杀死那只鹦鹉：基于深度行为的指纹探测识别

Coia, Sam, huige, Sekka

@acgdaily

摘要

随着 TLS（Transport Layer Security）加密协议的广泛应用，它已成为现代网络通信的主流方式和网络安全的基石。

然而，传统的基于明文特征的流量识别与安全检测方法在面对加密流量时面临巨大挑战。现有的 TLS 指纹识别技术（如 JA3/JA3S）主要依赖于握手阶段的明文特征，这使得它们容易受到特征伪装的影响，进而降低识别精度。

为了解决这一问题，本文提出了一种新的 TLS 客户端与服务器指纹识别方法，摒弃了对 JA3/JA3S 的依赖。

该方法通过对 TLS 握手和数据传输过程中的客户端与服务器行为特征进行数据库对比，在加密流量环境下实现高精度的实体识别。

相比传统方法，本方案无需破坏加密内容，并能够在中间盒环境中进行流量重定向、重放探测及异常行为识别。虽然注入和劫持式探测方法可能会导致连接中断，但它们能够提供深度行为分析，从而显著提升对加密流量的可观测性与安全性。

实验结果表明，该方法能有效区分不同应用和主机的行为特征，为网络安全提供了新的技术支撑，是网络安全防护体系中的关键基石，具备广泛的应用前景。

关键词：TLS，中间盒，指纹，深度行为检测

第一章 介绍

1.1 背景

随着网络安全的需求日益增加，基于传输层安全 (TLS) 协议的加密流量分析逐渐成为识别恶意行为和防止网络攻击的重要手段。

TLS ClientHello 中的指纹信息 (如协议版本、加密套件、扩展字段等) 成为了特征提取的关键。

然而，随着反侦察技术的发展，爬虫、代理及其他工具通过伪装 ClientHello 消息，成功避免了基于 JA3/JA4 指纹的检测。

Chromium 内核 (基于 BoringSSL) 的浏览器在 106 版本引入了 TLS 扩展随机排序[1]以避免指纹识别，但是这并不影响排序处理后的指纹。

例如，某些流行的工具（如 UTLS 库）会故意伪装 TLS 握手信息的特征，从而绕过传统的指纹分析方法，但是此类工具伪装的指纹缺乏大量测试，验证和审核，从而导致可以被直接被动检测 (附录 A)

因此，本研究旨在通过引入新的深度行为指纹和探测方法，不再依赖传统的 JA3(S) / JA4(S) 以改进现有的指纹识别技术，尤其是对伪装流量的检测能力，通过客户端/服务器的行为模式，进一步提高 TLS 指纹探测的准确性，为网络安全提供更有效的保障。

1.2 本文研究内容

通过对不同 TLS 实现中不严格遵循 RFC 规范，RFC 未定义的具体行为和 TLS 堆栈独有特征为依据，判断对方的 TLS 堆栈是哪种实现方案

第二章 深度行为探测

2.1 探测及指纹识别

不同的 TLS 实现（TLS 堆栈）在处理非规范或边缘情形时存在实现差异，因而在收到相同的异常或非标准记录层事件后会产生不同的响应行为。

本节目的在于通过对这些行为差异的被动与主动观测，构建一种以响应特征为核心的指纹识别方法，从而区分常见的 TLS 实现类型。

该方法的核心思想为：

1. 探测方法：以 “违反或偏离 RFC 定义的报文/记录层事件”、RFC 中明确应被忽略的兼容性保留情形（如中间盒兼容行为）以及各实现独有的错误处理特征为探测方式
2. 响应特征化：对每一次握手或会话交互，记录端点返回的高层响应类型（例如：连接中断、握手失败警告、错误码类别），并将这些可观测指标抽象为 “深度行为指纹” 用于比较。
3. 误判缓解：在探测时进行多次测试对比以避免外部干扰。
4. 隐私与合规措施：严格执行数据最小化与去标识化原则 —— 仅保留用于分类与统

