代数溯源 Agent 流控系统（Holographic Pass）开发全指南：从数学基石、嵌套架构、安全溯源、防御机制到压力测试的完整实施方案

引言：Agent 网络的“黑箱”危机与代数溯源的兴起

随着大语言模型（LLM）驱动的自主智能体（Agent）在企业级工作流中的大规模部署，软件架构正在经历从“确定性编排”向“概率性协作”的范式转移。在传统的微服务架构中，调用链是静态定义的，通过 Jaeger 或 Zipkin 等分布式追踪工具即可实现全链路的可观测性。然而，在 Agent 网络中，决策路径是动态生成的，数据流转不再遵循预设的硬编码逻辑，而是由 Agent 根据上下文实时推理决定。这种灵活性带来了前所未有的“黑箱”危机：当一个金融审批 Agent 批准了一笔异常交易时，是因为它的内部推理出现了幻觉，还是因为它接收到了上游数据清洗 Agent 传递的错误数据？传统的日志和堆栈跟踪（Stack Trace）在非确定性的 AI 系统中显得苍白无力，无法提供具有数学确定性的因果证据。

本报告旨在详细阐述一种全新的解决方案——代数溯源 **Agent** 流控系统（Algebraic Provenance Agent Flow Control System），工程代号为 **Holographic Pass**（全息通行证）。该系统不依赖于被动的日志记录，而是利用密码学累加器（**Cryptographic Accumulators**）、零知识证明（**ZKPs**）和同态加密（**Homomorphic Encryption**），将 Agent 的每一次交互、决策和状态变更映射为不可篡改的代数结构。Holographic Pass 就像一个携带了自身完整历史全息投影的数字令牌，它不仅记录了“谁做了什么”，还通过递归证明确保了整个因果链条的数学有效性，实现了从“基于日志的审计”到“基于密码学的验证”的跨越。

本指南将深入探讨该系统的数学基石、架构设计、安全防御机制以及在 Rust 环境下的高性能工程实施方案，为构建零信任（Zero-Trust）的 Agent 网络提供详尽的参考。

第一部分：数学基石——构建可验证的因果半环

Holographic Pass 的核心理念是将工作流的拓扑结构转化为代数方程。为了实现这一点，我们需要引入 provenance semiring（溯源半环）的概念，并利用 RSA 累加器和同态加密将其具象化。

1.1 溯源半环（**Provenance Semirings**）与代数模型

在理论计算机科学中，数据的血缘关系（Provenance）通常使用半环结构 $(K, \oplus, \otimes, 0, 1)$ 来建模 [1]。在这个代数结构中：

- 加法 **($\oplus$)** 代表**“选择”或“替代路径”**：如果一个结果是由 Agent A 或者 Agent B 产生

的，其溯源表达式为 $P(A) \oplus P(B)$。这对应于工作流中的分支（Branching）或多路复用。
- 乘法 **($\otimes$)** 代表“联合”或“路径组合”：如果一个结果需要经过 Agent A 然后 经过 Agent B 处理，其溯源表达式为 $P(A) \otimes P(B)$。这对应于工作流中的序列（Sequence）或依赖关系。

在 Holographic Pass 系统中，我们不仅仅是用符号记录这些关系，而是将其映射到具体的数论运算上：

- 乘法 **($\otimes$)** 映射为 **RSA** 模幂运算：路径的延伸通过在累加器上不断进行指数运算来实现。
- 加法 **($\oplus$)** 映射为集合并集或向量承诺：多条路径的汇聚通过零知识证明对多个累加器状态的聚合验证来实现。

这种映射使得工作流的每一次流转都变成了对一个全局代数方程的求解过程。

1.2 动态 RSA 累加器：状态的压缩与验证

Holographic Pass 的核心引擎是 **RSA** 累加器。与 Merkle Tree 相比，RSA 累加器具有两个在 Agent 流控中至关重要的特性：恒定大小的成员证明（**Constant-size Membership Proofs**）和动态更新能力（**Dynamic Updates**）[3]。

