基于 LLM 的前端加密自动分析框架

李金龙[1]，赵新元[1,2]
（1. 新疆师范大学　计算机科学技术学院，新疆　乌鲁木齐　830054；
2. 新疆智慧教育及应用工程技术研究中心，新疆　乌鲁木齐　830054）

摘　要：为提升网络攻击复杂性并防范内容威胁，很多网站利用前端脚本对数据包进行加密。但前端脚本的透明性导致其加密逻辑与密钥普遍面临被逆向解析的潜在风险。目前，针对此类风险的识别与分析主要依赖安全专家的手动逆向，效率低下且高度依赖专家经验，难以应对规模化安全评估需求。针对这一问题，提出一种基于大语言模型的前端加密自动化分析框架，该框架通过动态钩子技术捕获加密函数调用，并构造面向加密逆向专用提示词驱动 LLM 对混淆代码进行智能解析，从而能够自动重构出完整的加解密封装函数，并且引入可验证性约束，确保分析结果的可靠性。在六个真实匿名高校身份认证平台上的实验结果表明，该框架在含动态密钥获取在内的多类场景下均成功捕获加密细节，且显著提升了此类安全评估的整体效率。该研究为实现前端加密机制的高效、规模化安全评估提供了一条可扩展的技术路径。

关键词：前端加密；LLM；动态钩子；自动化；Web 安全
中图分类号：TP309　　**文献标志码**：A　　**DOI**：10. 19358/j. issn. 2097-1788. 2026. 07. 003

中文引用格式：李金龙，赵新元. 基于 LLM 的前端加密自动分析框架[J]. 网络安全与数据治理，2026，45(7)：16-23.
英文引用格式：Li Jinlong，Zhao Xinyuan. LLM-based automated analysis framework for front-end encryption[J]. Cyber Security and Data Governance，2026，45(7)：16-23.

LLM-based automated analysis framework for front-end encryption

Li Jinlong [1]，Zhao Xinyuan [1,2]
（1. School of Computer Science and Technology，Xinjiang Normal University，Urumqi 830054，China；
2. Xinjiang Engineering Technology Research Center for Smart Education and Applications，Urumqi 830054，China）

Abstract：To raise the barrier to cyberattacks and mitigate content-based threats，many websites use front-end scripts to encrypt data packets. However，the transparency of front-end scripts also exposes their encryption logic and keys to the potential risk of reverse engineering. Currently，the identification and analysis of such risks primarily rely on manual reverse engineering by security experts，which is inefficient，highly dependent on expert experience，and unable to meet the demands of large-scale security assessments. To address this issue，we propose an automated front-end encryption analysis framework based on large language models（LLMs）. This framework uses dynamic hooking techniques to capture calls to encryption functions and constructs prompts specifically designed for encryption reverse engineering to drive the LLM to intelligently parse obfuscated code. This ensures that the framework can automatically reconstruct complete encryption and decryption wrapper functions. Additionally，we introduce verifiability constraints to ensure the reliability of the analysis results. Experimental results on six real-world，anonymized university identity authentication platforms demonstrate that the framework successfully captures cryptographic details across various scenarios，including those involving dynamic key acquisition，and effectively improves the efficiency of such security assessments. This research provides a scalable technical approach for achieving efficient，large-scale security assessments of front-end encryption mechanisms.
Key words：front-end encryption；LLM；dynamic hooks；automation；web security

0　引言

面对不断变化的互联网安全威胁，HTTPS 协议已提供了传输层安全保障，但在高安全需求场景中，基于前端 JavaScript 的应用层加密技术仍是不可或缺的防御手段，其部署与应用具有必要性和实际价值。此机制的设计意图并非重复传输层加密功能，而是在客户

端增设一道独立防线，其核心价值在于：减轻客户端明文数据泄漏的风险；防范内部威胁与日志泄漏，确保即使调试日志、系统监控数据意外暴露，内部人员也无法直接获取用户明文密码；以及防止基于网络数据包捕获的简单重放攻击。它迫使攻击者从低成本的自动化扫描转向高成本的人工逆向工程，即攻击者必须投入大量时间成本解构 JavaScript 加密逻辑，提取相关密钥并重演整个请求的加密过程。

然而，当前针对前端加密机制的安全评估与风险识别手段存在明显短板。现有方法高度依赖安全专家的人工逆向分析，存在三方面的缺陷：其一，效率低下，单次完整分析平均耗时数分钟至数十分钟，难以应对现代 Web 应用快速迭代、加密方案频繁更新的规模化评估需求；其二，结果质量严重受限于分析者的个人经验与领域知识，不同人员对同一混淆代码的分析结论可能存在显著差异；其三，面对 Webpack 模块化封装、闭包嵌套、动态密钥获取等现代前端工程化手段，人工逐行调试的认知负担急剧增大，极易遗漏关键逻辑。上述局限使得前端加密机制的风险评估长期处于手工模型，缺乏可复现、可扩展的技术手段。

