。我们使用密码组AEAD-AES256GCM-SHA384进行对称加密，并使用曲线NIST P256进行Diffie-Hellman密钥交换。这两种算法都被TLS1.3[72]所认可。客户端代理代码的JavaScript行数（不包括配置文件）是~590。
对于完整性验证器，我们将其作为一个Chrome和Firefox的扩展来实现。它需要不到300行的JavaScript代码。一个复杂的问题是，目前Chrome的实现缺乏浏览器扩展API，webRequest.filterResponseData()，以读取响应体[3]，这在Firefox[13]中已经实现。来自Chrome浏览器开发者的讨论和建议表明，该API并没有引入新的漏洞，它的缺失是因为

技术上的复杂性[1, 2]。因此，我们提供了一个Chromium（Chrome的开源版本）的补丁来实现这个功能。
对于服务器代理，我们用C语言将其作为一个流行的网络服务器NGINX的模块来实现。这需要2000行代码。我们发布了我们的代码以促进InviCloak的部署[63]。我们在附录B中对我们的实现和配置文件样本进行了更详细的描述。

我们评估了InviCloak的原型实现的性能，并与相关工作进行了比较。首先，我们通过测量每个InviCloak操作的计算时间来对计算开销进行微观基准测试。其次，我们评估了增加的开销如何影响原点服务器和CDN边缘服务器的吞吐量，以及客户端现实网页的页面加载时间（PLT）。我们将这种开销与基于TEE的解决方案Phoenix[51]进行比较。我们还评估了InviCloak在一个使用Cloudflare的现代网络应用上的开销。除了性能开销，我们还利用网站部署InviCloak所需的配置行来估计在网站上部署InviCloak所需的努力。最后，我们分析了InviCloak的部署成本，并将其与Phoenix进行比较。我们没有直接将InviCloak的性能与基于TLS修改的解决方案[33, 59, 67]和双域解决方案[47]进行比较，因为它们的性能可以通过没有启用InviCloak的基线客户/服务器性能来近似[47, 67]。我们详细描述每个实验和评估结果。
测试平台。我们在实验中设置了一个由三台Dell Precision T3620机器组成的小型测试平台。这三台机器分别作为客户端、CDN和网站的源服务器。每台机器都有一个英特尔酷睿i7-7700的CPU和32GB的内存，并运行Ubuntu 16.04。这三台机器通过以太网相互连接。我们使用工具，Linux流量控制，（tc）[55]来配置机器之间的带宽和RTT值，如果需要的话。然后，服务器机运行NGINX的实现和配置，正如我们在第3.4节中描述的那样。为了模拟CDN的功能，我们在CDN机器上设置了一个NGINX代理或凤凰。在我们的实验中，我们使用Chromium（版本87.0.4280.88）作为浏览器。
6.1 计算开销
实验。我们让客户端机器将我们的客户端代理实现加载到Chromium中。它向原点服务器发送合成网络请求，而服务器则以合成网络响应来回应。为了测量有效载荷大小对计算开销的影响，我们将HTTP请求/响应的有效载荷大小从2KB改变到8MB，因为在这项工作中，90%的网页都小于（8MB）[9]。
我们对客户端代理、服务器代理和完整性验证器进行检测，以记录加密、解密和签名验证操作的计算时间。我们不显示签名生成的开销，因为网站可以离线签署静态文件。该测量包括完整的计算

```
(a) 客户端 (b) 服务器端
```

图2：该图显示了我们的测试平台机器在执行InviCloak的加密操作时所花费的计算时间。该图显示了由加密算法和其他必要的操作（如内存分配和复制）引起的开销。在本实验中，我们没有测量会话密钥设置的开销，因为它是每个会话的一次性成本。然而，我们确实在PLT实验中测量了它（§6.3）。我们将每个实验重复一百次，并使用平均值作为结果。
结果。图2显示了这些实验的结果。加密、解密和签名的开销随着有效载荷的大小几乎呈线性增长。在所有情况下，服务器的计算开销比客户端的计算开销要小一个数量级。造成这种性能差距的原因是，我们用C语言实现了服务器代理，而客户端代理是用JavaScript实现的，以方便部署。对于小于8MB的有效载荷，加密和解密在服务器上需要大约3毫秒。在客户端，加密需要不到50毫秒，而解密需要大约60毫秒。在客户端，解密比加密慢，因

