n-VM：采用共享身份与代币状态的多虚拟机第一层架构

Jian Sheng Wang
Yeah LLC
jason@yeah.app

2026年3月23日

arXiv:2603.23670v1 [cs.CR] 24 Mar 2026

摘要

多链生态系统面临身份碎片化、孤岛式流动性以及依赖跨链桥的代币转移等问题。我们提出 n-VM，这是一种第一层架构，可在共享共识与共享状态之上，将 n 个异构虚拟机作为地位平等的执行环境运行。该设计由三个部分组成：按操作码前缀路由交易的调度器；统一身份层，其中一个 32 字节承诺作为锚点，关联各虚拟机专属地址；以及统一代币账本，它在通用余额存储之上提供 ERC-20 和 SPL 等虚拟机原生接口。我们对路由、身份派生和代币转移语义进行了形式化，并证明在标准密码学假设下，跨虚拟机转移具有原子性，身份也能得到隔离。我们介绍了一种包含五个虚拟机的具体实现：原生运行时、以太坊虚拟机、Solana 虚拟机、比特币脚本和 TVM。我们还提出了基于上下文的分片方案，以及用于并行执行的写集调度器。根据分析性吞吐量模型，该架构在商用硬件上的预期吞吐量约为每秒 16,000 至 66,000 笔交易。

关键词： 多虚拟机区块链、统一身份、跨虚拟机互操作性、并行执行、上下文分片、操作码路由、ACE-GF

1 引言

1.1 动机

区块链生态系统分散在彼此不兼容的执行环境中。以太坊的以太坊虚拟机、Solana 的 Solana 虚拟机、比特币的 Script，以及波场的 TVM 各自定义了不同的账户模型、地址格式、交易语义和智能合约语言。希望在这些生态系统间进行操作的用户，必须维护独立的钱包、管理不同的密钥对，并依赖跨链桥——这类第三方系统历来是去中心化金融领域最大的单一安全故障来源。2021年至 2024 [4]期间，跨链桥攻击造成的损失超过28亿美元。

近期项目开始探索多虚拟机架构。Movement Labs 在基于 Move 的结算链之上部署兼容以太坊虚拟机的执行层；Eclipse 将 Solana 虚拟机执行环境作为以太坊 Rollup 运行；Sei v2 则在 Cosmos SDK 链中结合以太坊虚拟机与 CosmWasm。然而，这些方案有一个共同的

局限：一个虚拟机从属于另一个虚拟机（作为 Rollup 或兼容层），身份在不同虚拟机之间仍然彼此割裂，而执行环境之间的代币转账仍需依赖类似桥接的机制。

1.2 核心洞见：虚拟机无关的身份与授权分离

Ace-Gf 框架 [1] 将身份绑定与逐笔交易授权分离。用户的链上身份由单个 32 字节承诺表示：$\texttt{id_com} = \text{Poseidon(REV,salt,domain)}$，该承诺通过 HKDF 密钥流从根熵值（REV）派生。逐笔交易授权采用轻量级的基于 HMAC 的证明凭证（CPU 上每笔交易耗时 $\sim 1\,\mu$s），而非逐笔交易的加密签名，并将绑定证明推迟到非关键路径上的零知识证明中 [3]。

这种分离与虚拟机无关：身份承诺和证明机制独立于任何特定执行环境。因此，只要存在从 `id_com` 到每个虚拟机原生地址格式的确定性抗碰撞映射，单个 `id_com` 便可作为任意多个虚拟机上地址的锚点。

1.3 贡献

本文作出以下贡献：

1. **通用化n-Vm 架构**。我们定义了一个框架，可在单条第一层链上托管 n 个地位平等的虚拟机，并采用基于操作码前缀的交易路由、共享状态树和可插拔引擎接口（第 3 节）。2. **统一身份层**。我们形式化定义了如何从单一身份承诺中确定性地派生虚拟机专属地址，证明了在 HKDF 的伪随机函数假设下的上下文隔离，并介绍了旧版钱包的原始链入站验证（第 4 节）。3. **统一代币账本**。我们提出了一种单账本代币运行时，在同一底层余额存储之上提供 ERC-20 和 SPL 接口，并证明跨虚拟机代币转账具有原子性（第 5 节）。4. **并行执行**。我们介绍了一种基于写集的冲突检测调度器和一种基于上下文的分片方案，并分析吞吐量扩展（第 6 节）。5. **具体实现**。我们介绍了一个 Rust 实现，其中包含 $n = 5$ 个虚拟机（原生、以太坊虚拟机、Solana 虚拟机、BVM、TVM），以及身份预编译合约、跨虚拟机调用钩子和原始链签名验证（第 9 节）。

n-Vm 构建于 Ace-Gf 身份框架 [1] 之上，用于多流密钥派生；该框架还基于 Ace Runtime 执行层 [3]，以支持见证–执行–证明流水线，并基于 VA-DAR 恢复协议 [2]，以支持以电子邮件为锚定的钱包恢复。

