通过操作系统进行（跨）浏览器指纹识别硬件级功能

曹银芝
利哈伊大学
yinzhi.cao@lehigh.edu

宋莉
利哈伊大学
sol315@lehigh.edu

艾里克·威曼斯 †
圣路易斯华盛顿大学
erikwijmans@wustl.edu

抽象的—在本文中，我们提出了一种浏览器指纹识别技术，它不仅可以在单个浏览器中跟踪用户，还可以在同一台机器上的不同浏览器中跟踪用户。具体来说，我们的方法利用了许多新颖的操作系统和硬件级功能，例如来自显卡、CPU 和已安装的编写脚本的功能。我们通过要求浏览器执行依赖于相应操作系统和硬件功能的任务来提取这些功能。

我们的评估表明，我们的方法可以成功识别 99.24% 的用户，而在同一数据集上，单浏览器指纹识别的最新技术只能识别 90.84%。此外，我们的方法可以实现比文献中唯一具有类似稳定性的跨浏览器方法更高的唯一性率。

我引言

网络跟踪是一种有争议的技术，用于记住和识别过去的网站访问者。一方面，网络跟踪可以对用户进行身份验证，特别是可以结合使用不同的网络跟踪技术进行多因素身份验证以增强安全性。另一方面，网络跟踪还可用于提供个性化服务，如果服务不受欢迎，例如一些不受欢迎的定向广告，这种跟踪就是侵犯隐私的行为。无论我们是否喜欢网络跟踪，也无论它在当前网络中是否合法使用，超过 90% 的 Alexa Top 500 网站 [39] 都采用了网络跟踪，并引起了公众和媒体的广泛关注 [6]。

网络跟踪技术发展迅速。第一代跟踪技术采用有状态的、服务器设置的标识符，例如 cookies 和 evercookie [21]。此后，出现了第二代跟踪技术，称为指纹识别，从有状态标识符转向无状态标识符，即第二代技术不再设置新的标识符，而是探索浏览器中已经存在的无状态标识符，例如插件版本和用户代理。第二代技术通常与第一代技术一起使用，以恢复丢失的cookie。第一代和第二代跟踪都局限于单个浏览器，现在人们正在开发第三代跟踪技术，试图实现跨设备跟踪[16]。

本文的重点是介于第二代和第三代之间的 2.5 代技术，该技术不仅可以在同一浏览器中对用户进行指纹识别，还可以在同一台机器的不同浏览器中对用户进行指纹识别。使用多种浏览器的做法很常见，并得到 US-CERT [42] 和其他技术人员 [12] 的推广：根据我们的调查，[1]70% 的研究用户在同一台 计算机上安装并经常使用至少两种浏览器。

从积极的一面来看，所提出的 2.5 代技术可以用作更强大的多因素用户身份验证的一部分，甚至可以跨浏览器使用。从另一个角度来看，就像许多现有的针对新型网络攻击的研究一样，所提出的 2.5 代跟踪也可以帮助改进现有的隐私保护工作，我们将在第七部分简要讨论我们跨浏览器跟踪的防御。

现在，让我们抛开网络跟踪的优点、缺点和丑陋的用途，看看这项技术本身。要对安装在同一台机器上的不同浏览器进行指纹识别，一种简单的方法是使用对单个浏览器进行指纹识别的现有特征。由于许多现有特征都是特定于浏览器的，因此跨浏览器的稳定特征即使组合在一起进行指纹识别也不够唯一。这就是为什么唯一的跨浏览器指纹识别工作 Boda 等人 [14] 采用 IP 地址作为主要特征的原因。然而，作为网络级特征的 IP 地址在著名的 Panopticlick 测试 [5] 和许多其他相关工作 [10、20、26、32、34、36] 中被排除在现代浏览器指纹识别之外。原因是如果动态分配、通过移动网络连接或笔记本电脑切换位置（例如从家里到办公室），IP 地址就会发生变化，并且在匿名网络或代理后面不可用。

在本文中，我们提出了一种基于许多新颖的操作系统和硬件级特征（例如来自显卡、CPU、音频堆栈和已安装的编写脚本的特征）的（跨）浏览器指纹识别方法。具体来说，由于许多此类操作系统和硬件级功能都通过浏览器 API 暴露给 JavaScript，因此我们可以在要求浏览器通过这些 API 执行某些任务时提取特征。提取的特征可用于单浏览器指纹识别和跨浏览器指纹识别。

† 作者在利哈伊大学 REU 就读期间为本文做出了贡献。

NDSS '17，2017 年 2 月 26 日至 3 月 1 日，美国加利福尼亚州圣地亚哥，
版权所有 2017 互联网协会，ISBN 1-1891562-46-0 http://dx.doi.org/
10.14722/ndss.2017.23152

[1] 有关我们实验的更多详细信息，请参阅附录 A。

让我们以 WebGL 为例，这是一个在浏览器画布对象中实现的 3D 组件。虽然画布（尤其是 2D 部分）已用于单浏览器指纹识别 [9, 32]，但最近一项名为 AmIUnique [26] 的研究实际上认为 WebGL 即使对于单个浏览器来说也“太脆弱和不可靠”。得出这种结论的原因是 AmIUnique 选择一个随机的 WebGL 任务，并且不限制影响指纹识别结果的许多变量，例如画布大小和抗锯齿。

与 AmIUnique 得出的结论相反，我们表明 WebGL 不仅可用于单浏览器指纹识别，还可用于跨浏览器指纹识别。具体来说，我们要求浏览器使用精心选择的计算机图形参数（例如纹理、抗锯齿、光线和透明度）渲染 20 多个任务，然后从这些渲染任务的输出中提取特征。

我们的主要贡献是*菲第*一在单浏览器和跨浏览器指纹识别中使用许多新颖的操作系统和硬件功能，尤其是计算机图形功能。具体来说，在单浏览器指纹识别的同一数据集上，我们采用新功能的方法可以成功识别 99.24% 的用户，而 AmIUnique（即最先进的方法）的识别率为 90.84%。此外，我们的方法可以实现 83.24% 的唯一性和 91.44% 的跨浏览器稳定性，而 Boda 等人 [14] 排除 IP 地址后，唯一性和跨浏览器稳定性仅为 68.98% 和 84.64%。

我们的第二个贡献是，我们对单浏览器和跨浏览器指纹识别做出了一些有趣的观察。例如，我们发现当前的屏幕分辨率测量（例如 AmIUnique、Panopticlick [5, 17] 和 Boda 等人 [14] 所做的测量）是不稳定的，因为当用户放大或缩小网页时，Firefox 和 IE 中的分辨率会发生变化。因此，我们考虑了缩放级别，并根据屏幕分辨率规范了宽度和高度。再举一个例子，我们发现 DataURL 和 JPEG 格式在不同浏览器之间都是不稳定的，因为这些格式存在损失，并且在多个浏览器和服务器端的实现方式也不同。因此，我们需要在跨浏览器指纹识别中采用无损格式进行服务器-客户端通信。

我们的工作是开源的，可以在https://github.com/Song-Li/cross browser/上找到，工作演示可以在http://www.uniquemachine.org上找到。

本文的其余部分组织如下。我们首先在第二部分中介绍所有功能，包括从 AmIUnique 中采用和修改的旧功能以及我们提出的新功能。然后，我们在第三部分中介绍浏览器指纹识别的设计，包括总体架构、渲染任务和掩码生成。之后，我们将在第四部分讨论我们的实现，并在第五部分讨论数据收集。我们评估我们的方法并在第六部分展示结果。接下来，我们将在第七部分讨论指纹识别的辩护，在第八部分讨论一些道德问题，并在第九部分讨论相关工作。我们的论文在第十部分结束。

二、F可打印弗特點

在本节中，我们介绍本文中使用的可指纹特征。我们从先前工作中使用的特征开始，然后介绍一些需要修改的功能，特别是针对跨浏览器指纹识别。接下来，我们介绍我们新提出的功能。

虽然单浏览器指纹识别对特征没有限制，但我们的跨浏览器特征需要反映浏览器以下级别（即操作系统和硬件级别）的信息和操作。例如，顶点和片段着色器都暴露了操作系统中 GPU 及其驱动程序的行为；虚拟核心数量是 CPU 特征；安装的写入脚本是操作系统级特征。原因是操作系统和硬件级别的这些特征在各个浏览器之间相对更稳定：所有浏览器都运行在相同的操作系统和硬件之上。

注意，如果一个操作，尤其是操作的输出，是由浏览器和底层（操作系统和硬件）共同贡献的，我们可以用它来进行单浏览器指纹识别，但在跨浏览器指纹识别中需要摆脱浏览器因素。例如，当我们将图像渲染为立方体上的纹理时，纹理映射是 GPU 操作，但图像解码是浏览器操作。因此，我们只能使用无损格式 PNG 进行跨浏览器指纹识别。再比如，音频信号的动态压缩操作是由浏览器和底层音频堆栈共同执行的，我们需要提取底层特征。现在让我们介绍一下本文中使用的这些特征。

A. 先前可指纹识别的特征

在本节的这一部分中，我们介绍了从最新技术中采用的可指纹识别特征。AmIUnique 论文 [26] 的表 I 中列出了 17 个特征，我们将其全部用于单浏览器指纹识别。更多详细信息可参见他们的论文。由于许多此类特征都是浏览器特定的，因此我们采用一个包含 4 个特征的子集进行跨浏览器指纹识别，即屏幕分辨率、颜色深度、字体列表和平台。其中一些特征需要修改，下面将进行介绍。

B. 旧功能经过重大修改

一项先前的功能，即屏幕分辨率，需要针对单浏览器和跨浏览器指纹识别进行重构。然后，我们引入了另一项可指纹识别的功能，即 CPU 虚拟核心数。最后，两项先前的功能需要进行重大修改，才能针对跨浏览器指纹识别进行重构。

*屏幕分辨率。*目前屏幕分辨率的测量是通过 JavaScript 下的“screen”对象进行的。但是我们发现许多浏览器，尤其是 Firefox 和 IE，会根据缩放级别按比例更改分辨率值。例如，如果用户在 Firefox 和 IE 中使用“ctrl++”放大网页，则屏幕分辨率是不准确的。我们认为，无论是在单浏览器指纹识别中还是跨浏览器指纹识别中，都需要考虑缩放级别。

具体来说，我们追求两个不同的方向。首先，我们采用现有的工作[13]，根据div标签的大小和设备像素比检测缩放级别，然后相应地调整屏幕分辨率。其次，由于前一种方法并不总是可靠的，正如发明者所承认的，我们采用了一个新特征，即div和设备像素比之间的比率。

屏幕的宽度和高度，不会随着缩放级别而改变。

除了屏幕分辨率之外，我们还发现其他一些属性，例如availHeight、availWidth、availLeft、availTop 和screenOrientation，在单浏览器和跨浏览器指纹识别中都很有用。前四个表示浏览器可用的屏幕（不包括系统区域，例如 Mac OS 的顶部菜单和工具栏）。最后一个显示屏幕的位置，例如屏幕是横向还是纵向，以及屏幕是否颠倒。

