（跨）浏览器指纹通过OS和硬件级功能

曹银芝
里海大学
yinzhi.cao@lehigh.edu

宋丽
里海大学
sol315@lehigh.edu

埃里克·维曼斯（Erik Wijmans）†
圣路易斯华盛顿大学
erikwijmans@wustl.edu

抽象的—在本文中，我们提出了浏览器指纹
该技术不仅可以跟踪单个浏览器中的用户，还可以跟踪同一台计算机上不同浏览器中的用户。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别的功能，例如来自图形卡，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地识别99.24％的用户，而针对同一数据集的单浏览器指纹识别技术的最新水平是90.84％。此外，与具有类似稳定性的文献中唯一的跨浏览器方法相比，我们的方法可以获得更高的唯一性率。

我 简介

Web跟踪是一种可用于记住和识别过往网站访问者的有争议的技术。一方面，Web跟踪可以对用户进行身份验证-特别是可以将不同Web跟踪技术的组合用于多因素身份验证以增强安全性。另一方面，Web跟踪也可以用于提供个性化服务-如果该服务是不受欢迎的，例如某些不需要的，有针对性的广告，则这种跟踪会侵犯隐私。无论我们是喜欢网络跟踪还是在当前网络中合法使用它，超过90％的Alexa排名前500的网站[39]都采用了网络跟踪，并引起了公众和媒体的广泛关注[6]。

网络跟踪一直在迅速发展。第一代跟踪技术采用有状态的服务器设置标识符，例如cookie和evercookie [21]。此后，出现了第二代跟踪技术，即指纹识别，从有状态标识符变为无状态，

也就是说，第二代技术没有设置新的标识符，而是探索了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恢复丢失的Cookie。第一代和第二代跟踪都被限制在一个浏览器中，如今人们正在开发第三代跟踪技术，以尝试实现跨设备跟踪[16]。

本文的重点是在第二个和第三个之间的2.5代技术，该技术不仅可以在同一浏览器中而且可以在同一台机器上的不同浏览器中为用户提供指纹。使用多种浏览器的做法很普遍，并受到US-CERT [42]和其他技术人员[12]的提倡：根据我们的调查，[1个] 70％的受研究用户已在同一台计算机上安装并定期使用至少两个浏览器。

从积极的方面来看，提出的2.5代技术甚至可以用作跨浏览器的更强大的多因素用户认证的一部分。从另一个角度来看，正如许多有关新网络攻击的现有研究工作一样，提议的2.5代跟踪也可以帮助改善现有的隐私保护工作，我们将在第七节中简要讨论跨浏览器跟踪的防御。

现在，让我们撇开网络跟踪的优缺点和丑陋用法，看看技术本身。要对安装在同一台计算机上的不同浏览器进行指纹识别，一种简单的方法是使用可对单个浏览器进行指纹识别的现有功能。由于许多现有功能都是特定于浏览器的，因此跨浏览器稳定的功能即使结合在一起进行指纹打印也不够独特。这就是Boda等人唯一的跨浏览器指纹打印工作的原因。[14]，采用IP地址为主要特征。但是，在著名的Panopticlick测试[5]和许多其他相关工作[10、20、26、32、34、36]中，现代网络浏览器指纹排除了IP地址作为网络级功能。原因是IP地址如果动态分配，通过移动网络连接，

在本文中，我们基于多种新颖的操作系统和硬件级别功能（例如，来自图形卡，CPU，音频堆栈和已安装的编写脚本的功能），提出了（跨）浏览器指纹。具体来说，由于许多此类操作系统和硬件级别的功能都是通过浏览器API公开给JavaScript的，因此当要求浏览器通过这些API执行某些任务时，我们可以提取功能。提取的特征可用于单浏览器和跨浏览器的指纹打印。

† 作者在利哈伊大学（Lehigh University）的REU学生时期为论文做出了贡献。

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 Internet Society，ISBN 1-1891562-46-0 http://dx.doi.org/10.14722/ndss.2017.23152

1个 有关我们实验的更多详细信息，请参见附录A。

让我们以WebGL为例，它是在浏览器画布对象中实现的3D组件。尽管画布（尤其是2D部件）已用于单浏览器指纹打印[9，32]，但最近一项名为AmIUnique的研究[26]实际上将WebGL视为“太脆且不可靠”，即使对于单个浏览器也是如此。。得出这样的结论的原因是AmIUnique选择了randomWebGL任务，并且没有限制很多影响指纹打印结果的变量，例如画布大小和抗锯齿。

与AmIUnique得出的结论相反，我们表明WebGL不仅可以用于单浏览器，而且还可以用于跨浏览器的指纹打印。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 *首先* 在单浏览器和跨浏览器指纹打印中使用许多新颖的OS和硬件功能，尤其是计算机图形功能。特别是，我们采用新功能的方法可以成功为99.24％的用户显示指纹，而AmIUnique则为90.84％，

即最新技术，在同一数据集上进行单浏览器指纹打印。此外，我们的方法可以实现83.24％的唯一性，并具有91.44％的跨浏览器稳定性，而Boda等人则认为。[14]不包括IP地址的唯一性只有68.98％，跨浏览器的稳定性为84.64％。

我们的第二个贡献是，我们对单浏览器和跨浏览器的指纹打印做了几个有趣的观察。例如，我们发现当前屏幕分辨率的测量，例如在AmIUnique，Panopticlick [5，17]和Boda等人中进行的测量。[14]是不稳定的，因为当用户放大或缩小网页时，分辨率在Firefox和IE中会发生变化。因此，我们考虑了缩放级别，并以屏幕分辨率归一化了宽度和高度。再举一个例子，我们发现DataURL和JPEG格式在不同的浏览器中都是不稳定的，因为这些格式会丢失，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打印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上找到。com / Song-Li / cross browser /，可以在http://www.uniquemachine.org上找到有效的演示。

本文的其余部分安排如下。我们首先介绍所有功能，包括在AmIUnique中采用和修改的旧功能，以及我们在第二节中提出的新功能。然后，在第三节中介绍浏览器指纹的设计，包括总体体系结构，渲染任务和蒙版生成。之后，我们在第四节中讨论我们的实现，在第五节中讨论数据收集。我们评估我们的方法，并在第六节中介绍结果。接下来，我们将在第七节中讨论指纹的辩护，在第八节中讨论一些道德问题，并在第IX节中讨论相关工作。我们的论文在第十节结束。

二。F 不可印证 F EATURES

在本节中，我们介绍本文使用的可指纹识别功能。我们从先前作品中使用的功能开始，然后

然后介绍一些需要修改的功能，尤其是跨浏览器指纹打印时。接下来，我们介绍我们新提出的功能。

尽管对单浏览器指纹的功能没有任何限制，但我们的跨浏览器功能需要反映浏览器下方级别（即操作系统和硬件级别）的信息和操作。例如，顶点着色器和片段着色器都可以在操作系统中显示GPU及其驱动程序的行为。虚拟内核的数量是CPU的功能；安装的编写脚本是操作系统级别的功能。原因是操作系统和硬件级别的这些功能在各个浏览器中相对更稳定：所有浏览器都在相同的操作系统和硬件上运行。

请注意，如果某个操作（尤其是该操作的输出）是由浏览器和基础（操作系统和硬件）级别共同贡献的，我们可以将其用于单浏览器指纹打印，但需要摆脱浏览器因素在跨浏览器指纹打印中。例如，当我们在立方体上将图像作为纹理渲染时，纹理映射是GPU操作，而图像解码是浏览器。因此，我们只能使用无损格式PNG进行跨浏览器指纹打印。再举一个例子，音频信号的动态压缩操作是由浏览器和基础音频堆栈共同执行的，我们需要提取基础特征。现在让我们介绍本文中使用的这些功能。

A.先前的指纹功能

在本节的这一部分中，我们介绍了我们从最先进的技术中采用的可指纹识别功能。AmIUnique论文的表I中提出了17种功能[26]，而我们所有这些功能都用于单浏览器指纹打印。可以在他们的论文中找到更详细的信息。由于许多此类功能都是浏览器特定的，因此我们采用了具有4种功能的子集进行跨浏览器指纹打印，即屏幕分辨率，颜色深度，字体列表和平台。其中一些功能需要修改，下面介绍。

B.具有重大修改的旧功能

屏幕分辨率的一项先验功能需要针对单浏览器和跨浏览器的指纹进行重构。然后，我们介绍另一个可指纹识别的功能，即CPU虚拟内核的数量。最后，两个现有功能需要对跨浏览器指纹打印进行重大修改。

屏幕分辨率。 当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，则屏幕分辨率不正确。我们认为，在单浏览器指纹和跨浏览器指纹中都需要考虑缩放级别。

具体来说，我们遵循两个单独的方向。首先，我们基于div标签的大小和设备像素比率，采用有关缩放级别检测的现有工作[13]，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如发明人所承认的那样可靠，因此我们采用了一个新功能，即

屏幕的宽度和高度，不会随缩放级别而变化。

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的指纹打印中都非常有用。前四个代表浏览器的可用屏幕，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，以及屏幕是否上下颠倒。