1.2.1 定义与构造

设 $N = pq$ 为一个大整数模数，其中 p, q 为安全强素数（Safe Primes）。累加器的状态 A 定义为模 N 剩余类环中的一个元素。给定一组 Agent 的身份标识集合 $S = \{x_1, x_2,..., x_n\}$，累加器的值 A 计算如下：

$$A = g^{\prod_{x \in S} x} \pmod N$$
其中 g 是群 QR_N（模 N 的二次剩余群）的一个生成元。

这里的关键约束是：集合 **S** 中的每个元素 **x_i** 必须是素数。如果 Agent 的身份 ID 被映射为合数，攻击者可能通过分解合数来伪造路径。因此，系统必须包含一个确定的 **Hash-to-Prime** 映射机制，将任意字符串（如 Agent UUID）映射为素数域中的元素 [6]。

1.2.2 成员证明（**Membership Witness**）

当一个请求到达 Agent Z 时，Agent Z 需要验证该请求是否确实经过了 Agent X 的处理。在 Holographic Pass 中，这不需要查询中心化数据库，而是通过验证“成员证明”来实现。
对于集合 S 中的元素 x（Agent X 的素数 ID），其成员证明 w_x 是除 x 以外所有元素的累加值：

$$w_x = g^{\prod_{y \in S, y \neq x} y} \pmod N$$
验证过程仅需一步模幂运算：

$$(w_x)^x \equiv A \pmod N$$
如果等式成立，则数学上可以保证 Agent X 确实参与了累加器状态 A 的构建。这种验证是零知识的，因为验证者只需要知道当前状态 A 和 Agent X 的 ID，而无需知道路径中的其他 Agent（$y \in S, y \neq x$），从而保护了工作流的隐私 [5]。

1.2.3 动态性与无陷门更新

Camenisch 和 Lysyanskaya 在 2002 年提出的动态累加器方案允许在不重新计算整个累加器的情况下添加或删除元素 [3]。

- 添加元素：$A_{new} = A_{old}^x \pmod N$。这是一个 $O(1)$ 操作，Agent 可以极其高效地将自己“盖章”到通行证上。
- 更新证明：当累加器状态从 A 更新为 A'（添加了元素 y）时，旧的证明 w_x 可以通过 $w_x' = w_x^y \pmod N$ 进行更新，无需通过中心节点。

这使得 Holographic Pass 能够在完全去中心化的 Agent 网络中流转，每个节点都能独立更新状态和维护证明。

1.3 同态加密（Paillier）：状态的盲计算

RSA 累加器解决了“路径溯源”问题（即“经过了谁”），但无法解决“状态溯源”问题（即“计算了什么”）。例如，在信贷审批流中，我们需要累加多个 Agent 计算的风险评分，同时不希望中间的 Agent 窥探到总分。为此，我们引入 **Paillier** 同态加密系统 [9]。

Paillier 密码系统具有加法同态性（Additive Homomorphism）：

$$D(E(m_1) \cdot E(m_2) \pmod{n^2}) = m_1 + m_2 \pmod n$$

$$D(E(m)^k \pmod{n^2}) = m \cdot k \pmod n$$
在 Holographic Pass 中，除了累加器状态 A，还携带一个加密的状态向量 C_{state}。

- 初始化：入口网关生成 $C_{state} = E(0)$。
- 更新：Agent i 计算出本地评分 s_i，并将其加密为 $E(s_i)$（或者利用同态性质直接操作密文），然后更新 Pass 中的状态：$C_{new} = C_{old} \cdot E(s_i) \pmod{n^2}$。
- 结果：最终的 Pass 包含 $E(\sum s_i)$。只有持有私钥的终点节点（或审计员）解密得到总分。

通过将 RSA 累加器（拓扑证明）与 Paillier 密文（数据计算）绑定，Holographic Pass 实现了结构与内容的双重溯源 [12]。

1.4 递归零知识证明（Recursive SNARKs）：全息的本质

“全息”一词暗示了部分包含整体的信息。为了防止恶意的 Agent 虽然更新了累加器但没有正确

执行内部逻辑（例如，随便填了一个风险分），我们需要 **zk-SNARKs**（简洁非交互零知识论证 ）[14]。

在传统的 ZK 系统中，证明生成是昂贵的。如果 Agent链很长，生成一个包含所有步骤的证明是不切实际的。递归 **SNARKs**（如 Halo2 或 Nova 方案）允许我们将前一步的证明 π_{t-1} 作为输入，生成一个新的证明 π_t，验证“我验证了前一步的证明是正确的，并且我也正确执行了当前步骤”。

$$\pi_t = Prove(Circuit(State_{t-1}, State_t, \pi_{t-1}, Witness_t))$$
这样，无论 Agent 链有多长，Holographic Pass 中始终只携带一个常数大小的证明 π_{final}，它递归地包含了整个历史的有效性。这就是“全息”的数学实现 [16]。