*CPU 虚拟核心的数量。*核心数可以通过浏览器的一项新特性hardwareConcurrency 来获取，该特性提供了 Web Workers 的能力信息。现在，许多浏览器都支持该特性，但有些浏览器，尤其是早期版本的浏览器，并不支持。如果不支持，则存在一个侧通道 [1] 来获取核心数。具体来说，可以在增加 Web Workers 数量时监视 payload 的完成时间。当 Web Workers 数量达到一定水平时，完成时间显著增加，即达到硬件并发的极限，这时对核心数进行指纹识别就很有用了。需要注意的是，有些浏览器（例如 Safari）会将 Web Workers 可用的核心数减半，而我们需要将核心数翻倍才能进行跨浏览器指纹识别。

发明者知道内核数量是可以进行指纹识别的 [2]，这也是他们称之为硬件并发而不是内核的原因之一。然而，在浏览器指纹识别的现有技术中，该功能从未被使用或测量过。

*音频上下文。*AudioContext 借助操作系统和音频卡中的音频堆栈，提供了从信号生成到信号过滤的一系列音频信号处理功能。具体来说，现有的指纹识别工作 [18] 使用 OscillatorNode 生成三角波，然后将该波输入到 DynamicsCompressorNode，这是一个抑制大声音或放大小声音的信号处理模块，即产生压缩效果。然后，处理后的音频信号通过 AnalyserNode 转换为频域。

在同一台机器上，频域中的波形在不同浏览器之间有所不同。但是，我们发现峰值及其对应的频率在不同浏览器之间相对稳定。因此，我们创建了一个在频率轴和值轴上都具有小步长的箱列表，并将峰值频率和值映射到相应的箱中。如果一个箱包含频率或值，我们将该箱标记为 1，否则标记为 0：这样的箱列表就是我们的跨浏览器特征。

除了波形处理之外，我们还从目标音频设备获取以下信息：采样率、最大通道数、输入数、输出数、通道数、通道数模式和通道解释。请注意，据我们所知，现有的指纹识别工作中还没有使用过此类音频设备信息进行浏览器指纹识别。

*字体列表。*AmIUnique 的测量是基于 Flash 插件的，然而 Flash 正在迅速消失，这一点在他们的论文中也提到并承认了。在我们进行实验时，Flash 已经很少支持获取字体列表。相反，我们采用 Nikiforakis 等人 [36] 提到的侧信道方法，通过测量某个字符串的宽度和高度来确定字体类型。请注意，并非所有字体都是跨浏览器可指纹识别的，因为有些字体是 Web 特定的并由浏览器提供，我们需要应用第 III-C 节中所示的掩码来选择子集。另一件值得注意的是，我们知道 Fifield 等人 [20] 提供了 43 种字体的子集用于指纹识别，但他们的工作基于单浏览器指纹识别，不适用于我们的跨浏览器场景。

C. 新提出的原子指纹特征

在本小节和下一小节中，我们将介绍我们新提出的可指纹识别功能。我们首先从原子功能开始，原子功能是指浏览器直接向 JavaScript 公开 API 或组件。然后，我们将介绍复合功能，这通常需要多个 API 和组件进行协作。

*线条、曲线和抗锯齿。*直线和曲线是 Canvas（2D 部分）和 WebGL 都支持的 2D 特征。抗锯齿是一种计算机图形技术，用于通过平滑锯齿（即锯齿状或阶梯状线条）来减少锯齿，无论是在单个直线/曲线对象中还是在计算机图形模型的边缘。目前有许多用于抗锯齿的算法 [4]，例如第一性原理方法、信号处理方法和 mipmapping，这些算法使抗锯齿可指纹化。

*顶点着色器。*顶点着色器由 GPU 和驱动程序渲染，将 3D 模型中的每个顶点转换为 2D 裁剪空间中的坐标。在 WebGL 中，顶点着色器可以通过 3 种方式接受数据：来自缓冲区的属性、始终保持不变的统一值以及来自片段着色器的纹理。在渲染计算机图形任务时，顶点着色器通常与下面描述的片段着色器结合使用。

*片段着色器。*片段着色器由 GPU 和驱动程序渲染，它将片段（例如光栅化输出的三角形）处理为一组颜色和一个深度值。在 WebGL 中，片段着色器以以下方式获取数据：

- *制服。*在一次绘制调用期间，统一值对于片段中的每个像素都保持不变。因此，统一值是不可指纹识别的功能，我们在此列出它以确保完整性。
- *变化。*Varying 将顶点着色器中的值传递到片段着色器，片段着色器在这些值之间进行插值并光栅化片段，即绘制片段中的每个像素。插值算法在不同的计算机显卡中有所不同，因此 Varying 是可指纹识别的。
- *纹理。*给定顶点和纹理之间的映射设置，片段着色器会根据纹理计算每个像素的颜色。由于纹理的分辨率有限，片段着色器需要根据目标所包围的纹理中的像素为目标像素插值。纹理插值算法也因显卡而异，因此纹理可指纹识别。

WebGL 中的纹理可以进一步分为几类：（1）普通纹理，即我们

（2）深度纹理，即包含每个像素的深度值的纹理；（3）动画纹理，即包含视频帧而不是静态图像的纹理；（4）压缩纹理，即接受压缩格式的纹理。

*通过 Alpha 通道实现透明度。*透明度是 GPU 和驱动程序提供的功能，允许背景与前景混合。具体来说，值为 0 到 1 之间的 alpha 通道使用合成代数将背景和前景图像合成为单个最终图像。alpha 通道中有两个指纹点。首先，我们可以使用一个 alpha 值来指纹识别背景和前景之间的合成算法。其次，我们可以在 alpha 值从 0 增加到 1 时指纹识别透明效果的变化。由于某些显卡采用离散 alpha 值，因此在透明效果的变化中可能会观察到一些跳跃。

*图像编码和解码。*图像可以以不同的格式进行编码和压缩，例如 JPEG、PNG 和 DataURL。有些格式（例如 PNG）是无损的，而有些格式（例如 JPEG）在压缩时会丢失信息。压缩图像的解压是一个可指纹识别的特征，因为不同的算法在解压过程中可能会发现不同的信息。根据我们的研究，这是一个单浏览器特征，不能用于跨浏览器。

*已安装的写作脚本（语言）。*书写脚本（系统）或通常称为书面语言的语言，例如中文、韩语和阿拉伯语，由于库的大小和语言的本地性，需要安装特殊库才能显示。浏览器不提供用于访问已安装语言列表的 API，但可以通过侧信道获取此类信息。具体而言，安装了特定语言的浏览器将正确显示该语言，否则会显示多个框。也就是说，框的存在可用于识别该语言的存在。

D. 新提出的复合指纹特征

现在，让我们介绍一下我们新提出的复合指纹识别功能，它由多个浏览器 API 或组件呈现，有时还会在浏览器 API 之上构建额外的算法。

*建模和多种模型。*建模，或者本文中具体指 3D 建模，是一种通过三维表面以数学方式描述对象的计算机图形学过程。模型的顶点由顶点着色器处理，表面由片段着色器处理。不同的对象由不同的模型表示，并且可能相互交互，尤其是当技术 低，例如照明，存在。

*照明和阴影映射。*光照是计算机图形学中对光效的模拟，而阴影贴图是测试某个像素在某种光照下是否可见，并添加相应的阴影。光照有很多种，比如

环境光、定向光和点光，它们的光源不同。此外，许多效果都伴随着灯光，例如反射、半透明、光跟踪和间接照明我 国家，当灯光与一个或多个计算机图形模型交互时。WebGL 不

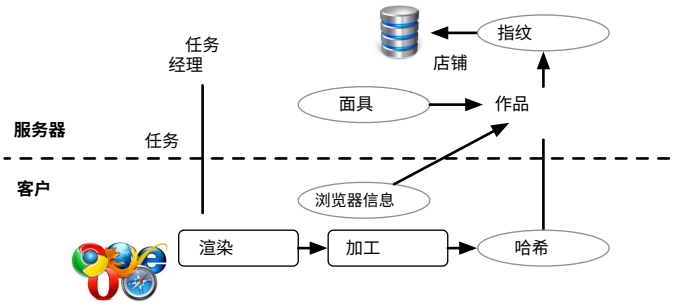

图 1：系统架构

提供了灯光和阴影的直接 API，而一些 WebGL 库（例如 three.js）则提供了在 WebGL 的顶点和片段着色器之上构建的用于灯光和阴影的高级 API。

*相机。*相机，或者说针孔相机模型，将空间中的 3D 点映射到图像中的 2D 点。在 WebGL 中，相机由顶点和片段着色器处理的相机投影矩阵表示，可用于旋转和放大和缩小对象。

*剪切平面。*裁剪将渲染操作限制在定义的关注区域内。在 3D 渲染中，裁剪平面与相机有一定距离且垂直，这样可以防止渲染距离相机太远的表面。在 WebGL 中，裁剪平面由顶点和片段着色器使用额外提供的算法执行。

三、D设计

A. 总体架构

图 1 显示了系统架构。首先，服务器端的任务管理器将各种渲染任务（例如绘制曲线和直线）发送到客户端。注意，渲染任务还涉及获取操作系统和硬件级别的信息，例如屏幕分辨率和时区。然后，客户端浏览器通过调用特定 API 或 API 组合来渲染这些任务，并产生相应的结果，例如图像和声波。然后，这些结果（尤其是图像）被转换为哈希值，以便可以方便地将它们发送到服务器。同时，浏览器还会收集特定于浏览器的信息，例如是否支持抗锯齿和压缩纹理，这些信息将在服务器端用于指纹合成。

接下来，当服务器收集到客户端的所有信息后，服务器将开始合成指纹。具体来说，指纹由客户端的哈希列表和与哈希列表对应的 1 或 0 列表的掩码生成——我们在哈希列表和掩码之间执行“与”运算，然后生成另一个哈希作为指纹。单一浏览器指纹识别的掩码是直的时长 tforward，所有 1 的列表。跨浏览器指纹识别的掩码由两个来源合成。

首先，收集到的浏览器信息将用于掩码：如果浏览器不支持抗锯齿，则所有涉及抗锯齿的任务的掩码中的位值为零。其次，我们将为每个浏览器对设置不同的掩码，例如 Chrome 与 Firefox 以及 Chrome 与 Windows Edge。

在接下来的两节中，我们首先介绍客户端的渲染任务，然后介绍指纹组成，特别是如何生成掩码。

B. 渲染任务

在本节中，我们将介绍本文提出的不同渲染任务。在此之前，让我们首先介绍下面的基本画布设置。画布的大小为 256 ×256. 画布的轴定义如下。[0, 0, 0] 是画布的中间，其中 x 轴是向右增加的水平线，y 轴是向下增加的垂直线，z 轴在远离屏幕时增加。存在一个功率为 [R: 0.3, G: 0.3, B: 0.3] 的环境光，其比例为 1，并且相机放置在 [0, 0, -7] 的位置。这两个组件是必需的，因为否则模型将完全是黑色的。在本文的其余部分，除非另有说明，例如具有 2D 特征的任务 (d) 和具有附加灯光的其他任务，否则我们在所有任务中使用相同的基本设置。