计的元数据摘要；对任何可能含敏感信息的原始日志或抓包进行脱敏或销毁；所有实验均在作者控制或经书面授权的隔离网络中进行。

本章所述方法着重于“基于响应行为”的分类与检测框架，旨在为实现指纹识别及伪装验证提供可重复的、可审计的学术方法。

该方法为学术与防御目的设计，严格遵守最小化数据原则，仅能收集深度行为指纹，且**不会导致任何加密后数据的泄露。**

注意: 使用以下探测方式时

1. 若果要采集具体行为指纹: 探测包不宜发送过快和过慢
   延迟以等待对方收到后以后再发送下一个
2. 仅检查 JA3 指纹与行为指纹是否一致: 可一次性发送多个包以加速流程
3. 堆栈可能会一次性打开多个 TCP 并使用 Close Notify 关闭到仅剩余一个

2.2 握手前探测 (明文)

TLS 记录: Alert (Level: Warning) <可携带无效 TLS/SSL 版本号>

注入位置: ServerHello 前任意位置

探测流程:

Client		Server
ClientHello	--->	
	<---	(Alert… <Level: Warning>)
(Alert*)	--->	
		ServerHello
		Certificate
		CertificateStatus*
		ServerKeyExchange*
		CertificateRequest*
	<---	ServerHelloDone
Certificate*		
ClientKeyExchange		

CertificateVerify*

ChangeCipherSpec

{Finished} --->

NewSessionTicket*

ChangeCipherSpec

<--- {Finished}

[Application Data] <---> [Application Data]

2.3 握手中探测 (明文)

2.3.1 TLS 1.0-1.2

注: 可在握手前探测的亦可在握手中探测

TLS 记录: Alert (Level: Warning)

注入位置: ChangeCipherSpec 前任意位置

探测流程:

Client Server

ClientHello --->

ServerHello

Certificate

CertificateStatus*

ServerKeyExchange*

CertificateRequest*

ServerHelloDone

<--- (Alert… <Level: Warning>)

(Alert*) --->

(Alert… <Level: Warning>) --->

<--- (Alert*)

Certificate*

ClientKeyExchange

CertificateVerify*

```
ChangeCipherSpec
{Finished}                    --->
                                                NewSessionTicket*
                                                ChangeCipherSpec
                              <---               {Finished}
[Application Data]            <--->              [Application Data]
```

2.3.2 TLS 1.3

TLS 记录: ChangeCipherSpec

注入位置: ClientHello/ServerHello 后 (非 0-RTT 客户端的建议等待服务器发送完 Finished)

探测流程:

```
Client                                          Server
ClientHello                   --->
                                                ServerHello
                              <---               (ChangeCipherSpec…)
(Alert*)                      --->
                                                ChangeCipherSpec*
                                                {EncryptedExtensions}
                                                {CertificateRequest*}
                                                {Certificate}
                                                {CertificateVerify}
                              <---               {Finished}
(ChangeCipherSpec…)           --->
                              <---               (Alert*)
ChangeCipherSpec*
{EncryptedExtensions*}
{Certificate*}
{CertificateVerify*}
{Finished}                    --->
```

[Application Data] <---> [Application Data]

2.4 握手后探测 (加密)

注: 可在握手中探测的亦可在握手后探测

2.4.1 TLS 1.0-1.2

TLS 记录: 重协商 HelloRequest (若支持) / 解密后无数据的 Application Data

2.4.2 TLS 1.3

TLS 记录: ChangeCipherSpec [明文/加密] / Alert (详见下文或附件) / 解密后无数据的 Application Data

2.5 常见 TLS 堆栈行为指纹参考

注: 仅供参考，实际行为可能随版本变化，不一定准确

表 2.5-1 计数器最大值

TLS 堆栈	Alert (Level: Warning)	ChangeCipherSpec (TLS 1.3)	Empty Application Data	Alert 响应
BoringSSL (Edge 122)	4	32*	32*	Unexpected Message
NSS (Firefox 143)	不限制	1	不限制	Unexpected Message
Safari 18.7	4	32*	32*	Unexpected Message
OpenSSL 1.1.1 (Nginx 1.24)	4	32*	32*	Unexpected Message
GnuTLS	0	不限制	不限制	无
Cloudflare	4	32*	32*	Unexpected Message