CPU虚拟核心数。 核心编号可以通过称为hardwareConcurrency的新浏览器功能获得，该功能为Web Workers提供功能信息。现在，许多浏览器都支持这种功能，但是某些浏览器（尤其是早期版本的浏览器）不支持。如果不支持，则存在侧通道[1]以获取编号。具体来说，当增加网络工作者的数量时，可以监视有效负载的完成时间。当完成时间在Web工作者的特定级别显着增加时，硬件并发达到了极限，这对于指纹打印核心数量很有用。请注意，某些浏览器（例如Safari）会将Web Worker的可用内核数量减少一半，而我们需要将跨浏览器指纹打印的数量增加一倍。

发明人知道内核的数量是可以打印的[2]，这是他们称其为hardwareConcurrency而不是内核的原因之一。但是，该功能从未在浏览器指纹打印的现有技术中使用或测量过。

AudioContext。 AudioContext借助OS和声卡中的音频堆栈，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，现有的指纹打印工作[18]使用OscillatorNode生成三角波，然后将波馈送到DynamicsCompressorNode，DynamicsCompressorNode是一种信号处理模块，可抑制大声或放大安静的声音，

即产生压缩效果。然后，经过处理的音频信号通过AnalyserNode转换到频域。

频域中的波动在同一台计算机上从一个浏览器到另一个浏览器是不同的。但是，我们发现峰值和它们对应的频率在浏览器中相对稳定。因此，我们创建了一个在频率轴和值轴上都具有小步长的bin列表，并将峰值频率和峰值映射到相应的bin。如果一个bin包含一个频率或值，则将bin标记为1，否则将其标记为0：此类bin列表用作我们的跨浏览器功能。

除波形处理外，我们还从目标音频设备获得以下信息：采样率，最大通道数，输入数量，输出数量，通道数，通道计数模式和通道解释。请注意，就我们所知，现有指纹技术都没有将这种音频设备信息用于浏览器指纹。

字体列表。 AmIUnique中的测量基于Flash插件，但是Flash消失得非常快，他们的论文中也提到并认可了Flash。在我们进行实验时，Flash已几乎不受支持

获取字体列表。相反，我们采用了Nikiforakis等人提到的旁道方法。[36]，其中测量某个字符串的宽度和高度以确定字体类型。请注意，并非所有字体都是跨浏览器可打印的字体，因为某些字体是Web特定的，并且由浏览器提供，并且我们需要应用第III-C节中所示的掩码来选择子集。另一点值得注意的是，我们知道Fifield等人。[20]提供了43种字体的子集用于指纹识别，但是它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.新提议的原子指纹特征

在本小节和下一个小节中，我们将介绍我们新提出的指纹可打印功能。我们首先从原子特征开始，从原子上说，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，这些功能通常需要多个API和组件进行协作。

线，曲线和抗锯齿。 线和曲线是Canvas（2D零件）和WebGL都支持的2D功能。抗锯齿是一种计算机图形技术，用于通过平滑单线/曲线对象或计算机图形模型边缘中的锯齿（即锯齿状或阶梯状线）来减少锯齿。现有许多用于抗锯齿的算法[4]，例如第一原理方法，信号处理方法和mipmapping，它们使抗锯齿指纹成为可能。

顶点着色器。 由GPU和驱动程序渲染的顶点着色器将3D模型中的每个顶点转换为2D剪贴空间中的坐标。在WebGL中，顶点着色器可以三种方式接受数据：来自缓冲区的属性，始终保持不变的制服以及来自片段着色器的纹理。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。 同样由GPU和驱动程序渲染的片段着色器将片段（例如由栅格化输出的三角形）处理为一组颜色和一个深度值。在WebGL中，片段着色器通过以下方式获取数据：

- *制服* 在一次绘制调用期间，片段中每个像素的统一值保持不变。因此，制服是不可指纹的特征，出于完整性考虑，我们在此列出。
- *变种。* 变量将值从顶点着色器传递到片段着色器，片段着色器在这些值之间进行插值并对片段进行栅格化，即绘制片段中的每个像素。插值算法在不同的计算机图形卡中有所不同，因此可以指纹显示。
- *纹理。* 给定顶点和纹理之间的映射设置，片段着色器将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包围的纹理中的这些像素为目标像素内插值。纹理插值算法从一张显卡到另一张显卡也有所不同，从而使纹理指纹可打印。

WebGL中的纹理可以进一步分为几类：（1）普通纹理，即我们

以上介绍；（2）深度纹理，即包含每个像素的深度值的纹理；（3）为纹理制作动画，
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通道的透明度。 透明度是GPU和驱动程序提供的功能，可以使背景与前景混合在一起。特别是，alpha通道的值在0到1之间，使用合成代数将背景图像和前景图像合成为一个最终的图像。alpha通道中有两个指纹打印点。首先，我们可以使用一个单独的alpha值在背景和前景之间标记合成算法。其次，我们可以在alpha值从0增加到1时显示透明效果的变化。由于某些图形卡采用离散的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。 可以采用不同的格式（例如JPEG，PNG和DataURL）对图像进行编码和压缩。某些格式（例如PNG）是无损的，而某些格式（例如JPEG）会在信息丢失的情况下进行压缩。压缩图像的解压缩是可指纹识别的功能，因为在解压缩期间，不同的算法可能会覆盖不同的信息。根据我们的研究，这是单浏览器功能，不能用于跨浏览器。

已安装的编写脚本（语言）。 由于库的大小和语言的本地性，编写脚本（系统）或通常称为书面语言（例如中文，韩文和阿拉伯文）的要求安装特殊的库才能显示。浏览器不提供访问已安装语言列表的API，但是可以通过辅助渠道获取此类信息。特别是，安装了特定语言的浏览器将正确显示该语言，否则将显示多个框。也就是说，可以使用框的存在来为该语言的存在打上指纹。

D.新提议的复合可打印特征

现在，让我们介绍我们新提议的可合成指纹的功能，这些功能可以由多个浏览器API或组件呈现，有时还可以在浏览器API之上构建其他算法。

建模和多种模型。 建模，即本文中的3D建模，是通过三维表面数学描述对象的计算机图形处理。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，并且可能彼此交互，尤其是当存在以下技术（例如照明）时。

照明和阴影贴图。 照明是计算机图形学中光效果的模拟，阴影映射是测试像素在特定光线下是否可见并添加相应的阴影。有许多类型的照明，例如环境照明，定向照明和点照明，它们的光源不同。此外，当灯光与一个或多个计算机图形模型交互时，许多效果都伴随有灯光，例如反射，半透明，光线跟踪和间接照明。WebGL不

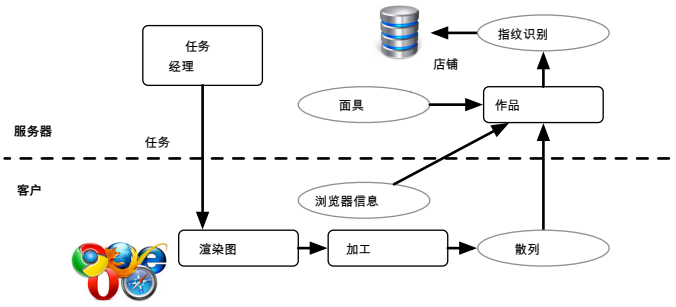

图1：系统架构

提供用于灯光和阴影的直接API，某些WebGL库（例如three.js）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。 相机或特定的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，摄影机由由顶点和片段着色器处理的摄影机投影矩阵表示，可用于旋转和放大和缩小对象。

修剪飞机。 裁剪将渲染操作限制在指定的感兴趣区域内。在3D渲染中，裁剪平面与摄影机相距一定距离，并与摄影机垂直，因此它可以防止渲染距离摄影机太远的表面。在WebGL中，剪切平面由顶点和片段着色器使用提供的其他算法来执行。

三，d 电子签名

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，绘制曲线和线条）发送到客户端。请注意，渲染任务还涉及获取操作系统和硬件级别信息，例如屏幕分辨率和时区。然后，客户端浏览器通过调用特定的API或API的组合来呈现这些任务，并产生相应的结果，例如图像和声波。然后，将这些结果（尤其是图像）转换为哈希，以便可以将其方便地发送到服务器。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组合。

接下来，当服务器从客户端收集所有信息时，服务器将开始合成指纹。具体来说，从客户端的哈希列表和掩码（对应于哈希列表的一或零的列表）生成指纹，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。单浏览器指纹的掩码很简单，列出了所有掩码。跨浏览器指纹打印的遮罩由两个来源组成。首先，收集的浏览器信息将有助于屏蔽：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，每个浏览器对都有一个不同的掩码，例如Chrome与Firefox和Chrome与Windows Edge。

在接下来的两节中，我们首先在客户端介绍渲染任务，然后介绍指纹组成，尤其是如何生成蒙版。

B.渲染任务

在本节中，我们介绍这项工作中提出的不同渲染任务。在此之前，让我们首先在下面介绍基本的画布设置。画布的大小是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轴是向底部增加的垂直线，而z轴远离移动时增加屏幕。功率为[R：0.3，G：0.3，B：

比例尺为1的[0.3]，并且在[0，0，-7]的位置放置了一个摄像头。这两个组件是必需的，因为否则该模型将完全是黑色的。在本文的其余部分中，除非另有说明，例如具有2D功能的任务（d）和具有附加灯光的其他任务，否则我们在所有任务中都使用相同的基本设置。