大语言模型（Large Lauguage Model，LLM）在代码理解与生成领域的进展[1]，为突破这一瓶颈提供了可能，但同时也带来了新的风险维度。LLM 展现出的卓越代码推理能力，使得复杂的混淆 JavaScript 代码正面临被智能化、自动化分析的风险。一旦代码分析与逻辑提取的门槛被系统性降低，前端应用层加密所依赖的防御机制的假设将不再成立。

针对这一新型威胁，本研究从风险识别与安全评估的视角出发，提出了一种基于 LLM 的前端加密逻辑自动化分析框架。该框架的核心创新体现在两个层面：在分析方法上，通过动态钩子技术与 LLM 语义推理的协同，实现了从运行时加密行为捕获到混淆代码智能解析的端到端自动化，突破了传统人工分析在效率与规模上的瓶颈；在提示词设计上，引入可验证性约束与闭包作用域提升机制，使 LLM 的输出结果具备可直接验证和注入执行能力，弥补了通用 LLM 在实际可用性上的不足。通过在真实匿名平台上的实验，本研究重新审视了在生成式 AI 时代下前端加密机制的有效性和脆弱性，为大规模前端安全评估提供了一条可扩展的技术路径。

1 研究现状

1.1 前端加密的安全分析

前端加密通常利用 JavaScript[1]（或 WASM[2]）技术，在浏览器端对用户输入的敏感数据载荷进行加密处理。然而，这一机制受到浏览器执行环境不可信的安全限制。浏览器的本质决定了其具有高度的开放性与透明度。在这一白盒环境下，客户端拥有对代码逻辑、内存状态及运行时数据的完全控制权[3]。这种高度可观测性，使得加密逻辑、密钥以及函数调用链路暴露于逆向分析、运行时监视或篡改[1,4]的威胁下。

现有研究已广泛探讨了算法层面的实现。Abdulla[5]详细介绍了 AES 的对称加密，而 Milanov[6]概述了 RSA 的非对称方法，两者都指出密钥管理是一个关键弱点。Mainanwal[7]等人提出了零知识 RSA 协议，但同样承认浏览器环境的可观察性会带来安全约束。Edler[8]与 Chechet[9]等人的研究进一步强调密钥存储或传输过程中的任何问题都会削弱客户端加密的有效性。因此，前端 JavaScript 加密的脆弱性并非源于算法本身，而是密钥保护机制与开放执行环境之间的矛盾。这也印证了前端 JavaScript 数据包加密仅能提高攻击者的逆向成本[9]，而难以从根本上保证敏感明文不被解析。为提升这一成本，开发人员常采用代码混淆[10]、打包或 WebAssembly[11]技术。然而 Schrittwieser 等人[10]的评估表明，虽然这些技术增加了攻击者的工作量，但在具备动态调试能力（如任意脚本执行、Hoo 技术注入）的攻击者面前，这些混淆措施效果会被削弱。

1.2 前端加密分析方法

前文揭示的“防护效果削弱”问题在当前 Web 威胁态势下显得尤为紧迫。根据 Verizon 2024 年数据泄露调查报告（DBIR），在基本的 Web 应用程序攻击中，77%采用窃取凭证攻击方式，而 21%采用暴力破解方式进行攻击。面对这一态势，前端加密机制作为一种有效的对策，通过对请求载荷进行混淆与加密，成功阻断了基于中间人和简单流量重放的自动化攻击。这一防御层的存在，迫使攻击者的视角发生转移，即从网络层面的流量分析转向可观测性更高的客户端端点。

针对这一攻击面的分析方法主要分为静态分析[1]与动态分析[12]。静态分析通过直接审查 JavaScript 源代码来推断加密逻辑，而动态分析通过动态插桩技术拦截运行时信息，例如通过 Hook 技术拦截 Web Crypto API 调用、监视加密函数的执行堆栈或者捕获密钥参数。由于此方法能绕过复杂的语法混淆，因此在实证中更具实效性。针对“客户端请求劫持[13]”和运行时 Hook[4]的实证研究表明，这些利用客户端白盒特性的

方法在真实网络中具有显著影响。在现实环境中，攻击者往往结合浏览器开发者工具（DevTools）通过断点调试逐层剥离加密外壳。

然而，上述方法高度依赖分析人员的手动操作和既有经验，这种依赖性导致分析实践耗时冗长，严重制约了安全评估的可扩展性，构成了当前前端安全领域的效率瓶颈。

1.3 LLM 在代码分析中的兴起

近年来，LLM 的崛起推动了代码理解与生成领域的转变，显著突破了传统静态分析工具在处理复杂混淆时的性能瓶颈。LLM 强大的语义理解与推理能力，能够基于上下文推断代码意图，而不是依赖固定规则或语法树（AST）匹配。Beste 等人[14]验证了 LLM 对 JavaScript 去混淆的有效性。更重要的是，LLM 在数据流分析领域的应用为自动化分析提供了理论支撑。LLMDFA[15]框架的提出展示了 LLM 并非仅能处理文本生成，更能通过推理来精确识别代码中的值依赖关系。这种能力对于追踪加密函数的输入参数（如密钥、初始向量）至关重要。同时，Jelodar 等人[16]在 2025 年发表的系统性综述中指出，LLM 在源代码分析中的应用已覆盖代码理解、生成、摘要、漏洞检测等七大核心任务，在代码分析中展现出巨大潜力。这种技术上的跨越，意味着传统的、依赖专家经验的混淆代码分析工程正面临被智能化、自动化取代的可能。这也为后续构建自动化前端加密分析框架提供了坚实的技术基础。

