经过身份验证的工作流：保护智能体人工智能的系统化方法

Mohan Rajagopalan
MACAW Security, Inc.
mohan@macawsecurity.com

Vinay
Rao*ROOST.tools*
vinay@roost.tools

arXiv:2602.10465v1 [cs.CR] 11 Feb 2026

摘要——**智能体人工智能系统可自动执行企业工作流，但现有防御措施——护栏和语义过滤器——具有概率性，且经常被绕过。我们提出了经过身份验证的工作流，这是首个面向企业级智能体人工智能的完整信任层。安全问题归结为保护四个基本边界：提示、工具、数据和上下文。我们在每次边界跨越时强制确保意图（操作符合组织策略）和完整性（操作具有密码学真实性），并结合以密码学方式消除攻击类别与运行时策略执行。这带来了确定性安全——操作要么携带有效的密码学证明，要么被拒绝。我们推出 MAPL，这是一种原生人工智能策略语言，可随着智能体的演进和调用上下文的变化，动态表达智能体约束；通过与工作流依赖项的密码学证明相结合的层级组合，它能够以 O(log M + N) 条策略扩展，相较之下，规则数量为 O(M×N)。我们通过一个通用安全运行时证明了其实用性：该运行时通过轻量适配器集成了九种领先框架（MCP、A2A、OpenAI、Claude、LangChain、CrewAI、AutoGen、LlamaIndex、Haystack），无需对协议进行任何修改。形式化证明确立了完备性和可靠性。实证验证显示，在 174 个测试用例中召回率达到 100%，且没有误报；能够防范 OWASP 十大安全风险中的 9 项，并完全缓解了两个影响严重的生产环境CVE漏洞。**

1. 引言

企业在将智能体人工智能系统部署到生产环境时面临重重困难。尽管这些系统有望自动处理复杂工作流——例如管理金融交易、患者记录和关键基础设施——但它们也带来了现有防御措施无法应对的安全挑战。例如，在 OpenAI Atlas 浏览器发布后的数小时内，研究员就证明，嵌入网页内容中的恶意指令可能会诱使助手窃取凭据 [1]。OpenAI 的首席信息安全官承认，“提示注入仍是一个尚未解决的问题” [2]。

这一挑战远不止提示注入。大语言模型无法区分指令与数据。非确定性的执行路径既无法预测，也无法进行静态分析。多轮交互会逐渐操纵上下文，而每条消息看起来都无害。现有方法要么寻找模式，要么依赖概率——这两种方法都需要枚举攻击，导致防御者难以应对未知的未知。

我们通过系统化的方法消除未知的未知：不再逐一罗列攻击手段，而是对系统进行确定性约束。我们的做法基于这样一个认识：智能体系统可以抽象为简单的拜占庭式分布式系统——多个实体在明确定义的边界间交互。因此，安全问题归结为保护边界跨越。从概念上说，我们的方法旨在确保每次边界跨越都满足两项属性：理解**意图**，确保操作符合组织策略；以及执行**完整性**验证，确保操作真实且未被篡改。两者缺一不可，单独任何一项都不足以保障安全。这种设计将智能体工作流转化为确定性分布式系统，其中边界由策略守护，跨越过程受到密码学保护。要攻破系统，就必须攻破密码学，而不是设计巧妙的提示。身份欺骗、会话重放、策略替换等许多类别的攻击可按设计消除——要想得逞，攻击者必须攻破密码学原语；而未授权数据访问、权限提升、凭据窃取等其他攻击则会在运行时被策略阻止。从架构层面看，**按策略执行**与**按设计消除**相辅相成，共同构成可组合的纵深防御。

我们研究了 100+ 智能体应用，并确定了几项关键挑战：**(1) 四个同时存在的攻击面**——工具、数据、提示、上下文——攻击会跨攻击面组合，实现任何单一攻击面都无法达成的目标。**(2) 异构框架**——智能体基于各种协议（MCP、A2A）、大语言模型接口（Claude、OpenAI）和编排器（LangChain、CrewAI、Au- toGen）构建。异构性是生态系统的长期特征。**(3) 组合式安全漏洞**——针对单个框架的检查会遗漏跨组合的违规行为。**(4) 开发者负担**——要求开发者在异构框架的每次交互中都加入安全逻辑，难以切实执行。

综合来看，攻击面扩大的速度超过了工具应对问题的能力，这凸显出一个严重缺口：人工智能信任层——一种位于应用层防御（无法组合）与基础设施原语（缺乏工作流语义）之间的安全基础设施。

为解决（1），我们引入**已认证工作流**——一种协议级原语，在每次交互中强制保障意图与完整性。我们证明，4 个攻击面

是最小且完备的。协议层定位解决了异构性（2），并消除了组合缺口（3）。通过为 9 种流行框架提供轻量封装，我们减轻了开发者负担（4），为企业级人工智能信任层奠定了基础。

经过身份验证的工作流结合密码学、零信任原则和运行时策略执行，确保每项操作都真实、经过授权且已获得证明。我们提出了一种新颖的设计：将分布式策略执行点（PEP）嵌入控制面，通过独立验证密码学证明，实现亚毫秒级开销，且无需集中式基础设施。每个控制面都会在执行前独立验证操作，从而构建纵深防御，即使某个控制面遭到攻破，也不会危及其他控制面。

为明确意图，我们推出 MAPL，这是一种面向 AI 的全新策略语言，能够让用户以动态、可扩展的方式表达智能体约束，即使智能体不断演进、调用上下文发生变化，也能适用。MAPL 提供三项关键能力：支持继承的可组合语法，使策略能够通过交集语义分层组合；大幅简化规范，通过层级组合和隐式主体解析，将规模从 $O(Musers \times Nresources)$ 缩减至 $O(\log M + N)$；以及通过证明实现动态状态——以签名声明证明操作已完成，从而支持诸如“仅在存在匿名化已完成证明后才导出数据”这类策略，并确保其具备密码学可执行性。这些进展克服了传统策略系统（OPA、AWS Cedar）在智能体环境中的局限。

我们的重点是针对操作边界——工具调用和数据检索——提供控制流保护，以补充 Rajagopalan等人 [3]，对流经提示和上下文的数据的保护。两者结合，为企业 AI 提供正式的信任层。我们通过一个通用安全运行时证明了其实用性：它通过轻量适配器（200–500 行代码）支持九种框架（第 5 节），无需更改协议，彰显了框架无关安全。

从形式上看，我们证明了这四个边界既完备又精简，分布式执行是可靠的，并且策略组合能够保持安全属性。从实证角度看，我们通过全面的攻击测试和生产环境CVE分析验证了这些属性。

当前的智能体防御——护栏、提示过滤器、语义分析（模式匹配、机器学习模型、启发式方法）——优先追求精确率，却只能达到较低甚至为零的召回率。对抗性提示、编码技巧和新颖模式常常利用语义盲点绕过这些防御，而假阳性则会阻止合法操作。相比之下，我们的密码学执行能够保证高精确率和高召回率：操作要么携带符合策略要求的有效签名，要么被直接拒绝。要攻破该系统，必须击破密码学原语，而非设计巧妙的模式。由此可实现完整的违规检测、零假阳性，并凭借计算困难性提供确定性信任。**贡献**。 我们提出了经过身份验证的工作流——一种协议级原语，可强制在

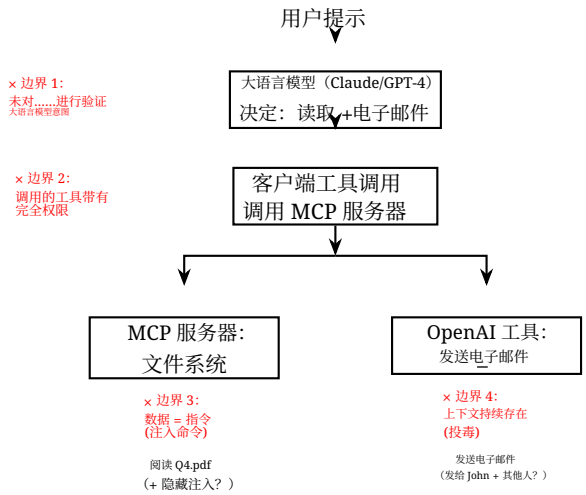


图1。 展示跨越四个边界的攻击级联的智能体工作流 界。

四个控制面（提示、工具、数据、上下文）上进行密码学验证，并已证明其完备且精简（引理6）。我们引入了MAPL，这是一种原生人工智能策略语言，通过证明提供密码学工作流依赖，并借助层级组合将策略规范从$O(M \times N)$降低到 $O(\log M + N)$（定理1-3）。我们通过轻量适配器（200-500代码行）实现了在九种异构框架（MCP、A2A、OpenAI、Claude、LangChain、CrewAI、AutoGen、LlamaIndex、Haystack）上的通用部署，无需修改协议；分布式策略执行点嵌入每个边界，以亚毫秒级开销提供零信任验证。形式化证明确立了完备性和可靠性（引理1-7）；实证验证则表明，该方案在涵盖OWASP 十大安全风险中九项的 174 个测试用例中，实现了确定性保证（召回率100%，假阳性0%），并完全缓解了两个生产环境CVE漏洞（OpenAI Atlas、GitHub MCP）。

2. 问题：智能体如何工作

要理解智能体系统为何难以保障安全，请看图1，其中展示了调用以下指令时涉及的实体：“读取第四季度财务文件，并将摘要通过电子邮件发送给 john@company.com。”用户的提示连同指令一起传递给大语言模型。大语言模型进行推理，并决定调用文件系统读取和电子邮件发送操作——这些操作可以通过 MCP 工具、OpenAI函数调用或自定义实现来完成。这些工具调用从文档存储中检索数据。在执行这个多轮工作流时，智能体会维护应用级状态，以跟踪交互历史。

这些交互跨越四个基本边界。**S1—提示** 将指令传递给大语言模型，引导其推理和工具选择。**S2—工具** 执行特权操作——文件系统访问、数据库查询、API调用、邮件发送。**S3—数据** 从外部来源流入智能体推理过程——文档存储库、RAG语料库、向量

数据库、网页抓取。**S4—上下文** 在多轮交互中维持对话状态。一个根本性挑战是：大语言模型对提示和文档内容一视同仁地进行处理——Q4.pdf中的恶意指令与用户请求无法区分。系统依赖隐式信任运行：上下文被视为有效，工具被假定已获授权，数据源则被视为无害。这些边界与框架无关——LangChain链和CrewAI团队都体现为提示→大语言模型→工具序列，且都会跨越相同的四个界面。这种普遍性使协议层安全成为可能，并能在异构系统中统一发挥作用。

任何表面都可能成为攻击入口，并导致影响在其他表面间层层扩散。在 OpenAI 的 Atlas 攻击中， [1], 嵌入数据（S3）中的恶意指令流入大语言模型推理（S1），进而生成工具调用（S2）以窃取凭据；这一切都作为正常执行记录在上下文（S4）中。不同攻击会通过不同表面进入。我们通过威胁模型对此进行形式化。

威胁模型。我们假设对手能力强大，并通过对手能力和信任假设对威胁模型进行形式化。

对手能力 A1—应用控制：对手可能控制应用层代码。**A2—内容注入：**他们会将恶意内容注入数据源、提示或上下文。**A3—组件失陷：**他们可能攻陷组件并获取私钥。**A4—网络攻击：**他们会拦截、重放或修改网络流量。**A5—组合式攻击：**他们会在多次交互中实施渐进式攻击，或跨越框架边界发动攻击；其中每项操作看似都经过授权，但合在一起却违反了策略。