2 背景

2.1 ACE-GF 身份原语

原子密码实体生成框架 (Ace-Gf) [1] 从一个称为根熵值 (REV) 的高熵秘密中，派生出确定性的、特定用途的密钥流。第 i 个密钥流为：

$$k_i \leftarrow \text{HKDF-SHA256}(\text{REV},\ info_i,\ salt_i). \tag{1}$$

ACE-GF 为多个密码学生态系统定义了规范流：

- 流 1：Ed25519（Solana 签名）
- 流 3：secp256k1（以太坊虚拟机链）
- 流 4：secp256k1（比特币）
- 流 7：ML-DSA-44（后量子签名）

链上身份由 Poseidon 哈希承诺表示：

$$\texttt{id_com} = \mathsf{Poseidon}(\text{REV},\ \text{salt},\ \text{domain}). \tag{2}$$

这个 32 字节的值是链上可见的主要原生身份凭证。在原生 Ace-Gf 路径中，可以延后或避免披露长期公钥，从而减少已发布公钥遭受“现在窃取、日后解密”式收集的风险。

2.2 见证–执行–证明流水线

Ace Runtime [3] 通过一个三阶段流水线处理交易：

1. **见证** (∼1–5 μ秒/笔交易，CPU)：验证每笔交易的轻量级基于 HMAC 的证明凭证。

2. **执行**（∼10–50 μ秒/笔交易）：针对状态树执行交易。

3. **生成证明**（不在关键路径上，由 GPU 执行）：生成零知识证明，将所有证明与各自的 `id_com` 值绑定。

阶段 1 和阶段 2 位于关键路径上，并决定区块时间（∼400毫秒）。阶段 3 异步运行，并在随后的时隙中生成密码学最终性证书。

n-Vm 分发器以路由层取代单体式执行阶段，由该路由层将任务委派给可插拔的 n 虚拟机引擎。

2.3 现有多虚拟机方案

我们简要概述现有的多虚拟机设计，并指出其局限性。

表 1：多虚拟机方案比较。

系统	VMs	身份	代币转移
Movement	以太坊虚拟机 +Move	每个虚拟机独立	桥接
Eclipse	以太坊上的 Solana 虚拟机	每个虚拟机独立	桥接（L1↔第二层）
Sei v2	以太坊虚拟机 + CosmWasm	指针合约	CosmWasm↔以太坊虚拟机指针
波卡	每个平行链	每个平行链	XCM 消息传递
n-Vm（本文工作）	n 地位平等的虚拟机	单一 `id_com`	统一账本（无桥接）

关键区别在于，n-Vm 将所有虚拟机都视为一等公民，共享同一个共识、身份和代币层，而不是将一个虚拟机置于另一个虚拟机之下。

3 架构

3.1 概述

n-Vm 架构由四层组成：

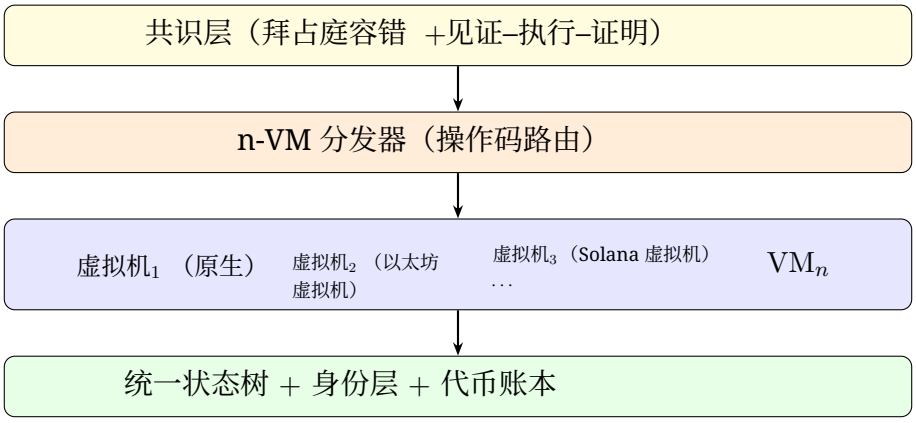


图1：分层 n-Vm 架构。

3.2 基于操作码的交易路由

每笔交易都携带一个有效载荷，其首字节决定目标虚拟机。我们将 256 个值的操作码空间划分为 n 个连续范围：

定义1 (操作码路由函数**)**。 设 $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ 为已注册虚拟机的集合。每个 V_i 都会被分配一个连续的操作码范围 $[l_i, u_i] \subset [0, 255]$，且这些范围两两不相交。路由函数为：

$$\mathsf{Route}(opcode) = \begin{cases} V_i & \textit{if } l_i \le opcode \le u_i, \\ \bot & \textit{otherwise (reject transaction).} \end{cases}$$