为在我们的实现中，解密代码更复杂，支持大的响应的精简解密。客户端验证一个8MB文件的签名大约需要60毫秒。我们使用我们开发的用于生成签名的简单工具进行测试，在服务器上生成一个8MB的文件需要16ms。

我们认为InviCloak的计算开销在服务器端和客户端都是可以接受的。此外，由于网页大小的中位数约为2MB[9]，远小于8MB，我们预计InviCloak的开销在实践中会很低。我们在第6.3节和第6.4节中通过评估PLTs来验证这一论点。

6.2 CDN和服务器吞吐量

实验。我们运行两组实验，分别测试CDN和服务器的吞吐量。我们使用GoHTTPBench[6]，它是ApacheBench[14]的多线程版本，从客户端机器向CDN机器或服务器机器总共发送50K HTTP请求。我们将来自CDN或服务器的HTTP响应的大小从1KB到8MB不等。GoHttpBench将以64个并发线程发送请求，以模拟64个并发客户端，这使得服务器机或CDN机的CPU使用率超过95%。

对于CDN的性能评估，我们测量Phoenix和一个基线（NGINX代理）的吞吐量。由于InviCloak不需要对CDN进行任何修改，我们使用基线性能来代表InviCloak部署时的CDN性能。对于起源服务器的性能，我们测量了

表1：抓取的网页中的对象大小之和。登录页面不包含任何私人内容。括号中的数字表示URL的数量。JS "指的是JavaScript。

	中位数	平均值
登录页面	所有	
	html	2476 kb (47)
		85 KB (4)
		3895 KB (63)
		225 KB (4)
	js	1794 kb (17)
		2815 kb (25)
私人页面	所有	
	私有的	
	html	5361 KB (74)
		210 KB (7)
		212 KB (4)
		6896 KB (93)
		505 KB (17)
		344 KB (6)
	js	3551 KB (30)
		5378 KB (34)

带InviCloak的NGINX代理和基线（不带InviCloak的NGINX代理）的吞吐量。由于Phoenix不需要对起源服务器进行任何修改，我们使用基线性能来代表部署Phoenix时起源服务器的性能。

结果。图3a显示了CDN性能实验的结果。我们可以看到，Phoenix给CDN服务器引入了相当大的开销。在每个实验中，它的吞吐量只有基线的三分之一左右，这与Phoenix原始论文[51]中的结果一致。图3b显示了起源服务器性能实验的结果。InviCloak在有效载荷大小为1KB到8MB的情况下，将服务器的吞吐量降低了不到5%。总之，由于设计不同，InviCloak和Phoenix拥有不同的优势。Phoenix保留了源服务器的吞吐量，但大大降低了CDN的吞吐量。InviCloak不会给CDN服务器带来开销，但会略微减慢源服务器的速度。

我们使用现实的网页来估计InviCloak如何影响用户的可感知的性能。我们从Alexa的前100名网站中抓取了能够注册和登录的前50名网站的两组网页。对于第一组，我们抓取了每个网站的登录页面。有些网站将登录表格嵌入主页内，而不是单独的登录页面。在这种情况下，我们抓取他们的主页作为登录页面。对于第二组，我们手动登录每个网站并抓取一个包含私人信息的网页，如个人资料、访问历史或最喜欢的项目列表。我们将这些网页镜像到我们的测试平台上。

对于登录页面，我们认为每个网页对象都是CDN可见的，因为它不需要用户登录。对于私人网页，我们使用HTTP Cache-Control头来识别不可缓存的网页对象。在我们的评估中，我们将所有不可缓存的对象视为私有对象，并在配置文件中包括其URL。InviCloak将在实验中对此类对象进行加密。对于每一个网页，我们对嵌入的网页对象的大小进行汇总。在抓取的网页中，我们计算出和值的中位数和平均值，如表1所示。登录页面和私人页面的平均大小分别为3895 KB和6896 KB。平均而言，一个私人网页包含505 KB的私人对象。

0 0

1 2 8 32 128 512 2048 8192 1 2 8 32 128 512 2048 8192

有效载荷大小(KB) 有效载荷大小(KB)

(a) CDN的吞吐量 (b) 服务器的吞吐量

图3：该图显示InviCloak和Phoenix如何影响服务器和CDN的吞吐量。吞吐量的单位是HTTP
响应/秒的单位。

```
0 0
0 2 4 6 8 0 0.5 1 1.5 2
PLT (s) PLT (s)
(a) Alexa网站 (b) DeathStarBench
```