请注意，与AmIUnique [26]中的设置不同，当当前窗口的状态改变时，我们的画布设置是可靠的。具体来说，我们测试了三种不同的更改：窗口大小，侧栏和缩放级别。首先，我们手动更改窗口大小，然后发现在哈希值方面，画布上的内容在视觉和计算上都保持不变。其次，我们放大和缩小当前窗口，发现内容根据定义在视觉上发生了变化，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，类似于更改窗口大小。现在让我们介绍从任务（a）到（r）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，经典的Suzanne Monkey Head模型[19]在具有随机生成纹理的画布上渲染。纹理，大小为256的正方形 × 通过随机选择每个像素的颜色来创建256。也就是说，我们在一个像素上为三种原色（红色，绿色和蓝色）统一生成介于0和255之间的三个随机值，将三种原色混合在一起，然后将其用作像素的颜色。

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功能。原因如下。当片段着色器将纹理映射到模型时，片段着色器需要对纹理中的点进行插值，以便可以将纹理映射到模型上的每个点。插值算法因一张显卡的不同而不同，并且当纹理的颜色急剧变化时，差异也会扩大。因此，我们生成了这种纹理，其中每对相邻像素之间的颜色变化很大。

任务（b）：变化。 如图2（b）所示，此任务旨在测试画布上片段着色器的变化功能。在多维数据集模型的六个表面上绘制了不同的变化颜色，每个表面上指定了四个点的颜色。我们选择这种变化的颜色来扩大每个单个表面上的颜色差异和变化。例如，当一个表面上的蓝色丰富（例如0.9，比例为1）在一个表面上时，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，

相机放置在[0，0，-5]的位置，以便与任务（c）进行比较。

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，我们在任务（b）中采用相同的任务，并添加了抗锯齿功能。如果我们放大图2（b'），我们将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，将投影矩阵馈入片段着色器。此任务中的每个设置都与任务（a）相同，只是相机已移至[-1，-4，-10]的新位置。相同的立方体看起来比任务（a）中的立方体要小，这是因为相机从立方体移得更远（z轴为-10，而不是-5）。

任务（d）：直线和曲线。 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线。具体来说，曲线服从

以下功能： $y = 256 - 100\ cos(\ 2.0\ \pi x/100.0) + 30\ cos(\ 4.0\ \pi x/100.0) + 6\ cos(\ 6.0\ \pi x/100.0)$ ， 其中[0，0]是
在画布的左侧和顶部，x轴向右增加，y轴向底部增加。三行的起点和终点是{[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直线和曲线，以便我们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）的反锯齿版本。

任务（e）：多模型。 图2（e）中的任务是测试不同模型在同一画布中如何相互影响。除了Suzanne模型外，我们还介绍了另一种看起来像单人武装沙发的模型（称为沙发模型），并将两个模型并行放置。遵循任务（a）中所述的相同过程，将另一个随机生成的纹理映射到沙发模型。

任务（f）：轻。 图2（f）中的任务是测试点状散射光和Suzanne模型之间的相互作用。点状漫反射光在照亮物体时会引起漫反射。具体地说，白色的光在RGB上具有相同的值，每种原色的光功率为2，光源位于[3.0，-4.0，-2.0]。

我们在此任务中选择白色光源，因为纹理是彩色的，并且单色光可以减少纹理上的一些细微差异。还必须谨慎选择光的强度，因为非常弱的光不会照亮Suzanne模型，使其不可见，但是非常强的光会使所有东西变白并减少所有可指纹的功能。在使用6台机器的小规模实验中，当功率从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。灯光位置是随机选择的，不会影响特征指纹的结果。

任务（g）：灯光和模型。 图2（g）中的任务是测试单个散射点光源和两个模型的相互作用，因为一个模型在被点光源照明时可能会在另一个模型上产生阴影。每种灯光设置都与任务（f）相同，并且模型与任务（e）相同。

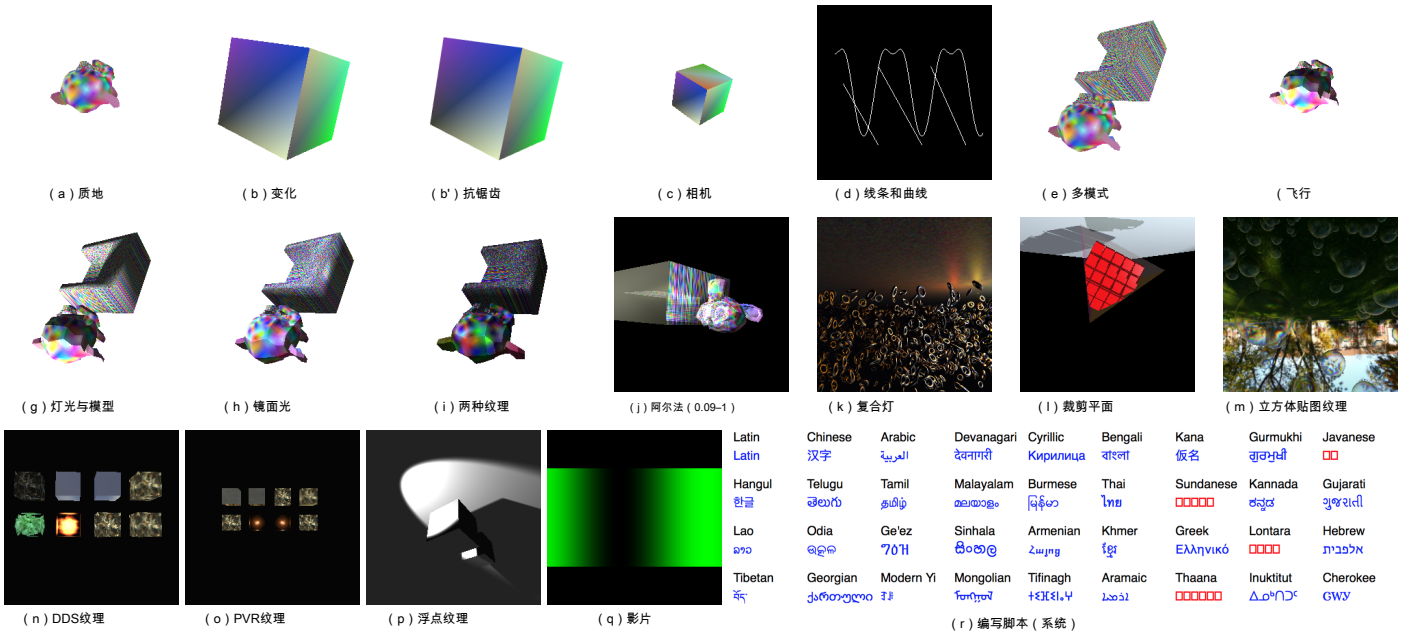

图2：以指纹为目的的客户端渲染任务

任务（h）：镜面光。 图2（h）中的任务是在两种模型上测试具有另一种颜色的漫反射点光和镜面反射点光的效果。类似于漫射点光，镜面反射点光将在对象上引起镜面反射。具体来说，两个灯都位于[0.8，-0.8，-

0.8]，则扩散点光的RGB为[0.75，0.75，1.0]，镜面光的RGB为[0.8，0.8，0.8]。

有两件事值得注意。首先，我们选择特定的相机位置，因为它离模型更近并且效果更大。特别是，人们可能会注意到镜面反射镜上的点被镜面反射点照亮。其次，尽管扩散点光的颜色偏向蓝色，但仍然具有很多红色和绿色。我们想测试其他颜色，但是考虑到纹理是彩色的，白光仍然是最适合指纹打印的颜色。

任务（h'）：抗锯齿+镜面反射光。 任务（h'）是任务（h）的反锯齿版本。

任务（h"）：抗锯齿+镜面反射+旋转。 任务（h'）与任务（h'）相同，但旋转90度。

任务（i）：两种纹理。 图2（i）中的任务是测试将两个不同纹理映射到同一对象的效果。在任务（h）的顶部（即，其他所有设置都相同），我们将随机生成的纹理的另一层映射到Suzanne和沙发模型。

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值的影响。具体来说，我们将Suzanne模型和沙发模型并行放置，并更改从该特定集合中选择的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，而1表示不透明。

同样，有两件事值得注意。首先，我们仔细选择此值集以反映不同的alpha值和较小的值变化：三个代表值{0.1、0.4，

0.8}及其附近的值被选中。值是

由于许多GPU不接受较小的步长，因此增加0.01。其次，将Suzanne和沙发模型放置在适当的位置，以使它们部分重叠，并且当模型变为透明时，可以看到沙发模型的隐藏结构。例如，从模型背面观看时，沙发模型的手臂部分可见。

任务（k）：复杂的灯光。 图2（k）中的任务是测试复杂的灯光功能，例如反射，移动灯光以及多个模型之间的光线追踪。具体来说，我们生成

5,000个具有不同角度的金属环模型随机放置在地面上并堆叠在一起。为了提高可靠性，我们每次都使用带有相同随机种子的种子随机数生成器，以便可以在不同的浏览器和机器上重复进行测试。朝向底部的两个点光源，黄色和红色在整个场景的右上角处盘旋。当灯光照亮下方的环时，其他环也会通过反射而照亮，并且来自不同光源的两种颜色会混合在一起。