2 自动化分析框架设计与实现

现有前端安全分析高度依赖人工操作，可扩展性受限。本文借助 LLM 在语义理解与代码推理方面的能力，提出并实现了一个基于 LLM 的自动化前端加密分析框架以提升分析效率。该框架构建了一个紧密的三阶段工作流，将动态运行时钩取、浏览器自动化技术与 LLM 驱动的代码语义分析进行了融合。其核心设计目标是建立一条从加密行为的实时捕获、逻辑解析到核心函数提取的自动化工作流，并在实证层面评估现有前端加密机制在智能化、自动化攻击威胁下的安全有效性。

2.1 系统架构与工作流程

本框架由浏览器自动化控制器（Browser Automation Controller，BAC）、加密钩子注入器（Crypto Hook Injector，CHI）和大语言模型辅助分析引擎（LLM-Assisted Engine，LAE）三大核心模块组成，整体架构如图 1 所示。

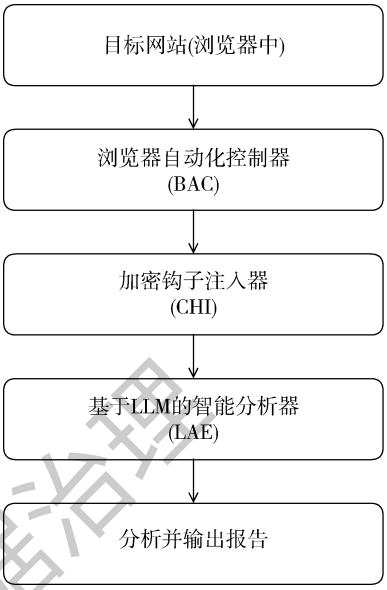

图 1　系统整体架构图

首先，BAC 负责初始化 Chrome 实例并导航到目标登录页面。为确保能够捕获早期加载阶段的加密逻辑，CHI 在文档对象模型（DOM）加载的最初阶段（document-start）即被注入。系统整体采用事件驱动机制：以带有特定前端［HOOK］的控制台输出作为信号触发器，一旦检测到加密函数调用的相关信号，系统便自动收集对应的 JavaScript 源码片段。随后，这些代码片段被结构化封装为特定格式的提示词（prompt），并提交给 LAE 进行深度解析。自动化提取过程可抽象为如下事件流，如式（1）所示：

$$E_i=(t_i,s_i,j_i,k_i) \tag{1}$$

其中 t_i 为时间戳，s_i 为调用堆栈，j_i 为 JS 源码片段，k_i 为密钥参数。所有采集到的事件 ε 被结构化为提示词集 P，输入 LAE 进行分析，如式（2）所示：

$$R=\mathrm{LAE}(P) \tag{2}$$

其中 R 为结构化分析结果。

自动化提取过程由算法 1 形式化描述。该算法在静默窗口 Δt 内持续监控新的加密活动信号。根据初步实证研究及主流 Web 自动化测试框架的基准，Δt 的设定能够覆盖 98%以上目标页面异步加载延迟，确保绝大多数前端加密行为都能够被及时捕获与分析。

算法 1　Automated Extraction and Synthesis

输入　target URL u，silence window Δt

输出　Algorithm A，Key_ source K，Encrypt_ wrapper EW，Decrypt_ wrapper DW，Usage_ example UE

1. launch Chrome with remote-debugging-port

```
2. navigate to u, inject CHI at document_start
3. Q ←Initialize message queue from Console
4. last_activity ← currentTime()
5. while (CurrentTime() - last_acitivity) < Δt do
6.         if Q is not empty then
7.                msg ← Q.dequeue()
8.                last_activity ← CurrentTime()
9.               if msg contains prefix "[Hook]" then
10.              src ← CollectJSSource(msg.url, msg.stack)
11.                       prompt ← ConstructPrompt(src)
12.                       EW, DW ← LAE(prompt)
13.                       if EW ≠ None then
14.                              return A, K, EW, DW, UE
15.                       end if
16.              end if
17.          end if
18. end while
19. return None
```