Golang	16*	16*	16*	Unexpected Message

注: 相同角标为共享计数器标识, 当发送第 n 个包后收到 Alert 响应或关闭则最大值为 n-1

表 2.5-2 握手前探测: 带有无效 TLS/SSL 版本号的 Alert (Level: Warning)

TLS 堆栈	SSLv3	TLS 1.3	无效版本号	Alert 响应
BoringSSL (Edge 122)	正常	正常	出错	Protocol Version
NSS (Firefox 143)	正常	正常	正常	N/A
Safari 18.7	正常	正常	出错	Protocol Version
Golang	正常	正常	正常或出错	无

表 2.5-3 握手后探测: TLS 1.3 ChangeCipherSpec

TLS 堆栈	明文	加密
BoringSSL (Edge 122)	Bad Record MAC	Unexpected Message
NSS (Firefox 143)	Unexpected Message	Unexpected Message
Safari 18.7	Bad Record MAC	Unexpected Message
OpenSSL 1.1.1 (Nginx 1.24)	Unexpected Message	Unexpected Message
GnuTLS	Unexpected Message	Unexpected Message
Cloudflare	Bad Record MAC	Unexpected Message
Golang	计入计数器	计入计数器

表 2.5-4 握手后探测: TLS 1.3 Alert 响应 (Description: UserCanceled)

TLS 堆栈	Warning	Fatal
BoringSSL (Edge 122)	计入计数器	按 Warning 计入计数器
NSS (Firefox 143)	计入计数器	出错并关闭 (Close Notify)
Safari 18.7	计入计数器	出错并关闭 (Close Notify)
OpenSSL 1.1.1 (Nginx 1.24)	计入计数器	按 Warning 计入计数器
GnuTLS	出错并关闭 TCP	出错并关闭 TCP
Cloudflare	计入计数器	出错并关闭 TCP
Golang	计入计数器	按 Warning 计入计数器

表 2.5-5 握手后探测: TLS 1.3 Alert 响应 (Level: Warning, Description: AccessDenied)

TLS 堆栈	响应
BoringSSL (Edge 122)	Decode Error
NSS (Firefox 143)	出错并关闭 (Close Notify)
Safari 18.7	Decode Error
OpenSSL 1.1.1 (Nginx 1.24)	出错并关闭 TCP
GnuTLS	出错并关闭 TCP
Cloudflare	Decode Error
Golang	出错

表 2.5-6 握手后探测: TLS 1.3 Alert 响应 (Level: Warning, Description: NoRenegotiation)

TLS 堆栈	响应
BoringSSL (Edge 122)	Decode Error
NSS (Firefox 143)	出错并关闭 (Close Notify)
Safari 18.7	Decode Error

OpenSSL 1.1.1 (Nginx 1.24)	出错并关闭 TCP
GnuTLS	出错并关闭 TCP
Cloudflare	Decode Error
Golang	出错

第三章 客户端识别与反爬虫

3.1 客户端识别

本方法**无需在客户端执行任何 JavaScript** 或额外代码，任何托管资源的站点均可在浏览器对资源的正常请求过程中被动采集必要元数据以完成识别与检测。

识别流程采用两类互补信号：一类为静态/半静态指纹信号（例如浏览器 User-Agent（UA）与 JA3 TLS 指纹，用于推断客户端所用的浏览器/TLS 堆栈类别）；另一类为基于第二章所述方法的 “深度行为指纹”， 用以验证该堆栈在面对非规范或边缘协议事件时的实际响应是否与已建立的基线相符。