请注意，与 AmIUnique [26] 中的设置不同，我们的画布设置在当前窗口的条件发生变化时仍然可靠。具体来说，我们测试了三种不同的变化：窗口大小、侧边栏和缩放级别。首先，我们手动更改窗口大小，发现画布中的内容在视觉上和哈希值计算上都保持不变。其次，我们放大和缩小当前窗口，发现内容在视觉上根据定义发生变化，但哈希值保持不变。最后，我们打开浏览器控制台作为侧边栏，发现画布内容也保持不变，类似于改变窗口大小。现在让我们从任务 (a) 到 (r) 介绍我们的渲染任务。

*任务（a）：纹理。*图 2(a) 中的任务是测试片段着色器中的常规纹理功能。具体来说，在画布上用随机生成的纹理渲染经典的 Suzanne Monkey Head 模型 [19]。纹理是一个大小为 256 的正方形 ×256，是通过为每个像素随机选取一种颜色而产生的。也就是说，我们在一个像素处为红、绿、蓝三原色均匀地生成三个介于 0 到 255 之间的随机值，将这三种原色混合在一起，并将其用作该像素的颜色。

我们选择这种随机生成的纹理而不是常规纹理，因为这种纹理具有更多可指纹识别的特征。原因如下。当片段着色器将纹理映射到模型时，片段着色器需要在纹理中插入点，以便将纹理映射到模型上的每个点。插值算法因显卡而异，当纹理颜色发生剧烈变化时，差异会放大。因此，我们生成这种纹理，其中每对相邻像素之间的颜色变化很大。

*任务（b）：变化。*此任务如图 2(b) 所示，旨在测试片段着色器在画布上的可变特性。在一个立方体模型的六个表面上绘制不同的可变颜色，每个表面上指定四个点的颜色。我们选择这种可变颜色来放大每个单个表面上的颜色差异和变化。例如，当表面的一个顶点上蓝色丰富（例如 0.9，比例为 1）时，另一个顶点将缺少蓝色（例如 0.1），并且具有更多的绿色或红色。此外，为了与任务（c）进行比较，将相机放置在[0, 0, -5]的位置。

*任务（b'）：抗锯齿+变化。*图 2(b') 中的任务是测试抗锯齿功能，即浏览器如何平滑模型的边缘。具体来说，我们采用与任务 (b) 相同的任务，并添加抗锯齿。如果我们放大图 2(b')，我们会发现两个模型的边缘都被平滑了。

*任务（c）：相机。*图 2(c) 中的任务是测试相机功能，即输入到片段着色器的投影矩阵。此任务中的每个设置与任务 (a) 相同，但相机除外，相机被移动到 [-1, -4, -10] 的新位置。同一个立方体看起来比任务 (a) 中的立方体小，因为相机从立方体移得更远（z 轴为 -10，而不是 -5）。

*任务（d）：线条和曲线。*图 2(d) 中的任务是测试直线和曲线。在画布上绘制一条曲线和三条不同角度的直线。具体来说，曲线遵循以下函数：*是*=256−100*余弦*（2）。0*πx*/100。0）+30*余弦*（4）。0*πx*/100。0）+6*余弦*（6）。0*πx*/100。0)，其中 [0, 0] 为画布的左侧和顶部，x 轴向右增加，y 轴向下增加。三条线的起点和终点分别为*{*[38.4, 115.2], [89.6, 204.8]*}*，*{*[89.6, 89.6], [153.6, 204.8]*}*， 和*{*[166.4, 89.6], [217.6, 204.8]*}*。我们选择这些特定的线条和曲线，以便可以测试不同的渐变和形状。

*任务（d'）：抗锯齿+直线和曲线。*任务（d'）是任务（d）的抗锯齿版本。

*任务（e）：多模型。*图 2(e) 中的任务是测试不同模型在同一个画布中如何相互影响。除了 Suzanne 模型之外，我们引入了另一个看起来像单人扶手沙发的模型（称为沙发模型），并将两个模型并行放置。按照任务 (a) 中描述的相同程序将另一个随机生成的纹理映射到沙发模型上。

*任务（f）：光。*图 2(f) 中的任务是测试漫反射点光源和 Suzanne 模型之间的相互作用。漫反射点光源在照亮物体时会引起漫反射。具体来说，光源为白色，RGB 值相同，每种原色的光源功率为 2，光源位于 [3.0, -4.0, -2.0]。

我们在本任务中选择了白色光源，因为纹理色彩丰富，单一颜色的光可能会削弱纹理上的一些细微差异。光的强度也经过精心选择，因为非常弱的光不会照亮 Suzanne 模型，使其不可见，但非常强的光会使一切变白并削弱所有可指纹识别的特征。在 6 台机器的小规模实验中，当将功率从 0 增加到 255 时，我们发现当光功率为 2 时，这些机器之间的像素差异最大。光的位置是随机选择的，不会影响特征指纹识别结果。

*任务（g）：灯光和模型。*图 2(g) 中的任务是测试单个漫射点光源和两个模型之间的相互作用，因为一个模型在被点光源照射时可能会在另一个模型上产生阴影。每种光源设置与任务 (f) 相同，模型与任务 (e) 相同。

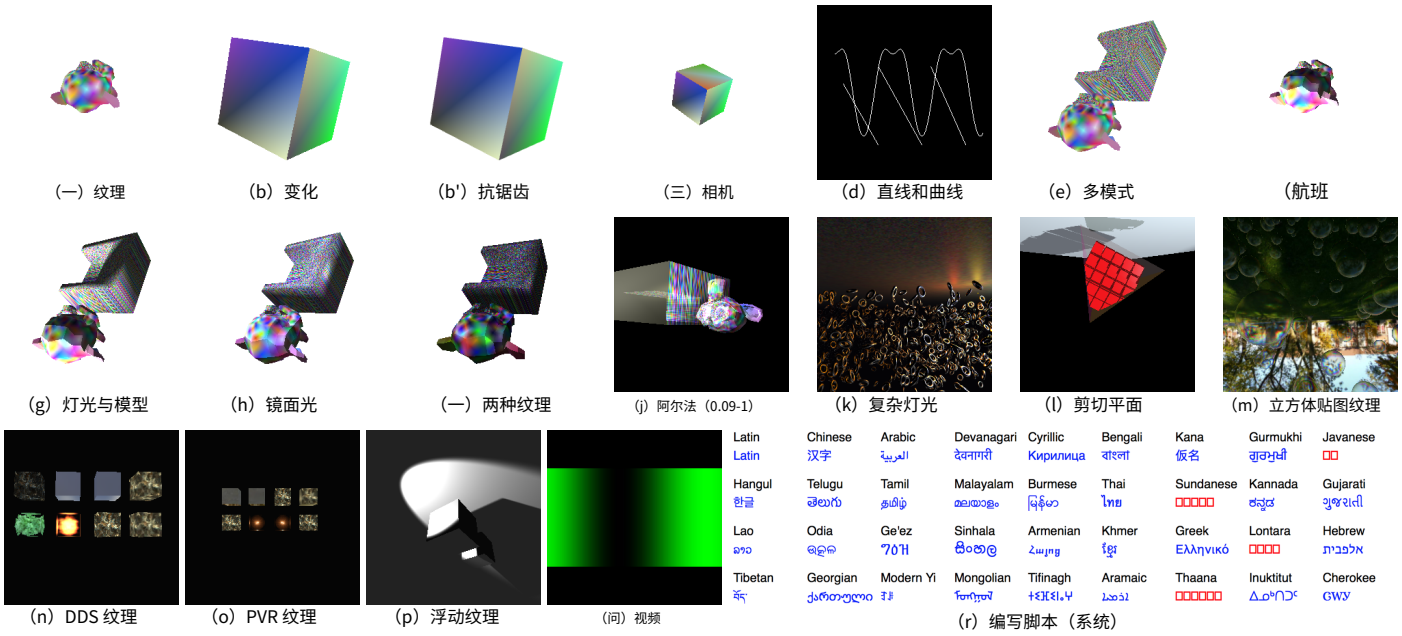

图 2：用于指纹识别的客户端渲染任务

*任务（h）：镜面光。*图 2(h) 中的任务是测试另一种颜色的漫射点光和镜面点光在两个模型上的效果。与漫射点光类似，镜面点光会在物体上引起镜面反射。具体来说，两种光都位于 [0.8, -0.8, - 0.8]，漫射点光的 RGB 为 [0.75, 0.75, 1.0]，镜面点光的 RGB 为 [0.8, 0.8, 0.8]。

有两点值得注意。首先，我们选择特定的相机位置是因为它离模型更近，效果更大。特别是，人们可能会注意到沙发模型背面被镜面点光照亮的斑点。其次，虽然漫射点光的颜色偏蓝，但仍然有很多红色和绿色。我们想测试其他颜色，但考虑到纹理色彩丰富，白光仍然是指纹识别的最佳选择。

*任务（h'）：抗锯齿+镜面光。*任务（h'）是任务（h）的抗锯齿版本。

*任务（h”）：抗锯齿+镜面光+旋转。*任务（h'）与任务（h'）相同，但旋转了 90 度。

*任务（i）：两种纹理。*图 2(i) 中的任务是测试将两种不同的纹理映射到同一物体上的效果。在任务 (h) 的基础上，即其他所有设置都相同，我们将另一层随机生成的纹理映射到 Suzanne 和沙发模型上。

*任务（j）：Alpha。*图 2(j) 中的任务由 8 个子任务组成，目的是测试不同 alpha 值的效果。具体来说，我们将 Suzanne 和 sofa 模型并行，并更改从这个特定集合中选择的 alpha 值，*{*0.09、0.1、0.11、0.39、0.4、0.41、0.79、1*}*，其中 0 表示完全透明，1 表示不透明。

同样，有两点值得注意。首先，我们精心选择了这组值来反映不同的 alpha 值和微小的值变化：三个代表值*{*0.1、0.4、0.8*}*以及它们附近的值都会被选中。值是 0.01 时增强，因为许多 GPU 不接受更小的步长。其次，Suzanne 和沙发模型的定位使得它们部分重叠，并且当模型变得透明时，沙发模型的隐藏结构可见。例如，从模型背面看时，沙发模型的扶手部分可见。

*任务（k）：复杂灯光。*图 2(k) 中的任务是测试复杂的光特性，例如反射、移动光和多个模型之间的光跟踪。具体来说，我们生成了 5,000 个不同角度的金属环模型，这些模型随机放置在地面上并堆叠在一起。为了确保可靠性，我们使用种子随机数生成器，每次都使用相同的随机种子，以便可以在不同的浏览器和机器上重复测试。靠近底部的两个点光源（黄色和红色）在整个场景的右上角盘旋。当灯光照亮下面的环时，其他环也会通过反射被照亮，来自不同光源的两种颜色混合在一起。