信任假设 L1——密码学困难性：密码学原语提供计算上的困难性——对手无法在没有私钥的情况下伪造签名。**L2——受信任的控制平面：**控制平面服务（智能体注册表、策略存储库、日志记录、路由）在精简的可信计算基中运行，并具备密码学完整性保证（审计日志采用哈希链，策略存储库采用默克尔树）和管理访问控制。**L3——执行完整性：**在框架边界处，策略执行点验证逻辑能够正确执行。在企业部署中，框架不可变且操作均通过经过插桩的 API 路由，因此拥有应用控制权（A1）的对手可以操纵业务逻辑，但无法绕过框架 API 或破坏策略执行点的记忆。绕过策略执行点需要攻陷框架二进制文件——这相当于内核失陷，不在讨论范围之内。这与操作系统内核安全的情形类似。系统级攻击（调试器、内存破坏）不在讨论范围之内——即便此类攻击绕过了本地策略执行点，远端仍会独立验证调用，从而将对手限制在其应用范围内，无法获得新的权限。

安全目标针对这一威胁模型，我们确立五项安全目标：**O1—完整性：**操作真实且未经篡改。**O2—策略执行：**所有操作均符合组织策略。**O3—权限不提升：**组合后的策略不能授予比个体策略更广泛的权限。**O4—上下文完整性：**会话状态在交互过程中具有防篡改性。**O5—问责性：**所有操作均有不可否认的审计追踪。第7节证明，在假设 L1-L3 成立的情况下，经过身份验证的工作流能够抵御具备 A1-A5 能力的对手，并实现 O1-O5。

范围我们聚焦于协议级授权：通过密码学手段强制确保每项操作都符合组织策略。我们提供实现机制，策略由最终用户提供。应用级安全（例如检测恶意代码）需要领域特定语义，因此不属于协议范围。我们将所有应用程序都视为不可信，并要求其遵守系统级策略。

当前方法都是被动响应式的——检测模式、过滤输入、对执行进行沙箱隔离。每出现一种新的攻击变体，都需要新增检测规则；面对无界的攻击空间，这会造成枚举问题。这些对手能力（A1-A5）针对四种攻击面：应用控制和内容注入会危及提示/工具/数据，而组合式攻击则利用多框架漏洞。**A1, A3, A4** 会使基于可观测性的解决方案失效。近期针对**A2**和**A4**的定制模型研究 [4]采用概率性方法，无法提供确定性的安全保证。

我们的方法在协议层，智能体→工具、智能体→大语言模型和智能体→数据操作看起来完全相同——都是一次边界跨越。我们的方法将这些操作分别转换为已认证工作流，并使其受策略约束（O2）、通过授权验证（O1、O3、O4），且在执行时提供证明（O5）。下一节（第 3 节）将介绍一种可执行 O3 的策略代数。按设计机制（加密签名、哈希链）可应对组件失陷（A3）和网络攻击（A4）。按策略机制（在边界处进行运行时验证）可应对应用控制（A1）、内容注入（A2）和渐进式攻击（A5）。第7节将证明，操作要么携带符合策略的有效密码学证明，要么在执行前被阻止——这是确定性保证，而非概率性检测。

重新审视示例攻击：每个边界都由独立策略保护。即使推理被破坏，受限文档仍无法访问。遭到入侵后，应用程序的行为仍受策略许可范围的限制。要攻破整个系统，就必须同时攻破所有强制执行层——这在计算上十分困难。

这个问题可以分解为两个问题：如何规范用于治理边界的策略？如何在运行时验证完整性？我们引入 MAPL（第 3 节）来表达意图，并引入经过身份验证的工作流（第 4 节）来强制保障完整性；这些机制通过通用信任层（第 6–7 节）实现。

3. 意图规范：MAPL策略语言

我们提出 MAPL，这是一种用于声明意图的原生人工智能策略语言。智能体系统具有动态性（智能体会改变身份、生成子智能体、委派能力）和规模性（每位用户的规模增长 10 至 100 倍），这会使传统策略引擎失效。三个特性对现有系统构成挑战：**上下文身份**——智能体会根据运行时上下文代表不同主体

，因此需要动态解析主体，而非依赖静态的 WHO 绑定。**双重视角**——要维护完整性，就必须分别验证每次调用是否符合调用方意图和资源约束；传统策略将调用方与资源紧密耦合，一旦调用方遭到入侵，攻击者便可获得资源访问权限。**可验证的工作流状态**——多步骤工作流需要顺序执行策略；由于应用程序报告的状态不可信，我们需要某种证明机制，以验证操作 A 已完成后再调用操作 B。

MAPL 通过三项设计选择来应对这些问题：(1) 在运行时根据经过身份验证的上下文推断主体——无需更新策略即可实现动态身份；(2) 分别表达调用方策略和资源策略，并通过交集进行组合——无需协调即可实现纵深防御；(3) 策略引用经过密码学验证的证明——使工作流依赖关系可证明。这些设计结合起来，构成了一个完整、可证明且可组合的意图表达模型。

策略语法。 MAPL（MAPL 智能体策略语言）策略遵循结构化语法，支持机器验证和自动组合：

策略 { policy_标识符： <唯一_标识符>, extends: <父级_策略_标识符>, resources: [<资源_模式>], denied_资源： [<拒绝_模式>], 约束：{ 参数：{<资源>:{<参数>: [<模式↪>]}}, denied_参数：{<资源>:{<参数>: [<↪ 模式>]}}, 证明： [<所需_证明_名称>]}

MAPL 策略包括：策略 *ID* 用于组合链和密码学绑定的唯一标识符，*extends* 用于层级组合的父策略引用，*resources*允许的操作模式；通配符 */** 分别匹配单层/递归层级，拒绝的资源 用于覆盖允许项的显式阻止规则，以及约束 （参数、拒绝的参数、模式和证明要求）。附录 A 展示了具体策略。

层级组合。 **extends**字段支持策略通过交集语义继承父策略。这使组织策略能够分层（基础→ 部门 → 团队），每个子策略都通过添加限制来细化其父策略，而绝不放宽约束。当一个策略继承另一个策略时，有效权限是父策略与子策略的交集——从而保持单调限制，具体定义如下文的组合代数。

表达能力。最小语法通过四种机制即可满足需求：（1）带通配符的层级资源可表达无界命名空间；（2）正向和负向规范可表达任意布尔组合；（3）参数约束可表达关于参数的任何可判定谓词；（4）证明通过密码学状态实现顺序约束。这涵盖组织层级、双视角防御、工作流依赖、例外模式和参数控制。

组合代数。我们形式化策略如何在运行时组合，并证明系统满足关键安全属性。

策略结构与操作。
策略 $P = (R, D, C)$ 由以下部分组成：

- R：允许的资源模式
- D：拒绝的资源模式
- C：操作约束（参数、证明）

策略交集 $P_1 \cap P_2 = (R',\ D',\ C')$ 其中：

- $R' = R_1 \cap R_2$ （仅当两项策略都允许时才允许）
- $D' = D_1 \cup D_2$ （任一项策略禁止时即拒绝）
- $C' =$ 最严格$(C_1,\ C_2)$ （应用最严格的约束）

最严格$(C_1,\ C_2)$ 对数值限制取最小值，对允许模式取交集，对必需证明和拒绝模式取并集。

策略执行点在运行时组合来自组织层级与双重视角的策略。这确保操作只能增加限制，而绝不能放宽限制。若某项策略缺失，则它不施加任何约束（最宽松解释，且不违反任何已声明的约束）。

形式化安全属性
定义权限函数：

$$\pi(P) = \{(r, op) \mid r \notin \text{匹配}(D) \wedge r \in \text{匹配}(R) \wedge op \text{ 满足 } C\}$$

定理 1（单调限制）： 对于组合 $P_0 \cap \ldots \cap P_n$：

$$\forall i, j : (i < j) \Rightarrow \pi(P_0 \cap \ldots \cap P_j) \subseteq \pi(P_0 \cap \ldots \cap P_i)$$

证明概要：根据构造，$P_{i+1} = P_i \cap P_{\text{next}}$。资源交集：$R_{i+1} \subseteq R_i$。拒绝并集：$D_{i+1} \supseteq D_i$。因此 $\pi(P_{i+1}) \subseteq \pi(P_i)$。根据归纳法，$\pi(P_j) \subseteq \pi(P_i)$ 对于 $i < j$。 □

定理 2（传递拒绝）： 如果资源 r 被任一策略拒绝，则它仍然被拒绝：

$$\exists i : r \in \text{Match}(D_i) \Rightarrow r \in \text{Match}(D_{\text{eff}})$$

证明概要： $D_{\text{eff}} = D_0 \cup D_1 \cup \ldots \cup D_n$。如果 $r \in D_i$ 对于任意 i，则 $r \in D_{\text{eff}}$ 由集合并得出。 □

定理 3（无权限提升）： 如果基础策略 P_0 拒绝r，则任何组合都不会授予访问权限：

$$(r \in \text{Match}(D_0)) \Rightarrow r \notin \text{Allowed}(P_{\text{eff}})$$

证明：根据定理 2，如果 $r \in$ 匹配(D_0)，则 $r \in$ 匹配(D_{eff})。根据 π 的定义，拒绝的资源不能被允许。 □

安全影响这些定理提供了形式化保证：**定理 1** 可防止权限扩展——添加策略只能缩小权限范围；**定理 2** 确保任何层的拒绝都具有绝对效力；**定理 3** 可防止通过组合策略授予已被拒绝的权限，从而避免权限提升。

综上，这些机制实现了第 2 节中的策略执行 (O2) 和权限不提升 (O3)。即使对手攻陷组件并获取其密钥

（A3），被攻陷的组件也只能在策略允许的范围内执行操作——单调限制可防止权限扩展，而传递性拒绝可确保根级拒绝沿所有推导路径传播。这些属性在数学上成立，不受攻击者能力或大语言模型行为的影响。要破坏这些属性，就必须攻破密码学原语（数字签名、哈希函数），而非进行操纵。**不允许任意覆盖**。MAPL 明确禁止覆盖，因为覆盖会破坏可证明性，使定理1-3不再成立。管理员可改为创建具有有效期限制的临时组（类似Unix组/权限）。这些组通过交集进行组合，在保持形式化保证的同时支持操作灵活性。附录 A 展示了限时紧急访问。

实现规模化：从 O(MxN) 条规则到 O(log M +N) 项策略 传统策略引擎需要为 M 个主体和 N 个资源制定 O(M×N) 条显式规则。管理呈组合式增长的规则并不实际；随着企业将智能体扩展到数千名用户，MAPL 提供了一种切实可行的解决方案。MAPL 的层级组合和动态主体绑定通过映照组织结构，消除了规则爆炸问题——当分支因子为 8-16 时，部门规模呈对数增长： $D \approx \log_{8-16}(M)$——比 M 小得多；团队则向上继承并融入层级树中。资源随功能增长，与用户数量无关。因此，MAPL 只需要$\log(M) + N$项策略，大幅减少了实际所需的策略数量。

总之，MAPL 的组合代数满足智能体系统的全部三项要求：通过运行时主体解析实现上下文身份，通过独立的策略组合实现双重视角，并通过密码学证明实现可验证的工作流状态。

4. 已认证工作流

我们现在介绍经过身份验证的工作流——一种在每一次智能体边界跨越处实现确定性验证的协议级机制。面对具备能力 A1-A5（第 2 节）的对手，经过身份验证的工作流提供四项**密码学保证**，共同实现安全目标 O1-O5：

真实性（实现 O1：完整性）：每一项被接受的调用都以密码学方式关联到发起它的主体以及所执行的操作。攻击者在不持有该主体私钥的情况下无法伪造调用，从而同时绑定谁（主体）与什么（操作）。

策略绑定（实现 O2-O3：策略执行、权限不提升）：管辖一项操作的策略与该调用构成密码学绑定，从而防止攻击者在不使签名失效的情况下替换策略。

篡改可检测性（实现 O4：上下文完整性）：对调用、上下文状态或策略绑定的任何修改，都可以通过哈希链和序列号以密码学方式检测出来。

不可否认性（实现 O5：问责性）：数字签名可建立主体与其操作之间不可否认的关联证明，从而借助防篡改审计日志开展取证分析和合规验证。

与已认证的系统调用 [5], [6], 一样，其核心思想在概念上十分简单：为每一次实体间调用附加策略与密码学签名，以确保完整性。

调用 = (Args, 策略, MAC) MAC =Sign(K, Args || 策略)