第二部分：全息通行证（Holographic Pass）的嵌套架构

将上述数学原语转化为工程架构，需要设计精确的数据结构和交互协议。本部分详细拆解 Holographic Pass 的系统架构。

2.1 数据结构定义

Holographic Pass 是一个在 HTTP Header 或 gRPC Metadata 中传递的二进制对象。以下是其基于 Protobuf 的定义建议：

Protocol Buffers

```
syntax = "proto3";

message HolographicPass {
  // 1. 拓扑溯源层 (RSA Accumulator)
  // 当前累加器的值 A_t (2048-bit 大整数，大端序字节流)
  bytes accumulator_tip = 1;
  // 上一步状态的成员证明，用于快速验证链式连续性
  bytes previous_witness = 2;

  // 2. 数据隐私层 (Paillier Encryption)
  // 同态加密的状态向量（例如：累积风险分、累积耗时、累积成本）
  bytes encrypted_state_vector = 3;
  // 用于防止重放攻击的随机数或时间戳元数据
  bytes flow_metadata = 4;
```

```
  // 3. 计算完整性层 (Recursive ZKP)
  // 递归 SNARK 证明，验证从创世状态到当前状态的所有状态转移均合法
  bytes recursive_proof = 5;

  // 4. 身份层
  // 当前 Agent 的 Prime ID (用于调试，实际验证依赖数学计算)
  string current_agent_id = 6;
}
```

2.2 质数身份层（Prime Identity Layer）

RSA 累加器的安全性严格依赖于 $gcd(x_i, x_j) = 1$，即所有 Agent 的 ID 必须互素。在分布式系统中，最简单的实现方式是将 Agent ID 映射为素数 [5]。

2.2.1 确定性 Hash-to-Prime 算法

系统不能依赖中心化的“发号器”来分配素数，因为这会成为单点故障和性能瓶颈。我们需要一个确定性算法 $H_{\mathbb{P}}: \{0,1\}^* \to \mathbb{P}$。

推荐算法流程：

1. 输入：Agent UUID (例如 credit-service-v2)。
2. 哈希：计算 $h = SHA256(UUID)$。
3. 搜索：以 h 为种子，设置计数器 $nonce = 0$。
4. 候选生成：计算 $candidate = SHA256(h + nonce)$，并强制设置最高位和最低位为1（确保是特定位长的大奇数）。
5. 素性测试：对 $candidate$ 进行 **Miller-Rabin** 素性测试。
   - 为了保证确定性，必须使用固定的底数集（Bases）。对于 2048 位以下的数，选取特定的底数集可以实现确定性判断 [19]。
   - 或者使用 **Pocklington** 判别法 生成可证明的素数，但这计算成本更高。
6. 输出：返回第一个通过测试的素数。

性能考量：
生成一个 2048 位的素数极为耗时（秒级），但生成 256 位素数则快得多（毫秒级）。鉴于 Agent ID 是静态的，Agent 应当在启动时（Cold Start）计算并缓存自己的 Prime ID，而不是在处理每个请求时实时计算 21。

2.3 嵌套 DAG 遍历逻辑

Holographic Pass 在 Agent 网络中的流转逻辑如下：

2.3.1 顺序节点 (A -> B)

当请求从 Agent A 传给 Agent B：

1. 接收验证：Agent B 接收 Pass，验证 recursive_proof 是否有效，确保 A 的状态合法。
2. 业务执行：Agent B 执行 LLM 推理或业务逻辑，得到结果。

3. 同态更新：Agent B 将自己的执行指标（如耗时、Token数）加密后同态加到 encrypted_state_vector。
4. 累加器更新：Agent B 计算 $A_{new} = (A_{tip})^{PrimeID_B} \pmod N$。
5. 递归证明生成：Agent B 调用 ZK Prover，输入 (π_A, A_{tip}, A_{new}, $PrimeID_B$, $PrivateInputs$)，生成新的证明 π_B。
6. 转发：构造新的 Pass 发送给下游。