注意我们选择单色光源是因为模型本身并不多彩，有颜色的灯光会照亮环上更多的细节。而且不同颜色的灯光会相互影响，创造出更细致的效果。

*任务（k'）：抗锯齿+复杂灯光。*任务（k'）是任务（k）的抗锯齿版本。

*任务（l）：剪切平面。*图 2(l) 中的任务是测试裁剪平面的移动和 FPS。具体来说，我们将一个静态正四面体放在地面上，用平行光照射它，然后移动裁剪平面，使观察者感觉到四面体在移动。当裁剪平面移动到该位置时，图 2(l) 中捕获的图像是上下颠倒的。

*任务（m）：立方体贴图纹理+菲涅尔效果。*图 2（m）中的任务是测试立方体贴图纹理和菲涅尔效果

光反射。具体来说，立方体贴图纹理 [7] 是一种特殊的纹理，它利用立方体的六个面作为贴图形状，而菲涅尔效应则是一种观察结果，即反射光量取决于视角。我们用一个正常的校园场景创建一个立方体贴图纹理，并在纹理顶部放置几个透明气泡以实现菲涅尔效果。所有气泡都在随机移动并在动画中相互碰撞。

*任务（n）：DDS 纹理。*DDS 纹理是指使用 DirectDraw 表面文件格式的纹理，这是一种采用 S3 纹理压缩 (S3TC) 算法的特殊压缩数据格式。S3TC 有五种不同的变体，从 DXT1 到 DXT5，每种格式都有一个启用 mipmapping 的选项，这是一种在纹理文件中将高分辨率纹理缩放为多种分辨率的技术。由于 DXT2 与 DXT3 相似，DXT4 与 DXT5 相似，因此任务 (n) 仅在每列中测试带和不带 mipmapping 的 DXT1、DXT3 和 DXT5，如图 2(n) 所示。为了进行比较，我们还在最右边的一列中包含了一个未压缩的 ARGB 格式的纹理。图 2(n) 中有两个灰色立方体，因为该特定机器不支持带 mipmapping 的 DXT3 和 DXT5。

*任务（o）：PVR 纹理。*PVR 纹理，或称为 PVRTC 纹理，是另一种纹理压缩格式，主要被移动设备采用，例如所有 iPhone、iPod Touch 和 iPad 以及一些 Android 产品。根据数据块的大小，有两种模式：4 位模式和 2 位模式。此外，还有两个流行的版本，v1 和 v3，我们还可以选择启用 mipmapping。总的来说，图 2(o) 所示的任务 (o) 有 8 个子任务，列举了位模式、版本和 mipmapping 的不同组合。同样，灰色立方体表示不支持该格式。

*任务 (p)：浮动纹理。浮*点纹理，或称为浮点纹理，使用浮点而不是整数来表示颜色值。一种特殊类型的浮点纹理是深度纹理，它包含特定场景的深度缓冲区中的数据。任务 (p)（如图 2(p) 所示）取自现有的在线测试 [15]，用于渲染浮点和深度纹理。

*任务（q）：视频（动画纹理）。*图 2(q) 中的任务是测试视频的解压。具体来说，我们从具有三种不同压缩格式（即 WebM、高质量 MP4 和标准 MP4）的 PNG 文件创建一段两秒的静态场景视频，将该视频作为动画纹理映射到立方体，并从视频中捕获六个连续帧。

注意，虽然所有视频都是用一个 PNG 文件创建的，但由于压缩算法有损，因此捕获的帧数不同。我们选择连续六帧是因为 JavaScript 仅提供 API 来获取特定时间的帧，而不是特定帧数的帧。根据我们的实验，连续六帧可以确保目标帧在集合内。

*任务（r）：编写脚本。*图 2(r) 中的任务是获取浏览器中支持的书写脚本列表，例如拉丁文、中文和阿拉伯文。由于现有浏览器均未提供获取支持的书写脚本列表的 API，因此我们采用旁通道来测试每种书写脚本的存在性。具体方法如下。

算法 1跨浏览器掩码生成

输入：
米：所有可能的掩码的集合。
赫浏览器,机器={哈希任务1、哈希任务2、哈希任务3，...}：哈希列出特定机器的一个浏览器上的所有渲染任务。
赫浏览器={赫浏览器,机器1, 高度浏览器,机器2，...}：哈希列表用于浏览器。
HS={H铬合金, 高度火狐, 高度歌剧，..}：总体哈希列表。 过程：

```
为了一切皆有可能{时长浏览器1，小时浏览器2} ⊂HS做
    最大限度独特←0
    最大限度面具←无效的
    为了面具在米做
        FS←{}
        数数←0
        为了米1∈时长浏览器1和米2∈时长浏览器2做
            如果米1&面具==米2&面具和米1&面具∈        / FS然后
                数数+ +
                FS.添加（米1&面具） 如
            果结束
        结束于
        尤尼奇←数量/尺寸（时长浏览器1） 如果
        Uniq > 最大独特然后
            最大限度独特←尤尼奇
            最大限度面具←面具
        如果结束
    结束于
    最大限度面具是浏览器 1 和 2 的掩码。
结束于
```

浏览器中会渲染每种书写脚本的语言版本。如果支持书写脚本，渲染将成功；否则，将显示一组框而不是脚本。因此，我们可以检测这些框来测试浏览器是否支持该脚本：例如，图 2(r) 显示，爪哇语、苏丹语、隆塔拉语和塔纳语在特定的测试浏览器中不受支持。我们当前的测试列表有 36 种书写脚本，这些脚本来自维基百科 [8]，并按其受欢迎程度进行排序。

C. 指纹组成

在本节中，我们将介绍如何基于客户端渲染任务的哈希值在服务器端形成指纹。如前所述，指纹是通过对所有任务的哈希列表和掩码进行“与”运算计算得出的哈希值。对于单浏览器指纹识别，掩码直接全部为 1，而对于跨浏览器指纹识别，掩码则由两个子掩码计算得出。我们在第三部分 A 中讨论了根据浏览器是否支持某些功能计算得出的第一个子掩码，现在将讨论第二个子掩码，每个浏览器对的掩码都不同。

每两个浏览器生成一个掩码是一种基于训练的方法。具体来说，我们使用一个小子集来获得一个同时优化跨浏览器稳定性和唯一性的掩码。请注意，与假阳性和假阴性类似，这两个数字，即跨浏览器稳定性和唯一性，是一枚硬币的两面：当跨浏览器稳定性增加时，唯一性会降低，反之亦然。让我们考虑两个极端的例子。如果我们使用单一浏览器特征，跨浏览器稳定性为零，但唯一性最高。相反，如果我们只使用一个特征，例如平台，跨浏览器稳定性是 100%，但唯一性非常低。

算法 1 展示了针对每个浏览器对的掩码的训练过程。我们采用强力搜索：虽然不是最有效的，但却是最有效和最完整的。由于

由于训练数据规模较小，我们意识到暴力破解是可行的，并且会产生最佳结果。具体来说，我们首先枚举每个浏览器对（第 1 行），然后枚举每个可能的掩码（第 4 行）。对于每个掩码，我们都会遍历训练数据（第 7 行），并确保选择最大化跨浏览器稳定性的掩码乘以唯一性（第 8-11 行和第 14-17 行）。

四、一实施

我们的开源实现（不包括所有开源库，例如 JavaScript 3D 库 three.js 和用于矩阵运算的 JavaScript 库 glMatrix）大约有 21K 行代码 (LoC)。具体来说，我们的方法涉及大约 14K 行 JavaScript、1K 行 HTML、2.4K 行 Coffeescript、500 行 C 代码和 3.7K 行 Python 代码。

我们现在将代码分为客户端和服务器，并在下面进行描述。客户端代码有一个 JavaScript 管理器，它由 Coffeescript 生成。管理器执行三项工作：(1) 加载所有渲染任务，(2) 收集渲染任务的所有结果以及浏览器信息，以及 (3) 将结果发送到执行哈希运算然后与服务器端代码通信的 JavaScript 代码段。任务 (n) 和 (o) 用 C 编写，并通过 Emscripten 转换为 JavaScript。所有其他渲染任务都直接用 JavaScript 编写：任务 (k)–(m) 是在 three.js 的帮助下编写的，其余任务直接使用 WebGL 或 JavaScript API。所有渲染任务都使用 glMatrix 进行矢量和矩阵运算。

我们实现的服务器端是用 Python 编写的，作为 Apache 服务器的一个模块。我们的服务器端代码可以进一步分为两部分：第一部分有 1.2K LoC，用于与客户端代码通信并将哈希值存储到数据库中并将图像存储到文件夹中；第二部分有 2.5K LoC，用于分析，例如在收集到的指纹上生成和应用掩码。

V. Data碳收藏

我们从两个众包网站收集数据，即 Amazon Mechanical Turks 和 MacroWorkers。具体来说，我们指示众包工作者自行选择通过两种不同的浏览器访问我们的网站，如果他们通过三种浏览器访问网站，他们将获得奖金。访问后，我们的网站将为每位工作者提供一个唯一的代码，以便她可以将其输入回众包网站以获得报酬和可选奖金。请注意，在我们的数据收集中，除了哈希之外，我们还将所有图像数据发送到服务器——如果部署我们的方法，则不需要这样的步骤。

为了确保我们拥有真实数据，我们会在每个众包工作者访问的 URL 中插入一个唯一标识符，例如 http://oururl.com/?id=ABC。该唯一标识符以 cookie 的形式存储在客户端浏览器中，这样如果用户再次访问我们的网站，她将获得相同的标识符。此外，我们只允许一个众包工作者承担一次工作。例如，MTurks 中的人类智能任务 (HIT) 数量为每个工作者一个。

三个月内，我们总共收集了 1,903 名用户的 3,615 个指纹。有些用户只是访问我们的网站

表一：我们的方法、AmIUnique 和 Panopticlick 收集的数据集的六个属性的归一化熵（最后两列从 AmIUnique 论文中复制而来）

	我们的	我是独一无二的	全景式
用户代理	0.612	0.570	0.531
插件列表	0.526	0.578	0.817
字体列表 (Flash) 屏幕	0.219	0.446	0.738
分辨率	0.285	0.277	0.256
时区	0.340	0.201	0.161
已启用 Cookie	0.001	0.042	0.019

仅使用一种浏览器，无法完成双浏览器任务。我们直接使用所有指纹进行单浏览器指纹识别。对于跨浏览器指纹识别，如果有足够的数据，数据集将为每个浏览器对平均分成十个部分：一个用于生成掩码，另外九个用于测试。

A. 将我们的数据集与 AmIUnique 和 Panopticlick 进行比较

本节的目的是将我们的数据集与 AmIUnique 和 Panopticlick 进行比较，使用 AmIUnique 论文中发明的归一化香农熵度量。具体来说，公式 1 显示了根据他们的论文的定义：

$$NH = \frac{赫（十）}{赫_{米}} = \frac{-\sum_{我磷（十我）\ 日志_2磷（十我）}}{日志_2（否）} \quad (1)$$

赫（*十*）是香农熵，其中*十*是一个具有可能值的变量*{十*$_1$*, x我，..}*并且 P(X) 是一个概率函数。*赫*$_{米}$是最坏的情况，其中每个指纹都有相同的概率，并且我们拥有最大熵。*否*是指纹总数。