识别流程：

1. 触发方式: 在页面中嵌入受控域名下的隐形或静态资源， 将探测唯一识别码嵌入域名 (以 SNI 在 ClientHello 中呈现)，浏览器按常规加载触发 TLS 握手；服务器端被动探测并记录与该请求关联的“深度行为指纹”。
2. 指纹信号：采集保存 UA 与 JA3 的必要字段，以便用于堆栈类别的初步判定。
3. 深度行为探测：基于第二章定义的 “深度行为指纹”，记录握手阶段的高层错误/告警类别、连接中断等行为性元数据；将这些响应行为与已知实现进行对比，评估实现行为是否符合预期。
4. 融合判定：将 UA、JA3 与行为指纹按预先校准的权重或策略融合，输出实现类别与置信度；对低置信度或高风险样本触发人工复核或后续二次验证策略。

隐私与安全保证：

1. 仅收集用于识别的元数据与摘要字段，**绝不记录或解析任何应用层加密负载或明文内容**；建议对 UA/JA3 等敏感标识取必要内容最小化保存。
2. 所有采集、存储处置活动应符合适用隐私法规（含告知/同意机制、最短保留期、访问控制审计）。

3. 该方法设计为检测与防御用途。

3.2 反爬虫

该方法在服务端集中采集并联合分析第 3.1 节中定义的被动元数据（包括 UA、JA3 及深度行为指纹），以“是否符合预期行为”为判据构建判定规则与风险评分模型。

通过跨维度特征的关联， 而非依赖单一可篡改字段，本方法能够显著降低由 UA 篡改、JA3 伪装以及对客户端 JavaScript 质询的逆向与规避所带来的漏判风险。

此外，所设计的特征提取与评分流程可无缝衔接 5 秒盾的风控决策链路，既支持前端可疑行为的实时标注，又为后端决策模块提供稳健的输入，从而在整体体系内实现对自动化爬取与恶意探测行为的高效识别与分层响应。

设计遵循以下原则：

1. 防护优先以最小侵扰为目标：优先记录与监控，只有在高置信度的异常下再执行更强的防护动作。
2. 不做堆栈指认：系统只判定“符合预期”或“不符合预期”，不输出或保存任何关于具体 TLS 堆栈厂商/版本的识别结论。
3. 隐私最小化：仅使用已经定义的必要字段，不处理或保存加密后的应用层内容。

架构概览:

- 数据采集层（被动）

  由任意网站在加载受控资源时触发的正常请求被服务器端记录为一次会话样本。服务器仅记录哈希/摘要与行为元数据。

- 行为评估层（实时）

  使用第 2.5 节定义的 “常见 TLS 堆栈行为行为” 对每个样本计算 “行为一致性分数”，表示与已知合规堆栈的相似度。该计算为纯统计/分类任务，不含具体堆栈判别输出。

- 决策引擎（阈值与策略）

  根据行为一致性分数与业务风险等级应用策略（记录、触发轻度挑战、限速、或人工复核）。

- 反馈与学习层

  将人工复核结果与后续观察作为标签回馈到基线更新流程，定期重训练以应对

正常客户端演进。

操作与合规细节:

1. **不可单一依赖 UA 或 JA3**：因 UA 易被伪造且 JA3 可能随实现版本变化，系统应采用多模态融合（UA/JA3/行为）并以行为一致性为核心判定。
2. **人工复核机制**：对被标记为“可疑”或“异常”的样本必须保留复核记录，人工复核结果用于模型精化与误判申诉处理。
3. **记录与保留策略**：仅保存必要的去标识化元数据，并严格限制访问与保留期限；在受监管区域遵守相关法律（例如用户告知/同意，数据最短保留期）。
4. **透明度与用户体验**：对正常用户尽量减少可见干扰；发生挑战时提供清晰的说明与申诉渠道以降低误判带来的负面影响。