2.3.2 路径合并与向量承诺 (The Merge Problem)

当工作流出现分支（Fork）后汇聚（Join），例如 Agent B 和 Agent C 并行处理，然后汇聚到 Agent D，这是一个代数难题。
RSA 累加器本质上是一个集合。$Path_B$ 包含集合 $\{A, B\}$，$Path_C$ 包含集合 $\{A, C\}$。Agent D 需要一个代表 $\{A, B, C\}$ 的状态。

- 难题：在不知道 N 的因子（无陷门）的情况下，已知 g^{AB} 和 g^{AC}，很难计算出 g^{ABC}（这涉及求根问题）[23]。
- 解决方案：向量承诺与 ZK 聚合。
  Agent D 不试图在代数上合并两个累加器值。相反，Agent D 在 ZK 电路中证明："我收到了两个有效的 Pass，一个是 $Pass_B$，一个是 $Pass_C$"。
  Agent D 生成的新 Pass 的累加器状态可能会被重置或映射到一个新的向量承诺根（Vector Commitment Root），该根承诺了分支的叶子节点状态。
  $$ Root = Hash(A_B |

| A_C) $$
随后的递归证明 π_D 将验证 $Root$ 的构造以及 π_B 和 π_C 的有效性。这种方法规避了 RSA 模幂的代数困难，利用 ZK 的通用计算能力处理复杂的 DAG 拓扑 14。

第三部分：安全溯源与防御机制

构建在数学之上的系统虽然理论完美，但在现实部署中面临诸多攻击向量。本部分详细阐述防御策略。

3.1 无陷门设置（Trapdoorless Setup）与 RSA-UFO

RSA 累加器的安全性基于强 RSA 假设（Strong RSA Assumption），前提是攻击者不知道模数 N 的因子 (p, q)。如果系统管理员知道 (p, q)，他们就可以随意伪造任意 Agent 的路径证明（计算 e-th root 变得微不足道）。

为了实现零信任（**Zero Trust**），必须消除陷门：

1. **RSA-UFO (Unknown Factorization Object)**：使用多方安全计算（MPC）仪式生成 N。只要参与者中有一个是诚实的并销毁了私钥分片，N 的因子就是未知的。这适用于联盟链式的企业环境 [26]。
2. 类群（**Class Groups of Imaginary Quadratic Fields**）：这是目前的学术界前沿方案。类群提供了一个天然的未知阶群（Group of Unknown Order），不需要 trusted setup。虽然其群

运算比 RSA 慢（约 10 倍差距），但它彻底消除了后门风险，是高安全等级场景的首选 [6]。

3.2 顺序保持与防篡改：哈希链技术

标准的 RSA 累加器是可交换的（Commutative）：$g^{a \cdot b} = g^{b \cdot a}$。这意味着它无法区分路径 A->B 和 B->A。在流控系统中，顺序至关重要（例如：必须先“鉴权”再“转账”）。

防御机制：依赖性哈希链（Dependent Hash Chaining）。
为了强制顺序，我们修改 Prime ID 的生成逻辑，使其依赖于前一个状态：
$$ P_{t} = HashToPrime(AgentID_t |
| A_{t-1}) $$$$ A_t = A_{t-1}^{P_t} \pmod N$$

通过将前一个累加器状态 A_{t-1} 混合到当前 Agent 的素数映射过程中，我们创造了一个密码学历史链。

- 如果是 A->B：$P_A = H(A)$, $P_B = H(B |

| A_A)$。

- 如果是 B->A：$P_B = H(B)$, $P_A = H(A |

| A_B)$。
显然，P_B 在两种情况下是完全不同的素数，导致最终的累加器值截然不同。这使得 Holographic Pass 不仅记录了集合成员，还通过素数生成的依赖性锁定了时间顺序 5。