请注意，我们选择单色光源是因为模型不是彩色的，并且带有颜色的光会照亮环上的更多细节。此外，具有不同颜色的灯光将相互影响，并产生更详细的效果。

任务（k'）：抗锯齿+复杂光源。 任务（k'）是任务（k）的反锯齿版本。

任务（l）：剪切平面。 图2（l）中的任务是测试剪切平面和FPS的运动。具体来说，我们将一个静态的正四面体放在地面上，用准直光照射它，然后移动剪切平面，以使观察者感觉到四面体正在移动。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+菲涅耳效果。 图2（m）中的任务是测试立方体贴图的纹理和菲涅耳效果。

轻反射。特别地，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，菲涅耳效应是反射光量取决于视角的一种观察。我们创建一个具有正常校园场景的立方体贴图纹理，并在该纹理之上放置几个透明气泡以产生菲涅耳效果。所有气泡在动画中随机移动并相互碰撞。

任务（n）：DDS纹理。 DDS纹理是指那些使用DirectDraw表面文件格式的纹理，DirectDraw表面文件格式是具有S3纹理压缩（S3TC）算法的一种特殊的压缩数据格式。从DXT1到DXT5，S3TC有五种不同的变体，每种格式都具有启用mipmapping的选项，mipmapping是一种将高分辨率纹理缩放为纹理文件中多种分辨率的技术。因为DXT2与DXT3类似，而DXT4与DXT5类似，所以任务（n）仅测试DXT1，DXT3和DXT5，在每列中有和没有进行映射，如图2（n）所示。为了进行比较，我们在最右边的列中还包含了ARGB格式的未压缩纹理。图2（n）中有两个灰色立方体，因为该特定机器不支持带有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，大多数移动设备都采用它，例如所有iPhone，iPod Touch和iPad以及某些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，有两个流行的版本，v1和v3，我们也可以选择启用mipmapping。总体而言，如图2（o）所示，任务（o）具有8个子任务，这些子任务枚举了位模式，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。浮点纹理的一种特殊类型是深度纹理，它包含来自特定场景的深度缓冲区的数据。图2（p）所示的任务（p）是从现有的在线测试[15]中采用的，目的是渲染浮纹和深度纹理。

任务（q）：视频（动画贴图）。 图2（q）中的任务是测试视频的解压缩。具体来说，我们使用三种不同的压缩格式（即WebM，高质量MP4和标准MP4）从PNG文件创建一个两秒钟的静态场景视频，将视频作为动画纹理映射到立方体，并从中捕获六个连续的帧。该视频。

请注意，尽管所有视频都是使用一个PNG文件创建的，但由于压缩算法存在损耗，因此捕获的帧有所不同。我们选择六个连续的帧，因为JavaScript仅提供一个API来在特定时间获取帧，而不提供特定的帧号-六个连续的帧可以根据我们的实验来确保目标帧在集合内。

任务（r）：编写脚本。 图2（r）中的任务是在浏览器中获取受支持的书写脚本的列表，例如拉丁文，中文和阿拉伯文。由于现有的浏览器均未提供API来获取受支持的书写脚本列表，因此我们采用辅助渠道来测试每个书写脚本的存在。具体地，该方法如下。的名字

算法1 跨浏览器掩码生成

输入：
M：所有可能的蒙版的集合。
$H_{浏览器，机器=\{杂凑_{任务1}，杂凑_{任务2}，杂凑_{任务3}，...\}}$：哈希列出特定机器的一个浏览器上的所有渲染任务。
$H_{浏览器=\{H_{浏览器，机器1}，H_{浏览器，机器2}，...\}}$：哈希表用于浏览器。
$HS = \{H_{组合金}，H_{火狐}H_{歌剧}，...\}$：整体哈希列表。
过程：
1：为了 所有可能的$\{H_{浏览器1}，H_{浏览器2}\} \subset HS$做
2：　*最大限度* $_{优衣库} \leftarrow 0$
3：　*最大限度* $_{面具} \leftarrow$ *空值*
4：　为了 *面具* 在 *中号* 做
5：　　*FS* $\leftarrow \{\}$
6：　　*数数* $\leftarrow 0$
7：　　为了 *米*$_{1个} \in H_{浏览器1个}$ 和 *米*$_{2个} \in H_{浏览器2个}$ 做
8：　　如果 *米*$_{1\&}$ *遮罩*$==m_{2\&}$ *面具* 和 *米*$_{1\&}$ *面具* $\in$ / *FS* 然后
9：　　*数++*
10：　　*FS.add*（*m*$_{1\&}$ *面具*）
11：　　万一
12：　　结束于
13：　　*优衣库* $\leftarrow$ *数*/*尺寸*（*h*$_{浏览器1}$）
14：　　如果 *Uniq*>*最大值* $_{优衣库}$ 然后
15：　　*最大限度* $_{优衣库} \leftarrow$ *优衣库*
16：　　*最大限度* $_{面具} \leftarrow$ *面具*
17：　　万一
18：　结束于
19：　*最大限度* $_{面具}$ 是浏览器1和2的掩码。
20：结束于

每个用自己的语言编写的脚本都会在浏览器中呈现。如果支持编写脚本，则渲染将成功；否则，渲染将失败。否则，将显示一组框而不是脚本。因此，我们可以检测方框以测试浏览器是否支持该脚本：例如，图2（r）显示该特定经过测试的浏览器不支持Javanese，Sudanese，Lontara和Thaana。我们当前的测试列表中有36种从Wikipedia [8]获得的书写脚本，并按其受欢迎程度进行排名。

C.指纹组成

在本节中，我们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，指纹是根据所有任务和掩码的哈希列表的“与”运算计算得出的哈希。对于单浏览器指纹，该掩码完全是所有掩码，由跨浏览器指纹的两个子掩码计算得出。在第III-A节中，我们已经讨论了根据浏览器是否支持某些功能这一事实计算出的第一个子掩码，现在将讨论第二个子掩码，这对每个浏览器对都不同。

每两个浏览器生成掩码是一种基于培训的方法。具体来说，我们使用一小部分子集来获得可同时优化跨浏览器稳定性和唯一性的蒙版。请注意，类似于假阳性和阴性，这两个数字，即跨浏览器的稳定性和唯一性，是硬币的两个方面：当跨浏览器的稳定性增加时，唯一性降低，反之亦然。让我们考虑两个极端的例子。如果使用单浏览器功能，则跨浏览器的稳定性为零，但唯一性最高。相反，如果仅使用平台等功能，则跨浏览器的稳定性为100％，但唯一性很低。

算法1显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：虽然不是最有效的搜索方法，但是却是最有效和完整的搜索方法。由于

训练数据的大小很小，我们意识到蛮力是可能的，并且可以产生最佳效果。具体来说，我们首先枚举每个浏览器对（第1行），然后列举每个可能的掩码（第4行）。对于每个蒙版，我们都要遍历训练数据（第7行），并确保选择一个最大化跨浏览器稳定性并乘以唯一性的蒙版（第8-11行和第14-17行）。

IV。一世 实施

除了所有开源库（例如，Three.js，一个JavaScript 3D库和glMatrix，一个用于矩阵运算的JavaScript库），我们的开源实现都有大约21K行代码（LoC）。具体来说，我们的方法涉及大约14,000行JavaScript，1K HTML，2.4K Coffeescript，500行C代码和3.7K Python代码。

现在，我们将代码分为客户端和服务器，并在下面进行描述。客户端代码具有从Coffeescript生成的JavaScript管理器。管理器执行三个工作：（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的JavaScript代码段，然后进行通信与服务器端代码。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。

我们的实现的服务器端是用Python编写的，用作Apache服务器的模块。我们的服务器端代码可以进一步分为两部分：第一个具有1.2K LoC，用于与客户端代码进行通信并将哈希存储在数据库中，并将图像存储在文件夹中，第二个

2.5K LoC用于分析，例如在收集的指纹上生成和应用蒙版。

V.D ATA C 污染

我们从两个众包网站（Amazon Mechanical Turks和MacroWorkers）收集数据。具体来说，我们指示众包工作者自行选择通过两个不同的浏览器访问我们的网站，如果他们通过三个浏览器访问该网站，他们将获得奖金。访问后，我们的网站将为每个工人提供唯一的代码，以便她可以将其输入回众包网站以获取报酬和可选的奖金。请注意，在我们的数据收集中，除了散列之外，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，则无需执行此步骤。

为了确保我们拥有基本的真实数据，我们在每个众包工作者访问的URL中插入一个唯一的标识符，例如http://oururl.com/?id=ABC。唯一标识符以cookie的形式存储在客户端浏览器中，因此，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，我们只允许一名众包工人上班一次。例如，MTurks中的每个人的智能任务（HIT）数量为一个。

在三个月内，我们总共从1,903位用户那里收集了3,615张指纹。一些用户只是访问我们的网站

表I：由我们的方法AmIUnique和Panopticlick收集的数据集的六个属性的归一化熵（后两列摘自AmIUnique论文）

	我们的	AmI唯一	Panopticlick
用户代理	0.612	0.570	0.531
插件清单	0.526	0.578	0.817
字体列表（Flash）屏幕分辨	0.219	0.446	0.738
率	0.285	0.277	0.256
时区	0.340	0.201	0.161
Cookie已启用	0.001	0.042	0.019