其中，消息认证码 (MAC) 签名使用机密或私钥 K，以密码学方式将参数与策略标识符绑定在一起。在接收端，验证分三步进行：验证签名以确保真实性和完整性，检索并验证策略绑定以防止替换，然后评估该操作是否符合策略规则。

4.1. 设计原则

实现经过身份验证的工作流需要遵循四项原则，这些原则相互组合，可提供密码学保证：

零信任身份：每个实体——智能体、工具、数据源、大语言模型——都拥有唯一的密码学身份（密钥对）。即使实体处于同一进程中，也会彼此独立地验证。不会传播任何隐式信任。我们将企业身份与智能体级验证相结合：外部主体通过企业身份与访问管理进行身份认证；运行时将这些身份作为防篡改的会话上下文传递。在内部，轻量级临时签名用于标识智能体，以应对动态身份挑战（第 3 节）。

边界验证：每次实体间通信——提示→大语言模型、智能体→工具、工具→数据——都需要独立的密码学验证。操作携带签名和策略标识符；接收方使用已注册的公钥验证签名、检索策略并评估约束。验证与传输方式无关——无论是 MCP、LangChain、OpenAI 应用程序编程接口，还是自定义协议，验证方式都相同。

策略执行点：策略执行点会在每个边界进行独立验证——实体无法验证自身的操作。策略执行点可以部署在进程内（作为链接库）或以边车形式部署，并在执行前验证调用，从而实现水平可扩展性、零信任验证、防篡改性和执行独立性。这依赖于 L3（执行完整性）。

可信的认证上下文：运行时提供防篡改的会话状态，例如来自 [3] 的状态，确保完整性，包括将操作绑定到会话范围（上下文 ID、会话 ID）、防止重放攻击（序列号）、检测篡改（哈希链），并支持工作流依赖（通过证明确认前置操作已完成）。

协议。这些原则组合成一个四阶段协议，使每次边界跨越都成为经过身份验证的工作流。图 2 展示了完整流程。

注册：每个实体都会进行注册，并获得（智能体 ID、密钥对）。注册采用最细粒度：智能体内的单个工具各自获得独立的（工具 ID、密钥对），从而最大限度缩小爆炸半径。

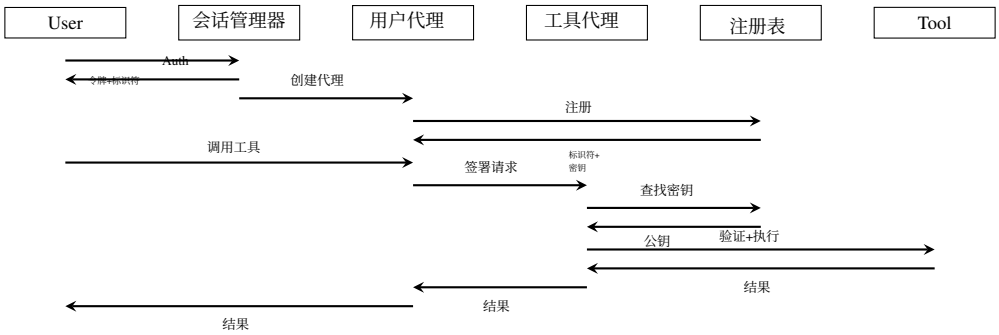


图2. 注册与调用流程，展示身份认证、实体注册、签名调用和双向验证。

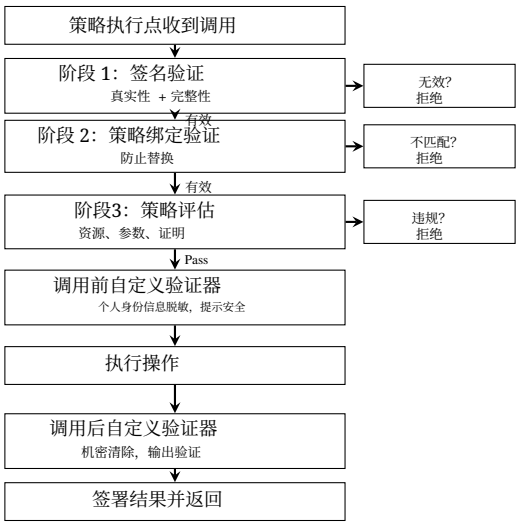


图 3. PEP 验证流程：包含三个阶段的密码学验证和可选的自定义验证器。

调用：调用方构造签名调用，将操作、参数、策略和会话上下文绑定在一起。

验证：接收方策略执行点通过三个阶段独立进行验证：(1) 签名验证（真实性+完整性），(2) 策略绑定验证（防止替换），(3) 策略评估（资源权限、参数约束、证明）。图 3 详细说明了验证流程。

证明：操作完成后，服务会对结果签名。上下文会更新为包含以密码学方式签署的证明，以证明操作已完成，并为下游操作维护经过身份验证的上下文。

服务对结果签名；调用方在接受结果前验证服务签名——由此实现类似于双向 TLS 的双向身份认证。该无状态协议确保每一跳都经过独立验证，不会在多步骤工作流中隐式传递信任。

4.2. 验证流程

图 3 展示了验证流水线——我们完整性主张的核心。该流水线始终强制执行且无法规避：每一次调用在执行前都须经过密码学验证，由接收方独立使用从注册表检索到的公钥验证调用方。

策略构建：阶段3由接收方通过双视角执行，独立构建有效策略。策略执行点检索组织策略（公司、业务部门、团队）和特定资源的约束，并计算 $P_{\text{eff}} = \text{意图} \cap \text{资源}$（第 3 节）。调用方声明的意图会根据资源约束独立验证——调用方无法影响策略评估。交集代数（定理1-3）保证单调限制：组合后的策略只会收窄权限，绝不会扩大权限。

自定义验证器：MAPL 的声明式约束可处理资源权限和参数，但有些安全检查需要针对调用上下文执行命令式逻辑。自定义验证器可通过个人身份信息检测、SQL 注入防护、借助大语言模型分类器（作为裁判的大语言模型）实现的提示安全、速率限制和地理位置检查，扩展策略评估。调用前验证器在策略评估后执行；调用后验证器则处理结果。验证器基于以密码学方式验证的上下文运行，是由系统管理员检入的可信代码，而非应用开发者编写的代码。一旦在验证流程中声明，策略执行点就会强制执行这些验证器——无法绕过。这提供了特定领域的可扩展性（HIPAA、GDPR、SOC 2），同时维持完整性保证。

四种生产环境验证器展示了这一架构：(1) MemoryIntegrityVerifier 通过速率限制、受保护字段强制执行和密码学完整性检查，防止记忆投毒（OWASPLLM01）；(2) Work- flowIntegrityVerifier 通过强制执行前置步骤并维护证明链，防止工作流劫持（OWASP LLM04）；(3) ToolAuthorizationVerifier 通过基于角色的访问控制和危险模式检测（例如，文件写入→命令执行），防止工具滥用（OWASP LLM06）；(4) StorageIntegrityVerifier 通过路径遍历防护和强制加密，防止数据外泄（OWASP LLM02）。附录 A 提供了实现细节。

这种分离至关重要：核心验证流水线（阶段 1-3：签名验证、策略绑定、策略评估）通过以密码学方式强制执行 MAPL 的声明式约束（引理 1、4），提供确定性保证——召回率达到 100%，且零假阳性。

自定义验证器（阶段 4–5）是可选的管理扩展，可能会使用启发式方法（基于机器学习的个人身份信息检测、大语言模型安全分类器）；如果配置不当，可能引入假阳性，但不会削弱核心保证——它们只能增加限制，绝不会移除限制。验证器仅在密码学验证通过后执行，处理的是经身份验证的调用和防篡改状态，而非不受约束的应用程序输入。即使验证器失效或遭到破坏，阶段 1-3 仍会强制保障密码学完整性和策略约束。要绕过验证器，必须破坏受信任的管理员代码（L3），而不是操纵应用程序逻辑（A1）。

安全保证。针对对手 A1-A5（第 2 节），协议机制通过四个相互独立的密码学层组合成纵深防御：**完整性**（签名确立真实性，抵御网络攻击 A4）；**授权**（策略交集强制落实意图——即使签名有效，操作也必须满足组合后的策略，从而防止应用程序控制下的权限提升 A1）；**来源追溯**（哈希链和序列号使上下文篡改可被检测，抵御内容注入 A2 和多轮操纵 A5）；**合规性**（自定义验证器基于经过密码学验证的上下文进行领域特定验证，从而提供不可否认性）。这些机制共同实现了安全目标 O1-O5。

协议提供纵深防御：每一层都在每个边界独立运作（引理 7，第7节）。攻破签名验证不会绕过策略评估；篡改上下文不会击败工具策略执行点；绕过某个验证器不会使密码学检查失效。第7节将这些独立性保证形式化。

5. 通用安全抽象

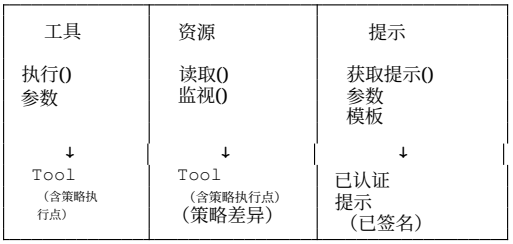
经过身份验证的工作流为智能体人工智能提供了一种通用安全抽象——我们通过保护涵盖三个架构层的九种框架来验证这一主张：协议标准（MCP、A2A）、大语言模型 API（OpenAI、Claude）和编排框架（LangChain、CrewAI、AutoGen、LlamaIndex、Haystack）。无论框架架构如何，每次智能体交互都可以完整映射为一次由我们的四阶段协议管理的可验证边界跨越：注册 → 调用 → 验证 → 证明。

5.1. 框架集成

表 1 展示了九种框架如何映射到经身份验证的工作流原语。我们重点介绍各架构层中的代表性示例：

协议层：MCP。MCP 服务器会公开工具、资源和提示。资源包含静态数据集，旨在为大语言模型提供额外的上下文；MCP 工具则是会产生副作用并带有参数的主动函数。在我们的系统中，服务器对应智能体，提示对应经身份验证的提示 [3], ，而资源和工具则抽象为由不同类型策略保护的经验证工具——每个工具都有独立且唯一的身份和密钥对。

MCP 服务器 → 已注册工具的智能体



协议层：A2A。A2A（智能体对智能体协议）直接映射到经过身份验证的工作流。智能体使用密码学身份注册。委托会转变为带有范围约束的已签名令牌，并遵循 MAPL 的交集代数——受委托方不能授予超出其所获权限范围的权限。证明通过密码学证明建立工作流依赖，确保操作按要求的顺序完成。

大语言模型接口：OpenAI 和 Claude。OpenAI 和 Claude 遵循相同的模式。API 端点注册为智能体。大语言模型推理操作（生成、补全、嵌入）注册为带有嵌入式策略执行点的工具。两种部署变体提供了灵活性：双侧封装 将大语言模型 API 和客户端应用程序都视为智能体——API 侧策略执行点验证传入请求是否满足提供方策略，而客户端侧策略执行点验证函数调用是否满足应用程序策略；单侧封装仅封装客户端侧工具，将大语言模型 API 视为直通组件；当大语言模型提供方在提示安全方面值得信任，且重点是保护客户端侧工具调用时，这种方式可简化部署。这解决了大语言模型不可信、无法用于授权决策这一挑战——验证必须在函数执行边界（S2）处进行，由策略执行点独立验证签名并评估策略。

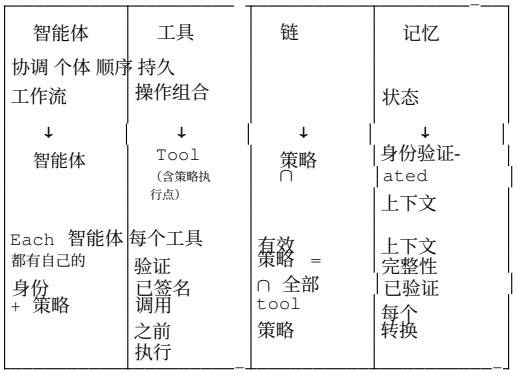
编排层：LangChain。编排框架跨工具和大语言模型协调多步骤工作流，提供多个控制点：链组合逻辑、智能体推理、内存操作以及智能体间通信。仅保护单个工具会使组合和状态管理处于未保护状态。