3.3 女巫攻击（Sybil Attack）防御

攻击场景：攻击者生成 10,000 个虚假的 Agent 节点，试图通过大量的噪声数据淹没真实的溯源信息。

防御机制：计算成本不对称性。
由于 Hash-to-Prime 算法寻找一个 2048 位的素数需要显著的 CPU 时间（例如 0.5 秒），攻击者要伪造一条包含 1,000 个节点的链，需要预先计算 500 秒的素数生成工作量证明（PoW）。这种天然的计算门槛有效地遏制了大规模女巫攻击。此外，验证者（Verifier）持有一份合法 Agent 的 Registry 白名单（或 Merkle Root），任何不在白名单中的 Prime ID 生成的转换都会在 ZK 验证阶段失败 22。

3.4 重放攻击与 Epoch 机制

攻击场景：攻击者截获了一个“已批准”的高权限 Pass，并试图多次提交给支付 Agent。

防御机制：基于 Epoch 的动态生成元。
系统每隔一定时间（如 10 分钟）切换一次累加器的生成元 $g_{epoch} = Hash(EpochID)$。

- Pass 中包含的时间戳必须与当前的 Epoch 匹配。
- 过期的 Pass 由于 g 值不同，其累加器状态在数学上无法通过新 Epoch 的验证。
- 此外，在 Paillier 加密状态中嵌入一个唯一的 Request_Nonce，并在终点进行布隆过滤器（Bloom Filter）去重，确保每个 Pass 只能被消费一次 [25]。

第四部分：工程实施与 Rust 生态深度解析

理论必须落地为代码。由于涉及到大数运算和密码学原语，**Rust** 是唯一能同时满足内存安全和极致性能的语言选择。本部分将深入探讨库的选型与性能权衡。

4.1 大数运算库选型：性能生与死

在 Holographic Pass 中，最频繁的操作是 2048 位的模幂运算（Modular Exponentiation, $b^e \pmod m$）。其性能直接决定了 Agent 的延迟。

我们对比 Rust 生态中的三大主流库：num-bigint、ibig 和 rug (基于 GMP)。

表 4.1：Rust 大数库模幂运算性能基准测试 (2048-bit) [30]

库名称	语言/底层	模幂延迟 (Latency)	吞吐量 (Ops/sec)	优点	缺点
num-bigint	Pure Rust	~7.0 ms	~140	纯 Rust，编译简单，无 FFI	极慢。缺乏高级算法（如蒙哥马利乘法优化不足）。
ibig	Pure Rust	~1.0 ms	~1,000	纯 Rust，性能优秀，算法先进	相比 GMP 仍有差距，生态稍小。
rug (GMP)	C Bindings	**~0.25 ms**	**~4,000**	极致性能，工业标准	依赖系统级 GMP 库，构建复杂 (FFI)，Unsafe 代码风

					险。

实施建议：

- 对于高性能网关和核心 **Agent**：必须使用 **rug**。在 100 跳的深度链路中，rug 仅引入 25ms 的总加密延迟，而 num-bigint 会引入 700ms，这对实时系统是不可接受的。
- 对于边缘设备或 **WASM** 环境：使用 ibig 或 dashu，因为 GMP 难以移植到 WASM。

4.2 ZKP 后端集成：Halo2 vs Plonky2

为了实现递归证明，我们需要选择支持“证明之证明”（Proof-of-Proof）的 ZK 系统。

- **Groth16**：验证快，但需要针对每个电路进行 Trusted Setup。不适合动态递归。
- **Halo2 (ECC)**：Zcash 团队开发，无需 Trusted Setup，支持递归（Accumulation Scheme）。Rust 生态支持极好（halo2_proofs）。
- **Plonky2 (FRI)**：Polygon Zero 团队开发，基于 Goldilocks 域，针对递归速度进行了极致优化。证明生成速度极快（毫秒级 ）。

实施建议：
鉴于 Holographic Pass 需要极低的延迟，Plonky2 是目前的最佳选择。它能在消费级硬件上在 100-200ms 内生成递归证明，且原生支持 Rust。
开发者需要编写一个 Circuit，该 Circuit 将 Old_Accumulator、Prime_ID、New_Accumulator 作为 Private Inputs，并在电路内部约束 Old ^ Prime == New (mod N)。
注意：在 ZK 电路中做 2048 位大整数运算极其昂贵。工程上通常使用 VDF 验证逻辑 或 SNARK-friendly Groups 来间接验证，或者将 RSA 模数运算拆解为多个小域运算（CRT, 中国剩余定理）33。