图 2 展示了 CHI 的内部逻辑，强调在无行为改变的情况下捕获元数据。

图 2　Hook_JS 代码内部逻辑图

2.2　动态编排与运动时钩取机制

本框架的自动化分析能力依赖于 BAC 与 CHI 的协同工作。BAC 基于事件驱动模型，担任系统核心编排器，负责浏览器全生命周期的任务调度。在浏览器初始化阶段，BAC 动态注入 CHI，并进入日志轮询状态。系统通过监听控制台输出，实时捕获带有［HOOK］前缀的控制台信号，一旦检测到相关事件，BAC 立即触发捕获程序，自动定位并提取相关 JavaScript 源码及其上下文信息。依据算法 1 定义的静默窗口机制，当在 Δt 时间内无新信号产生时，自动终止采集并进入后续分析流程。

CHI 作为运行时中枢，核心功能包括：

（1）调用堆栈捕获

每当加密相关 API（如 CryptoJS、JSEncrypt 等）被调用时，CHI 自动记录其 JavaScript 调用堆栈，形成集合 S，为后续加密逻辑追溯和分析提供数据基础，如式（3）所示：

$$S=\{s_1,s_2,\cdots,s_n\} \tag{3}$$

（2）外层封装 JS 文件抽取

CHI 通过解析调用堆栈中的 URL，自动识别并抽取最外层（即最终负责加密逻辑的）JavaScript 文件，记为 j_i。所有事件组成集合 J，用于后续源码分析与算法识别，如式（4）所示：

$$J=\{j_1,j_2,\cdots,j_n\} \tag{4}$$

（3）密钥与参数抓取

CHI 持续监听 DOM 变化，自动提取所有隐藏表单字段（input[type=hidden] 等）及其值，形成密钥集合 K 并对字段名进行敏感词检测（如“key”“iv”“token”等），实现密钥与参数的高效捕获，如式（5）所示：

$$K=\bigcup_{t\in[t_0,t_1]}\text{ExtractKeys}(\text{DOM}_t) \tag{5}$$

（4）事件结构化

对于每一次加密行为，CHI 将捕获到的时间戳、调用堆栈、最外层 JS 文件源码、密钥参数等信息结构化为事件 $E_i=(t_i,s_i,j_i,k_i)$，并将所有事件组成事件流 $\varepsilon=\{E_1,E_2,\cdots,E_n\}$。在采集窗口［$t_0$，$t_1$］内，最终获得的 J 和 K 分别为所有事件的并集，如式（6）和式（7）所示：

$$J=\bigcup_{i=1}^{n}j_i \tag{6}$$

$$K=\bigcup_{i=1}^{n}k_i \tag{7}$$

在实现层面，CHI 采用非侵入式挂钩技术，通过重写原生对象，在不破坏原有原型链的前提下建立捕获层。该机制确保页面原有业务逻辑的完整性，同时实现加密行为的高保真采集。

2.3　LLM 驱动的语义分析工程

LAE 构成了本框架的语义分析核心。针对前端混淆代码普遍存在超长上下文和复杂依赖问题，系统选用了支持 256K token 上下文的 Kimi（kimi-k2-0905-preview）模型作为基础推理模型，API 调用温度参数设为 0.3，以在保持输出确定性的同时保留一定的推理

灵活性。

在提示词设计层面，本模块针对前端加密逆向任务的特点进行了多项优化。为抑制模型分析混淆代码时常见的输出与实际行为不一致问题，提示词要求 LLM 在返回加密算法与密钥参数后，必须提供一组满足 encrypt（明文）= = =密文且 decrypt（密文）= = = 明文的自洽性加解密示例，这一可验证性约束将开放式的代码理解任务转化为结果可校验的推导过程。同时，考虑到前端加密逻辑普遍嵌套于 Webpack 模块闭包内，提示词在核心要求中明确规定“优先使用原始代码中的函数，只需提升至 windows 作用域”，并在输出格式中指定 global_functions 字段，引导模型将闭包内的加密、解密及辅助函数逐一暴露为全局可调用对象，从而省去了人工剥离闭包与执行环境的步骤。此外，面对动辄数十万字符的混淆代码，提示词通过“核心要求-分析步骤-输出格式”三层结构化引导实现信息聚焦，核心要求将分析目标限定于加密、解密函数及辅助函数，分析步骤将推理过程分解为定位加密核心函数、定位解密核心函数、识别辅助函数以及梳理函数依赖四个阶段，输出格式则以 JSON Schema 严格约束最终输出的字段结构，使得单次 LLM 推理即可将冗余源代码提纯为包含算法类型、密钥来源、可运行封装代码及使用示例的结构化蓝图，支持后续的自动化测试脚本生成。提示词摘录如下：

> 你是资深的网络安全专家，请分析以下前端加密代码。以下是一段前端 JS 代码，当用户点击登录时，会将密码加密后发送。
>
> 核心要求：
>
> 1. 找到代码中已存在的加密/解密函数（不要重新实现）；
> 2. 找到辅助函数（如时间戳生成、requestIds、签名等）；
> 3. 优先使用原始代码中的函数，只需提升到 windows 作用域；
> 4. 如果某些函数需要参数，提供默认值或从代码中提取；
> 5. 若源码中未提供解密函数，请严格根据加密函数反向实现解密函数，要求：必须保持完全相同的算法、模式、填充、编码、密钥派生逻辑；必须在返回的 JSON 里给出一条可验证的加解密示例（明文、密文、密钥、IV 等），确保复制粘贴后即可运行并验证‘encrypt（明文）= = = 密文’且‘decrypt（密文）= = = 明文’。