我们的设计从四个层面解决安全问题。智能体通过密码学身份进行注册。工具独立注册，并嵌入策略执行点（PEP）——每次工具调用都要经过签名验证和策略评估。链——由操作顺序组合而成——通过交集实施策略组合（第 3 节）；如果某个工具的策略拒绝某些操作，将该工具组合进链中也无法放宽限制——有效策略只会通过交集变得更加严格，绝不会更加宽松。记忆——跨交互持久保存的状态——通过经过身份验证的上下文 [3],确保上下文完整性，防止上下文投毒（S4）。

框架	层	映射到经身份验证的工作流原语
MCP	协议	服务器→智能体，工具→工具（含策略执行点），资源→工具（策略区分），提示→已认证提示
A2A	协议	智能体→智能体（密码学身份），委托→已签名令牌（范围约束），证明→证明（工作流依赖）
OpenAI API	大语言模型 API	端点→智能体、操作（生成/补全/嵌入）→工具（含策略执行点），两种部署变体（双侧/单侧封装侧/单侧封装）
Claude	大语言模型 API	端点→智能体，操作（生成/补全/嵌入）→工具（带策略执行点），两种部署变体（双侧/单侧封装）
LangChain	编排	智能体→智能体，工具→工具（带策略执行点），链→策略交集，记忆→已认证上下文（哈希链可防止投毒）
CrewAI	编排	团队成员→智能体、角色→证明（密码学绑定）、任务→签名调用（基于角色的授权）基于角色的授权）
AutoGen	编排	对话智能体→智能体、代码执行→工具（带策略执行点）、策略→可插拔验证器（AST 分析、允许列表、沙箱约束）
LlamaIndex	编排	RAG 流水线→智能体、查询/检索操作→工具（带策略执行点，通过执行文档访问控制来防止通过检索发起提示注入）
Haystack	编排	文档流水线→智能体，流水线节点→工具（带策略执行点），顺序执行→策略交集（节点不能放宽约束）

表 1. 框架到经身份验证的工作流原语的映射。

LangChain 编排器（带工具的智能体）-



其他编排框架（CrewAI、AutoGen、LlamaIndex、Haystack）也遵循类似模式——角色映射为证明，以实现密码学角色绑定；代码执行使用可插拔验证器进行 AST 分析；RAG 流水线则实施文档访问控制，详见附录 A

5.2. 协议层普适性

在协议层，所有九种框架都归结为一种统一的抽象：智能体通过签名调用操作；策略执行点验证签名和策略；操作要么执行，要么被拒绝。这种统一性使异构组合也能实现一致的安全性——例如 LangChain 编排 OpenAI 调用 MCP 工具，并在每个边界进行独立验证。

当工作流跨越多个框架时，攻破某个框架的验证机制并不能绕过其他框架。每个边界的策略执行点都会独立验证：(1) 签名有效性（真实性）；(2) 身份（防止冒充）；(3) 对资源权限、参数约束和证明进行策略评估（授权）。这种组合式方法——保护边界而非框架——可在异构系统中实现统一防护。第7节通过引理6-7将这些保证形式化。

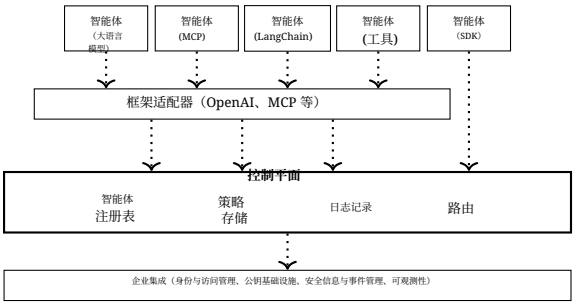


图 4. 智能体AI信任层。

6. 智能体AI信任层

本节将介绍智能体AI信任层，其中包括经过身份验证的工作流，以及经身份验证的提示和上下文，详见 [3]。所有九种框架都通过轻量级框架适配器接入信任层。信任层由以下部分组成：提供核心服务（身份、策略、日志记录和路由）的控制平面；管理与企业基础设施（身份与访问管理、公钥基础设施、存储等）的集成；验证网关（边车策略执行点）；以及将策略执行点嵌入智能体应用的客户端库。由于篇幅所限，我们将重点讨论关键设计要素，而非面面俱到地概述整个架构。

6.1. 架构

图 4 展示了智能体与控制平面交互的信任层架构。

框架适配器。每个框架都通过轻量适配器进行集成，将框架原生操作转换为经过身份验证的工作流。适配器实现四阶段协议——注册、调用签名、验证和证明——同时提供框架的原生 API。集成过程是透明的；开发者使用标准框架 API（MCP 服务器、OpenAI客户端、LangChain智能体），而密码学强制执行会自动完成。

控制平面服务。控制平面提供四项核心服务：

- 智能体注册表：存储智能体标识符和公钥，以及智能体内工具的标识符和公钥，从而保留密码学溯源关系。支持即时撤销凭证。策略执行点在签名验证期间检索公钥。
- 策略存储库：防篡改策略存储库，策略执行点通过交集检索并组合分层策略（例如，公司、业务部门、团队）。
- 路由：在跨越网络边界的智能体之间路由签名调用，使被调用方能够获取调用方的公钥以进行验证。
- 日志记录：通过防篡改哈希链记录所有安全事件，并保留其密码学溯源信息。可与企业级安全信息与事件管理（SIEM）平台集成。被阻止的操作会生成结构化日志，说明拒绝原因（签名无效、策略违规、缺少证明、验证器故障），提供足够的调试细节，同时避免泄露策略。审计日志有助于进行根因分析和合规报告。

分布式执行。这些服务以高可用配置运行，并与企业基础设施（身份与访问管理、 公钥基础设施、安全信息与事件管理、可观测性平台）集成。控制平面支持多租户——组织可以部署共享基础设施，同时维持租户之间的密码学隔离。该架构实现了零信任分布式验证——每个策略执行点都使用来自注册表的公钥和来自存储库的策略，独立验证调用。默认情况下，策略执行点以进程内嵌入方式部署（链接库），也可部署为边车，从而支持水平扩展。采用嵌入式策略执行点时，不存在集中式瓶颈；每个边界处都在本地完成验证，开销低于一毫秒。管理员可通过控制平面即时撤销凭据、更新策略或隔离智能体。受篇幅所限，本文不详述完整架构；有兴趣的读者可参阅我们的技术报告，了解完整的实现细节。

6.2. 设计决策

在生产环境中实现已认证工作流和信任层，暴露出了若干实际挑战，需要我们采取明确的设计方案——本文将讨论应对这些挑战的六项最有意思的设计决策。

身份桥接。企业身份需要在不同框架之间进行桥接，因为每个框架都有自己的身份认证模型（OAuth、API密钥、SAML）。我们不针对每个框架分别解决这一问题，而是在协议层进行桥接。用户完成身份认证后，信任层会签发与其会话绑定的临时密钥对，并在所有框架之间传播经过密码学验证的主体。签名调用会通过委托链携带用户身份；每个策略执行点都会根据智能体注册表，独立验证发起调用的主体。这样便能集中处理身份与访问管理集成——各框架无需分别集成企业身份系统，信任层只需处理一次身份认证。当使用 OAuth 进行身份认证的 Alice 调用通过 API密钥进行身份认证的 Claude，而 Claude 又调用通过 MCP 进行身份认证的工具时，日志会显示“Alice 通过 Claude 访问了文件”，同时保留审计追踪。在协议层进行桥接可提供更简单的编程模型，并实现一致的主体传播，消除各框架分别集成所带来的复杂性。

会话管理。许多框架缺少原生会话概念（MCP、LangChain 都是无状态的），但跨轮次关联对于安全至关重要。我们没有修改每个框架来添加会话，而是在协议层使用对框架透明的上下文标识符和序列号进行跟踪。无论框架是否支持会话，信任层都会维护已认证上下文——不支持会话的框架会获得不可见的协议层跟踪；支持会话的框架则直接映射。多租户架构确保每个会话的上下文在密码学上相互隔离，从而防止会话间污染。这样，无需修改各个框架的协议，就能在多框架工作流中实现统一的审计追踪。

可验证委托。多跳委托会带来权限提升风险——遭到入侵的中间代理可能声称拥有超出授予范围的权限。我们没有采用基于信任的委托，而是通过密码学方式强制收窄范围。已签名的委托令牌包含委托者身份、受托者身份、范围约束和到期时间。信任层通过取交集来强制收窄范围：委托范围 = 委托者授予的权限 ∩ 受托者拥有的权限，从而扩展了 MAPL 的代数体系（第 3 节）。遭到入侵的智能体无法声称拥有超出授予范围的权限；下游服务会以密码学方式验证委托的有效性。这使异构框架之间的最小权限原则得到可证明的保障。

前向保密。长期有效的 API 密钥会带来运维和安全风险——密钥很少轮换，一旦泄露，就会危及所有会话。我们不依赖运维层面的密钥轮换策略，而是从设计上采用临时密钥。每个会话都会获得唯一的临时密钥对，并在到期后自动失效。密钥轮换通过会话生命周期自然完成，无需修改基础设施，从而实现会话级前向保密。一个会话的密钥遭到泄露不会影响其他会话，因此能够限制影响范围，同时不会增加运维开销。

大规模服务账户。智能体系统会在运行时动态生成实体——编排器生成智能体，智能体生成工具，工作流创建子工作流。传统身份与访问管理（IAM）的手动配置无法适应运行时动态变化。我们不采用手动配置，而是在注册期间（第 4 节）自动配置密码学身份，细化至工具级粒度——分配（工具 ID、密钥对）并记录在智能体注册表中。无需手动配置，无需分发凭证，也不存在身份与访问管理瓶颈。每个工具都能独立实现密码学问责，从而最大限度地缩小影响范围。这使得智能体能够在企业级规模下动态部署。

双向问责。单向身份验证会在智能体环境中造成问责漏洞——调用方无法证明服务正确执行了操作；服务无法证明调用方请求了操作。我们不采用单向签名，而是使用双重签名来实现相互不可否认性，

以实现相互不可否认性。调用方对请求签名；服务对结果签名。每项操作都会生成双方的密码学证明，从而提供符合合规性要求的审计轨迹——服务无法否认其操作，调用方无法否认其请求。结合身份桥接与防篡改日志记录，这便能在异构身份验证边界之间实现完整的问责性。

7. 形式化分析

我们证明，在假设 L1-L3 成立的情况下，经过身份验证的工作流能够抵御对手 A1-A5，实现 O1-O5。此证明由围绕意图和完整性组织的七个引理构成。完整性属性（引理 1、2、3）确保操作真实且篡改可被发现。意图属性（引理 4、5）确保操作符合策略且能安全组合。完备性属性（引理 6、7）确保所有操作均已验证，且验证过程相互独立。

主要安全定理。定理（经过身份验证的工作流安全性）：在信任假设 L1-L3 成立的情况下，经过身份验证的工作流能够抵御具备 A1-A5 能力的对手，并实现完整性（O1）、策略执行（O2）、权限不提升（O3）、上下文完整性（O4）和问责性（O5）。

完整性属性。 这些引理确保操作真实、篡改可被发现且不可抵赖，从而应对具备组件失陷（A3）和网络攻击（A4）能力的对手。

引理 1（真实性）：所有已接受的调用都具有有效的密码学签名，将主体与操作、参数、策略及上下文状态绑定起来。

证明：策略执行点使用从智能体注册表（第 5 节）检索到的主体公钥验证签名。根据密码学困难性假设 L1，没有私钥而伪造签名在计算上不可行——即使对手拥有应用控制权（A1）、网络访问权限（A4）或已攻陷组件（A3），也无法为其他主体的身份生成有效签名。这实现了完整性（O1）：操作在密码学意义上是真实的。 □