4.3 关键代码实现逻辑（Rust）

以下是使用 rug 库实现累加器更新的核心逻辑片段：

Rust

```rust
use rug::{Integer, ops::Pow};

pub struct HolographicAccumulator {
    modulus: Integer, // N
}

impl HolographicAccumulator {
```

```
    // 确定性 Hash-to-Prime 实现
    pub fn hash_to_prime(&self, input: &[u8]) -> Integer {
        let mut nonce = 0;
        loop {
            // 1. 生成候选数
            let mut hasher = sha2::Sha256::new();
            hasher.update(input);
            hasher.update(&nonce.to_le_bytes());
            let hash = hasher.finalize();

            // 2. 转换为 Integer 并设置高低位以保证是大奇数
            let mut candidate = Integer::from_digits(&hash, rug::integer::Order::Msf);
            candidate.set_bit(2047, true); // 确保是 2048 位
            candidate.set_bit(0, true);    // 确保是奇数

            // 3. Miller-Rabin 素性测试 (25 轮)
            // rug::is_probably_prime 底层调用 GMP，速度极快
            if candidate.is_probably_prime(25)!= rug::integer::IsPrime::No {
                return candidate;
            }
            nonce += 1;
        }
    }

    // 累加器更新: A_new = A_old ^ prime mod N
    pub fn add_element(&self, current_acc: &Integer, element_prime: &Integer) -> Integer {
        // pow_mod 是高度优化的
        current_acc.clone().pow_mod(element_prime, &self.modulus).unwrap()
    }
}
```

第五部分：压力测试与性能基准

在将 Holographic Pass 部署到生产环境之前，必须进行详尽的压力测试，以评估其对系统吞吐量和延迟的影响。

5.1 延迟分析模型

系统的总附加延迟 T_{total} 可以分解为：

$$T_{total} = T_{hash2prime} + T_{update} + T_{zk\_prove} + T_{paillier}$$

- **$T_{hash2prime}$**：~100ms（一次性，冷启动）。在稳态运行中，Agent 使用缓存的 Prime ID，此项为 **0ms**。
- **T_{update} (RSA)**：使用 rug 为 **0.25ms**。
- **$T_{paillier}$**：同态加法耗时极短，约 **0.05ms** [35]。
- **T_{zk_prove}**：这是主要瓶颈。使用 Plonky2 在 CPU 上约 **150ms**。

结论：对于每个 Agent 节点，Holographic Pass 引入的额外延迟约为 **150ms**。对于对延迟敏感的实时系统，必须采用**异步证明（Asynchronous Proving）**策略。

- 同步路径：仅更新 RSA 累加器和 Paillier 状态（< 1ms 延迟），请求立即转发。
- 异步路径：ZKP 证明在后台生成，或者是 "Optimistic" 模式——每隔 10 个请求或在链的终点才生成一次聚合证明。

5.2 大规模并发测试

测试场景：100 个 Agent 节点，拓扑深度 10，并发请求 1,000 QPS。
硬件环境：AWS c6i.8xlarge (32 vCPU), NVIDIA T4 GPU (可选)。
测试结果预估：

- **CPU** 瓶颈：主要集中在 ZK Proof 生成。如果不使用 GPU 加速，32核 CPU 可能会在 200 QPS 时饱和。
- 内存瓶颈：累加器状态和证明都很小（KB 级别），内存压力主要来自 ZK Prover 的证明密钥（Proving Key）加载，可能占用数 GB 内存。
- 网络带宽：Pass 大小约 2-5KB（取决于证明大小），1k QPS 产生约 5MB/s 的额外流量，对现代数据中心网络影响微乎其微。

5.3 硬件加速方案

为了突破 CPU 的 ZKP 生成瓶颈，建议引入硬件加速：

- **GPU** 加速：使用 CUDA 实现的大数乘法（NTT 算子）可以将 ZKP 生成速度提升 10-50 倍 [36]。对于 Holographic Pass，这意味着单跳延迟可以降低到 **10ms** 以内。
- **FPGA** 加速：对于专用硬件环境，FPGA 可以提供更低的延迟和更高的能效比，特别适合在边缘计算节点上运行 Agent。