表一显示了比较结果，其中 AmIUnique 和 Panopticlick 的统计数据来自 AmIUnique 论文的表三。我们观察到，除了字体和时区列表外，我们的数据集的归一化熵值与过去方法中使用的数据集非常相似。

首先，字体列表的归一化熵比 AmIUnique 下降了 0.22，比 Panopticlick 下降了 0.52。AmIUnique 解释的原因是 Flash 正在消失。当我们收集数据时，与 AmIUnique 相比，支持 Flash 的浏览器百分比下降得更多。为了进一步验证我们的数据集，我们还计算了 JavaScript 收集的字体列表的归一化熵。该值为 0.901，非常接近 Panopticlick 的值。

其次，时区标准化熵比 AmIUnique 增加了 0.139，比 Panopticlick 增加了 0.179。原因是 MicroWorkers 的众包工作者非常国际化，从非洲和欧洲到亚洲和拉丁美洲。具体来说，MicroWorkers 允许我们创建针对世界各地不同地区的活动，我们确实为每个大洲创建了活动。

另一件值得注意的事情是，对于我们的数据集，启用 cookie 的归一化熵几乎为零。原因是我们从众包网站收集数据，其中

表 II：比较 AmIUnique、Boda 等人（不包括 IP 地址）和我们的方法的总体结果（“唯一”表示唯一指纹占总数的百分比，“熵”表示香农熵，“稳定性”表示跨浏览器稳定的指纹百分比。我们没有在表中列出 AmIUnique 的跨浏览器数量和 Boda 等人的单浏览器数量，因为这些数字非常低，而且他们的方法不是为此目的而设计的。）

	单一浏览器		跨浏览器		
	独特的	熵	独特的	熵	稳定
我独特 [26]	90.84%	10.82			
Boda 等人 [14] 我们的	99.24%	10.95	68.98% 83.24%	6.88 7.10	84.64% 91.44%

工作者需要启用 cookie 才能获得报酬。如果他们禁用 cookie，他们甚至无法登录众包网站。相比之下，AmIUnique 和 Panopticlick 都吸引了一般网络用户，其中一小部分人可能会禁用 cookie。一般来说，禁用 cookie 的人很少，因为 cookie 对许多现代网络功能至关重要。

六、R结果

在本节中，我们首先概述我们的结果，然后根据不同的浏览器对和功能细分结果。最后，我们提出一些有趣的观察结果。

A. 概述

我们首先概述了单浏览器指纹识别和跨浏览器指纹识别的结果。具体来说，我们将单浏览器指纹识别与 AmIUnique 进行了比较，这是最先进的，而跨浏览器指纹识别与 Boda 等人的指纹识别（不包括 IP 地址）进行了比较。请注意，尽管许多新功能（例如 AmIUnique 中的这些功能）是在 Boda 等人之后出现的，但这些功能是特定于浏览器的，我们发现 Boda 等人使用的功能仍然是跨浏览器稳定性最高的功能。

现在介绍如何复现这两项工作的结果。AmIUnique 是开源的 [3]，我们可以直接从 github 下载源代码。Boda 等人提供了一个开放的测试网站 (https://fingerprint.pet-portal.eu/)，我们可以直接下载指纹识别 JavaScript。我们相信直接使用他们的源代码可以最大限度地减少所有可能的实现偏差。

AmIUnique、Boda 等人和我们的方法的总体结果如表 II 所示。首先让我们看一下单浏览器指纹识别。我们在唯一性和熵方面将我们的方法与 AmIUnique 进行了比较。唯一性是指唯一指纹占指纹总数的百分比，熵是香农熵。评估表明，我们的方法可以唯一地识别 99.24% 的用户，而 AmIUnique 为 90.84%，增加了 8.4%。对于熵，最大值为 10.96，两种方法，尤其是我们的方法，都非常接近最大值。也就是说，两种方法中的非唯一指纹都分散在小的匿名组中。

然后，让我们看看跨浏览器指纹识别的指标。除了唯一性和熵之外，我们还计算了另一个称为跨浏览器稳定性的指标，这意味着在同一台机器上不同浏览器之间稳定的指纹百分比。尽管我们选择的功能大多数时候在浏览器之间都很稳定，但不同浏览器的指纹可能仍然不同。例如，如果用户在两个浏览器中选择不同的缩放级别，Boda 等人的屏幕分辨率可能会有所不同。再举一个例子，如果一个浏览器采用硬件渲染，而另一个浏览器采用软件渲染，则我们的方法的 GPU 渲染可能会有所不同。

现在让我们看看 Boda 等人和我们的方法的跨浏览器指纹识别结果。表 II 显示，我们的方法可以识别 83.24% 的用户，而 Boda 等人的方法只能识别 68.98%。这是一个巨大的增长，相差 14.26%。跨浏览器稳定性也从 Boda 等人的 84.64% 增加到我们的方法的 91.44%。原因之一是我们使现有功能（例如屏幕分辨率和字体列表）在不同浏览器之间更加稳定。熵也从 Boda 等人的 6.88 增加到我们的方法的 7.10。

B. 按浏览器对细分

在本节的这一部分，我们将结果按表三所示的不同浏览器对进行细分。浏览器分为六种类型，还有一类称为其他，其中包括一些不常见的浏览器，如Maxthon、Coconut和UC浏览器。由于该表具有对称性质，它是一个下三角矩阵：如果我们列出所有数字，上三角与下三角完全相同。该表的主对角线表示单浏览器指纹，另一部分表示跨浏览器指纹。有两个N/A，因为Apple放弃了对Windows上Safari的支持，而Microsoft从不支持Mac OS上的Internet Explorer和Edge浏览器，即Safari不能与IE和Edge共存。其他和Edge / IE / Safari也有两个破折号，因为我们在数据集中没有观察到任何这样的对。

让我们首先看一下主对角线。单个浏览器的稳定性显然是100%，因为我们正在将浏览器与其自身进行比较。独特性最低的浏览器是 Mozilla Firefox，因为 Firefox 出于隐私原因隐藏了一些信息，例如 WebGL 渲染和供应商。IE 和 Edge 的独特性为 100%，表明这两种浏览器都具有很高的可指纹性。Opera、Safari 和其他浏览器的独特性也是 100%，但由于我们的数据集中的样本数量较少，我们无法对这些浏览器得出进一步的结论。

然后，我们查看矩阵中除主对角线以外的下三角，这显示了跨浏览器指纹的唯一性和稳定性。首先，所有对的跨浏览器稳定性都非常高（>85%），其他浏览器和 Opera 与 IE 除外。由于此类对的数量很少，我们很难生成具有合理跨浏览器稳定性的掩码。

其次，与其他浏览器相比，IE 和 Edge 与其他浏览器的独特性相对较低。原因是 IE 和 Edge 都是由微软独立实现的，开源库较少。也就是说，IE/Edge 与其他浏览器共享的部分比其他浏览器少得多。相比之下，IE 和 Edge 之间的独特性

表三：跨浏览器指纹唯一性和稳定性分析（按浏览器对划分）

浏览器	铬合金	火狐	边缘	IE	歌剧	Safari	其他
铬合金	99.2%（100%）						
火狐	89.1%（90.6%）	98.6%（100%）					
边缘	87.5%（92.6%）	97.9%（95.9%）	100%（100%）				
IE	85.1%（93.1%）	91.8%（90.7%）	100%（95.7%）	100%（100%）			
歌剧	90.9%（90.0%）	100%（89.7%）	100%（100%）	100%（60.0%）	100%（100%）		
Safari	100%（89.7%）	100%（84.8%）	不适用	不适用	100%（100%）	100%（100%）	
其他	100%（22.2%）	100%（33.3%）	-	-	100%（50%）	-	100%（100%）

注意：每个单元格的格式如下 - 唯一性（跨浏览器稳定性）。

IE 和 Edge 的稳定性非常高：100% 的独特性和 95.7% 的跨浏览器稳定性，这意味着 IE 和 Edge 可能共享大量代码。

第三，比较 IE 和 Edge 很有趣。在所有浏览器对中，Edge Browser 的独特性都高于 IE。原因是 Edge Browser 引入了更多功能，例如完全实现符合标准的 WebGL，这暴露了更多的指纹识别方面。

C. 按功能细分

在本节的这一部分，我们将结果按不同的特征进行细分，并将其显示在表 IV 中。具体来说，表 IV 可以分为两部分：第一部分位于 AmIUnique 行上方，显示 AmIUnique 采用的特征，第二部分位于第一部分下方，显示我们的方法提出的所有新特征。现在让我们看看不同的特征。

*1）屏幕分辨率及比例：*单一浏览器
屏幕分辨率和比例的熵为 7.41，而宽度和高度比例的熵则显著下降到 1.40。原因是许多分辨率，例如 1024 ×768 和 1280 ×960，比例相同。屏幕分辨率的跨浏览器稳定性很低（9.13%），因为如前所述，用户经常放大和缩小网页。宽高比的跨浏览器稳定性较高（97.57%），但低于100%，因为有些用户采用双屏，将两个浏览器放在不同的屏幕上。

*2）字体列表：*由于 Flash 的消失，
从 Flash 获得的字体列表的熵低至 2.40，而从 JavaScript 获得的列表的熵高达 10.40。这意味着字体列表仍然是一个高度可指纹化的特征，我们需要在未来使用 JavaScript 来获取该特征。

请注意，尽管 JavaScript 字体列表的熵很高，但它在我们的指纹识别中占比并不大。当我们删除此功能时，我们方法的单一浏览器唯一性仅从 99.24% 下降到 99.09%，差异不到 0.2%。也就是说，即使没有字体列表功能，我们的方法仍然可以高精度地对用户进行指纹识别。

*3）抗锯齿：*任务 (b)、(b')、(d)、(d')、(h)、(h')、(k) 和 (k') 与抗锯齿有关。当添加抗锯齿时，(b)、(d) 和 (h) 的单浏览器指纹熵会增加，但 (k) 的熵会降低。原因是 (b)、(d) 和 (h) 的边缘较少，而抗锯齿会增加更多的可指纹内容；相反，（k）在每个豆子上包含许多小边缘，并且抗锯齿将占据豆子的内容并减少豆子内部的一些可指纹内容。

现在让我们看看跨浏览器指纹识别。跨浏览器稳定性与单浏览器熵相反：对于 (b)、(d) 和 (h)，跨浏览器稳定性会降低，但对于 (k)，跨浏览器稳定性会提高。原因是同一台机器上并非所有浏览器都支持抗锯齿功能，因此 (b)、(d) 和 (h) 的稳定性会降低。出于类似的原因，由于抗锯齿功能会减少 bean 内部的一些可指纹识别内容，因此 (k) 的跨浏览器稳定性会提高。

*4）直线和曲线：*任务（d）测试线条和
曲线。由于直线和曲线都是简单的二维操作，在不同浏览器和机器上差异不大，因此熵值较低（1.09），跨浏览器稳定性较高（90.77%）。我们手动比较了不同机器或浏览器上差异较大的情况，发现主要差异在于起点和终点，那里有一两个像素的偏移。