引理2（篡改可检测性）：对上下文状态或审计日志的修改可通过密码学方式检测，并且能够防止重放攻击。

证明：上下文状态通过连接顺序状态的哈希链来维护：$h_i = \text{哈希}(h_{i-1} \parallel \text{状态}_i \parallel \text{序列}_i)$。每次调用都包含一个单调递增且绑定在签名中的序列号。修改任何状态都需要找到哈希碰撞；由于策略执行点会拒绝小于或等于此前已处理值的序列号，因此重放旧调用会被检测到。在密码学困难性（L1）的前提下，找到碰撞在计算上不可行。试图逐步进行状态投毒或重放操作的多轮操纵攻击（A5），会因篡改破坏哈希链或违反序列单调性而被检测到。审计日志使用相同的哈希链机制。这实现了上下文完整性（O4）。□

引理 3（不可否认性）：主体不能否认执行过记录在审计日志中且带有有效签名的操作。

证明：只有持有私钥的主体才能生成有效签名（引理 1）。审计日志会保留具备篡改可检测性的记录（引理2）。因此，审计日志中已签名的操作构成主体行为的密码学证明。这实现了问责性（O5）。 □

意图属性。这些引理确保操作符合策略并能安全组合，从而应对具备应用控制（A1）、内容注入（A2）和渐进式攻击（A5）能力的对手。

引理 4（策略执行）：所有已执行的操作均符合由组织层级和资源约束组合而成的有效策略，且策略标识符以密码学方式绑定在签名中。

证明：每次调用都在签名载荷中包含一个策略标识符。策略执行点会检索策略，并通过对所有适用策略取交集来计算有效策略 P_{eff} ——包括组织层级策略（基础、部门、团队）和特定资源策略（第 3 节）。试图发动策略替换攻击的对手（能力 A1、A4）会失败，因为修改策略标识符会使签名失效（引理 1）。策略交集确保单调限制（第 3 节，定理1-3）——组合后的策略只会变得更加严格，而不会更加宽松。由此实现策略执行（O2）和防止权限提升（O3）。 □

引理 5（组合安全性）：当工作流跨越多个框架时，安全属性在框架边界之间得以保留。

证明：考虑框架 F1 上的操作 O1 调用框架 F2 上的操作 O2。两次调用都带有用于绑定主体和策略的签名（引理 1）。框架 F2 的策略执行点会独立验证 O2 的签名，并将有效策略计算为 F1 策略与 F2 策略的交集（引理 4）。框架 F1 无法绕过 F2 的验证，原因如下：(a) F2 的策略执行点会独立验证签名（引理 7，证明见下文）；(b) 策略交集确保 O2 必须同时满足 F1 和 F2 的约束。在 N 个框架之间进行组合，会创建 N 个独立的验证点，并组合策略 $P_{\text{eff}} = P_1 \cap P_2 \cap \ldots \cap P_N$，从而实现单调限制。第 5 节展示了这一机制在九种框架组合中的应用。 □

完备性属性。 这些引理确保所有操作都经过验证，并且在各控制面之间独立进行验证。

引理6（控制面完备性与最小性）：四个控制面 {S1、S2、S3、S4} 是完备的（所有资源访问操作都至少经过一个控制面），且具有最小性（每个控制面都是必不可少的）。

通过枚举与必要性证明：

完备性：我们枚举了所有资源访问操作，并表明每项操作至少涉及一个攻击面：(1) 计算资源——大语言模型推理涉及 S1（提示中包含指令）；工具执行涉及 S2（特权操作）。(2)数据资源——检索增强生成检索、数据库

查询、网页抓取涉及 S3（外部数据流入推理过程）——内容注入攻击会利用这一点（A2）。(3)状态资源——会话状态、工作流上下文、记忆涉及 S4（跨轮次的持久状态）——多轮操纵会以此为目标（A5）。(4)跨智能体通信——委托涉及 S1（指令传播）或 S2（调用），以及 S4（上下文继承）。不访问外部资源的操作（纯计算）无需防护——它们无法泄露数据或违反策略。

最小性：每个面都不可或缺——移除任何一个面都会使攻击失去防护：移除 S1（提示）：攻击者可通过 S3 数据实施间接提示注入，绕过大语言模型的验证。移除 S2（工具）：即使提示通过验证，仍可能发生未经授权的工具执行。移除 S3（数据）：即使 S1/S2 通过验证，遭投毒的检索增强生成数据仍可能导致工具调用。移除 S4（上下文）：多轮工作流中的会话劫持和上下文投毒。因此，{S1、S2、S3、S4} 是完备且最小的。第 5 节通过涵盖三个架构层的九种框架，对此进行了实证验证。□

引理 7（策略执行点独立性）：攻陷一个策略执行点不会削弱其他策略执行点的验证。

证明：每个策略执行点均独立验证操作，仅使用：(1) 用于验证签名和哈希链的密码学原语（L1）；(2) 从受信任的控制平面检索的公钥和策略（L2）；以及 (3) 自身的验证逻辑（L3）。策略执行点之间不共享运行时状态，也不进行协调。攻陷某个面上的策略执行点的对手（能力 A3），只能绕过该面上的验证——其他面会使用各自的策略执行点实例独立验证。多面攻击需要攻陷多个相互独立、且各自受到 L3 保护的策略执行点。这实现了纵深防御：影响范围仅限于被攻陷策略执行点所在的面。 □

主定理证明。主定理证明：根据引理6，所有资源访问都至少经过一个控制面。每个控制面都嵌入了一个策略执行点，以强制执行独立验证（引理1至4、7）。多框架工作流会跨框架边界组合验证（引理 5）。验证通过策略评估（引理 4）来强制执行意图，并通过签名（引理 1）和哈希链（引理 2）确保完整性。证明通过已签名、经哈希链链接的声明，提供工作流依赖关系。

每个资源访问操作有两种结果：（1）授权执行——操作满足所有密码学检查（有效签名、未被篡改的上下文、有效序列号）和策略约束（允许访问的资源、参数处于界限内、具备必需证明），得以执行，并记录在不可否认的审计追踪中（引理 3）；（2）阻止——操作至少未通过一项检查（签名无效、策略违规、上下文遭篡改、序列违规或缺少证明），因此在执行前被拒绝，并记录违规情况。

具体示例：在 Q4攻击（第 2 节）中，投毒数据指示大语言模型窃取凭据。当大语言模型生成对 credentials.db 的文件系统读取请求时，工具的策略执行点会评估策略——拒绝匹配 *credential* 的路径——即使该操作带有经过身份验证的大语言模型的有效签名，仍会阻止该操作。无论签名有效性如何，策略执行都会防止未经授权的访问。即使多个边界遭到破坏，每个界面仍会执行独立验证（引理 7），从而限制影响范围。

因此，在 L1-L3 假设下，具备 A1-A5 能力的对手无法在不被检测到的情况下进行未经授权的资源访问，从而实现 O1-O5。 □

实际意义。这些形式化属性为第 6 节中的操作解决方案提供了基础：引理 1 支持主体传播，引理 4+5 支持可验证委托，引理 2 支持会话级前向保密，而引理 3 则支持双向问责。

8. 攻击防御验证

第7节证明，在 L1-L3 假设下，经过身份验证的工作流能够抵御 A1-A5 对手，实现 O1-O5。我们通过 OWASP十大安全风险覆盖率（10 项风险中覆盖 9 项）、系统化攻击测试（11 种模式、174 个测试用例、100% 召回率、0% 假阳性），以及生产环境CVE分析来验证这些形式化保证；分析结果表明，当基准系统失效时，本方案仍能提供保护。

2025年OWASP大语言模型应用十大风险覆盖率。表 2 将我们的防御机制映射到 2025 [7], 2025年OWASP大语言模型应用十大风险，展示了系统化的覆盖率，并明确列出防御机制和形式化保证。

这些防御措施应对了对手的所有能力（第 2 节）：按策略的机制通过运行时强制执行阻止 A1、A2、A5（引理 4、5）；按设计的机制通过密码学困难性消除 A3、A4（引理 1、2、3）。两者结合实现了 O1-O5。这验证了双重框架：任何一种机制单独使用都不足以提供保护；两者结合才能实现全面保护。

系统化攻击验证。除了 OWASP 宽泛的风险类别外，我们还分析攻击机制——即攻击如何在工作流组合中呈现。我们确定了 8 种机制类别，涵盖提示操纵、工具链式调用、凭证窃取、数据外泄、恶意软件/代码执行、资源耗尽、多智能体攻击以及策略违规。详细分类体系见附录 C。表3展示了 11 种攻击模式，提供了具有代表性的覆盖范围。

每种攻击模式都包含多个测试场景：数据外泄测试涵盖批量导出、多资源访问、个人身份信息提取和未经授权的访问；推理攻击测试涵盖属性攻击、成员推断攻击和重构攻击。我们的 174 个测试用例涵盖不同的框架组合（OpenAI、Anthropic、LangChain 等）和配置变体。

表3展示了防御分类。按设计的机制实现完整性属性（引理 1、2、3），从密码学角度使以下攻击不可能发生：身份欺骗、会话重放、策略替换、上下文篡改、审计操纵和证明伪造。按策略的机制实现意图属性（引理 4、5），通过运行时验证阻止违规行为，包括提示注入、权限提升、凭据外泄、数据收集、供应链攻击、资源耗尽和跨智能体攻击。复杂的

表 2. 2025年OWASP大语言模型应用十大风险覆盖率

OWASP 风险	我们的攻击测试	防御机制	章节
LLM01：提示注入	提示注入，Atlas	按策略：策略会阻止未经授权的操作 + 记忆完整性验证器可防止目标操纵	§4，§7
LLM02：信息泄露	令牌劫持、数据窃取、侧信道，推理	按策略：资源拒绝 + 存储完整性验证器可检测外泄模式	§4、§5
LLM03：供应链	恶意工具、供应链	按设计 + 按策略：代码签名 + 工具注册表，含审-批策略	§5，§7
LLM04：数据投毒	上下文篡改	按设计：哈希链 + 工作流完整性验证器强制执行步骤序列	§4, §7
LLM05：输出处理	（横切）	按策略：调用后验证器会清理输出	§5
LLM06：过度代理	王国之钥，混淆代理	按策略：策略组合 + ToolAuthorizationVerifier 通过模式检测强制执行基于角色的访问控制	§3，§7
LLM07：提示词泄漏	提示词变体	按设计：密码分段（AuthPrompt）	§4
LLM08：向量弱点。	（检索增强生成场景）	按策略：证明可验证来源；策略限制访问	§3，§5
LLM09：错误信息。	（间接相关）	可插拔验证器；非主要安全重点	§5
LLM10：无界一致性。	拒绝服务	按策略：速率限制 + 资源配额	§5

覆盖率：9 项中的 10 项 OWASP 风险。LLM04 部分相关（已防止运行时投毒；训练数据不在范围内）。LLM09 间接相关（质量与安全性对比 y).