图4：（a）50个Alexa网站的PLT的CDF（§6.3）。(b)
(b) DeathStarBench的PLTs来自六个地理分布的客户（§6.4）。每个图的X轴的单位是秒。

为了测量每个测试网页的PLT，我们在客户端机器的浏览器上设置仪器，将所有请求发送到CDN机器上。CDN机器用它缓存的CDN可见的网页对象进行回复，并将私人请求转发给服务器机器。为了提出一个更现实的评估，我们将机器之间的带宽限制为100Mbps。根据[32]，客户与Cloudflare的边缘服务器之间的平均RTT为36毫秒。因此，我们使用这个值作为客户端和CDN机器之间的RTT。我们将CDN和服务器之间的RTT设定为100毫秒，这大约是位于美国东海岸和西海岸的两台机器之间测量的RTT。我们在Chromium中对每个网页加载五次，并计算出每个网页的平均PLT。

我们通过测量抓取的登录页面作为InviCloak的登陆页面时的PLT增量来评估InviCloak的启动开销。启动过程包括验证CDN可见的可执行对象、安装客户端代理，以及第3.5节中讨论的异步密钥交换。结果显示，InviCloak的启动使登录页面的
所有网站的PLT小于8%。具体来说，47个网站的开销
47个网站的开销都低于5%。我们注意到，这样的开销是一次性的，因为CDN可见的对象和InviCloak的会话状态会在用户第一次访问登陆页面后被浏览器缓存。

至于抓取的私人网页，我们在多种设置下进行实验，以显示各种计算开销对PLT的影响。在第一个设置中，我们测量了InviCloak和Phoenix都被禁用时的PLTs，作为基线（Baseline）。在第二种设置中，客户机只运行客户代理，不运行完整性验证器

（Enc）。对于第三种，客户机同时运行客户代理和完整性验证器（Enc&Sign）。在InviCloak的这两种设置中，客户端代理已经在登陆页面中与服务器代理设置了会话密钥。为了将InviCloak与现有的解决方案进行比较，我们测量了启用Phoenix（凤凰）时的PLTs。上述所有的设置都使用了带有暖缓存的CDN。我们还测量了冷缓存时的PLTs，即CDN缓存被禁用（CDN-off）。

图4a显示了50个网站在每种设置下的结果。在Baseline、Enc和Enc&Sign设置下的PLTs具有相似的分布，它们与CDN-off曲线之间存在很大的差距。这一结果表明，InviCloak在很大程度上保留了CDN的性能优势。此外，InviCloak的开销比Phoenix要低。当Phoenix开启时，超过32%的PLT超过4.0s，但在Baseline、Enc和Enc&Sign设置下，超过92%的PLT小于4.0s。使用Phoenix的PLT中值为2.8s，而Baseline、Enc和Enc&Sign的PLT中值分别为2.0s、2.1s和2.1s。

在实验室环境下用抓取的网页评估InviCloak可能无法完全反映InviCloak在现实中的性能。为了解决这一局限性，我们运行了一个现代网络应用程序DeathStarBench[44]，并通过互联网上地理分布的客户端来评估InviCloak。我们使用Cloudflare，最大的CDN供应商之一作为CDN。在这次评估中，我们不能与Phoenix进行比较，因为TEE目前还没有在任何CDN上提供。

DeathStarBench是作为研究现代云服务性能的一个基准而开发的[44]。它采用了流行的微服务架构，被一些学术和工业机构使用[44]。我们在评估中使用了DeathStarBench提供的社交网络，它提供的功能与Twitter类似。除了现实的网络应用外，我们还使用现实的网络内容进行评估。我们从Facebook导入了一个有962个节点和18,812条边的小型社交图到社交网络[75]。我们还通过Twitter API[24]从随机选择的962个Twitter用户中抓取了612,455条推文，并将这些推文导入数据库。

我们在弗吉尼亚州AWS的一台虚拟机上部署了带有抓取工作负载的DeathStarBench。我们使用六个地理上分布的AWS虚拟机作为客户端。这些客户端分布在六个AWS地区，包括加利福尼亚、蒙特利尔、巴黎、新加坡、圣保罗和巴林。本实验中使用的每台虚拟机都有4个vCPU和8GB的内存，运行Ubuntu 20.04。我们使用Cloudflare作为CDN来缓存客户端附近的静态资源。