结论

Holographic Pass 系统不仅是一个技术栈的堆叠，更是一种对 AI 治理哲学的实践。它承认 Agent 行为的不可预测性，并试图用数学的确定性来包围这种不确定性。

通过集成 **RSA** 累加器 建立拓扑溯源、**Paillier** 加密 实现隐私计算、以及 递归 **SNARKs** 确保证明传递，我们构建了一个具备**自我解释（Self-Explaining）和自我验证（Self-Verifying）**能力的流控系统。尽管实施该系统面临着无陷门参数生成、ZKP 计算成本和复杂工程落地的挑战，但对于金融、医疗和关键基础设施领域的 Agent 网络而言，这是从“黑箱操作”走向“可信自治”的必由之

路。

未来的演进方向将聚焦于基于类群（**Class Groups**）的无陷门累加器优化，以及更高效的硬件加速 **ZK** 证明，最终实现几乎零感知的代数溯源层。

引用的著作

1. PROV-AGENT: Unified Provenance for Tracking AI Agent Interactions in Agentic Workflows This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a non-exclusive - arXiv, 访问时间为 十二月 19, 2025, https://arxiv.org/html/2508.02866v1
2. A Process Algebra Approach to Provenance, 访问时间为 十二月 19, 2025, https://homepages.inf.ed.ac.uk/jcheney/presentations/lablunch.pdf
3. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials - Brown Computer Science, 访问时间为 十二月 19, 2025, https://cs.brown.edu/people/alysyans/papers/camlys02.pdf
4. zkSNARKS and Cryptographic Accumulators - Coinbase, 访问时间为 十二月 19, 2025, https://www.coinbase.com/blog/zksnarks-and-cryptographic-accumulators
5. Universal Accumulators with Efficient Nonmembership Proofs - CS@Purdue, 访问时间为 十二月 19, 2025, https://www.cs.purdue.edu/homes/ninghui/papers/accumulator_acns07.pdf
6. A Deep Dive on RSA Accumulators. by Georgios Konstantopoulos - Medium, 访问时间为 十二月 19, 2025, https://medium.com/good-audience/deep-dive-on-rsa-accumulators-230bc84144d9
7. Cryptographic Accumulators: Part1 | by Amit Panghal | Medium, 访问时间为 十二月 19, 2025, https://medium.com/@panghalamit/cryptographic-accumulators-part1-3f23172d3fec
8. Real-World Performance of Cryptographic Accumulators - Brown Computer Science, 访问时间为 十二月 19, 2025, https://cs.brown.edu/research/pubs/theses/ugrad/2013/tremel.pdf
9. Automating CKKS Configuration for Practical Encrypted Inference via an LLM-Guided Agentic Framework - arXiv, 访问时间为 十二月 19, 2025, https://arxiv.org/html/2511.18653v1
10. Fully Homomorphic Encryption – Making it Real - Duality Tech, 访问时间为 十二月 19, 2025, https://dualitytech.com/blog/homomorphic-encryption-making-it-real/
11. Paillier cryptosystem - Wikipedia, 访问时间为 十二月 19, 2025, https://en.wikipedia.org/wiki/Paillier_cryptosystem
12. Differentially Private Aggregation of Distributed Time-Series with Transformation and Encryption - Microsoft, 访问时间为 十二月 19, 2025, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/paper-8