这种设计有几个优点：

- $O(1)$ **路由**。只需比较一个字节即可确定目标虚拟机，为交易处理增加的开销微乎其微。
- **可扩展性**。注册一个引擎即可添加新的虚拟机，为其分配一个尚未占用的操作码范围，无需修改现有引擎。
- **确定性**。路由决策仅取决于交易载荷，确保所有验证者得出相同的分派决策。

3.3 虚拟机引擎接口

每个虚拟机引擎都实现一个最小化接口：

定义 2 (虚拟机引擎)。虚拟机 E_i 的虚拟机引擎 V_i 实现：

1. 虚拟机_标识符() $\to V_i$：返回虚拟机标识符。*2.* 执行(状态，交易) $\to$(收据 Δ)：针对状态树 *state* 执行交易 *tx*，返回收据和一组状态变更 Δ。

调度器维护一个注册表 $\mathcal{E}: \mathcal{V} \to E$ 将每个虚拟机标识符映射到其引擎。交易执行遵循算法 1。

算法 1 n虚拟机交易分派

要求： 带有有效载荷 P 的交易 tx，状态树 S
确保： 收据 R

操作码 $\leftarrow P[0]$
$V \leftarrow$路由(操作码)
如果 $V = \perp$，**则**
    返回 拒绝（“未知操作码”）
结束如果
$E \leftarrow \mathcal{E}[V]$
$S' \leftarrow$快照（S）　　▷ 保存状态以便回滚
(R， Δ) $\leftarrow E$.执行(S， 交易)
如果 R.成功 $=$ `false` **则**
    回滚(S， S')　　▷ 恢复执行前状态
结束 if
返回 R

快照/回滚机制确保任意虚拟机中的失败交易都不会损坏共享状态树，从而维护虚拟机之间的隔离。

3.4 区块执行

区块 $B = [$交易$_1$,交易$_2$，.....，交易 $_m]$是一个有序交易序列，其中的交易可能针对不同的虚拟机。调度器按顺序（在基线模式下）处理每笔交易，并以原子方式应用状态变更：

$$S_j = \begin{cases} S_{j-1} \cup \Delta_j & \text{if } tx_j \text{ succeeds,} \\ S_{j-1} & \text{otherwise,} \end{cases} \qquad j = 1, \ldots, m. \tag{3}$$

给定初始状态 S_0 和区块 B，最终状态 S_m 和有序收据列表 $[R_1, \ldots, R_m]$ 是确定性的。

4 统一身份层

4.1 跨虚拟机地址派生

核心身份原语是从单一 `id_com` 到各虚拟机专属地址的确定性映射。

定义 3 (地址派生**)**。 对于原生地址长度为 ℓ_i 字节的虚拟机 V_i ，派生地址为：

$$\alpha_{V_i} = \mathsf{Truncate}_{\ell_i}\big(\mathsf{SHA\text{-}256}(\mathsf{tag}_i \| \mathit{id_com})\big)$$

其中 标签$_i$ 是虚拟机专属的域分隔字符串，而 截断$_\ell$ 会取哈希输出的最后 ℓ 字节。

对于 $n = 5$ 实例化：

$$\alpha_{\mathrm{EVM}} = \mathsf{SHA\text{-}256}(\texttt{"evm:"} \| \texttt{id_com})[12{:}32] \quad (20 \text{ bytes}) \tag{4}$$
$$\alpha_{\mathrm{SVM}} = \mathsf{SHA\text{-}256}(\texttt{"svm:"} \| \texttt{id_com}) \quad (32 \text{ bytes}) \tag{5}$$
$$\alpha_{\mathrm{BVM}} = \mathsf{SHA\text{-}256}(\texttt{"bvm:"} \| \texttt{id_com}) \quad (32 \text{ bytes}) \tag{6}$$
$$\alpha_{\mathrm{TVM}} = \mathsf{SHA\text{-}256}(\texttt{"tron:"} \| \texttt{id_com})[12{:}32] \quad (20 \text{ bytes}) \tag{7}$$
$$\alpha_{\mathrm{native}} = \texttt{id_com} \quad (32 \text{ 字节}) \tag{8}$$

定理 4.1 （地址隔离）。 对于任意两个不同的虚拟机标签 $\mathsf{tag}_i \neq \mathsf{tag}_j$ 以及任意$\mathit{id_com}$, 派生地址 α_{V_i} 和 α_{V_j} 在 *SHA-256* 的抗碰撞性保证下具有计算独立性。