在我们的实验中，我们从数据库中随机选择了10个用户，并让每个客户端以每个用户的身份登录并访问该用户的三个私人页面。这三个页面包括一个显示用户的追随者的最多100条最新推文的页面，一个显示用户的最多100条最新推文的页面，以及一个显示用户的所有追随者和粉丝的页面。为了显示一个保守的结果，我们把所有的推文、关注者名单和追随者名单视为私人数据。它们是加密的，并且没有缓存在Cloudflare上。其他资源是公开的，在Cloudflare上缓存。为了保守起见，我们不仅签署了HTML和JavaScript文件，还签署了CSS文件。正如第6.3节中所定义的，我们在四种设置下运行实验。基线、Enc、Enc&Sign以及CDN-off。总的来说，六个客户端中的每一个都会访问所有10个用户的三个页面中的每一个5次。我们计算出每个页面的平均PLT。因此，我们在每种设置下有6×3×10=180个PLT值。

如图4b所示，我们可以在曲线中找到三个明确的阶段。$S1$,$S2$,$S3$，这源于不同地点之间PLT的差异。加州、蒙特利尔和巴黎的低网络延迟导致了$S1$中显示的小PLT。由于网络延迟较大，$S2$包括圣保罗和巴林的PLT，而$S3$则显示新加坡的PLT。除了这些阶段，CDN-off的曲线和其他曲线之间存在很大的差距，表明InviCloak保留了CDN的性能优势。此外，Baseline、Enc和Enc&Sign有相似的分布，表明InviCloak的开销很低。具体来说，在Invi-Cloak的任何一个设置中，InviCloak都具有较高

的性能。
Cloak，InviCloak对所有180个PLT值中的170个引入了不到5%的开销。因此，我们得出结论，InviCloak的性能开销很低。

在本节中，我们估计了InviCloak的部署工作。
我们使用代码行（LoC）来衡量抓取的网站和DeathStarBench的配置开销。对于50个网站，我们将每个网站的私有URL添加到服务器代理配置文件（nginx.conf）中，并编辑客户端代理配置文件（configure.js）以包括私有URL。我们在每个网站的登录页面中插入服务工注册的代码。nginx.conf的LoC从4到62不等，平均为13.7，中值为8.0。对于configure.conf，LoC的范围是10到68，平均为19.7，中值为14.0。对于所有的网站，注册的LoC是4。对于DeathStarBench，我们也使用4个LoC来在登录页面注册一个服务工作器。此外，我们在现有的nginx.conf中添加了一个带有9个LoC的configure.js，并插入了14个LoC。
InviCloak的部署要求网站开发者明确地将私人URL和公共URL分开。我们在两种情况下讨论这个过程以及网站开发者可能出现的配置错误的影响。
首先，如果网站开发人员已经将URLs分类为私有和公共目录（如/private和/public），他们可以在配置文件中通过通配符或正则表达式列出私有URLs。这种情况下的配置工作很低。如果网站开发人员错误地对URL进行分类，这些错误将被传播到InviCloak的配置中。
第二，如果URL没有分类，我们提出一种方法，通过Cache-Control头自动生成InviCloak的配置，正如我们在第6.3节的实验中所做的那样。这种方法是有效的，因为目前的网站开发人员通过HTTP头来管理不应该被缓存的URL[4, 58]。用这样的方法，配置的工作量也很低。
在这种情况下，有两个错误的来源。1）开发者错误地配置了Cache-Control头；2）网站可能提供不可缓存的公共对象或可缓存的私有对象。第一个来源是由网站对CDN的错误使用引起的，而不是由InviCloak引起的。至于第二个来源，我们的方法会将不可缓存的公共URL视为私有，可缓存的私有URL视为公共。前者只影响InviCloak的效率。后者则会导致隐私泄露。然而，对于那些希望保护用户隐私的网站来说，允许CDN缓存私人对象是一种错误的配置[26, 46, 64]。因此，这种类型的错误也源于网站的URLs错误配置。
总之，InviCloak的配置工作很低，因为配置可以从现有的网站URL分类中继承下来。因此，网站URL分类中的任何现有错误都可能被传播到InviCloak的配置中，其中一些错误可能会泄露用户隐私。