使用一个浏览器，并且不会完成两个浏览器的任务。我们将所有指纹直接用于单浏览器指纹。对于跨浏览器指纹打印，如果有足够的数据，则每个浏览器对将数据集平均分为十个部分：一个用于生成掩码，另一个用于测试。

A.将我们的数据集与AmIUnique和Panopticlick进行比较

本部分的目的是在AmIUnique论文中发明的归一化Shannon熵度量中，将我们的数据集与AmIUnique和Panopticlick进行比较。具体来说，等式1根据他们的论文给出了定义：

$$NH = \frac{高(X)}{H_{中号}} = \frac{-\sum_{一世}(一世)日志_{2个}(一世)}{日志_2(N)} \quad (1)$$

*高（X）*是香农的熵 *X*是一个变量
可能的值{ X_1，$X_{一世}$，..}P（X）是一个概率函数。
$H_{中号}$在最坏的情况下，每个指纹都具有相同的概率，而我们的熵最大。 *ñ*是

指纹总数。

表I显示了比较结果，其中从AmIUnique论文的表III中获得了AmIUnique和Panopticlick的统计信息。我们观察到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，除了字体列表和时区。

首先，字体列表的规范化熵从AmIUnique下降0.22，从Panopticlick下降0.52。AmIUnique解释的原因是Flash消失了。与AmIUnique相比，到我们收集数据时，支持Flash的浏览器的百分比下降得更多。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。值是0.901，非常接近Panopticlick的值。

第二，时区的归一化熵增加
来自AmIUnique的0.139和来自Panopticlick的0.179。原因是我们来自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。特别是，MicroWorkers允许我们创建针对全球不同地区的活动，并且确实为每个大陆创建了活动。

另一点值得注意的是，对于我们的数据集，启用的Cookie的规范化熵几乎为零。原因是我们从众包网站收集数据，

表II：BoA等人比较AmIUnique的总体结果。不包括IP地址，我们的方法（“唯一”表示唯一指纹在总数中所占的百分比，“熵”表示香农熵，而“稳定性”则表示在所有浏览器中稳定的指纹所占的百分比。我们没有列出跨浏览器的数量表中Boda等人使用的是AmIUnique和单浏览器编号，因为这些编号非常低，并且其设计方法并非为此目的而设计的。）

	单浏览器		跨浏览器		
	独特的	熵	独特的	熵	稳定
独特的[26]	90.84％	10.82			
博达（Boda）等人。[14]			68.98％	6.88	84.64％
我们的	99.24％	10.95	83.24％	7.10	91.44％

工人需要在启用Cookie的情况下获得报酬。如果他们禁用cookie，他们甚至无法登录众包网站。相反，AmIUnique和Panopticlick都吸引了普通的Web用户，其中一小部分人可能会禁用Cookie。通常，禁用cookie的人很少，因为cookie对许多现代Web功能都是必不可少的。

六。[R 结果

在本节中，我们首先概述结果，然后按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。

A.概述

我们首先概述单浏览器和跨浏览器指纹打印的结果。具体来说，我们将现有技术的AmIUnique单浏览器指纹技术与Boda等人的跨浏览器指纹技术进行了比较。不包括IP地址。请注意，尽管Boda等人之后出现了许多新功能，例如AmIUnique中的这些功能，但这些功能是浏览器特定的，我们发现Boda等人中使用的功能。仍然是跨浏览器稳定性最高的。

现在，我们介 绍如何重现这两部作品的结果。AmIUnique是开源的[3]，我们可以直接从github下载源代码。博达（Boda）等人。提供了一个开放的测试网站（https://finngerprint.pet-portal.eu/），我们可以直接下载finingprinting JavaScript。我们相信直接使用其源代码可最大程度地减少所有可能的实现偏差。

表II显示了AmIUnique，Boda等人的整体结果以及我们的方法。让我们首先看一下单浏览器指纹打印。在唯一性和熵方面，我们将我们的方法与AmIUnique进行了比较。唯一性是指唯一指纹在指纹总数中所占的百分比，而熵是香农熵。评估表明，我们的方法可以唯一地识别

99.24％的用户与AmIUnique的90.84％相比，增长了8.4％。对于熵，最大值为10.96，并且两种方法（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，让我们看一下跨浏览器指纹打印的指标。除了唯一性和熵外，我们还计算了另一个指标，称为跨浏览器稳定性，这意味着在同一台计算机上的不同浏览器中稳定的指纹百分比。尽管我们选择的功能通常在大多数浏览器中都稳定，但是来自不同浏览器的指纹可能仍会有所不同。例如，如果用户在两个浏览器中选择不同的缩放级别，则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种软件渲染，则GPU渲染对于我们的方法可能有所不同。

现在让我们看一下Boda等人的跨浏览器指纹打印结果。和我们的方法。表二表明，我们的方法可以识别83.24％的用户，而Boda等人则为68.98％。这是一个巨大的增长，相差14.26％。跨浏览器的稳定性也从Boda等人的84.64％提高。达到91.44％。原因之一是我们使现有的功能（例如屏幕分辨率和字体列表）在不同的浏览器中更加稳定。熵也从Boda等人的6.88增加。我们的方法为7.10。

B.按浏览器对分类

在本节的这一部分中，我们按表III中所示的不同浏览器对细分结果。有六种不同的浏览器类型，另一类称为其他浏览器，包括一些不常见的浏览器，例如Maxthon，Coconut和UC浏览器。该表由于其对称特性而成为较低的三角形矩阵：如果我们列出所有数字，则较高的三角形与较低的三角形完全相同。表格的主要对角线代表单浏览器指纹打印，另一部分代表跨浏览器打印。有两个N / A，因为Apple放弃了Windows上对Safari的支持，而Microsoft从不支持Mac OS上的Internet Explorer和Edge浏览器，即Safari不与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要的对角线。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。唯一性最低的浏览器是Mozilla Firefox，因为Firefox隐藏了一些信息，

例如出于隐私原因的WebGL渲染和供应商。IE和Edge的唯一性是100％，表明这两个浏览器都可以高度指纹识别。Opera，Safari和其他浏览器的唯一性也是100％，但是由于我们的数据集中的样本数量很少，因此我们无法为这些浏览器得出进一步的结论。

然后，我们看一下除主对角线外的矩阵的下三角，它显示了跨浏览器指纹打印的唯一性和稳定性。首先，除其他浏览器以及Opera与IE之外，所有浏览器对的跨浏览器稳定性都非常高（> 85％）。由于此类对的数量很少，因此我们很难生成具有合理的跨浏览器稳定性的掩码。

其次，与其他版本相比，IE和Edge与其他版本的唯一性相对较低。原因是IE和Edge都是由Microsoft用更少的开源库独立实现的。也就是说，在IE / Edge和其他浏览器之间共享的共同部分比其余浏览器之间的共享要少得多。相反，两者之间的唯一性

表III：浏览器对的跨浏览器指纹唯一性和稳定性细分

浏览器	铬合金	火狐浏览器	边缘	IE	歌剧	苹果浏览器	其他
铬合金	99.2%（100%）						
火狐浏览器	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%）		
苹果浏览器	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很高：唯一性为100%，跨浏览器稳定性为95.7%，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。对于所有浏览器对，Edge浏览器的唯一性都高于IE。原因是Edge浏览器引入了更多功能，例如完全遵循标准的WebGL实现，从而暴露了更多的指纹方面。

C.按功能细分

在本节的这一部分中，我们将结果划分为不同的功能，并将其显示在表IV中。具体而言，表IV可以分为两部分：AmIUnique行上方的第一部分显示了AmIUnique所采用的功能，第二部分下方的第一部分显示了我们方法提出的所有新功能。现在让我们看一下不同的功能。

1）屏幕分辨率和比率： 屏幕分辨率和比率的单浏览器熵为7.41，而宽度和高度比率的熵显着下降至1.40。原因是许多分辨率，例如1024 × 768和1280 × 960，份额相同。跨浏览器的屏幕分辨率稳定性非常低（9.13%），因为用户经常如前所述放大和缩小网页。由于某些用户采用两个屏幕并将两个浏览器放在不同的浏览器中，因此跨浏览器的宽高比稳定性较高（97.57%）但低于100%。

2）字体列表： 由于Flash的不断消失，从Flash获得的字体列表的熵低至2.40，相反，从JavaScript获得的字体列表的熵高至10.40。这意味着字体列表仍然是可高度指纹识别的功能，将来我们需要使用JavaScript获得该功能。

请注意，尽管来自JavaScript的字体列表的熵很高，但是在指纹打印中它并不占很大的比例。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%降至99.09%，相差不到0.2%。也就是说，没有字体列表功能，我们的方法仍可以高精度地为用户打印指纹。

3）抗锯齿： 任务（b），（b'），（d），（d'），（h），（h'），（k）和（k'）与抗锯齿相关。添加了抗锯齿后，（b），（d）和（h）的单浏览器指纹打印的熵增加，但（k）的熵减小。原因是（b），（d）和（h）的边较少，并且抗锯齿会增加可打印的内容；相反，（k）在每个bean上都包含许多小边缘，并且抗锯齿将占据bean的内容并减少bean内部的一些可打印内容。