第四章 中间盒探测与攻击

4.1 概览

本节概述针对 TLS 握手链路的中间盒攻击方法及其在指纹采集与堆栈识别场景中的探测流程。

中间盒通过在客户端与服务器之间插入、劫持或修改 TLS 握手流程，能够诱导或触发可区分的协议层与实现层行为，从而暴露、放大或伪造客户端/服务器指纹信息。

我们将中间盒攻击划分为四类技术路径：

1. 重定向（基于 SNI 的流量引导并观测服务器响应）
2. 劫持式中间人（主动劫持客户端并注入握手信息以观察端点行为）
3. 重放客户端 ClientHello（将捕获的 ClientHello 重放到目标服务器以收集服务器对该 Hello 的响应特征）
4. 已建链路注入（在已建立的 TLS 会话中随机注入记录并监测端点反应）

下文分别对这四类方法进行简要阐述，并提供针对不同方案实际可行的探测流程。

4.2 重定向式探测

探测目标: 客户端

适用类型: ClientHello 单向客户端认证协议

导致连接中断: 否

成本: 中等

探测流程:

1. 截取客户端数据包 ClientHello，提取报文的 TCP 目标端口号和 TLS SNI
2. 中间盒解析 SNI 对应的 IP 地址，将该 TCP 流重定向到解析的地址
3. 观察行为，在握手完成后服务器是否有 Alert 发送 (TLS 1.3 判断加密数据是否为 1+2+16 字节)
4. 如果存在 Alert 则说明该客户端为 FakeTLS 代理

当前情况: 已被绝大多数实现修复

被动探测方案: 不符合 TLS 流量规范的流量且没有 Alert 则是代理

1. 对于 CBC 套件，所有流量应该被 PADDING 到套件规定的整数倍
2. 对于非 AEAD 套件，所有流量应该包含 MAC 大小
3. 对于 AEAD 套件，所有流量应该包含 AEAD Tag 大小
4. 对于 TLS 1.1+，则还需要包含显式 IV 大小
5. （需主动探测一次）对于同一个 ClientHello 更换不同的密钥/忽略字段，服务器响应长度应该一致

4.3 劫持式攻击

探测目标: 客户端

适用类型: 全部

导致连接中断: 是

成本: 较高

探测流程:

1. （随机采样）将客户端的 TCP 连接重定向到中间盒搭建的行为测试服务器
2. 按第二章提供的 握手前/握手中 方式探测

   对于探测 TLS 1.3: 发送 ServerHello 但是不发送 Certificate
3. 不停的发送探测包观察行为, 直到对方发送 Alert 关闭或达到最大计数器 (附件提供的工具是 100)
4. 根据 Alert 行为和数据库进行对比（可配合 JA3 确定是否是伪装）

4.4 重放式探测

探测目标: 服务器

适用类型: 基于 ClientHello 认证客户端，并进行服务器认证协议

导致连接中断: N/A

成本: 低

探测流程:

1. 捕获一个 ClientHello 数据包，进行重放，按 2.2 内容继续进行探测
2. 简单修改捕获的数据包 (例如 Random/SessionID/KeyShare 以及 RFC 规定的忽略字段) 使认证失败被回落，然后按 2.3 内容进行服务器行为探测
3. 对比两次识别到的行为特征指纹，如果不一致则是伪装（可反复测试验证）

4.5 注入式攻击

探测目标: 全部

适用类型: TLS 1.3 握手完成后 (劫持式攻击亦可用于注入)

导致连接中断: 大概率会

成本: 最高

探测流程:

1. 等待 TLS 握手完成
2. 向客户端/服务器注入明文的 ChangeCipherSpec 消息
3. 观察是否有 Alert (Application Data 长度为 19) 和关闭连接的消息发出

第五章 总结与展望

5.1 总结

本报告围绕基于 TLS 堆栈的深度行为探测展开，提出并验证了以被动元数据（如 UA、JA3 及深度行为指纹）为基础、以 “是否符合预期行为” 为核心的反爬与可疑访问识别框架；同时讨论了通过中间盒攻击对握手链路进行定向探测以扩充指纹空间的四类方法（重定向式、劫持式、重放式、注入式），并阐明了这些方法在指纹可测性、侵入性与对抗性方面的差异。以下总结本研究的主要结论、局限性与未来研究方向。