至此，本研究通过将 CHI、BAC 与 LAE 进行融合，成功构建了一个自动化的前端加密分析框架。这种混合架构确保了系统的可重复性与可扩展性，从而为第 3 节的实证验证奠定了方法论基础。

3 实例验证

本节通过实证数据评估前端加密机制的安全性现状，并检验本研究提出的自动化分析框架在实际环境下的效能。实验设计涵盖广度和深度两个维度：在广度层面，对近 80 所大学公开可访问的平台进行了初步检验，揭示其普遍采用的加密方式以及潜在的风险；在深度层面，选取 6 所大学（记为 U1～U6，均已去标识化）具有代表性的典型平台作为深入分析对象。

在受控实验环境中，部署第 2 节构建的基于 LLM 的自动化框架，开展对比实验。实验重点围绕加密入口识别、密钥提取以及加密逻辑重构三个关键阶段，评估框架的准确率和时间效率，验证 LLM 驱动自动化攻击在真实场景中的可行性。

3.1 实验设置与评估

本实验模拟红队场景，所有测试均限定在浏览器级别的访问权限，不涉及服务器修改或零日漏洞利用。该设计旨在真实反映攻击者利用浏览器可观测性进行请求伪造或数据泄露的实际威胁。为确保研究焦点明确，实验排除了网络层攻击（如 TLS 中间人拦截），严格限定于浏览器层面的安全分析。

实证评估测试流程包括以下三个步骤：

（1）运行时数据捕获：使用 Chrome 调试协议（CDP）注入 CHI，在用户登录等关键交互期间，实时记录加密函数的调用及运行时加密逻辑。

（2）加密与混淆识别：对捕获的 JavaScript 代码进行混合分析，以确定加密入口点、加密算法类型（如 AES、RSA）以及动态密钥生成模式。同时，依据混淆复杂度将样本按 1～4 的等级评分[17]，其中，Level 1 为轻度混淆如语法级混淆，常采用标识符重命名、间接属性访问、字符串字面量混淆等；Level 4 表示重度混淆，采用了多层混淆策略。

（3）自动化逻辑恢复：将预处理后的代码片段输入 LAE，通过结构化提示词引导模型恢复关键的函数名称和核心加密逻辑。

为确保研究的合规性，所有测试均严格限定于非生产环境，遵循相关伦理准则和数据保护要求。

表 1 中的对比性分析证实了前端 JavaScript 加密机制存在高度的相似性。实验样本所示，多数平台采用 CryptoJS 或 JSEncrypt 等主流开源库来实现 AES 或 RSA 加密算法，其底层实现逻辑普遍源自相同的开源模板。

表 1　各大学身份认证平台加密机制比较

平台代号	加密算法	混淆复杂度	密钥位置
U1	国密 SM2	Level 1	公钥编码在 HTML 表单中
U2	RSA-PKCS1	Level 2	公钥编码在 HTML 表单中
U3	无加密	无	无
U4	3DES-CBC	Level 1	公钥编码在 HTML 表单中
U5	AES-CBC+Base64	Level 2	动态密钥编码在 HTML 中
U6	RSA-PKCS1	Level 1	动态公钥但可获得

更为严重的是，在前五个样本中密钥均直接暴露于前端浏览器环境，第六个样本虽然通过接口请求获取公钥，但攻击者可构造请求包实现自动化提取。尽管都尝试隐藏密钥引用，但密钥仍被成功提取。

在混淆策略方面，尽管 U2、U5 平台采用了中等强度的代码混淆技术，LLM 驱动的语义分析依然能够在短时间内恢复主要核心函数名称与核心加密逻辑。

这些实验结果揭示：随着 LLM 能力的持续跃升，传统前端 JavaScript 加密机制在高校统一身份认证系统中所能提供的实际安全增益已大幅削弱，其对整体防护水平的提升效果愈发有限。

3.2　实验结果与分析

基于 3.1 节的案例研究结果，进一步运用第 2 节构建的 LLM 驱动自动化框架，在六个匿名平台（U1~U6）上开展受控实验。为保障实验的一致性，所有测试均在一个统一且受控的环境中进行，具体配置如下：硬件环境为一台配备 AMD Ryzen 9 7945HX 处理器、32 GB 内存的电脑，操作系统为 Windows11；软件层面采用 Chrome 浏览器（v1200.6099.130）、通过 DrissionPage 库驱动浏览器并注入 Hook 脚本，并以 Kimi-k2-0905-preview 模型作为 LAE 模块推理核心。所有自动化测试均在同一网络条件下进行，每个样本重复执行 3 次取平均结果。