7.pdf
13. homomorphic encryption - Paillier VS RSA - Cryptography Stack Exchange, 访问时间为 十二月 19, 2025, https://crypto.stackexchange.com/questions/57766/paillier-vs-rsa
14. Ledgerless Digital Currency Using DAG + ZKP + Merkle Trees : r/CryptoTechnology - Reddit, 访问时间为 十二月 19, 2025, https://www.reddit.com/r/CryptoTechnology/comments/1j728gh/ledgerless_digital_currency_using_dag_zkp_merkle/
15. Recursive SNARKs - Berkeley RDI, 访问时间为 十二月 19, 2025, https://rdi.berkeley.edu/zkp-course/assets/lecture10.pdf
16. Recursive zkSNARKs: Exploring New Territory - 0xPARC, 访问时间为 十二月 19, 2025, https://0xparc.org/blog/groth16-recursion
17. Recursive SNARKs: A Comprehensive Primer - Michael Straka, 访问时间为 十二月 19, 2025, https://www.michaelstraka.com/recursivesnarks
18. Using Prime numbers to map n integers uniquely to an integer x and allowing an easy reverse mapping - Math Stack Exchange, 访问时间为 十二月 19, 2025, https://math.stackexchange.com/questions/297698/using-prime-numbers-to-map-n-integers-uniquely-to-an-integer-x-and-allowing-an-e
19. Hashing to Prime in Zero-Knowledge - Newcastle University, 访问时间为 十二月 19, 2025, https://research.ncl.ac.uk/cascade/outputs/cryptography/2022-853-2.pdf
20. Miller–Rabin primality test - Wikipedia, 访问时间为 十二月 19, 2025, https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test
21. Hash to prime numbers? - Cryptography Stack Exchange, 访问时间为 十二月 19, 2025, https://crypto.stackexchange.com/questions/63013/hash-to-prime-numbers
22. Which is the fastest algorithm to find prime numbers? [closed] - Stack Overflow, 访问时间为 十二月 19, 2025, https://stackoverflow.com/questions/453793/which-is-the-fastest-algorithm-to-find-prime-numbers
23. Secure Accumulators from Euclidean Rings without Trusted Setup, 访问时间为 十二月 19, 2025, https://kodu.ut.ee/~lipmaa/papers/lip12b/cl-accum.pdf
24. Distributed Identity Based Short Linkable Ring Signature - EPFL, 访问时间为 十二月 19, 2025, https://www.epfl.ch/labs/dedis/wp-content/uploads/2020/01/report-2017-2-kasra_edalat-accumulators.pdf
25. Notus: Dynamic Proofs of Liabilities from Zero-knowledge RSA Accumulators - USENIX, 访问时间为 十二月 19, 2025, https://www.usenix.org/system/files/usenixsecurity24-xin.pdf
26. Efficient Accumulators without Trapdoor Extended Abstract | Request PDF - ResearchGate, 访问时间为 十二月 19, 2025, https://www.researchgate.net/publication/225679636_Efficient_Accumulators_without_Trapdoor_Extended_Abstract
27. Verifiable Delay Functions for Rate-limiting Systems - ETH Zurich Research Collection, 访问时间为 十二月 19, 2025,

https://www.research-collection.ethz.ch/bitstreams/47edebf2-4480-460b-ab5b-d3d8470e16fe/download
28. Identification Codes via Prime Numbers - arXiv, 访问时间为 十二月 19, 2025, https://arxiv.org/pdf/2408.12455
29. A Deep Dive on RSA Accumulators. by Georgios Konstantopoulos - Good Audience, 访问时间为 十二月 19, 2025, https://blog.goodaudience.com/deep-dive-on-rsa-accumulators-230bc84144d9
30. bigint-benchmark - crates.io: Rust Package Registry, 访问时间为 十二月 19, 2025, https://crates.io/crates/bigint-benchmark/0.2.2
31. bigint-benchmark - crates.io: Rust Package Registry, 访问时间为 十二月 19, 2025, https://crates.io/crates/bigint-benchmark
32. A new big integer library: ibig - announcements - The Rust Programming Language Forum, 访问时间为 十二月 19, 2025, https://users.rust-lang.org/t/a-new-big-integer-library-ibig/56828
33. Efficient Zero-Knowledge Proofs: Theory and Practice - UC Berkeley EECS, 访问时间为 十二月 19, 2025, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-20.pdf
34. SoK: VDFs for proving passage of time in ZK games - Paima Blog, 访问时间为 十二月 19, 2025, https://blog.paimastudios.com/vdf-sok/
35. Big Data Analytics over Encrypted Datasets with Seabed - Microsoft, 访问时间为 十二月 19, 2025, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/09/paper.pdf
36. TFHE-rs: A library for safe and secure remote computing using fully homomorphic encryption and trusted execution environments - ResearchGate, 访问时间为 十二月 19, 2025, https://www.researchgate.net/publication/357375391_TFHE-rs_A_library_for_safe_and_secure_remote_computing_using_fully_homomorphic_encryption_and_trusted_execution_environments