5.2 主要结论

1. 多维被动指纹的联合分析可显著提升对伪装与篡改行为的鲁棒性

   通过将 UA、JA3 与深度行为指纹进行跨维度融合，并以 “是否符合预期行为” 的判据构建判定规则及风险评分，能够从多角度放大实现差异，从而降低仅基于单一字段（如 UA 或 JA3）导致的误判与漏判概率。
2. 服务端集中采集与对比校验是实用且可部署的方案

   在服务端统一采集握手及后续交互的细粒度行为数据，结合时序与上下文校验，既可抵御客户端本地级别的篡改（例如 UA 修改、JA3 伪装），也便于将检测逻辑无缝嵌入现有风控链路（如 5 秒盾），实现实时标注与后端决策输入的协同。
3. 中间盒探测方法能有效扩展指纹空间但存在权衡。

   重定向与 ClientHello 重放为偏被动、低侵入的探测手段，适于大规模数据部署探测

   劫持与注入更具侵入性和探测深度，可暴露端点在异常时序与错误处理上的差异，但同时带来稳定性风险与连接中断。

5.3 局限性

1. 对抗性演进的威胁

   伪装者可针对本框架调整策略（如更复杂的 JA3 模拟、行为模仿、对中间盒行为进行检测与规避），因此检测模型需要持续更新与对抗训练。
2. 标准与协议演进带来的不确定性

   TLS 协议及其扩展（新版本、QUIC 的广泛部署）可能降低基于握手可见度的检测信号，需要探索对新协议栈的适配方案。
3. 数据依赖性与泛化能力

   指纹模型对训练样本的覆盖度敏感，跨地域、跨运营商与跨设备环境的泛化能力需通过多源数据与持续采样来保证。

5.4 未来工作

1. 对抗性鲁棒性研究

   采用对抗训练、生成对抗网络或动态白盒测试方法以提升模型对伪装/蓄意规避的鲁棒性。

2. 行为扩展

研究对 HTTP/2、QUIC 及 HTTP/3 指纹等新兴协议和应用层指纹的适配策略，探索基于流量行为的跨协议指纹统一建模。

5.5 写在最后

基于 TLS 行为的深度行为探测为识别自动化爬取、伪装客户端与路径性异常提供了有效手段。

通过多维指纹融合、服务端集中分析与有选择的中间盒探测，可以在保持系统可用性的前提下显著提升检测能力。

然而，这是一场持续的技术对抗：随着协议演进与对抗者策略的变化，防护体系必须保持动态演进、严格评估并遵循法律与伦理约束。

我们期待在未来的工作中，进一步强化对抗鲁棒性、扩展至新协议栈并推动行业间关于测量、评估与合规的协作。

参考

电子资源

[1] GitHub. refraction-networking/utls | feat: Chrome 106 Shuffled Fingerprint #133

https://github.com/refraction-networking/utls/pull/133

[2] RFC 2246: The TLS Protocol Version 1.0

https://www.rfc-editor.org/rfc/rfc2246

[3] RFC 8466: The Transport Layer Security (TLS) Protocol Version 1.3

https://www.rfc-editor.org/rfc/rfc8446

[4] GitHub: Google BoringSSL

https://github.com/google/boringssl

[5] GitHub: Mozilla Network Security Services (NSS)

https://github.com/nss-dev/nss

[6] GitHub: Go Programming Language | crypto/tls

https://github.com/golang/go/tree/master/src/crypto/tls

致谢

1. The Parrot is Dead: Observing Unobservable Network Communications
   https://people.cs.umass.edu/~amir/papers/parrot.pdf