*5）相机：*比较单个浏览器的熵
对于任务 (b) 和 (c)，我们发现添加相机时熵会降低。原因是添加相机的目的是缩小立方体，从而减少表面上的细微差异。由于 (b) 和 (c) 之间的相似性，(b) 和 (c) 的跨浏览器稳定性非常相似。

*6）纹理：*首先让我们比较一下普通、DDS、PVR、立方体-映射和浮点纹理。浮点和立方体贴图纹理的熵高于所有其他纹理，因为浮点和立方体贴图纹理具有更多信息，例如浮点纹理中的深度和立方体贴图纹理的立方体映射。PVR 纹理的熵非常低（0.14），因为 PVR 纹理主要在 Apple 移动设备（如 iPhone 和 iPad）上受支持。由于我们的数据集是从众包工作者那里收集的，因此很少有人会使用 Apple 移动设备来执行众包任务。另一个有趣的观察是 DDS 纹理的跨浏览器稳定性很低（68.18%）。原因是许多浏览器不支持 Microsoft 格式 DDS。

其次，我们来看两个纹理，即任务 (i)。与任务 (h) 相比，增加了另一层纹理，但单浏览器和跨浏览器指纹识别的熵都下降了。原因是我们任务中使用的纹理是经过精心创建的，因此它可以包含更多可指纹识别的特征。

表 IV：按特征划分的熵和跨浏览器稳定性

特征	单一浏览器	跨浏览器	
	熵	熵	稳定
用户代理	6.71	0.00	1.39%
接受	1.29	0.01	1.25%
内容编码	0.33	0.03	87.83%
内容语言	4.28	1.39	10.96%
插件列表	5.77	0.25	1.65%
已启用 Cookie	0.00	0.00	100.00%
使用本地/会话存储时区	0.03	0.00	99.57%
	3.72	3.51	100.00%
屏幕分辨率和颜色深度字体列表	7.41	3.24	9.13%
（Flash）	2.40	0.05	68.00%
HTTP 标头列表平台	3.17	0.64	9.13%
	2.22	1.25	97.91%
不追踪	0.47	0.18	82.00%
帆布	5.71	2.73	8.17%
WebGL 供应商	2.22	0.70	16.09%
WebGL 渲染器	5.70	3.92	15.39%
使用广告拦截器	0.67	0.28	70.78%
我是独一无二的	10.82	0.00	1.39%
屏幕比例	1.40	0.98	97.57%
字体列表 (JavaScript)	10.40	6.58	96.52%
AudioContext	1.87	1.02	97.48%
CPU 虚拟核心	1.92	0.59	100.00%
规范化的 WebGL 渲染器任务 (a)	4.98	4.01	37.39%
纹理	3.51	2.26	81.47%
任务（b）变化	2.59	1.76	88.25%
任务 (b') 变化+抗锯齿 任务 (c) 相	3.24	1.66	73.95%
机	2.29	1.58	88.07%
任务（d）直线和曲线	1.09	0.42	90.77%
任务 (d') (d)+抗锯齿 任务	3.59	2.20	74.88%
(e) 多模型	3.54	2.14	81.15%
任务（f）光	3.52	2.27	81.23%
任务（g）灯光与模型	3.55	2.14	80.94%
任务 (h) 镜面光 任务 (h')	4.44	3.24	80.64%
(h)+抗锯齿 任务 (h”) (h')	5.24	3.71	70.35%
+旋转 任务 (i) 两个纹理 任	4.01	2.68	75.09%
务 (j) Alpha (0.09)	4.04	2.68	75.98%
	3.41	2.36	86.25%
任务 (j) Alpha (0.10) 任务 (j)	4.11	3.02	75.31%
Alpha (0.11) 任务 (j) Alpha	3.95	2.84	75.80%
(0.39) 任务 (j) Alpha (0.40)	4.35	3.06	82.75%
任务 (j) Alpha (0.41) 任务 (j)	4.38	3.10	82.58%
Alpha (0.79) 任务 (j) Alpha	4.49	3.13	81.89%
(1) 任务 (k) 复杂灯光 任务	4.74	3.12	72.63%
(k') (k)+抗锯齿 任务 (l) 裁剪	4.38	3.07	82.75%
平面 任务 (m) 立方体贴图纹	6.07	4.19	66.37%
理 任务 (n) DDS 纹理 任务	5.79	3.96	74.45%
(o) PVR 纹理 任务 (p) 浮点纹	3.48	1.93	76.61%
理	6.03	3.93	58.94%
	4.71	3.06	68.18%
	0.14	0.00	99.16%
	5.11	3.63	74.41%
任务（q）视频	7.29	2.32	5.48%
任务 (r) 编写脚本（支持） 任务 (r)	2.87	0.51	97.91%
编写脚本（图像）	6.00	1.98	5.48%
所有跨浏览器功能 所有功能	10.92	7.10	91.44%
	10.95	0.00	1.39%

当我们将两种纹理加在一起时，其中一些特征会减弱，从而使得双纹理任务更难进行指纹识别。

*7）型号：*让我们比较一下任务（a）和（e），以及模型效果任务（f）和（g）。相比于（a）和（f），在（e）和（g）中加入了沙发模型，熵值略有增加，两个任务都增加了0.03。结论是沙发模型确实引入了更多可指纹化的特征，但增加的幅度非常有限。

*8）光照：*任务（a）、（e）、（f）、（h）和（k）与灯光。让我们首先看一下任务 (f)，其中将漫反射点光添加到任务 (a)。对于单浏览器和跨浏览器指纹识别，熵仅增加 0.01，表明漫反射点光对指纹识别影响不大。作为比较，镜面光的影响更明显，因为任务 (h) 的熵增加了>与任务 (e) 相比，在单浏览器和跨浏览器指纹识别中，平均熵为 0.9。最后，让我们看看任务 (k)，这是一个复杂的光线示例。任务 (k) 的熵是除视频之外所有任务中最高的，因为有 5,000 个模型，不同颜色的光线在所有模型之间反射并混合在一起。

*9）阿尔法：*任务 (j) 测试 alpha 值从 0.09 到 1。它有趣的是，不同的 alpha 值具有非常不同的熵。一般来说，趋势是当 alpha 值增加时，熵也会增加，但会出现许多回退。我们没有在大规模实验中测试连续的 alpha 值，而是在五台机器之间进行了小规模实验。具体来说，我们比较了每个 Alpha 值图像与标准图像之间的不同像素，发现回退主要是由软件渲染引起的，软件渲染会近似 alpha 值。此外，我们在回退中观察到了一些模式，这些模式发生在大约 0.1 的增量步骤中。

*10)剪切平面：*任务（l）是测试剪辑的效果平面，产生 3.48 个单浏览器熵和 1.93 个跨浏览器熵，稳定性为 76.61%。熵与纯纹理的熵相似，因为剪切平面是用 JavaScript 实现的，对指纹识别的贡献不大。

*11）旋转：*任务（h”）是任务（h'）的轮换。熵值降低，跨浏览器稳定性提高。原因是 Suzanne 模型的正面和沙发模型的内部有更多细节。当我们将两个模型旋转到另一个角度时，可指纹化的细节会减少，相应地稳定性会提高。

*12）音频上下文：*我们测量的 AudioContext 是跨浏览器稳定的，即目标音频设备信息和转换后的波。熵为 1.87，远小于 Englehardt 等人 [18] 测量的整个波的整个熵 5.4。

*13)视频：*任务（q）正在测试视频特征。熵视频的熵在所有渲染任务中最高（7.29），因为解码视频是浏览器、驱动程序，有时还有硬件的组合。相比之下，视频的跨浏览器稳定性非常低（5.48%），熵也下降到 2.32。原因是与图像编码和解码类似，WebM 和 MP4 视频格式都是有损的，并由浏览器解码。我们没有找到与图像一样的通用无损视频格式。

*14）编写脚本：*编写脚本在任务 (r) 中进行测试。为了进行跨浏览器指纹识别，我们进一步将任务（r）分为两部分。第一部分，我们称之为编写脚本（支持），仅包含某些编写脚本是否受支持的信息，即一个由 0 和 1 组成的列表，其中 1 表示支持，0 表示不支持。如前所述，我们通过框检测获取信息。第二部分，我们称之为编写脚本（图像），是在客户端渲染的图像。编写脚本（图像）的单浏览器熵比编写脚本（支持）的熵大 3.13。也就是说，图像确实包含比编写脚本是否受支持更多的信息。编写脚本（支持）的跨浏览器稳定性是根据应用我们的掩码后的结果计算出来的，因为有些编写脚本是随浏览器附带的，并不是跨浏览器稳定的。相应地，编写脚本（支持）的跨浏览器熵低于单浏览器熵。

*15）CPU虚拟核心：*虚拟CPU数量 cores 仅根据 HardwareConcurrency 值计算得出（如果不支持，则值为“未定义”），单浏览器指纹的熵为 1.92。我们预计未来熵会增加，因为就在我们提交之前，Firefox 48 开始支持新功能。跨浏览器稳定性为 100%，因为我们可以检测浏览器是否支持 HardwareConcurrency 并应用自定义掩码。由于数据大小，跨浏览器熵与单浏览器熵不同，两者的归一化熵非常相似。

*16）规范化的 WebGL 渲染器：*WebGL 渲染器无法跨浏览器指纹识别，部分原因是不同浏览器提供的信息水平不同。我们从不同的浏览器中提取出共同的信息，并将信息对齐为标准格式。与熵值为 5.70 的原始 WebGL 渲染器相比，标准化后的 WebGL 渲染器的熵为 4.98。下降的原因是提取将丢弃一些信息（例如 Chrome），以与其他浏览器（例如 Edge 浏览器）对齐。相应地，跨浏览器稳定性从原始 WebGL 渲染器的 15.39% 提高到标准化后的 37.39%。

这里有两点需要注意。首先，WebGL 供应商提供的信息并不比 WebGL 渲染器多。也就是说，当我们将两个值组合在一起时，得到的熵就是 WebGL 渲染器的熵。其次，我们的 GPU 任务比 WebGL 供应商和渲染器提供的信息多得多。有些浏览器，例如 Firefox，不提供 WebGL 供应商和渲染器信息，这给了我们很大的空间来填补这个空白。此外，即使浏览器提供了这些信息，我们的 GPU 任务组合在一起的熵也是 7.10，比 WebGL 渲染提供的 5.70 熵大得多。原因是渲染是软件和硬件的结合，而 WebGL 渲染器只提供硬件渲染的硬件信息。

D. 观察结果

在我们的实验和实施过程中，我们观察到了一些有趣的事实，并在本小节中展示了它们：

观察 1：我们的指纹特征高度可靠，即删除一个单一特征对指纹识别结果影响不大。

在这一部分中，我们展示了从 AmIUnique 和我们的方法中删除单个特征的影响，然后测量两者的独特性。结果表明，当删除表 IV 中的任何单个特征（包括 AmIUnique 中的所有旧特征和我们的新特征）时，我们的指纹的独特性仍然高于 99%。相比之下，如果删除以下六个属性中的任何一个，AmIUnique 的独特性就会降至 84% 以下，即用户代理、时区、插件列表、内容语言、HTTP 标头列表以及屏幕分辨率和颜色深度。总之，就使用的特征而言，我们的方法比 AmIUnique 更可靠。