6.6 估算硬件部署成本

我们通过估算CDN提供商或网站需要购买的新硬件数量来比较Phoenix和InviCloak的货币硬件部署成本。我们注意到，这一估算并不包括网站部署InviCloak或CDN升级到使用TEE时的运营成本变化，因为这些数字很难获得。
Phoenix要求CDN支持SGX，但它不需要对网站的源服务器进行任何升级。为了做一个下限估计，我们假设CDN的所有现有边缘服务器都支持SGX。因此，CDN不需要用支持SGX的服务器替换旧的服务器。我们以最大的CDN供应商之一Akamai为例，说明增加边缘服务器的成本。
如图3a所示，Phoenix的吞吐量约为CDN边缘服务器的三分之一。因此，当启用Phoenix时，CDN应至少增加两倍于现有边缘服务器的数量，以达到相当的吞吐量。由于Akamai有超过290K的边缘服务器分布在世界各地[77]，它需要增加大约290K×2=580K的边缘服务器来维持其目前的吞吐量。由于每个支持SGX的英特尔至强处理器至少需要200美元[20]，Akamai的成本下限为580K×200美元=116,000K美元，超过1亿美元。

InviCloak是一个端到端的解决方案，因此它不需要对CDN进行任何修改。然而，与Phoenix不同的是，它为网站的源服务器引入了计算开销。为了估算一个网站部署InviCloak的升级成本，我们以美国排名前50的电子商务网站Etsy[19]为例进行估算。

根据[90]，Akamai观察到Esty在18小时内的流量为395GB，即每秒4909KB。现有的研究表明，74.2%的请求字节是可以缓存的[90]。因此，我们估计起源服务器Esty每秒响应4909×1-7474.2.%2%=1706 KB的有效载荷。请注意，InviCloak在服务器端充当反向代理，因此InviCloak增加的开销与应用程序的逻辑无关。根据图3b，即使每个响应的有效载荷大小小到1KB，InviCloak服务器代理每秒可以提供超过5000×1=5000KB的有效载荷。因此，如果我们假设像Etsy这样的网站将服务器代理部署在一台单独的机器上，那么它只需要像我们测试平台中的那些机器一样增加一台机器来支持其流量负荷。在进行这项工作时，我们测试平台中每台机器的成本不到600美元。此外，在我们的实验中，由于服务器代理仅使原生服务器的吞吐量降低了5%，如果一个网站的原生服务器没有满负荷运行，网站可以将服务器代理与原生服务器放在一起，而无需购买任何新机器。我们省略了内部部署的WAF的硬件成本，因为根据一项调查[7]，内部部署的防火墙在网络服务中很受欢迎。超过98.1%的人已经托管了内部部署的防火墙。
我们注意到，这种成本比较并不是苹果对苹果的比较，因为这两种解决方案的设计本质上是不同的。用Phoenix升级CDN将保护所有使用CDN的网站，而用InviCloak升级网站只保护网站本身的安全。我们不对所有使用CDN的网站的成本进行汇总，因为网站升级的成本是在其相应的组织之间分配。每个组织都会做出独立的决定，从而实现逐步部署的过程。