附录 A 漏洞披露: UTLS

A.0 UTLS 相关项目被暂时暂停

图 A.0 UTLS 项目网站（https://tlsfingerprint.io）截图

A.1 BoringSSL 系 (Chrome/Edge) ECH GREASE 指纹泄漏

影响范围: 全部 UTLS Chrome/Edge 带有 ECH 指纹

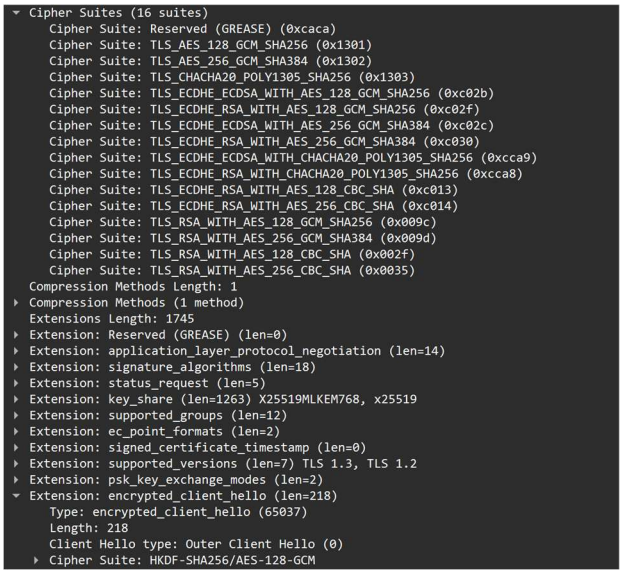
影响版本: 2023/23 – 2025.10 (由本频道披露并修复)

匹配概率: 50%

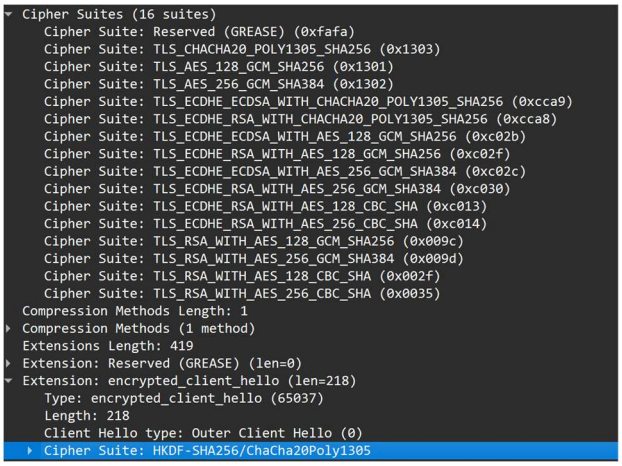
BoringSSL 的 ECH GREASE 密码套件偏好与握手偏好一致[1]

```
const EVP_HPKE_AEAD *aead =
    has_aes_hw ? EVP_hpke_aes_128_gcm() : EVP_hpke_chacha20_poly1305();
```

图 A.1-1 BoringSSL ECH GREASE 源代码



(a) AES 优先



(b) CHACHA20 优先

图 A.1-2 Edge 122 ECH 指纹

UTLS 只有 AES 优先，但是随机选择 ECH 密码套件，会导致密码套件不匹配[2]

```go
// BoringGREASEECH returns a GREASE scheme BoringSSL uses by default.
func BoringGREASEECH() *GREASEEncryptedClientHelloExtension {
    return &GREASEEncryptedClientHelloExtension{
        CandidateCipherSuites: []HPKESymmetricCipherSuite{
            {
                KdfId:  dicttls.HKDF_SHA256,
                AeadId: dicttls.AEAD_AES_128_GCM,
            },
            {
                KdfId:  dicttls.HKDF_SHA256,
                AeadId: dicttls.AEAD_CHACHA20_POLY1305,
            },
        },
        CandidatePayloadLens: []uint16{128, 160, 192, 224}, // +16: 144, 176, 208, 24
    }
}
```

图 A.1-3 UTLS ECH GREASE 源代码

[1] https://github.com/google/boringssl/blob/88d0c0f4772f3abe74f4f1012fe580fa85bab417/ssl/encrypted_client_hello.cc#L732

[2] https://github.com/refraction-networking/utls/blob/a5511b382b818deb970dd402090369dba8a59856/u_ech.go#L296

A.2 BoringSSL 系 (Chrome/Edge) ECH 指纹无 Padding

影响范围：全部 UTLS Chrome/Edge 带有 ECH 的非 PQC 指纹

影响版本: 2023/12 – 2026/01 (由本频道披露并修复)