现在让我们看一下跨浏览器指纹打印。跨浏览器的稳定性与单浏览器熵相反：它对于（b），（d）和（h）减小，但对于（k）增大。原因是同一机器上的所有浏览器均不支持抗锯齿，这会降低（b），（d）和（h）的稳定性。出于类似的原因，由于抗锯齿减少了bean内部的一些可指纹显示的内容，因此跨浏览器的稳定性对于（k）会增加。

4）线和曲线： 任务（d）测试直线和曲线的效果。熵较低（1.09），跨浏览器的稳定性较高（90.77%），因为直线和曲线都是简单的2D操作，并且在浏览器和计算机之间的差异不大。我们手动比较了不同机器或浏览器的情况，发现主要区别在于开始点和结束点有一个或两个像素偏移。

5）相机： 当比较任务（b）和（c）的单浏览器熵时，我们发现当添加摄像机时，熵降低。原因是添加的相机的目的是缩小立方体，从而减少了表面上的细微差异。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）质地： 让我们首先比较法线，DDS，PVR，立方体贴图和平面纹理。平面和立方体贴图的熵高于所有其他纹理，因为平面和立方体贴图具有更多信息，例如，平面纹理的深度和立方体贴图的立方体贴图。PVR纹理的熵非常低（0.14），因为在iPhone和iPad等Apple移动设备上大多数都支持PVR纹理。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。另一个有趣的发现是，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DDS（Microsoft格式）。

其次，让我们看一下两个纹理，即任务（i）。与任务（h）相比，添加了另一层纹理，但是单浏览器和跨浏览器指纹的熵都减小了。原因是我们在任务中使用的纹理是精心创建的，因此可以包含更多可指纹识别的功能。

表IV：按功能划分的熵和跨浏览器的稳定性

特征	单浏览器	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%
插件清单	5.77	0.25	1.65%
Cookies已启用	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（Flash）	7.41	3.24	9.13%
	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%
不跟踪	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%
Aml唯一	10.82	0.00	1.39%
屏幕比例	1.40	0.98	97.57%
字体列表（JavaScript）AudioCo	10.40	6.58	96.52%
ntext	1.87	1.02	97.48%
CPU虚拟核心	1.92	0.59	100.00%
标准化的WebGL渲染器	4.98	4.01	37.39%
任务（a）纹理	3.51	2.26	81.47%
任务（b）变体	2.59	1.76	88.25%
任务（b'）变体+抗锯齿	3.24	1.66	73.95%
任务（c）相机	2.29	1.58	88.07%
任务（d）线和曲线	1.09	0.42	90.77%
任务（d'）（d）+抗锯齿	3.59	2.20	74.88%
任务（e）多模型	3.54	2.14	81.15%
任务（f）轻	3.52	2.27	81.23%
任务（g）灯光和模型	3.55	2.14	80.94%
任务（h）高光任务（h'）（h）+抗	4.44	3.24	80.64%
锯齿	5.24	3.71	70.35%
任务（h“）（h'）+旋转	4.01	2.68	75.09%
任务（i）两个纹理任务（j）阿尔	4.04	2.68	75.98%
法（0.09）任务（j）阿尔法（0.1	3.41	2.36	86.25%
0）任务（j）阿尔法（0.11）任务	4.11	3.02	75.31%
（j）阿尔法（0.39）任务（j）阿	3.95	2.84	75.80%
尔法（0.40）任务（j）阿尔法	4.35	3.06	82.75%
（0.41）任务（j）阿尔法（0.79	4.38	3.10	82.58%
）任务（j）阿尔法（1）	4.49	3.13	81.89%
	4.74	3.12	72.63%
	4.38	3.07	82.75%
任务（k）复合灯任务（k'）（k）+	6.07	4.19	66.37%
抗锯齿	5.79	3.96	74.45%
任务（l）裁剪平面任务（m）立方体	3.48	1.93	76.61%
贴图纹理任务（n）DDS纹理任务（o	6.03	3.93	58.94%
）PVR纹理任务（p）浮动纹理	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）编写脚	2.87	0.51	97.91%
本（图像）	6.00	1.98	5.48%
跨浏览器的所有功能	10.92	7.10	91.44%
所有功能	10.95	0.00	1.39%

当我们将两个纹理加在一起时，其中的某些功能会减少，从而使两个纹理的任务无法显示指纹。

7）型号： 让我们比较任务（a）和（e）以及任务（f）和（g）的模型效果。与（a）和（f）相比，将沙发模型添加到（e）和（g），并且熵稍微增加一点，即对于两个任务，其熵均为0.03。结论是，Sofa模型确实引入了更多可指纹识别的功能，但是增加的幅度非常有限。

8）灯光： 任务（a），（e），（f），（h）和（k）与灯光有关。让我们首先看一下任务（f），其中在任务（a）中添加了一个漫射的点光源。对于单浏览器和跨浏览器的指纹，熵仅增加0.01，这表明漫射的点光源对指纹的影响很小。作为比较，镜面反射光的效果更加明显，因为在单浏览器和跨浏览器指纹打印中，与任务（e）相比，任务（h）的熵增加> 0.9。最后，让我们看看任务（k），这是一个复杂的示例。任务（k）的熵是除视频之外的所有任务中最高的，因为存在5,000个模型，并且所有模型之间都反射具有不同颜色的灯光并将它们混合在一起。

9）Alpha： 任务（j）测试从0.09到1的alpha值。有趣的是，不同的alpha值具有非常不同的熵。通常，趋势是当alpha值增加时，熵也增加，但有很多回落。我们没有在大规模实验中测试连续的alpha值，而是在五台机器中执行了一个小规模的实验。具体来说，我们将每个Alpha值图像和一个标准图像之间的像素进行比较，发现后退主要是由软件渲染引起的，该渲染近似于alpha值。此外，我们在后备广告中观察到了一些模式，这些模式大约会发生

0.1增量步。

10）裁剪平面： 任务（l）是测试剪切平面的效果，产生3.48单浏览器熵和1.93跨浏览器熵，其稳定性为76.61％。熵与纯纹理的熵相似，这是因为剪切平面在JavaScript中具有重要意义，并且对指纹图谱的贡献不大。

11）轮换： 任务（h“）是任务（h'）的轮换。熵减少，跨浏览器的稳定性增加。原因是Suzanne模型的前部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，可指纹显示的细节都会减少，因此稳定性会相应增加。

12）AudioContext： 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形。熵为1.87，远小于整个波的整个熵（Englehardt等人测量为5.4）。[18]。

13）视频： 任务（q）正在测试视频功能。在所有渲染任务中，视频的熵最高（7.29），因为对视频进行解码是浏览器，驱动程序，有时还包括硬件的组合。相反，视频的跨浏览器稳定性非常低（5.48％），并且熵也下降到2.32。原因是类似于图像编码和解码，WebM和MP4视频格式都丢失并且由浏览器解码。我们没有像图像那样找到通用的视频无损格式。

14）文字剧本： 编写脚本在任务（r）中进行了测试。为了跨浏览器指纹打印，我们将任务（r）进一步分为两部分。第一部分，我们称之为书写脚本（支持），仅包含有关某些书写脚本是否受支持的信息，即零列表以及一个表示支持而零表示不支持的列表。如前所述，我们通过盒子检测获得信息。第二部分，我们称其为编写脚本（图像），是在客户端渲染的图像。用于编写脚本（图像）的单浏览器熵比用于编写脚本（支持）的单浏览器熵大3.13。即，图像是否包含比是否支持书写脚本更多的信息。应用遮罩后，根据结果计算出跨浏览器的脚本编写稳定性（支持）。因为某些编写脚本随浏览器一起提供，并且跨浏览器不稳定。相应地，用于编写脚本（支持）的跨浏览器熵低于单浏览器。

15）CPU虚拟核心： 仅从HardwareConcurrency值（如果不支持，该值“未定义”）中计算出的CPU虚拟核的数量，对于单浏览器指纹打印的熵为1.92。我们希望信息熵在将来会增加，因为在我们提交Firefox 48之前，Firefox 48就开始支持该新功能。跨浏览器的稳定性为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。由于数据的大小，跨浏览器的熵不同于单浏览器的熵，并且两者的归一化熵非常相似。

16）标准化的WebGL渲染器： WebGL渲染器无法跨浏览器打印指纹，部分原因是不同的浏览器提供了不同级别的信息。我们从不同的浏览器中提取通用信息，并以标准格式对齐信息。与具有5.70熵的原始WebGL渲染器相比，归一化的WebGL渲染器的熵为4.98。删除的原因是，提取将丢弃某些信息（例如，供Chrome浏览器使用）以与其他浏览器（如Edge浏览器）对齐。相应地，跨浏览器的稳定性从原始WebGL渲染器的15.39％增加到标准化的WebGL渲染器的37.39％。

这里有两点值得注意。首先，WebGL供应商没有提供比WebGL渲染器更多的信息。也就是说，当我们将两个值组合在一起时，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。某些浏览器（即Firefox）不提供WebGL供应商和渲染器信息，这为我们提供了填补这一空白的很大空间。此外，即使浏览器提供了此类信息，我们GPU任务组合在一起时的熵也为7.10，远大于