表 2 详细记录了该自动化框架在 U1~U6 样本上的执行结果，涵盖捕获到的加密类型、调用文件、密钥来源及密钥捕获情况。框架在所有检测到加密行为的样本上均实现了完整捕获：U1 为国密 SM2[18]，公钥在 login 页面中捕获，Hook 脚本通过拦截 setPublickey 调用并扫描页面隐藏字段捕获了公钥值及调用文件；U2、U4、U5 分别采用 RSA-PKCS1、3DES-CBC、AES-CBC+Base64，密钥均源自 login 页面中捕获，框架直接通过 JSEncrypt.encrypt、setPublicKey 或 CryptoJS.AES.encrypt 等钩子截获密钥参数，并定位到具体调用文件；U3 未检测到加密行为，与人工核实结果一致。U6 为 RSA-PKCS1 加密，公钥需异步请求/rsa 接口获取，框架的 Hook 脚本对 JSEncrypt.prototype.setPublicKey 进行了运行时封装，成功提取了完整的动态公钥及其调用链信息，验证了框架对动态公钥获取场景的有效覆盖。

表 2　自动化框架检测结果（去标识化）

平台代号	检测类型	调用文件	密钥来源	结果
U1	国密 SM2	/dist/sm2Utils.js	动态公钥	已捕获
U2	RSA-PKCS1	/static/common/jsencrypt.js	固定硬编码密钥	已捕获
U3	无	无	无	未捕获
U4	3DES-CBC	/comm/js/des.js	固定硬编码密钥	已捕获
U5	AES-CBC+Base64	/custom/js/encrypt/aes.js	固定硬编码密钥	已捕获
U6	RSA-PKCS1	/comm/js/login_stander_new.js	动态公钥	已捕获

为进一步辨析自动化插桩分析与纯 AI 推理的能力差异，本研究补充了纯 AI 分析的受控实验。该实验通过定制爬虫完整抓取六个目标平台前端全部 HTML 和 JavaScript 源代码，合并后统一提交至同一 LLM（Kimi-k2-0905-preview），要求模型识别加密算法、密钥来源及完整的加密流程。分析结果如表 3 所示。在 U1、U2、U4、U5 固定编码密钥的样本上，纯 AI 分析能直接从源代码中提取出算法类型和具体密钥值，准确率与自动化框架相当；但在 U6 的动态密钥场景下，源代码中仅存在 fetch('/rsa') 的请求逻辑，密钥值依赖运行时返回，LLM 无法从静态文本中获取实际公钥，因此未能完成端到端的密钥复原以及完整的加解密函数识别，突显了纯 AI 分析在面对运行时依赖时的固有局限。

表 3　纯 AI 静态分析结果（去标识化）

平台代号	检测类型	调用文件	密钥来源	结果
U1	国密 SM2	/dist/sm2Utils.js	动态公钥	已捕获
U2	RSA-PKCS1	/static/common/jsencrypt.js	固定硬编码密钥	已捕获
U3	无	无	无	未捕获
U4	3DES-CBC	/comm/js/des.js	固定硬编码密钥	已捕获
U5	AES-CBC+Base64	/custom/js/encrypt/aes.js	固定硬编码密钥	已捕获
U6	RSA-PKCS1	/comm/js/jsencrypt.min.js	动态公钥	未捕获密钥

为评估框架相对于传统静态反混淆工具的实用性，开展与静态反混淆工具的对比实验。采用 de4js 依次对从六个样本中提取出的加密相关 JS 代码进行反混淆与可读性还原处理。de4js 能够处理较简单的混淆模型，但对于 U1 和 U6 中采用现代 Webpack 模块化封装或嵌套三元运算，反混淆后仍残留大量不可读变量名。相比之下，本框架首先凭借 BAC 模块在运行时拦截加密参数调用，绕过了对混淆代码的完全解耦；更重要的是，LAE 模块利用 LLM 强大的代码理解能力，对捕获到的混淆上下文进行语义级推理，即使代码未能彻底反混淆，也能准确提取加密逻辑，实现了较传统工具更高的定位效率和准确性。

此次受控实验揭示了前端加密机制存在脆弱性。首先，CHI 在五个有加密行为样本下均实现了对完整加密过程（包括密钥初始化、加密算法及文件调用）的实时捕获，尤其在 U6 这类通过异步接口获取密钥并运行时调用 setPublicKey 的场景下，凭借对加密库原型方法的 Hook，框架仍能完整截获动态密钥，展现出相比纯静态分析的显著优势。相比之下，纯 AI 分析因缺乏运行时状态无法获取动态密钥；传统静态反混淆工具面对现代混淆时往往卡在代码还原阶段，难以直接锁定密钥逻辑。本文提出的框架则通过运行时插桩直接定位执行路径，绕过了混淆障碍，大幅提高了逻辑定位的效率。此外，实验验证了 3.1 节中发现的安全隐患：多数平台存在“前端公开密钥或全局变量泄露”的问题。自动化框架不仅能够静态识别这些特征，更能在运行时准确捕获并利用相关信息，实现对加密逻辑的重现。