观察2：软件渲染也可以用于指纹识别。

对于 WebGL 的一个普遍理解是，软件渲染可能会消除显卡造成的所有差异。然而，我们的实验表明，即使是软件渲染也可以用于指纹识别。具体来说，我们选择了所有由 SwiftShader 渲染 WebGL 的数据，SwiftShader 是 Google 发明的开源软件渲染器，Chrome 在硬件渲染不可用时会使用它。我们计算一个特殊的指纹，仅包含我们所有的 GPU 渲染任务，即任务 (a)-(p)，不包括编写脚本和视频。

由于硬件渲染的采用率很高，我们仅使用 SwiftShader 收集了 88 个案例，并找到了 11 个不同的 GPU 指纹，其中 7 个是独一无二的。软件渲染的独 特性肯定比硬件渲染低得多，但仍然不是零。也就是说，我们在采用软件渲染时需要小心谨慎，以减轻基于 WebGL 的指纹识别。

观察 3：WebGL 渲染是软件和硬件的结合，其中硬件的贡献大于软件。

在这次观察中，我们考察了与软件渲染相比的另一个极端，即 Microsoft Basic Rendering。Microsoft Basic Rendering 为各种显卡提供了通用驱动程序，也就是说，使用 Microsoft Basic Rendering 将最大限度地减少软件驱动程序的影响，并显示硬件带来的影响。与软件渲染的实验类似，我们选择使用 Microsoft Basic Rendering 的渲染并计算指纹。

由于软件渲染中存在类似的原因，我们仅收集了 32 个使用 Microsoft Basic Rendering 的案例，并找到了 18 个不同的 GPU 指纹，其中包含 15 个唯一值。Microsoft Basic Rendering 的唯一性低于使用普通显卡驱动程序的唯一性，这意味着 WebGL 由软件和硬件同时渲染。同时，我们认为硬件的贡献更大，因为 Microsoft Basic Rendering 的唯一性高于软件渲染器的唯一性。

观察4：DataURL 的实现在不同的浏览器有所不同。

在本观察中，我们研究了 DataURL，这是先前指纹识别中用于表示图像的常见格式。令人惊讶的是，我们发现 DataURL 在浏览器中的实现方式非常不同，即，如果我们将图像转换为 DataURL，则表示形式在不同的浏览器之间会有很大差异。这对于单浏览器指纹识别来说是个好消息，但对于跨浏览器来说却是个坏消息。如表 IV 所示，Canvas 的跨浏览器率非常低（8.17%），因为我们采用了 AmIUnique 的代码，其中 DataURL 用于存储图像。

观察 5：渲染结果之间的一些差异非常细微，即只有一个或两个像素的差异。

在最后的观察中，我们手动比较了渲染结果之间的差异，发现虽然有些差异很大，尤其是在软件和硬件渲染之间，但有些差异非常细微，尤其是当两张显卡彼此相似时。例如，iMac 和另一台 Mac Pro 渲染的 Suzanne 模型在纹理上仅相差一个像素，如果我们旋转模型，差异就会消失。

VII. D防御磷罗波塞德弗印记

在本节中，我们将讨论如何防御我们提出的浏览器指纹识别。我们将首先从现有的防御措施，即著名的 Tor 浏览器开始，然后介绍我们防御措施的一些设想。

Tor 浏览器规范化了许多浏览器输出，以减轻现有的浏览器指纹识别。也就是说，许多功能在 Tor 浏览器中不可用 - 根据我们的测试，只有以下功能（尤其是我们新提出的功能）仍然存在，其中包括屏幕宽度和高度比以及音频上下文信息（例如，采样率和最大通道数）。我们相信 Tor 浏览器很容易规范化这些剩余的输出。

另外值得一提的是，Tor 浏览器默认禁用画布，并会要求用户允许使用画布。如果用户确实允许使用画布，她仍然可以被指纹识别。Tor 浏览器文档还提到了一种尚未实现的软件渲染解决方案，但如第 VI-D 节所述，软件渲染的输出在同一浏览器中也存在很大差异。我们仍然相信这是可行的方法，但需要更仔细的分析才能涵盖所有软件渲染库。

总的来说，防御浏览器指纹识别的理念可以概括为虚拟化，我们需要找到一个正确的虚拟化层。考虑一个极端的解决方案，即浏览器在虚拟机中运行——虚拟机中的一切都是标准化的，浏览器的输出在不同的物理机器上是相同的。然而，缺点是机器虚拟化是重量级的。Tor 浏览器是另一个极端——一切都被虚拟化为浏览器的一部分。这种方法是轻量级的，但我们需要找到所有可能被指纹识别的地方，比如画布和音频上下文：如果一个地方缺失，浏览器仍然可能以某种方式被指纹识别。我们把探索正确的虚拟化层作为我们未来的工作。

八、D讨论埃泰国科学技术信息中心我问题

我们已经与所在机构的机构审查委员会 (IRB) 讨论了伦理问题，并获得了 IRB 的批准。具体来说，虽然网络跟踪可用于获取隐私信息，但我们从众包工作者那里获得的标识符（例如计算机显卡的行为）本身并不隐私。只有当标识符与隐私信息（例如浏览历史）相关联时，这种组合才被视为隐私。然而，这一步超出了研究范围。我们的调查部分，即附录 A 中关于多种浏览器使用情况统计的研究，包含用户的浏览习惯。为了确保隐私，调查是匿名的，我们不存储来自 MicroWorkers 的用户 ID。

IX. R兴高采烈西兽人

在本节中，我们讨论现有网络跟踪和反跟踪技术的相关工作。

A. 网络跟踪技术和测量

我们首先讨论第一代跟踪，即基于 cookie 或超级 cookie，然后讨论第二代跟踪，即浏览器指纹跟踪。

*1）基于 Cookie 或超级 Cookie 的跟踪：*有很多现有工作侧重于基于 cookie 或超级 cookie 的网络跟踪技术的测量或研究。Mayer 等人 [28] 和 Sanchez 等人 [40] 对第三方跟踪进行了全面的讨论，包括跟踪技术、商业模式、防御选择和政策辩论。Roesner 等人的另一项重要测量工作提出了一个全面的分类框架，用于部署在现实世界网站中的不同网络跟踪 [39]。Lerner 等人从 1996 年到 2016 年对网络跟踪进行了考古研究，包括基于 cookie 和超级 cookie 以及浏览器指纹识别 [27]。Soltani 等人和 Ayenson 等人测量了非基于 cookie 的状态跟踪的流行程度，并展示了跟踪公司如何使用多个客户端状态来重新生成已删除的标识符 [11, 41]。Metwalley 等人 [30] 提出了一种无监督的网络跟踪测量方法。除了跟踪行为和技术之外，Krishnamurthy 等人[22–25]重点关注网络跟踪造成的危害风险，表明不仅用户的浏览历史记录，而且其他敏感个人信息（例如姓名和电子邮件）都可能泄露。

*2）浏览器指纹识别：*现在让我们讨论一下浏览器的 gerprinting，即第二代网络跟踪。我们首先讨论现有的测量研究。Yen 等人和 Nikiforakis 等人讨论了现有指纹识别工具中使用的不同第二代跟踪技术及其在工作中的有效性 [36, 46]。Acar 等人 [9] 对三种先进的网络跟踪机制进行了大规模研究，一种是第二代网络跟踪，即画布指纹识别，另外两种是第一代网络跟踪，即 evercookies 和结合使用“cookie 同步”与 evercookies。Fifield 等人 [20] 专注于第二代网络跟踪的一个特定指标，即字体。FPDetective [10] 通过使用其框架专注于字体检测，对数百万个最受欢迎的网站进行了大规模研究。Englehardt 等人 [18] 还对 1

数百万个网站，发现了许多新的指纹特征，例如 AudioContext。我们在论文第二部分中也使用了他们新发现的指纹特征作为先前特征的一部分。

现在让我们来谈谈浏览器指纹识别的工作。Mowery 等人 [32] 可能是最早提出基于画布的指纹识别的工作之一。其他一些工作 [31、33] 专注于浏览器 JavaScript 引擎的指纹识别。Nakibly 等人 [34] 的立场文件提出了几种基于硬件的跟踪，包括麦克风、运动传感器和 GPU。他们的 GPU 跟踪仅包括基于时间的功能，不如本文中的技术可靠。Laperdrix 等人 [26]，即 AmIUnique，对具有 17 个属性的浏览器指纹识别进行了最广泛的研究，我们在整篇论文中与它们进行了比较。Boda 等人 [14] 试图实现跨浏览器跟踪，但他们的功能是来自包括 IP 地址在内的单浏览器跟踪的旧功能。如上所述，当机器使用 DHCP、位于 NAT 后面或移动到新位置（如笔记本电脑）时，IP 地址是不可靠的。

与现有方法相比，我们的方法在操作系统和硬件层面引入了许多新功能。例如，我们引入了许多 GPU 功能，如纹理、变量、灯光和模型。再例如，我们还引入了一个侧通道来检测已安装的编写脚本和 AudioContext 中的一些新信息。所有这些新功能都有助于我们实现高指纹唯一性和跨浏览器稳定性。

B. 现有的反追踪机制

我们首先讨论针对第一代跟踪的现有反跟踪措施，然后再讨论第二代跟踪。

1）基于Cookie或Super-cookie的反跟踪技术： Roesner 等人 [39] 提出了一种名为 Share-MeNot 的工具，用于防御社交媒体按钮跟踪，例如 Facebook 的“点赞”按钮。隐私浏览模式 [44，45] 通过单独的用户配置文件将正常浏览与隐私浏览隔离开来。类似地，TrackingFree [37] 采用基于配置文件的隔离，并提出了用于配置文件创建的入度有界图。“禁止跟踪”（DNT）[43] 标头是一种选择退出方法，需要跟踪器遵从性。如先前的研究 [28，39] 所示，DNT 无法有效保护用户免受现实世界中的跟踪。用户还可以禁用大多数浏览器支持的第三方 cookie，以避免基于 cookie 的跟踪。Meng 等人 [29] 设计了一项策略并授权用户控制是否被跟踪，但他们必须依赖现有的反跟踪技术。

上述所有研究都侧重于基于 cookie 或超级 cookie 的网络跟踪，可以完全或部分阻止此类跟踪。它们都无法阻止本文提出的指纹识别，因为本文提出的指纹识别属于第二代，不需要服务器端有状态的标识符。

2）针对浏览器指纹的反跟踪： Tor 浏览器 [38] 可以成功防御许多浏览器指纹识别技术，包括我们论文中提出的功能。有关更多详细信息，请参阅第七节。除了 Tor 浏览器中提出的规范化技术外，PriVaricator [35] 还在可指纹识别的输出中添加了随机噪声。由于 PriVaricator 不是开源的，我们无法测试我们的指纹识别以抵御他们的防御。