WebGL render提供的5.70熵。原因是渲染是软件和硬件的组合，并且WebGL渲染器仅提供用于硬件渲染的硬件信息。

D.观察

在我们的实验和实现过程中，我们观察到了一些有趣的事实，并在本小节中对其进行了展示：

观察1：我们的指纹功能高度可靠，
也就是说，删除一个单一特征对指纹结果几乎没有影响。

在这一部分中，我们展示了从AmIUnique和我们的方法中删除单个功能，然后测量两者的唯一性的影响。结果表明，当删除表IV中的任何单个功能（包括AmIUnique的所有旧功能和我们的新功能）时，指纹的唯一性仍高于99％。相反，如果删除以下六个属性中的任何一个，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，则AmIUnique的唯一性将降至84％以下。总之，就使用功能而言，我们的方法比AmIUnique更可靠。

观察2：软件渲染也可以用于指纹打印。

WebGL的一种普遍理解是，软件渲染可以减少图形卡引起的所有差异。但是，我们的实验表明，即使软件渲染也可以用于指纹打印。具体来说，我们选择由SwiftShader渲染WebGL的所有数据，SwiftShader是Google发明的一种开源软件渲染器，在硬件渲染不可用时被Chrome使用。我们计算出一个仅包含我们所有GPU渲染任务的特殊指纹，即任务（a）-（p），不包括编写脚本和视频。

由于硬件渲染的高度采用，我们仅使用SwiftShader收集了88个案例，并找到了11种不同的GPU指纹以及7种独特的GPU指纹。软件渲染的唯一性肯定比硬件渲染的唯一性低得多，但仍不为零。也就是说，在采用软件渲染来减轻基于WebGL的指纹时，我们需要小心。

观察点3：WebGL渲染是软件和硬件的组合，其中硬件的贡献要大于软件。

在此观察中，我们将比较与软件渲染相比的另一种极端情况，即Microsoft Basic Rendering。Microsoft Basic Rendering为所有类型的图形卡提供了通用驱动程序，即，使用Microsoft Basic Rendering可以最大程度地减少软件驱动程序的影响并显示硬件带来的影响。与软件渲染实验类似，我们选择使用Microsoft Basic渲染的软件并计算指纹。

出于类似的原因，在软件渲染中，我们仅使用Microsoft Basic Rendering收集了32个案例，并发现了18个具有15个唯一值的不同GPU指纹。Microsoft Basic Rendering的唯一性低于使用普通图形卡驱动程序的唯一性，这意味着WebGL由软件和硬件两者呈现。同时，我们认为硬件做出了更大的贡献，因为Microsoft Basic渲染的唯一性高于软件渲染器的唯一性。

观察4：DataURL在不同浏览器中的实现方式有所不同。

在此观察中，我们看一下DataURL，它是在先前的指纹打印中用来表示图像的一种通用格式。令人惊讶的是，我们发现DataURL在浏览器中的实现方式大不相同，即，如果将图像转换为DataURL，则表示在不同浏览器中的变化很大。对于单浏览器指纹打印来说，这是个好消息，但对跨浏览器来说则是坏消息。如表IV所示，由于我们采用了AmIUnique的代码，其中DataURL用于存储图像，因此Canvas的跨浏览器比率非常低（8.17％）。

观察结果5：渲染结果之间的某些差异非常微妙，即存在一两个像素差异。

在最后的观察中，我们手动比较了渲染结果之间的差异，发现尽管其中一些比较大，尤其是在软件和硬件渲染之间，但有些却非常微妙，特别是当两个图形卡彼此相似时。例如，一台iMac和另一台Mac Pro渲染的Suzanne模型在纹理上仅相差一个像素，如果我们旋转该模型，则差异将消失。

七。d 的感觉 P 定位 F 专利

在本节中，我们讨论如何保护我们建议的浏览器指纹。我们将首先从现有的防御（即著名的Tor浏览器）开始，然后对我们的防御进行一些展望。

Tor浏览器将许多浏览器的输出标准化，以减轻现有浏览器的指纹。也就是说，Tor浏览器中没有许多功能-根据我们的测试，仅以下功能（尤其是我们新提议的功能）仍然存在，包括屏幕的宽度和高度比以及音频上下文信息（例如采样率和最大声道）数数）。我们相信Tor浏览器很容易将这些剩余的输出标准化。

值得一提的另一件事是，Tor浏览器默认情况下会禁用画布，并会要求用户允许使用画布。如果用户确实允许使用画布，则仍然可以打印指纹。Tor浏览器文档中也提到了未实现的软件渲染解决方案，但是，如第VI-D节所述，在同一浏览器中，软件渲染的输出也存在显着差异。我们仍然相信这是追求的方式，但是需要更仔细的分析才能包括所有软件渲染库。

总体而言，捍卫浏览器指纹保护的想法可以概括为虚拟化，我们需要找到正确的虚拟化层。考虑一种极端的解决方案，即在虚拟机内部运行的浏览器-一切都在虚拟机中进行了标准化，并且浏览器在不同物理机上的输出是相同的。但是，缺点是机器虚拟化很重。Tor浏览器是另一个极端，所有内容都虚拟化为浏览器的一部分。这种方法是轻量级的，但是我们需要找到所有可能的可指纹显示的位置，例如画布和音频上下文：如果缺少一个位置，仍然可以以某种方式对浏览器进行指纹显示。我们将其作为探索正确的虚拟化层的未来工作。

八。d 讨论 E 思想 一世 诉讼

我们已经与我们机构的机构审查委员会（IRB）讨论了道德问题，并获得了IRB的批准。具体而言，尽管可以使用网络跟踪来获取私人信息，但是我们从众包工作者那里获得的标识符（例如计算机图形卡的行为）本身并不是私人的。仅当标识符与诸如浏览历史记录之类的私人信息相关联时，该组合才被视为私人信息，但是，此步骤不在研究范围之内。我们的调查部分（即附录A中有关多种浏览器使用情况的统计数据的研究）包含用户的浏览习惯。为了确保隐私，调查将匿名进行，我们不存储来自MicroWorkers的用户ID。

九。[R 兴高采烈 w ^ 兽人

在本节中，我们讨论有关现有Web跟踪和反跟踪技术的相关工作。

A. Web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，然后是第二代浏览器指纹。

1）基于Cookie或超级Cookie的跟踪： 现有许多工作集中在cookie或基于超级cookie的Web跟踪技术的测量或研究上。梅耶等。[28]和Sanchez等。[40]对第三方跟踪进行全面的讨论，包括跟踪技术，业务模型，防御选择和政策辩论。Roesner等人的另一项重要测量工作。提出了一个针对现实网站中部署的不同Web跟踪的综合分类框架[39]。Lerner等。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以及浏览器指纹的搜索[27]。Soltani等。和Ayenson等。衡量基于非cookie的状态跟踪的普遍性，并展示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。[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外，Krishnamurthy等。[22-25]集中于Web跟踪导致的损害风险，表明不仅用户的浏览历史记录，而且其他敏感的个人信息（例如姓名和电子邮件）也可能被泄露。

2）浏览器指纹： 现在让我们讨论浏览器指纹打印，第二代网络跟踪。我们首先讨论现有的测量研究。颜等。和Nikiforakis等。讨论了现有指纹打印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]。阿卡（Acar）等。[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，即画布指纹打印，另一种是第一代网络跟踪，即evercookies和“ cookies”的使用。同步”与evercookies结合使用。现场等。[20]专注于第二代网络跟踪的特定指标，即字体。FPDetective [10]通过关注字体检测及其框架来对数百万个最受欢迎的网站进行大规模研究。Englehardt等。

数百万个网站，并找到了许多新的指纹打印功能，例如AudioContext。我们还将其新发现的指纹打印功能用作本文第二部分中先前功能的一部分。

现在让我们谈谈浏览器指纹的工作原理。Mowery等。[32]可能是提出基于画布的指纹的非常早期的作品之一。其他一些著作[31，33]则集中在指纹浏览器JavaScript引擎上。讨厌地等。[34]是一份立场文件，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跟踪仅包含基于时序的功能，可靠性不如本文中的技术。Laperdrix等。[26]，

即AmIUnique，对具有17个属性的浏览器指纹进行了最广泛的研究，我们在整篇论文中都将它们与它们进行了比较。博达（Boda）等人。[14]尝试实现跨浏览器跟踪，但是它们的功能是单浏览器跟踪（包括IP地址）中的旧功能。如前所述，当计算机在NAT后面使用DHCP或移动到笔记本电脑等新位置时，IP地址不可靠。

作为与现有作品的一般比较，我们的方法在操作系统和硬件级别上引入了许多新功能。例如，我们介绍了许多GPU功能，例如纹理，变化，灯光和模型。再举一个例子，我们还引入了一个辅助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的稳定性。