为了对比 AI 协同分析所带来的效率提升，设计并执行了多方案对比测试。将本混合框架与纯手动分析（由两名经验丰富的测试员执行，取平均值）、纯 AI 分析（即前述全量源代码 LLM 分析）以及静态反混淆工具（de4js 处理并辅以人工理解）进行对比。此次测试结果展示了四者之间的显著差异。手动分析平均耗时 502.5 s（置信区间：463～542 s），纯 AI 分析平均耗时 292 s（置信区间：264～320 s），静态反混淆结合人工解读平均耗时 482 s（置信区间：429～535 s），而自动化框架仅需 135.5 s（置信区间：86～185 s），如图 3 所示。纯 AI 分析平均耗时之所以远超预期，系因各平台前端源代码体量巨大，加上统一提交时拼接的分析提示词，单次 LLM 请求需处理上下文极长，推理时间显著膨胀，尽管如此，其耗时较手工分析降低约 41.89%，但缺陷在于面对 U6 等动态密钥场景时无法提取有效密钥。静态反混淆结合人工解读的总耗时与纯手动分析基本持平（仅缩短约 4.08%），说明传统工具其自动还原效果有限，后续仍需大量人工介入。相比之下，本框架依靠运行时 Hook 精准截获加密参数，仅将小范围的代码上下文交由 LAE 模块推理，从而将分析耗时压缩，较纯手动分析提速约 73.03%，较纯 AI 分析提速约 53.60%。这一结果印证了本研究的核心论点：随着 LLM 驱动的自动化攻击工具的成熟，传统前端加密机制正面临系统性的失效风险。

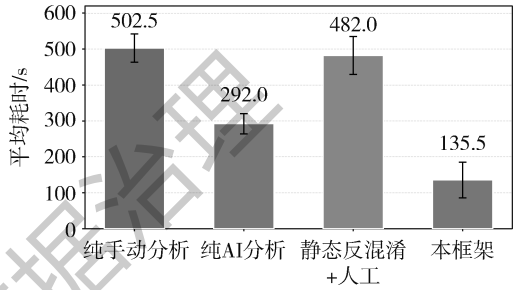

图 3　各方案前端加密分析耗时对比

为验证本框架对底层大语言模型的通用性，即提示词设计策略与运行时捕获机制是否能在不同模型上均保持有效性，本实验在保持 CHI 和 BAC 模块不变的前提下，分别替换 LAE 推理引擎为 DeepSeek-V4-pro、Qwen3.6-flash 两种国内模型，在六个样本中重复加密分析实验，使用完全相同的提示词模板与 Hook 代码。评估指标包括输出可验证性（即提供的加解密示例是否自洽）以及平均分析耗时，结果如表 4 所示。

表 4　不同 LLM 模型的加密分析结果对比

模型	密钥捕获成功率/%	验证通过率/%	平均耗时/s
Kimi-k2-0905-preview	100	100	135.5
DeepSeek-V4-pro	100	100	325.2
Qwen3.6-flash	100	66.7	70.6

由表 4 可见，三种模型在密钥捕获成功率上均达到了 100%，表明框架的运行时 Hook 机制与提示词中的“优先使用原始函数并提升至 windows 作用域”策略在不同模型上均能有效引导 LLM 正确定位加密入口及密钥参数，密钥捕获环节不依赖于特定模型能力。

在输出可验证性方面，DeepSeek-V4-pro 通过率为 100%，与本文中所使用的 Kimi 表现一致；Qwen3.6-plus 则仅为 66.7%，有两个样本未能编写正确的解密函数，说明该模型在加密算法反向推导的准确性上存在一定波动。这一差异恰好验证了本框架引入可验证性约束的必要性。在耗时维度上，Qwen3.6-plus 的平均耗时最短，但综合可验证性通过率来看，其速度优

势以牺牲部分准确性为代价；DeepSeek-V4-pro 平均耗时最长，但准确率较高。

上述结果表明，本框架的核心设计在不同 LLM 上均能维持密钥捕获能力，具备良好的模型通用性，但不同模型在可验证性通过率和响应稳定性上的差异，也表明提示词中的自洽性约束对筛选可靠分析结果具有重要价值。

4 结论

本研究证实了一个核心论点：在 LLM 驱动的自动化分析面前，传统依赖混淆技术的前端 JavaScript 加密机制存在脆弱性。为验证这一论点，本文提出并实现了一种融合动态运行时钩取与 LLM 语义推理的自动化分析框架。实验结果表明，该框架能高效解析多数现有加密实现方案，并在分析效率上较人工专家提升约 73.03%，显著降低了前端代码审计的技术门槛。跨模型对比实验进一步表明，框架的运行时捕获与结构化提示词策略具备较好的模型通用性。