X.C.结论

总之，我们提出了一种新颖的浏览器指纹识别方法，它不仅可以识别使用一个浏览器的用户，还可以识别在同一台机器上使用不同浏览器的用户。我们的方法采用了操作系统和硬件级别的功能，包括 WebGL 公开的显卡、Audio-Context 的音频堆栈和 hardwareConcurrency 的 CPU。我们的评估表明，对于单浏览器指纹识别，我们的方法可以唯一地识别比 AmIUnique 更多的用户，对于跨浏览器指纹识别，我们的方法可以唯一地识别比 Boda 等人更多的用户。我们的方法非常可靠，即删除任何单个特征只会使准确度最多降低 0.3%。

一个致谢

作者要感谢匿名审稿人的深思熟虑的评论。这项工作部分由美国国家科学基金会 (NSF) 资助，资助编号为 CNS-1646662 和 CNS-1563843。本文中的观点和结论均为作者的观点和结论，不应被解释为必然代表 NSF 的官方政策或认可（无论是明示的还是暗示的）。

R参考文献

[1] 核心估计器。https://github.com/oftn-oswg/core-estimator。

[2] [电子邮件线程] 提案：navigator.cores。https://lists.w3.org/Archives/Public/public-whatwg-archive/2014May/0062.html。

[3] [github] 我是独一无二的吗？https://github.com/DIVERSIFYproject/amiunique。

[4] [graphics wikia] 抗锯齿。http://graphics.wikia.com/wiki/Anti-Aliasing。

[5] Panopticlick：你的浏览器是否可以安全抵御追踪？https://panopticlick.eff.org/。

[6] 关注：华尔街日报隐私报告。http://www.wsj.com/public/page/what-they-know-digital-privacy.html。

[7] [维基百科]立方体映射。https://en.wikipedia.org/wiki/Cube map。 _

[8] [维基百科] 书写系统列表。https://en.wikipedia.org/wiki/书写系统列 _ 表。 _ _

[9] G. Acar、C. Eubank、S. Englehardt、M. Juarez、A. Narayanan 和 C. Diaz，“网络永远不会忘记：野外持久追踪机制”，*2014 年 ACM SIGSAC 计算机和通信安全会议论文集*，爵士。 CCS '14，2014 年，第 674–689 页。

[10] G. Acar、M. Juarez、N. Nikiforakis、C. Diaz、S. Gürses、F. Piessens 和 B. Preneel，《FPDetective：为指纹识别者扫清网络障碍》，*2013 年 ACM SIGSAC 计算机和通信安全会议论文集*，爵士。 CCS '13，2013 年，第 1129–1140 页。

[11] M. Ayenson、D. Wambach、A. Soltani、N. Good 和 C. Hoofnagle，“Flash cookies 和隐私 ii：现在有了 html5 和 etag 重生”，*可在 SSRN 1898390 获得*，2011年。

[12] S. Berger. 你应该安装两个浏览器。http://www.compukiss.com/internet-and-security/you-should-install-two-browsers.html。

[13] T. Bigelajzen。跨浏览器缩放和像素比率检测器。https://github.com/tombigel/detect-zoom。

[14] K. Boda、AM Földes、GG Gulyás 和 S. Imre，“通过跨浏览器指纹识别在网络上进行用户跟踪”，载于*第十六届北欧信息安全技术应用会议论文集*，爵士。 NordSec'11，2012 年，第 31-46 页。

[15] F. Boesch。软阴影映射。http://codeflow.org/entries/2013/feb/15/soft-shadow-mapping/。

表五：浏览器使用情况统计

单身的浏览器	> 2 浏览器	> 3 浏览器	铬合金 & 火狐	铬合金 & 微软 IE/Edge
30%	70%	13%	33%	20%

[16] FT 委员会。跨设备追踪。https://www.ftc.gov/newsevents/events-calendar/2015/11/cross-device-tracking。

[17] P. Eckersley，“你的网络浏览器有多独特？”*第十届隐私增强技术国际会议论文集*, 系列。PETS'10，2010 年。

[18] S. Englehardt 和 A. Narayanan，“在线追踪：百万个站点的测量与分析”，*第 22 届 ACM SIGSAC 计算机和通信安全会议论文集*，系列 CCS '16，2016 年。

[19] A. Etienne 和 J. Etienne。Blender 中的经典 Suzanne Monkey，让你的游戏从 Threex 开始。Suzanne。http://learningthreejs.com/blog/2014/05/09/classical-suzanne-monkeyfrom-blender-to-get-your-game-started-with-threex-dot-suzanne/。

[20] D. Fifield 和 S. Egelman，“通过字体指标识别网络用户”，*金融密码与数据安全*。施普林格，2015 年，第 107-124 页。

[21] S.卡姆卡。埃弗饼干。 http://samy.pl/evercookie/。

[22] B. Krishnamurthy、K. Naryshkin 和 C. Wills，“隐私泄露与保护措施：日益加剧的脱节”，*Web 2.0 安全与隐私研讨会*，2011年。

[23] B. Krishnamurthy 和 C. Wills，“网络上的隐私传播：纵向视角”，*第 18 届万维网国际会议论文集*.ACM，2009 年，第 541-550 页。

[24] B. Krishnamurthy 和 CE Wills，“在互联网上留下隐私足迹”，*第六届 ACM SIGCOMM 互联网测量会议论文集*.ACM，2006，第 65-70 页。

[25] ——“在线社交网络中隐私的特征”，*第一次在线社交网络研讨会论文集*.ACM，2008 年，第 37-42 页。

[26] P. Laperdrix、W. Rudametkin 和 B. Baudry，“美女与野兽：利用现代网络浏览器构建独特的浏览器指纹”，在*第 37 届 IEEE 安全与隐私研讨会 (S&P 2016)*，2016 年。

[27] A. Lerner、AK Simpson、T. Kohno 和 F. Roesner，“互联网琼斯和失踪追踪者的袭击者：1996 年至 2016 年网络追踪的考古学研究”，*第 25 届 USENIX 安全研讨会 (USENIX Security 16)*，德克萨斯州奥斯汀，2016 年。

[28] JR Mayer 和 JC Mitchell，“第三方网络跟踪：政策和技术”，*安全与隐私（SP），2012 IEEE 研讨会*.IEEE，2012 年，第 413-427 页。

[29] W. Meng、B. Lee、X. Xing 和 W. Lee，“Trackmeornot：实现对网络跟踪的灵活控制”，*第 25 届万维网国际会议论文集*，爵士。 WWW '16，2016 年，第 99–109 页。

[30] H. Metwalley 和 S. Traverso，“网络跟踪器的无监督检测”，*全球通讯*，2015 年。

[31] K. Mowery、D. Bogenreif、S. Yilek 和 H. Shacham，“JavaScript 实现中的指纹信息”，2011 年。

[32] K. Mowery 和 H. Shacham，“像素完美：html5 中的指纹画布”，2012 年。

[33] M. Mulazzani、P. Reschl、M. Huber、M. Leithner、S. Schrittwieser、E. Weippl 和 F. Wien，“使用 JavaScript 引擎指纹识别快速可靠地识别浏览器”，*W2SP*，2013 年。

[34] G. Nakibly、G. Shelef 和 S. Yudilevich，“使用 html5 进行硬件指纹识别”，*arXiv 预印本 arXiv:1503.01408*，2015 年。

[35] N. Nikiforakis、W. Joosen 和 B. Livshits，《私掠者：用善意的谎言欺骗指纹识别者》*第 24 届万维网国际会议论文集*，爵士。 WWW '15，2015 年，第 820–830 页。

[36] N. Nikiforakis、A. Kapravelos、W. Joosen、C. Kruegel、F. Piessens 和 G. Vigna，“无 Cookie 怪物：探索基于 Web 的设备指纹识别”，*IEEE 安全与隐私研讨会*，2013 年。

[37] X. Pan、Y. Cao 和 Y. Chen，“我不知道你去年夏天访问了什么——使用 trackingfree 浏览器保护用户免受第三方网络跟踪”，*国家发展战略支持系统*，2015 年。

[38] M. Perry、E. Clark 和 S. Murdoch，“Tor 浏览器的设计和实施 [草案] [在线]，美国”，2015 年。

[39] F. Roesner、T. Kohno 和 D. Wetherall，“检测和防御防止网络上的第三方跟踪”，*第九届 USENIX 网络系统设计与实现会议论文集*
化，爵士。 NSDI'12，2012 年，第 12-12 页。

[40] I. Sánchez-Rola、X. Ugarte-Pedrero、I. Santos 和 PG Bringas，“像没有明天一样跟踪用户：当前互联网上的隐私”，*国际联合会议*。施普林格，2015 年，第 473-483 页。

[41] A. Soltani、S. Canty、Q. Mayo、L. Thomas 和 CJ Hoofnagle，“Flash cookies 和隐私”，*AAAI 春季研讨会：智能信息隐私管理*，2010 年。

[42] US-CERT。保护您的网络浏览器。https://www.us-cert.gov/publications/securing-your-web-browser。

[43] 维基百科。“不跟踪政策”。http://en.wikipedia.org/wiki/Do Not Track _ Policy。 _

[44] ——. 隐私模式。http://en.wikipedia.org/wiki/Privacy mode。

[45] M. Xu, Y. Jang, X. Xing, T. Kim 和 W. Lee，“Ucognito：无需流泪的隐私浏览”，*第 22 届 ACM SIGSAC 计算机和通信安全会议论文集*，爵士。 CCS '15，2015 年，第 438–449 页。

[46] T.-F. Yen、Y. Xie、F. Yu、RP Yu 和 M. Abadi，“网络上的主机指纹识别和跟踪：隐私和安全影响”，*NDSS 会议纪要*，2012 年。

一个附录一个
年代调查磷欧普 ‘年代乌圣人米多重乙划船者

在本附录中，我们研究了在同一台机器上使用多种浏览器的人的统计数据。请注意，这是一项小规模、独立于本文所有其他设计和实验的研究。我们进行这项研究是为了加强本文的动机。我们的结果表明，人们确实在同一台机器上使用多种浏览器，并且花费了相当多的时间。

现在让我们介绍一下我们在众包网站 MicroWorkers 上的实验设置。我们进行一项开放式调查，询问调查对象他们拥有并通常使用哪种浏览器，以及他们在每个浏览器上花费的时间占比。他们可以自由地在多行文本框中写入任何内容。

以下是我们的实验结果。我们收集了 102 个答案，其中一个答案只是复制了我们的调查链接，另一个答案提到了一个不存在的浏览器。排除这两个无效答案后，我们总共有 100 个。95% 的受访用户安装了两个以上的浏览器，因为 IE 或 Edge 是默认安装的。我们进一步统计了他们定期使用两个或两个以上浏览器的百分比，即他们在其中一个浏览器上花费的时间至少超过 5%。

表五显示了人们使用浏览器的情况。70%的受访者经常使用两种或两种以上的浏览器，只有30%的人使用单一浏览器。调查答案中的浏览器类型包括Chrome、Firefox、IE、Edge、Safari、椰子浏览器和傲游。结果表明，人们确实使用多种浏览器，跨浏览器指纹识别非常重要且必要。