影响版本(默认值): 2023/12 – 2025/05

匹配概率: 25%-50%

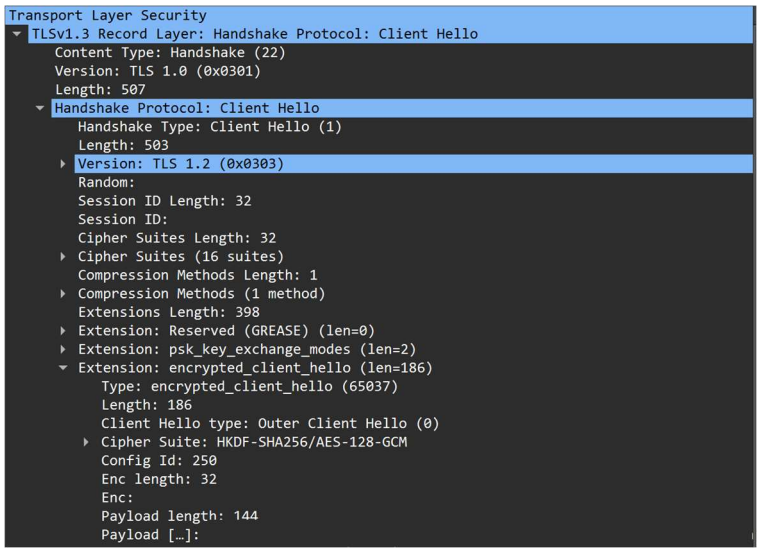


图 A.2-1 UTLS Chrome 120 ECH 指纹

BoringSSL 小于 512 长度的 ClientHello 会被 Padding 到最少 512 字节

指纹提交者刚刚好抓到了足够大小的 ECH 指纹从而忽略了可能会有的 Padding。

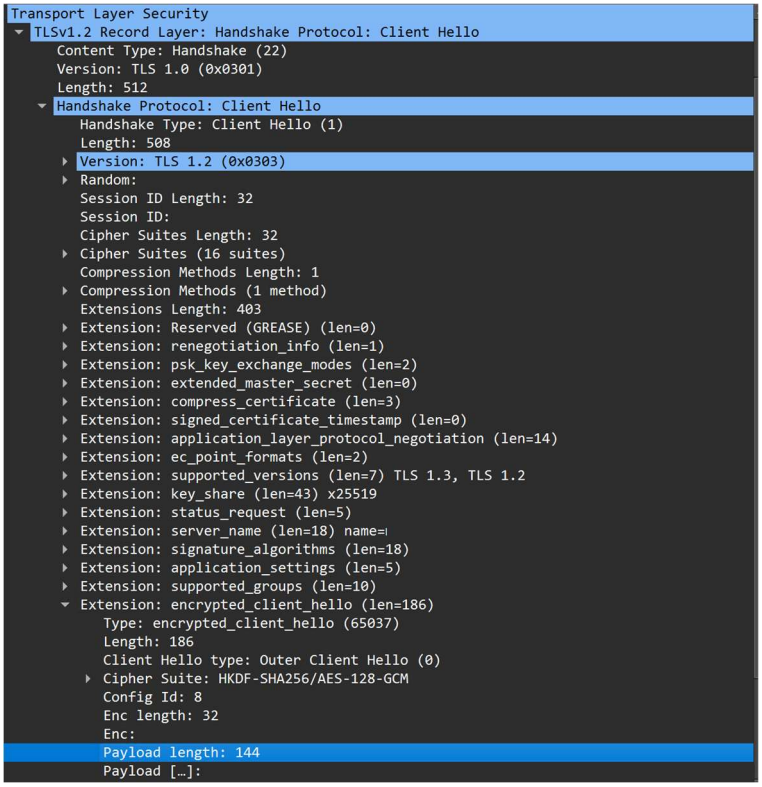


图 A.2-2 Edge 122 ECH 指纹