在Chrome中缺少扩展API，在移动浏览器中扩展支持不足。InviCloak的完整性验证器需要一个在Chrome中缺失的扩展API[2, 3, 13]。这个缺失的API目前限制了该扩展在Chrome上的部署，但InviCloak仍然为希望保护其私人通信的网站和用户提供了一个可行的选择。这是因为桌面版和移动版的Firefox都已经实现了该API[13]。如果用户担心主动攻击，她可以切换到Firefox。此外，我们设想这项工作可以促进移动版Chrome浏览器中扩展功能的启用，我们对Chromium补丁的实现可以帮助Chrome浏览器实现缺失的API。此外，在今天的互联网中，通过DoH和DNSSEC的组合安全地分发新的密钥对的关键想法不受扩展的限制。完整性验证器可以作为一个与浏览器分离的进程来实现。
限制CDN的WAF功能。InviCloak对用户的敏感请求进行加密，因此CDN不能对这些请求应用WAF规则。我们认为这种限制是可以接受的，因为关注隐私的网站已经在其源服务器上为私人数据建立了防火墙。2019年来自573名专业人士的Firemon报告显示，40%的受访者只拥有内部防火墙，58.1%的受访者同时托管内部和云端防火墙[7]。因此，98.1%的受访专业人士已经托管了内部防火墙。使用云防火墙的网站在部署InviCloak后，有可能需要扩展其目前的内部防火墙功能，但并不是从零开始。如果该网站认为InviCloak带来的安全利益超过了扩展其防火墙的开销，它仍然可能认为InviCloak是一个可行的选择。
性能开销。InviCloak在客户端增加了加密操作的开销。具体来说，启用InviCloak的网站的所有用户都能从被动攻击防御中受益，但要付出加密/解密的开销，因为InviCloak可以在不进行任何客户端操作的情况下防止被动攻击。此外，签名验证的开销只影响到安装了扩展并受益于主动攻击防御的用户。如第3.7节所述，当扩展检测到用户正在访问一个没有InviCloak的网站时，它不会进行验证。尽管有开销，InviCloak保留了CDN的性能优势，因为它不对公共数据进行加密，CDN仍然可以在边缘服务器上缓存它们。对于私有数据，InviCloak使CDN能够在InviCloak会话中以加密格式缓存它们。如第3.6节所述，该会话在多个页面访问中存在。此外，可以通过优化技术减少计算开销，但代价是增加设计和实现

的复杂性。一种技术是用哈希值取代网络对象的预签名数字签名，并分发一个包括哈希值列表的签名文件。总的来说，我们相信InviCloak目前的设计和实现在性能、可部署性和可用性之间实现了适当的权衡，根据我们在第6.3节和第6.4节的评估，其开销对大多数网站和用户来说是可以接受的。

第三方服务安全。一个支持InviCloak的网站(如bank.com）可能会使用第三方服务（如service.com），但如果InviCloak没有部署在该服务上，InviCloak就不能保护对service.com的请求。当bank.com的一个页面向service.com发送请求，并且该URL在bank.com的configure.js中被列为私有，bank.com的客户端代理可以拦截该请求，但会转发该请求而不进行加密。这是因为service.com没有在特定的TLSA记录中分发一个新的公钥。只有当service.com也部署了InviCloak时，该请求才能得到保护。我们认为对第三方服务的这种决定是合理的，因为一个网站没有责任和能力来保护其他服务的数据。

对服务工作者和CSS的攻击。最近的研究探讨了对Service Worker的攻击[35, 60, 70]。然而，这些攻击利用的是网站Javascript代码中的漏洞。正如第2.4节所讨论的，这种攻击与我们的工作是正交的，因为我们假设网站的代码是可信的。如果攻击者修改了网站的Javascript代码，使其包含一个漏洞，InviCloak可以通过签名验证来检测它。

除了Service Worker，现有的研究表明，CSS也可能存在漏洞[49, 69]。然而，这些攻击都是利用网站的HTML或JavaScript代码中的特定漏洞。虽然InviCloak默认不对CSS文件进行签名，而且CDN可以修改CSS代码，但我们不知道如何在HTML或Javascript代码没有漏洞的情况下，仅用CSS注入来发动此类攻击。此外，我们可以

没有会话密钥的保护。Cloudflare已经部署了一个名为无密钥SSL的解决方案[79, 80]，允许网站保留其TLS私钥，代价是每一个TLS连接都需要网站的服务器参与。

TLS连接。然而，它仍然会向CDN透露TLS会话密钥。在这个解决方案中，CDN将客户端的TLS握手信息转发给持有所需私钥的网站。无密钥SSL要求网站在本地托管一个密钥服务器来解密或签署由CDN转发的握手信息。因此，不知道私钥的CDN可以完成握手。

Akamai也有一个类似的专利[45]，而WASP是Goh等人提出的类似方法[65]。除了无密钥SSL，Liang等人提出的解决方案采用DANE[42]来告知客户网站对CDN提供商的委托，这样客户就会接受CDN的证书[62]。总的来说，这些解决方案并不能保护用户的私人数据，因为CDN仍然可以获得HTTPS连接的会话密钥。

基于TEE的解决方案。另一个研究方向是利用一种叫做可信执行环境（TEE）的技术，例如，英特尔软件防护扩展（Intel SGX）[29, 38]。STYX[85]和Harpocrates[27]是两个基于英特尔SGX的解决方案，但无密钥SSL的问题同样存在：CDN仍然知道HTTPS会话密钥。