B.现有的反跟踪机制

我们首先讨论针对第一代跟踪的现有反跟踪，然后讨论第二种。

1）基于Cookie或超级Cookie的反跟踪技术： Roesner等。[39]提出了一种名为ShareMeNot的工具，该工具可以防御社交媒体按钮跟踪，例如Facebook Like按钮。私人浏览模式[44，45]通过单独的用户配置文件将正常浏览与私人浏览隔离开来。同样，TrackingFree [37]采用基于特征的隔离，并为特征创建提出了一个度数限制图。不跟踪（DNT）[43]标头是一种选择退出的方法，它要求跟踪器合规。如先前的工作所示[28，39]，DNT无法有效地保护用户免受现实世界中的跟踪。用户还可以禁用第三方cookie，大多数浏览器都支持该第三方cookie，以避免基于cookie的跟踪。孟等人。[29]设计策略并授权用户控制是否要跟踪，但是他们必须依靠现有的反跟踪技术。

前面提到的所有工作都集中在基于cookie或基于超级cookie的Web跟踪上，并且可以完全或部分阻止这种跟踪。在本文中，它们都不能阻止提议的指纹打印，因为提议属于第二代，不需要服务器端的状态标识符。

2）针对浏览器指纹的反跟踪： 托尔浏览器[38]可以成功捍卫许多浏览器指纹打印技术，包括本文提出的功能。有关更多详细信息，请参阅第VII节。除Tor浏览器中提出的归一化技术外，PriVaricator [35]在可指纹打印的输出中添加了随机噪声。由于PriVaricator不是开源的，因此我们无法针对他们的防御来测试指纹。

X.C 列入

总而言之，我们提出了一种新颖的浏览器指纹打印功能，它不仅可以识别一个浏览器后面的用户，还可以识别在同一台计算机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公开的图形卡，AudioContext的音频堆栈和hardwareConcurrency的CPU。我们的评估表明，对于单浏览器指纹打印，与AmIUnique相比，与Boda等人相比，我们的方法可以唯一地标识更多的用户。用于跨浏览器指纹打印。我们的方法是高度可靠的，也就是说，删除任何单个功能只会使准确性最多降低0.3％。

一种 知识点

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）在CNS-1646662和CNS-1563843资助下提供支持。本文包含的观点和结论是作者的观点和结论，不应解释为必然代表NSF的官方政策或认可，无论是明示或暗示。

[R 参会

[1] 核心估算器。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] 是 一世 独特的？ https://github.com/DIVERSIFY-项目/独特的。

[4] [图形Wikia]抗锯齿。http://graphics.wikia.com/wiki/反锯齿。

[5] Panopticlick：您的浏览器可以安全进行跟踪吗？https：// panopticlick。eff.org/。

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

[7] [维基百科] 立方体 映射。 https://zh.wikipedia.org/wiki/多维数据集 _ 映射。

[8] [wikipedia]书写系统列表。https://zh.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：持久的持久跟踪机制”，在 *2014年ACM SIGSAC计算机和通信安全性会议论文集*，系列 CCS '14，2014年，第674页– 689。

[10] G. Acar，M。Juarez，N。Nikiforakis，C。Diaz，S。Gürses，F。Piessens和B. Preneel，“ FPDetective：为指纹打印机撒粉”*2013 ACM SIGSAC计算机和通信安全性会议论文集*，系列 CCS '13，2013年，第1129-1140页。

[11] M. Ayenson，D。Wambach，A。Soltani，N。Good和C. Hoofnagle，“ Flash Cookie和隐私ii：现在具有html5和etag的重生功能”，*可在SSRN 1898390获得*，2011。

[12] S.伯杰。您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，AMFöldes，GGGulyás和S. Imre，“通过跨浏览器指纹在网络上跟踪用户”，*第16届北欧应用信息安全技术会议论文集*，系列 NordSec'11，2012年，第31-46页。

[15] F. Boesch。柔和的阴影贴图。http：// code fl ow.org/entries/2013/feb/ 15 / soft-shadow-mapping /。

表V：浏览器使用情况统计

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

[16] 金融时报委员会。跨设备跟踪。https://www.ftc.gov/news- events / events-calendar / 2015/11 / cross-device-tracking。

[17] P. Eckersley，“您的网络浏览器有多独特？”在 *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，“在线跟踪：一百万个站点的测量和分析”， *22Nd ACM SIGSAC计算机和通信安全性会议论文集，* 系列 CCS '16，2016年。

[19] A. Etienne和J. Etienne。搅拌机的经典苏珊猴到得到你的游戏开始了和苏珊 http://learningthreejs.com/blog/2014/05/09/classical-suzanne-monkey- from-blender-to-get-your-game-started-with-threex-dot-suzanne /。

[20] D. Fifield和S. Egelman，“通过字体指标对网络用户进行指纹识别”，在 *金融密码学和数据安全。* Springer，2015年，第107-124页。

[21] S.坎卡尔。Evercookie。http://samy.pl/evercookie/。

[22] B. Krishnamurthy，K。Naryshkin和C. Wills，“隐私泄漏与保护措施：日益增长的脱节”，在 *Web 2.0安全和隐私研讨会，* 2011。

[23] B. Krishnamurthy和C. Wills，“网络上的隐私传播：纵向观点”，第 *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，“在互联网上产生隐私足迹”， *第六届ACM SIGCOMM互联网测量会议论文集。* ACM，2006年，第65-70页。

[25] ——表征在线社交网络中的隐私”，在 *第一届在线社交网络研讨会的论文集。* ACM，2008年，第1卷，第37-42页。

[26] P. Laperdrix，W。Rudametkin和B. Baudry，“美女与野兽：转移现代Web浏览器以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器的袭击：1996年至2016年的网络追踪考古研究”，载于 *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”，在 *安全和隐私（SP），2012 IEEE研讨会。* IEEE，2012年，第413–427页。

[29] W. Meng，B。Lee，X。Xing和W. Lee，“ Trackmeornot：启用对Web跟踪的灵活控制”，第 *第25届万维网国际会议论文集，* 系列 WWW '16，2016年，第99-109页。

[30] H. Metwalley和S. Traverso，“网络跟踪程序的无监督检测”，在 *Globecom，* 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：在html5中对画布进行指纹识别”，2012年。

[33] M.Mulazzani，P.Reschl，M.Huber，M.Leithner，S.Schrittwieser，E. Weippl和F. Wien，“使用javascript引擎指纹进行快速可靠的浏览器识别”，在 *W2SP，* 2013。

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

[35] N. Nikiforakis，W。Joosen和B. Livshits，“专论者：用很少的白色谎言欺骗指纹打印机”，在 *第24届国际互联网会议论文集，* 系列 WWW '15，2015年，第820-830页。

[36] N. Nikiforakis，A。Kapravelos，W。Joosen，C。Kruegel，F。Piessens和G. Vigna，“无饼干怪兽：探索基于Web的生态系统设备指纹”，在 *IEEE安全与隐私研讨会，* 2013。

[37] X. Pan，Y。Cao和Y. Chen，“我不知道您去年夏天访问过什么-使用无跟踪浏览器保护用户免受第三方Web跟踪的影响，” *NDSS，* 2015年。

[38] M. Perry，E。Clark和S. Murdoch，“ tor浏览器[草稿] [在线]的设计和实现，美国”，2015年。

[39] F. Roesner，T。Kohno和D. Wetherall，“检测与捍卫反对网络上的第三方跟踪”，在 *第9届USENIX网络系统设计与实现大会论文集tion，* 系列 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”，在 *国际联席会议。* 施普林格，2015年，第473页–483。

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

[42] 美国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。不追踪政策。http://en.wikipedia.org/wiki/不追踪政策。

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

[45] M. Xu，Y。Jang，X。Xing，T。Kim和W. Lee，《无痕：无泪的私人浏览》， *22Nd ACM SIGSAC计算机和通信安全性会议论文集，* 系列 CCS '15，2015年，第438–449页。

[46] 电话 Yen，Y。Xie，Y。F. Yu，RP余和M. Abadi，“网络上主机指纹的打印和跟踪：隐私和安全的影响”，第1期， *NDSS的程序，* 2012。

一种 附录 一种

小号 调查 P 人 ' 小号 ü 贤者 中号 终极 乙 抢劫者

在本附录中，我们研究在同一台计算机上使用多个浏览器的人员的统计信息。请注意，这是与论文的所有其他设计和实验分开的小规模研究。我们进行研究以增强论文的动机。我们的结果表明，人们在相当多的时间里确实在同一台机器上使用多个浏览器。

现在，让我们在众包网站MicroWorkers上介绍我们的实验设置。我们进行的调查带有一个开放性问题，询问被调查者他们拥有和通常使用的浏览器，以及在每个浏览器上花费的时间百分比。他们可以自由地将任何内容写到多行文本框中。

这是我们的实验结果。我们已经收集了102个答案，其中一个答案只是复制了我们的调查链接，另一个提到了不存在的浏览器。排除这两个无效答案后，我们总共有100个确切的答案。95％的受访用户安装了两个以上的浏览器，因为默认情况下安装了IE或Edge。我们还会定期使用两个或多个浏览器来计算它们的百分比，

也就是说，他们在至少一个浏览器上花费了至少5％的时间。

表V中显示了使用浏览器的人的结果。70％的受访者经常使用两个或多个浏览器，只有30％的人使用单个浏览器。调查答案中的浏览器类型包括Chrome，Firefox，IE，Edge，Safari，Coconut Browser和Maxthon。结果表明，人们确实使用了多种浏览器，跨浏览器指纹打印非常重要和必要。