表3。 经验证的攻击模式与防御类别 分类

攻击模式	类别	防御类别	OWASP
提示注入	提示操纵	按策略	LLM01
王国之钥	工具链式调用	按策略	LLM06
混淆代理	工具链式调用	按策略	LLM06
令牌劫持	凭证窃取	按策略	LLM02, LLM06
会话固定	凭证窃取	按设计	超越 OWASP
数据外泄	数据外泄	按策略	LLM02
侧信道	数据外泄	按策略	LLM02
推理攻击	数据外泄	按策略	LLM02
恶意工具	恶意软件/代码执行	按策略	LLM03
供应链	恶意软件/代码执行	按策略	LLM03
拒绝服务	资源耗尽	按策略	LLM10
总计：*11* 种模式、*18* 种明确变体、*174* 个测试用例（包括框架排列组合）			

覆盖率：8 类机制中明确实现了 6 类 ；多智能体攻击和策略违规攻击通过组合机制加以应对（见正文）。

攻击（如供应链攻击）需要两者兼备——代码签名加工具审批策略。六个类别有明确的实现；多智能体攻击则通过独立的 PEP 验证加以防范（引理 7）。

实证结果：我们的评估在 174 个测试用例中实现了 100% 召回率，且没有假阳性。按设计消除可提供确定性保证——违规会以密码学方式检测，而非通过模式匹配。要破坏验证机制，必须解决计算上难以处理的问题（L1），而不是构造对抗性输入。这也解释了为何绕过语义防御的攻击（Atlas、GitHub MCP CVE）会被经过身份验证的工作流确定性地阻止。

真实世界攻击验证。我们针对两起曾攻破生产系统的攻击验证了经过身份验证的工作流——OpenAI 的 Atlas 智能体浏览器攻击和 GitHub 的模型上下文协议服务器攻击。这两起攻击都成功绕过了采用语义护栏的基线防御；而经过身份验证的工作流通过在控制面进行独立验证，完全阻止了这两起攻击。随后，我们通过企业集成场景展示了该方案的部署可行性。

OpenAI Atlas 浏览器攻击。OpenAI 的 Atlas 智能体浏览器 [1] 遭到提示注入攻击，导致凭据外泄（ OWASP LLM01、LLM06、LLM02）。

我们模拟了这种攻击：合法工作流读取财务报告；恶意输入嵌入了隐藏指令，以泄露凭据。该攻击成功绕过了语义护栏，因为它们无法区分对抗性提示与合法的文档内容。

策略执行阻止了这条攻击链。文件系统策略拒绝访问凭证路径；网络策略限制外部端点。无论大语言模型如何推理，PEP 验证（引理 4）都会在执行前拒绝未经授权的操作。我们测试了允许列表、拒绝列表以及跨框架变体的组合。所有方案都阻止了攻击，且没有产生误报。

GitHub MCP 提示注入。GitHub 的模型上下文协议服务器 [8] (558,846+ 次下载)遭到恶意提示攻击，这些提示利用 GitHub议题调用 MCP文件系统工具以窃取凭据（ OWASP LLM01、LLM03、LLM02）。

多重防护措施阻止了这条攻击链。首先，工具授权要求提供来自获准注册表的密码学签名——恶意工具在发现阶段即遭拒绝（引理 1）。其次，文件系统策略阻止凭证访问（引理 4）。第三，跨框架组合确保文件系统策略执行点能够独立验证操作（引理 5）。通过在多个层面进行独立验证，该攻击被阻止（引理 7）。

这两种攻击都表明，仅靠应用层防御并不足够——尽管设置了语义护栏，生产系统仍遭到入侵。经过身份验证的工作流通过纵深防御实现全面保护：必须同时攻破多个相互独立的加密屏障。所有测试场景的召回率均保持在 100%，且没有假阳性，这验证了形式化保证

（第7节）可转化为生产部署。第10节将这些结果与现有防御措施进行了比较。

9. 性能

我们通过在商用硬件（8 核、16GB RAM）上使用 ECDSA-256 签名和 SHA-256 哈希开展微基准测试，以验证实际开销。**密码学操作**的开销通常低于 ∼0.2ms（ECDSA 签名生成耗时 ∼102μs，验证耗时 ∼89μs，每次操作的哈希链更新耗时 <15μs）。**策略操作**：检索延迟取决于后端：内存中（∼5-50μs）、文件系统（∼30-200μs）、Redis（∼50-300μs）；策略交集取决于组合方式——对于典型的 3 至 5 层深度层级，耗时低于 100μs。**自定义验证器**：确定性检查（个人身份信息检测、路径验证）会增加 30-400μs 的耗时；基于大语言模型的验证器（提示安全）会增加 150-500ms 的耗时，启用时将成为主要开销，但可提供与密码学保证互补的语义分析。**端到端影响**：对于受网络延迟限制的操作（大语言模型推理、远程 API），密码学开销（∼0.2ms）相较于网络延迟（50-500ms）可以忽略不计。对于本地工具调用，不使用大语言模型验证器时，PEP 验证的总开销低于 1ms。审计日志记录通过缓冲 I/O 异步进行。

一篇配套论文提供了全面的系统评估，包括生产工作负载特征分析、涵盖全部九种框架的框架集成成本、大规模部署经验，以及基于生产规模基准测试的详细性能分析。

10. 相关工作

近期综述 [9]–[12]指出了智能体 AI 安全领域的四个严重缺口：不可预测的多步骤工作流、跨框架组合、可变环境，以及与不可信实体的交互。这些综述揭示了碎片化问题，却缺乏系统层面的解决方案。以往工作要么关注应用语义，要么关注基础设施原语；我们则提出一个介于两者之间的信任层，通过协议层原语保护四个基础控制面（提示、工具、数据、上下文），所有智能体与资源的交互都发生在这些控制面上。

应用层防御在组合时失效。特定框架的安全机制各自独立运行。LangChain 提供模式和验证 [13]；MCP 实现身份认证和权限 [14]；OpenAI 强制实施速率限制和内容审核 [15]。近期漏洞暴露了组合层面的缺口：CVE-2024-8309（LangChain 提示注入→SQL，CVSS 9.0） [16]，CVE-2024-36480（通过未经净化的 eval() 实现远程代码执行） [17]，CVE-2025-6514（MCP 供应链，558K+次下载） [8],Atlas 浏览器攻击（2025年10月） [1], [2]。框架边界之间不存在密码学绑定。

近期学术研究探讨了提示注入防御 [18]–[20]，工具使用安全 [21]，以及 RAG 检索投毒 [22]，[23]——嵌入文档中的恶意指令会损害推理能力。语义防御（护栏 [24]–[26]）的检测率达到 60-80%，但仍然可以被绕过 [27]，[28]。我们通过纵深防御提供确定性保证（100% 召回率、零假阳性）：按设计消除使攻击在密码学上无法实现；按策略执行则阻止违规行为。这些方法相辅相成——语义防御缩小攻击面，密码学强制执行则提供不可绕过的验证。

基础设施原语缺乏工作流语义。基础设施原语可提供强有力的保障，但缺乏工作流语义。身份系统（SPIFFE [29],AWS IAM [30]）和策略引擎（OPA [31]，Cedar [32]）通过交集组-合策略，却无法表达时间依赖关系——“操作 B 需要证明操作 A 已完-成。”Cedar 通过交集组合策略，但缺少：(1) 基于证明的工作流依赖，(2) 在运行时解析的动态主体，(3) 跨四个控制面的密码学绑定。如果没有前置操作的密码学证明，策略就必须信任由应用程序报告、攻击者可以伪造的状态。

MAPL 通过证明解决这一问题——证明是不可伪造的密码学凭证，能够证明操作已完成，从而实现可证明正确的多步骤工作流。与通过度量值证明代码完整性的平台代理证明（TPM [33]，SGX [34]）不同，MAPL 证明能够证明工作流已完成——即证明操作已执行并产生特定结果的密码学凭证，从而支持基于度量值的证明无法实现的时间依赖关系。我们将经过身份验证的系统调用 [5] 和内联引用监视器 [6], [35]从内核边界扩展到四个控制面，并通过经过身份验证的上下文为工作流依赖添加有状态验证，同时提供可插拔验证器以进行特定领域的检查（附录 A）。

正交方法。人工智能安全 [36]–[38]在训练阶段发挥作用；我们提供运行时强制执行。机密计算 [34]，[39] 保护数据机密性；我们应对工作流完整性。这些方法是互补的。

11. 结论

我们提出了经过身份验证的工作流——面向企业级智能体人工智能的首个完整信任层。我们的关键架构洞见是：定位于协议层——介于在组合时失效的应用层防御与缺乏工作流语义的基础设施原语之间——能够在保持工作流上下文的同时，跨异构框架实现统一保护。所有智能体与资源的交互都会经过四个基本控制面（提示、工具、数据、上下文）；对这些控制面进行统一保护，便可实现任一相邻层单独都无法达成的组合式安全。

三项贡献实现了这一愿景：MAPL通过证明提供密码学工作流依赖，使系统无需信任应用程序报告的状态即可实现顺序约束；嵌入式独立策略执行点实现纵深防御，在这种防御中，多个加密屏障必须同时被攻破；通过轻量适配器将九个框架集成起来，验证了表面完备性。形式化

证明（引理1-7、定理1-3）确立了安全属性；实证验证表明，在174个测试用例中召回率达到100%、假阳性为零，并且能够防御两项曾攻破基准系统的生产环境CVE漏洞。这回应了第1节提出的五项挑战：跨框架的密码学完整性、规模化的策略执行、通过组合实现权限不提升、通过哈希链维护上下文完整性，以及通过不可否认性实现完整的问责。随着智能体人工智能迈入生产环境，经过身份验证的工作流提供了安全基础，使安全部署成为可能；而目前的组合缺口正阻碍着这一技术的采用。