同时，本研究也认识到方法的边界与局限性。当前框架的有效性主要集中于标准的 JavaScript 环境，对于编译型的 WebAssembly 模块或涉及复杂数学运算的后量子密码学库，自动化分析的精度与适用性仍有限。此外，LLM 的随机性导致输出可能包含语法正确但语义错误的代码，特别是在处理极高强度的对抗性混淆时，仍需保留人机回环的验证机制。这些局限反映了网络安全领域持续攻防对抗的特性。未来的研究工作将致力于扩展框架对 WASM 二进制分析的支持，并探索将可验证性约束从被动校验升级为自动纠错的反馈闭环，以适应日益复杂的 Web 安全威胁。

参考文献

[1] 史橹，吴毅坚，赵文耘. JavaScript 代码分析技术综述[J]. 计算机应用与软件，2018，35(11)：16-25.

[2] 庄骏杰，胡霜，华保健，等. WebAssembly 安全综述[J]. 计算机研究与发展，2024，61(12)：3027-3053.

[3] Kambhampati V，Mohammed N H，Fard A M. Characterizing JavaScript security code smells[EB/OL]. (2024-11-28). https://arxiv.org/abs/2411.19358.

[4] Li Zhe，Xie Fei. Concolic testing of front-end JavaScript[C]// International Conference on Fundamental Approaches to Software Engineering. Cham：Springer Nature Switzerland，2023：67-87.

[5] Abdullah A M. Advanced encryption standard (AES) algorithm to encrypt and decrypt data[J]. Cryptography and Network Security，2017，16(1)：11.

[6] Milanov E. The RSA algorithm[J]. RSA Laboratories，2009，1(11).

[7] Mainanwal V，Gupta M，Upadhayay S K. Zero knowledge protocol with RSA cryptography algorithm for authentication in web browser login system (Z-RSA)[C]//2015 Fifth International Conference on Communication Systems and Network Technologies. New York：IEEE，2015：776-780.

[8] Edler D. Client side encryption in the browser and key management[D]. Darmstadt：Technische Universität Darmstadt，2017.

[9] Chechet A S，Chernykh M V，Panasiuk I S，et al. Front-end security architecture：protection of user data and privacy[J]. Systems and Technologies，2024，68(2)：102-111.

[10] Schrittwieser S，Katzenbeisser S，Kinder J，et al. Protecting software through obfuscation：can it keep pace with progress in code analysis?[J]. ACM Computing Surveys，2016，49(1)：1-37.

[11] Bhansali S，Aris A，Acar A，et al. A first look at code obfuscation for webassembly[C]//Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks. New York：Association for Computing Machinery，2022：140-145.

[12] Kim H C，Choi Y H，Lee D H. JsSandbox：a framework for analyzing the behavior of malicious JavaScript code using internal function hooking[J]. KSII Transactions on Internet and Information Systems，2012，6(2)：1-15.

[13] 赵国锋，陈勇，王新恒. 针对 HTTPS 的 Web 前端劫持及防御研究[J]. 信息网络安全，2016(3)：15-20.

[14] Beste D，Menguy G，Hajipour H，et al. Exploring the potential of LLMs for code deobfuscation[C]//International Conference on Detection of Intrusions and Malware，and Vulnerability Assessment. Cham：Springer Nature Switzerland，2025：267-286.

[15] Wang Chengpeng，Zhang Wuqi，Su Zian，et al. LLMDFA：analyzing dataflow in code with large language models[J]. Advances in Neural Information Processing Systems，2024，37：131545-131574.

[16] Jelodar H，Meymani M，Razavi-Far R. Large language models (llms) for source code analysis：applications，models and datasets[EB/OL]. (2025-03-21). https://arxiv.org/abs/2503.17502.

[17] Zhou Dongchao，Ying Lingyun，Chai Huajun，et al. From obfuscated to obvious：a comprehensive JavaScript deobfuscation tool for security analysis[EB/OL]. (2025-12-22). https://arxiv.org/abs/2512.14070.

[18] GM/T 0003.2-2012 SM2 elliptic curve public key cryptography algorithm[S]. Beijing：China Cryptography Administration，2012.

（收稿日期：2026-05-07）

作者简介：

李金龙（2002-），男，本科，主要研究方向：多智能体协同渗透测试、网络信息安全。

赵新元（1974-），通信作者，男，博士，副教授，主要研究方向：人工智能应用、网络信息安全。E-mail：1621000@qq.com。

版权声明

凡《网络安全与数据治理》录用的文章，如作者没有关于汇编权、翻译权、印刷权及电子版的复制权、信息网络传播权与发行权等版权的特殊声明，即视该文章署名作者同意将该文章的汇编权、翻译权、印刷权及电子版的复制权、信息网络传播权与发行权授予本刊，本刊有权授权本刊合作数据库、合作媒体等合作伙伴使用。同时，本刊支付稿酬已包含上述使用的费用，特此声明。

《网络安全与数据治理》编辑部