Herwig等人使用英特尔SGX设计并实现了第一个真正的 "无密钥CDN"，名为Phoenix[51]。他们的工作实现了保留CDN的全部功能的目标，同时使网站的TLS私钥和HTTPS会话密钥不被不信任的CDN访问。然而，部署Phoenix及其安全保障需要CDN升级其所有的边缘服务器和软件以支持英特尔的SGX。部署成本可能会使CDN不愿意迁移到这样的解决方案（6.6节）。除了财务成本，Phoenix的边缘服务器的吞吐量比不使用SGX的服务器低三到四倍（第6.2节）。这是一个潜在的性能瓶颈，会降低CDN在加速网络访问方面的效益（§ 6.3）。在我们的评估中，InviCloak显示出比Phoneix更低的开销，但没有保留CDN的WAF功能。

双域解决方案。另一个解决方案是使用两个独立的域名，分别用于CDN可见的内容和私有内容。然而，这种解决方案面临着一些安全、性能和可用性的挑战。首先，也是最重要的，目前的网站更喜欢使用CDN来提供他们的基础HTML文件，以实现页面加载加速

[86]，正如我们的测量结果所示（§2.1）。因此，双域解决方案不能防止CDN中的主动攻击者篡改基础HTML文件以暴露私人内容。
除了安全问题，双域解决方案还可以防止CDN缓存加密的私人内容，如加密的私人照片，因为私人内容是在未共享的TLS会话中，CDN无法区分HTTP请求。相比之下，InviCloak保留了目前网站和CDN之间的TLS会话做法，因此CDN可以缓存加密的HTTP主体和头文件，而无法访问内容。
此外，网站必须采取一些额外的步骤来部署双域解决方案。它们包括 (1) 已经将其私钥分享给CDN的网站需要撤销证书并重新申请两个不同的TLS证书。对于商业证书来说，这样的程序可能是昂贵和耗时的[62]。(2) 网站需要与CDN提供商协商两种不同类型的服务（一个用于缓存，另一个用于转发[81]） (3) 域名分离可能需要大量的网络重组和修改，以便在URL中包含新的域名。
CDN-on-Demand。Gilad等人使用双域解决方案来建立一个低成本的按需CDN[47]。在他们的设计中，他们强迫客户端通过私有域从原点服务器获取所有基本的HTML文件。虽然这种设计可以防止主动攻击，但它与目前网站使用CDN来提供基础HTML文件的做法不相容[86]，而且会增加网站登陆页面的页面加载时间。最后，它继承了上面讨论的双域解决方案在功能和部署方面的其他缺点。

TLS修改。研究人员还提议修改TLS，使中间盒在TLS握手过程中可见。Naylor等人提出了mcTLS[67]，以提供不同的上下文密钥，并使用这些密钥来控制中间盒可以读取或写入的内容。Bhargavan等人发现了mcTLS的安全漏洞，并通过正式证明提供了一个替代方案[33]。Lee等人将mcTLS扩展到maTLS，这使得中间盒可以被审计[59]。与InviCloak相比，这些解决方案面临着重大的部署挑战，因为它们修改了HTTPS/TLS协议栈。终端用户需要在TLS握手中明确授权符合条件的中间箱的证书，这可能会引起可用性的担忧。客户端、服务器和CDN都需要升级他们的TLS库并调整他们的应用程序代码以使用新的协议。
总结：我们认为InviCloak在隐私、性能、用户界面和部署成本之间取得了独特的平衡。附录中的表2比较了各种解决方案之间的特点。与mcTLS和双域解决方案相比，InviCloak没有改变当前的网络界面，并且需要在网络生态系统中进行较少的改变。与TEE解决方案相比。
InviCloak不需要CDN提供商的硬件升级。
9 结论
我们已经介绍了InviCloak，一个允许网站使用CDN进行DDoS保护和网络加速的系统，而不暴露它与用户交换的敏感数据。InviCloak对客户端和源服务器之间传输的敏感数据进行加密，这样，不被信任的CDN就无法窃听他们的通信。InviCloak引入的开销很低，而且很容易部署。一个网站的单边部署可以防止CDN中的被动窃听者。如果用户安装了InviCloak的浏览器扩展，InviCloak可以防止CDN内的主动攻击者窃听或篡改其私人通信。