致谢

我们感谢匿名评审提供的宝贵反馈。本工作部分得到了 [REDACTED FOR BLIND REVIEW]的资助。

参考文献

[1] Security Researchers, 《OpenAI Atlas 浏览器中的提示注入漏洞》, 安全公告，2025年10月21日披露。[2] OpenAI, 《OpenAI 首席信息安全官关于提示注入挑战的声明》, 官方声明，2025年10月。[3] M. Rajagopalan 和 V. Rao, 《保护上下文与提示：面向非确定性人工智能的确定性安全》, 发表于第一届人工智能安全与评估国际研讨会论文集（*IASEAI'26*）, 与 *IEEE* 安全与隐私研讨会同期举行。MACAW Security, Inc. 和 ROOST.tools，2026年，即将发表。[4] G. Li、H. A. A. K. Hammoud、H. Itani、D. Khizbullin 和 B. Ghanem, 《CAMEL：用于大规模语言模型社会心智探索的交互式智能体》, *arXiv* 预印本 *arXiv:2303.17760*，2023年。[5] M. Rajagopalan、M. A. Hiltunen、T. Jim 和 R. D. Schlichting, 《面向分布式组件系统的高效灵活安全支持》, *IEEE* 可靠与安全计算汇刊，第3卷，第4期，第352–369页，2006年。[6] Ú. Erlingsson 和 F. B. Schneider, 《用于安全策略执行的内联引用监视器方法》, *ACM* 编程语言与系统汇刊，第28卷，第1期，第2–44页，2004年。[7] OWASP 基金会, 《2025年OWASP大语言模型应用十大风险》, https://genai.owasp.org/，2025年。[8] Invariant Labs, 《CVE-2025-6514：GitHub 模型上下文协议服务器中的提示注入》, https://www.invariant.sh/blog/cve-2025-6514，2025年5月，2025年5月26日披露；558，846+ 次下载。[9] Y. Wang、X. Chen、J. Li 和 W. Zhang, 《受威胁的人工智能智能体：关键安全挑战与未来路径综述》, *ACM* 计算调查，2024年，arXiv:2406.02630。[10] Y. Zhou、T. Liu 和 J. Wang, 《智能体安全综述：应用、威胁与防御》, *arXiv* 预印本 *arXiv:2510 06445*，2025年。。[11] N. Gong、Y. Liu 和 Y. Zhang, 《提示注入攻击与防御的形式化及基准测试》, 发表于第*33*届 *USENIX* 安全研讨会论文集。USENIX，2024年。[12] Y. Liu、X. Chen 和 Z. Wang, 《从提示注入到协议漏洞利用：大语言模型驱动的人工智能智能体工作流面临的威胁》, *arXiv* 预印本 *arXiv:2506.23260*，2025年6月。[13] L. Inc., 《LangChain：使用大语言模型构建应用》, https://www.langchain.com/，2023年。[14] Anthropic, 《模型上下文协议（MCP）》, https://modelcontextprotocol.io/，2024年。[15] OpenAI, 《OpenAI Assistants API》, https://platform.openai.com/docs/ assistants/，2024年。[16] Huntr Dev, 《CVE-2024-8309：LangChain 的 GraphCypherQAChain 中的提示注入导致数据库完全失陷》, https: //nvd.nist.gov/vuln/detail/CVE-2024-8309，2024年。[17] NVD, 《CVE-2024-36480：LangChain 中的远程代码执行》, https://nvd.nist.gov/vuln/detail/CVE-2024-36480，2024年，CVSS 9.0——严重。[18] F. Perez 和 I. Ribeiro, 《忽略先前的提示：语言模型的攻击技术》, *arXiv* 预印本 *arXiv:2211.09527*，2022年。[19] Y. Liu、G. Deng、Z. Xu、Y. Li、Y. Zheng、Y. Zhang、L. Zhao、T. Zhang 和 Y. Liu, 《针对集成大语言模型的应用程序的提示注入攻击》, 载于 *2023*年 *ACM SIGSAC*计算机与通信安全会议（*CCS*）论文集。ACM，2023年，第 1–15 页。[20] N. Jain、A. Schwarzschild、Y. Wen、G. Somepalli、J. Kirchenbauer、P.-y. Chiang、M. Goldblum、A. Saha、J. Geiping 和 T. Goldstein, 《针对对齐语言模型的对抗攻击的基线防御》, *arXiv* 预印本 *arXiv:2309.00614*，2023年。[21] Y. Ruan、H. Dong、A. Wang、S. Pitis、Y. Zhou、J. Ba、Y. Dubois、C. J. Maddison 和 T. Hashimoto, 《使用语言模型模拟沙箱识别语言模型智能体的风险》, *arXiv* 预印本 *arXiv:2309 15817*，2023年。。[22] W. Zou、R. Geng、B. Wang 和 J. Jia, 《PoisonedRAG：针对大语言模型检索增强生成的知识投毒攻击》, 载于 *arXiv* 预印本 *arXiv:2402.07867*，2024年。[23] Z. Cheng、G. Li、J. Liu、Q. Zhang 和 J. Shao, 《PARDEN：迈向检索增强生成中的后门攻击检测》, *arXiv* 预印本 *arXiv:2409.01573*，2024年。[24] NVIDIA, 《NeMo Guardrails：面向大语言模型应用程序的可编程护栏》, https://github.com/NVIDIA/NeMo-Guardrails，2023年。[25] Rebuff.ai, 《Rebuff：提示注入检测》, https://github.com/protectai/rebuff，2023年。[26] H. Inan、K. Upasani、J. Chi、R. Rungta、K. Iyer、Y. Mao、M. Tontchev、Q. Hu、B. Fuller、D. Testuggine 和 M. Khabsa, 《Llama Guard：用于人机对话的基于大语言模型的输入输出防护》, 2023年。[27] A. Wei、N. Haghtalab 和 J. Steinhardt, 《Jailbroken：大语言模型安全训练为何会失效？》神经信息处理系统进展，第 36 卷，2023年。[28] A. Zou、Z. Wang、J. Z. Kolter 和 M. Fredrikson, 《针对对齐语言模型的通用且可迁移的对抗攻击》, *arXiv* 预印本 *arXiv:2307.15043*，2023年。[29] CNCF, 《SPIFFE：面向所有人的安全生产身份框架》, https://spiffe.io/，2023年。[30] 亚马逊云科技, 《AWS 身份与访问管理：策略组合与授权》, https://aws.amazon.com/iam/，2023年。[31] CNCF, 《Open Policy Agent：基于策略的控制》, https://www.openpolicyagent.org/，2023年。[32] A. W. Services, 《Cedar：用于授权的策略语言》, https: //www.cedarpolicy.com/，2023年。[33] 可信计算组织, 《TPM 2.0 库规范》, 可信计算组织，技术报告，2016年，规范系列“2.0”，级别 00，修订版 01。38。[34] V. Costan 和 S. Devadas, 《Intel SGX 解析》, *IACR* 密码学电子印刷档案，第 2016 卷，第 086 页，2016年。[35] F. B. Schneider, 《可执行安全策略》, *ACM* 信息与系统安全汇刊，第 3 卷，第 1 期，第 30–50 页，2000年。[36] Y. Bai、S. Kadavath、S. Kundu、A. Askell、J. Kernion、A. Jones、A. Chen、A. Goldie、A. Mirhoseini、C. McKinnon 等, 《宪法式人工智能：来自人工智能反馈的无害性》, *arXiv* 预印本 *arXiv:2212.08073*，2022年。

[37] L. Ouyang、J. Wu、X. Jiang、D. Almeida、C. Wainwright、P. Mishkin、C. Zhang、S. Agarwal、K. Slama、A. Ray 等，《通过人类反馈训练语言模型遵循指令》，神经信息处理系统进展，第 35 卷，27 744，2022年 27730pp. – 。

[38] D. Ganguli, L. Lovitt, J. Kernion, A. Askell, Y. Bai, S. Kadavath, B. Mann, E. Perez, N. Schiefer, K. Ndousse 等，“红队测试语言模型以减少危害：方法、扩展行为与经验教训”，*arXiv* 预印本 *arXiv:2209.07858*，2022。

[39] C. Gentry，《一种全同态加密方案》，博士论文，斯坦福大学，2009。

附录

附录 A：MAPL 策略示例

基础组织策略

```
{ "策略_标识符": "acme:base", "资源": [ "工具:*", "↪ 大语言模型:*" ],"拒绝的_资源": [ "*↪ 凭证*" ],"约束": { "证明": [ "↪用户已认证_↪ " ]} }
```

包含凭证阻止规则的基线。

部门策略（财务）

```
{ "policy_id": "acme:finance↪ ", "extends": "acme:base", "资源": [ "数据:↪ 财务:**" ],"约束": { "证明": [ "↪ 多因素认证_已验证" ]} }
```

继承基础策略，添加多因素认证。

证明链

```
// 步骤 1：分析 {"policy_id": "fin:analyze↪ ", "resources": [ "tool:analyze↪ " ]," constraints": {"↪ attestations":["user_authenticated"]}} // -> 生成：↪ 分析_完成 // 步骤 2：要求步骤 1 {"policy_id": "fin:report", "resources": [ "tool:report↪ " ],"constraints": {"↪ attestations":["analysis_done"]}} // -> 生成：报告_完成 // 步骤 3：要求步骤 2 {"policy_id": "fin:send", "resources": [ "tool:email↪ " ],"constraints": {"↪ attestations":["report_done"]}}
```

密码学证明可防止工作流劫持。

资源策略（数据库）

```
{
  "策略_标识符": "数据库:↪ 客户_数据库",
  "约束": {
    "拒绝_参数": {
      "查询": ["DROP", "↪ DELETE"]
    }
  }
}
```

双视角约束。

委派策略

```
{ "策略_标识符": "智能体:↪ 子_智能体", "extends": "acme:finance", "资源": ["数据:↪ finance:reports↪ :*" ],"约束": { " 参数 ": {"↪ 日期 _ 范围 ": ["2024年-Q1", "2024年-Q2"]} } }
```

收窄的委派范围。

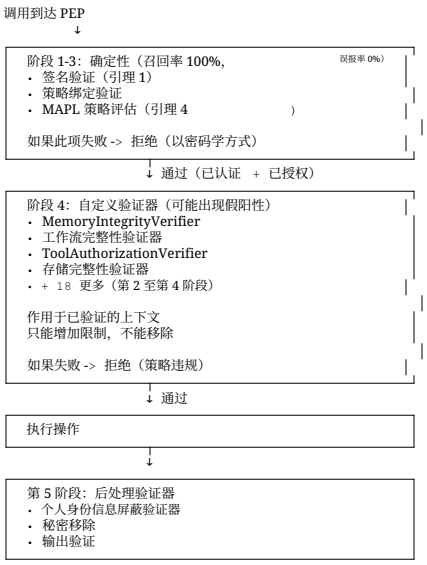
紧急访问

```
{
  "策略_标识符": "紧急:↪ 事件",
  "扩展": "acme:base ",
  "有效期": {
    "不晚于": "2024年10月25日_↪ T16:00:00Z"
  },
  "约束": {"↪ 证明",
    ["ciso_已批准"]}
}
```

限时窗口）。（2小时

附录 B：自定义验证器实现验证流水线架构

PEP 验证流水线由五个阶段组合而成：



即使自定义验证器遭到入侵，阶段 1-3 仍会强制执行密码学完整性和策略约束。

生产环境验证器

四项生产环境验证器通过对以密码学方式验证的上下文进行特定领域检查，应对 OWASP十大安全风险威胁：

MemoryIntegrityVerifier防止记忆投毒（*OWASP LLM01*）
•速率限制（每分钟 10 次更新）•受保护字段（目标、系统提示词）
• 模式检测（可疑更新）
• 密码学完整性检查

ToolAuthorizationVerifier防止工具滥用（*OWASP LLM06*）•基于角色的访问控制（角色到工具的映射）
•危险模式（写入→执行）
• 使用速率限制（每分钟 30 个工具）
• 权限提升防护

WorkflowIntegrityVerifier
防止工作流劫持（*OWASP LLM04*）
• 序列强制（前置条件）
• 状态跟踪（防止重新执行）
•证明链（密码学证明）
• 时长限制（最长 3600 秒）

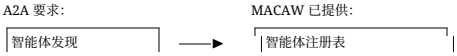
存储完整性验证器
防止数据外泄（*OWASP LLM02*）
•路径遍历防护
• 数据分类（机密/敏感）
• 强制加密
• 外泄检测（批量读取）

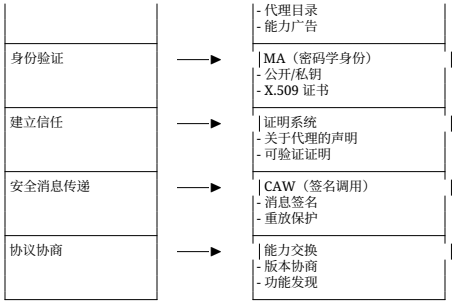
附录 C：框架集成详情

详细映射展示了九种框架中的框架构造如何对应经身份验证的工作流原语。

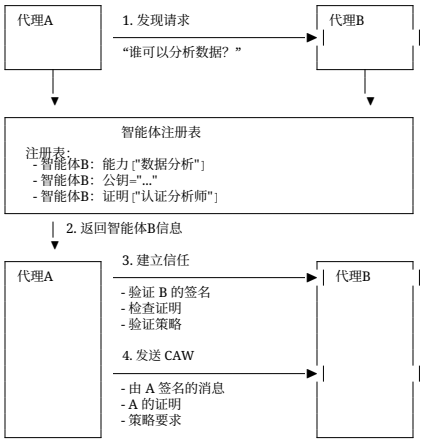
A2A（代理间协议）

A2A 为智能体发现、信任建立和通信定义了标准。MACAW 提供了 A2A 所需的密码学基础设施：





完整的 A2A 流程：

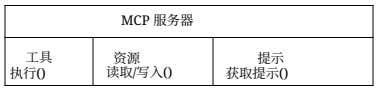


安全挑战：智能体必须在委派权限前验证身份和能力。证明提供加密声明，使我们无需依赖集中式权威机构即可做出信任决策。

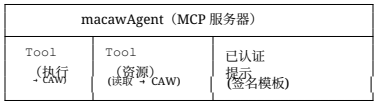
MCP（模型上下文协议）

MCP 服务器提供三种能力类型：工具（可执行函数）、资源（支持读取/写入/监视操作的数据访问）和提示（可检索模板）。MACAW 映射：

MCP 核心概念：



MACAW 映射：



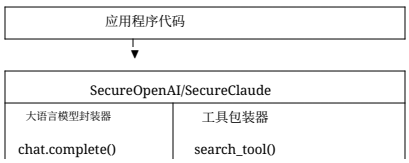
资源流：客户端：“读取 file.txt” → SecureMCP：调用工具()→ 策略执行点：签名有效吗？ → 策略：允许读取吗？→ 如果允许则执行

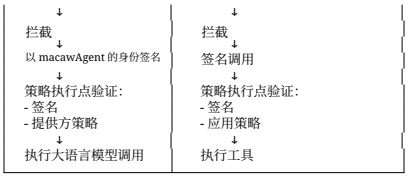
安全挑战：资源代表只读数据访问，而工具可以修改状态。这种映射通过策略实施只读约束，防止权限从数据检索提升到数据修改。

OpenAI 和 Claude（大语言模型接口）

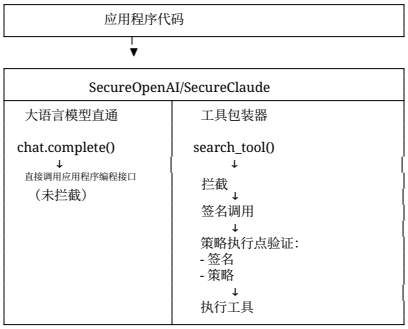
大语言模型提供商通过函数调用提供推理 API。根据不同的信任假设，存在两种部署变体：

双侧封装：大语言模型 API 和客户端均作为 macawAgents





单侧封装：仅将客户端工具作为 macawAgents

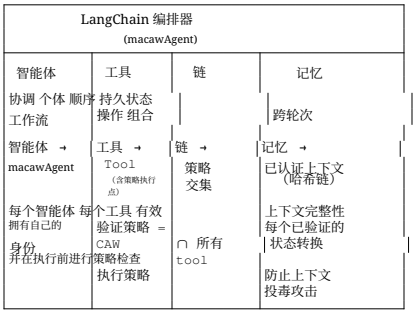


安全挑战：大语言模型在授权决策方面不可信。即使经过提示工程，对抗性提示仍可能导致未经授权的函数调用。策略执行点无论大语言模型做出何种决策，都会在函数执行边界进行验证。

LangChain（多智能体编排）

LangChain 协调多跨工具和大语言模型进行协调。多层集成涵盖四个层级：

LangChain 架构：



工作流流程：用户：“分析文档并发送摘要”↓LangChain 智能体（macawAgent）↓步骤 1：读取文档→ 工具调用 → CAW → PEP 检查 → 执行↓步骤 2：分析内容→ 大语言模型调用 → CAW → PEP 检查 → 执行↓步骤 3：发送电子邮件→ 工具调用 → CAW → PEP 检查 → 执行

通过交集进行策略组合：如果 tool_1 的策略允许 {A, B, C}，且 tool_2 的策略允许 {B, C, D}，并且链将 tool_1 → tool_2 组合起来，那么有效策略为 = {B, C}（交集）。链组合不能放宽工具策略。

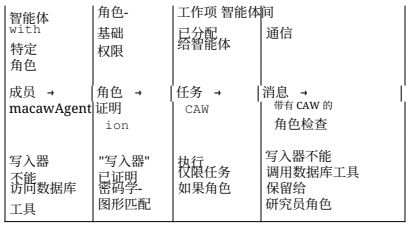
安全挑战：编排器除了单个工具之外，还暴露了多个控制点——链组合逻辑、智能体推理、内存操作。仅保护工具会使组合和状态管理得不到保护。

CrewAI（基于角色的多智能体）

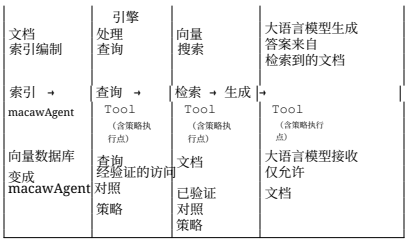
CrewAI 将智能体组织成团队，每个成员都承担一个角色，协同执行任务：

CrewAI架构：





基于角色的流程：团队：研究员 + 写入器↓任务：“撰写关于第四季度数据的报告”↓研究员（带有“研究员”证明的 macawAgent）：→ 调用数据库查询工具→ PEP 检查：签名有效吗？角色=是“研究员”吗？→ 策略：“数据库查询”要求提供 [“研究员”证明]→ 允许：执行查询↓写入器（带有“写入器”证明的 macawAgent）：→ 接收来自研究员的数据→ 尝试使用数据库查询工具→ PEP 检查：签名有效吗？角色=是“写入器”吗？→ 策略：“数据库查询”要求提供 [“研究员”证明]→ 拒绝：写入器缺少所需的证明



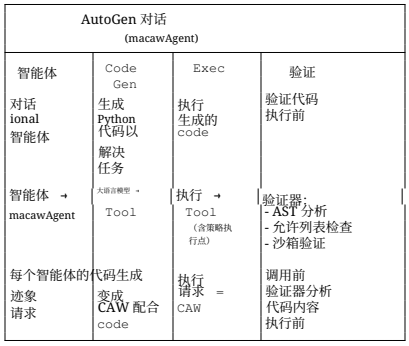
带安全机制的 RAG 流程：用户：“第四季度收益是多少？”↓查询引擎（带 PEP 的工具）→ CAW：查询=“第四季度收益”→ PEP 检查：- 签名有效吗？- 策略允许此查询模式吗？- 拒绝模式：* 密码 * 、 * 凭据*→ 已批准：继续↓检索器（带 PEP 的工具）→ CAW：检索与“第四季度收益”匹配的文档→ PEP 检查：- 是否可以访问这些文档？- 允许的文件：* 公开 * 、 * 报告 * - 拒绝的文件：* 私有 * 、 * 机密 *→ 已批准：检索文档↓生成器（带 PEP 的工具）→ CAW：根据检索到的文档生成答案→ PEP 检查：输出验证→ 向用户返回答案

安全挑战：必须以密码学方式强制执行基于角色的权限。“写入器”角色不应访问为“研究员”保留的数据库，即使驱动写入器的大语言模型尝试进行未授权调用。证明以密码学方式将角色与智能体身份绑定。

AutoGen（自主代码生成）

AutoGen 支持自主代码生成与执行。其安全挑战在于不受限制的代码执行：

AutoGen 架构：



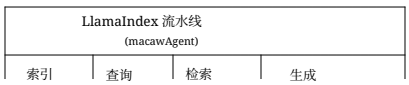
代码执行流程：用户：“计算斐波那契(10)”↓AutoGen Agent 生成代码：def fib(n): return n if n<2 else fib(n-1)+fib(n-2)↓执行请求 → CAW↓PEP 调用执行前验证器：1. AST 分析：检测危险模式（os.system、subprocess、eval、exec、import sys）2. 允许列表检查：仅允许 math/基本操作 3. 沙箱验证：禁止网络/文件访问↓验证器批准 → 在沙箱中执行 → 返回结果↓恶意尝试：“os.system('rm -rf /')”↓AST 分析检测到 os.system → 拒绝

安全挑战：不受限制的代码执行可能导致任意操作，包括数据外泄和权限提升。可插拔验证器会在执行前通过 AST 解析分析代码，并强制执行允许列表和沙箱约束。

LlamaIndex（检索增强生成流水线）

LlamaIndex 提供用于文档处理和问答的检索增强生成（RAG）流水线：

LlamaIndex 架构：

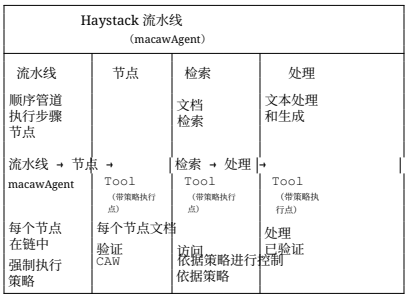


安全挑战：检索增强生成流水线会访问不受信任的文档存储库。恶意文档可能包含提示注入或敏感数据。检索前，策略执行点会验证文档访问模式和查询约束。

Haystack（文档处理管道）

Haystack 提供文档处理管道，其中包含用于检索、预处理和生成的节点：

Haystack 架构：



流水线流程：用户：“处理 ./reports/ 中的文档”↓流水线(macawAgent)↓节点 1：文件读取器（带 PEP 的工具）→ CAW：读取（"./reports/"）→ PEP 检查：- 允许的路径：*reports*、*public* - 拒绝的路径：*credentials*、*private*→ 已批准：读取文件↓节点 2：预处理器（带 PEP 的工具）→ CAW：清理文本，移除个人身份信息→ PEP 检查：个人身份信息检测验证器→ 执行脱敏↓节点 3：生成器（带 PEP 的工具）→ CAW：生成摘要→ PEP 检查：输出验证→ 返回结果 跨节点策略组合：如果节点 1 允许 {A, B, C}，且节点 2 允许 {B, C, D}，并且流水线将节点 1 → 节点 2 串联起来，那么有效策略为 = {B, C}（交集）

安全挑战：文档处理管道会访问文件系统和外部数据源。节点必须强制执行路径约束和内容验证，以防止未经授权的访问或数据外泄。

附录 D：AI 智能体威胁分类

ID	攻击	描述与影响
类别 A：提示注入与输入操纵		
T-A1	隐藏式提示注入	通过白底白字文本、CSS 和 HTML 注释嵌入。影响：命令执行、凭证窃取。
T-A2	语义混淆	改述以绕过过滤器，使用 Base64 编码。影响：绕过过滤器。
T-A3	多轮投毒	在多个轮次中逐步注入，伪造历史记录。影响：权限提升。
T-A4	延迟激活	条件触发器（定时炸弹）。影响：规避即时检测。
B类：工具链式调用与推导漏洞利用		
T-B1	权限提升via链式调用	组合良性操作（搜索→列表→读取）。影响：规避策略。
T-B2	语义漂移	逐渐偏离意图。影响：解释错误不断累积。
T-B3	推导深度耗尽	深度递归链。影响：资源耗尽、策略漂移。
类别C：凭证与会话攻击		
T-C1	令牌提取	从上下文中读取凭据。影响：会话劫持、云访问。
T-C2	凭据外泄	工具滥用序列（读取→电子邮件）。影响：持续未授权访问。
T-C3	OAuth流程操纵	诱骗 OAuth 为攻击者应用授权。影响：账户接管。
T-C4	会话重放	重复使用有效的已签名提示。影响：利用过期的身份认证。
类别 D：数据外泄与隐私泄露		
T-D1	对话历史挖掘	从历史记录中提取个人身份信息/机密。影响：违反 GDPR/HIPAA、知识产权盗窃。
T-D2	跨主体泄漏	上下文隔离不足。影响：未经授权的跨用户访问。
T-D3	嵌入空间投毒	污染向量数据库/检索增强生成。影响：持久性数据污染。
T-D4	文档收集	系统性文件提取。影响：大规模数据窃取。
类别 E：恶意软件与代码执行		
T-E1	AI 调用的恶意软件	下载/执行恶意代码。影响：勒索软件、后门。
T-E2	代码生成利用	生成伪装的恶意脚本。影响：资源劫持。
T-E3	持久化安装	创建后门（.bashrc、cron）。影响：持续性入侵。
T-E4	通过软件包实施供应链攻击	安装被植入木马的依赖项。影响：依赖混淆。
类别 F：资源耗尽与成本攻击		
T-F1	计算资源耗尽	递归操作、无限循环。影响：拒绝服务、账单膨胀。
T-F2	API 速率限制耗尽	通过循环耗尽配额。影响：服务降级、数千美元的+ 成本。
T-F3	存储／带宽饱和	无限制下载/生成。影响：系统故障。
类别 G：多智能体系统攻击		
T-G1	智能体间感染	通过 MCP/A2A 投毒提示。影响：级联入侵。
T-G2	工作流劫持	精心构造的代理间请求。影响：分布式策略绕过。
T-G3	拜占庭代理行为	持有有效凭证的恶意代理。影响：已授权攻击。
T-G4	信任锚点入侵	破坏注册表/身份提供商。影响：完全入侵。
类别 H：策略与合规违规		
T-H1	策略混淆	利用规则歧义。影响：授权漏洞。
T-H2	审计跟踪操纵	篡改日志、伪造记录。影响：取证证据丢失。
T-H3	合规绕过	违反 GDPR/HIPAA/SOC 2。影响：法律责任、4% 的营收罚款。
T-H4	证明伪造	伪造的安全检查证明。影响：工作流安全绕过。

涵盖八个类别（*T-A* 至 *T-H*）中 *25* 种攻击变体的全面分类体系。第 *8* 节验证覆盖率。