证明概要。 将 SHA-256 建模为随机预言机。由于 $\mathsf{tag}_i \neq \mathsf{tag}_j$，输入 $\mathsf{tag}_i \| \texttt{id_com}$ 和 $\mathsf{tag}_j \| \texttt{id_com}$ 互不相同，因此它们的输出相互独立分布。截断保留了计算独立性：获知一个截断后的输出，并不能帮助预测另一个不同哈希值的不同截断结果。 □

这意味着，用户在某个虚拟机上的地址遭到泄露，并不会透露其在其他虚拟机上的地址信息，除共享的 `id_com`已公开的信息外。

4.2 反向解析

为与遗留工具（例如显示以太坊虚拟机地址的区块浏览器）实现互操作性，调度器会维护一个反向索引：

$$\mathsf{ace_id_from_evm}(\alpha_{\mathrm{EVM}}) = \mathsf{SHA\text{-}256}(\texttt{"ace_from_evm:"} \| \alpha_{\mathrm{EVM}}) \tag{9}$$

这是一种确定性的单向映射，允许 EVM 合约通过预编译合约查询 EVM 地址对应的 ACE 侧身份，而不会暴露原始值`id com`。_

4.3 原生链入口

为了实现冷启动采用，n-Vm 链必须接受由不了解 Ace-Gf 证明机制的旧版钱包（例如用于以太坊虚拟机的 MetaMask、用于 Solana 的 Phantom 等）签名的交易。

定义 4（原生链交易）。原生链交易除了标准载荷外，还携带原生链签名（例如，以太坊虚拟机使用椭圆曲线数字签名算法，*Solana* 使用 *Ed25519* 等）。调度器执行以下操作：

1. 签名验证：使用链原生验证算法恢复签名者的公钥或地址。*2.* 身份映射：为恢复的地址计算确定性 id_com。*3.* 载荷重构：验证交易载荷是否与根据原始字节推导出的规范形式相符。*4.* 账户绑定：确保推导出的 id_com 在状态树中拥有账户，必要时创建账户。

此机制支持四种入口路径：

表 2：原生链入口验证。

链	签名	地址格式	身份派生
EVM	椭圆曲线数字签名算法/secp256k1	20 字节 Keccak	SHA-256(`"legacy_evm:"`‖地址)
Solana	Ed25519	32 字节公钥	SHA-256(`"legacy_sol:"`‖公钥)
比特币	椭圆曲线数字签名算法/Schnorr	P2WPKH/Taproot	SHA-256(`"legacy_btc:"`‖公钥)
Tron	椭圆曲线数字签名算法/secp256k1	20 字节（T 前缀）	SHA-256(`"legacy_tron:"`‖地址)

原生链入口提供了无缝的迁移路径：用户可以使用现有钱包开始与 n-Vm 链交互，之后还可以选择升级为 Ace-Gf 身份，以实现跨虚拟机统一和基于证明的授权。

5 统一代币账本

5.1 设计

不同虚拟机之间代币标准的碎片化，是多链系统复杂性的主要来源之一。以太坊虚拟机使用 ERC-20（通过余额映射和授权额度管理余额）；Solana 虚拟机使用 SPL 代币（采用独立的代币账户，并包含铸造/所有者元数据）。两者在语义上等价，但结构上并不兼容。

n-Vm 代币账本将所有代币状态存储在状态树中的单个内置程序账户内，并将虚拟机原生接口作为该共享存储的视图对外提供：

定义 5 (统一代币状态**)**。对于铸造标识符为 M（*32* 字节）的代币，其规范状态由以下部分构成：

- 总供应量(M)：总供应量（*uint64*）。

- 小数位数(M)：小数精度（*uint8*）。
- 铸币权限(M)*:* 铸币权限 id_com （*32* 字节）。
- 余额(M, id_com)*:* 身份的余额（*uint64*）。
- 委托额度$(M, id_com_{\text{所有者}}\ id_com_{\text{支出者}})$*:* 委托额度（*uint64*）。

存储槽通过域分离哈希派生：

$$\text{slot}_{\text{balance}} = \text{SHA-256}(\texttt{"balance:"} \| M \| \texttt{id_com}) \tag{10}$$

$$\text{slot}_{\text{allowance}} = \text{SHA-256}(\texttt{"allowance:"} \| M \| \texttt{id_com}_{\text{owner}} \| \texttt{id_com}_{\text{spender}}) \tag{11}$$

5.2 多接口访问

同一底层余额可通过以太坊虚拟机和 Solana 虚拟机接口访问：

ERC-20 接口。 每个铸币 M 都有一个确定性的 ERC-20 合约地址：

$$addr_{\text{ERC20}} = \text{SHA-256}(\texttt{"erc20-addr:"} \| M)[12{:}32] \tag{12}$$

当以太坊虚拟机合约通过此地址调用 `balanceOf`(α_{EVM}) 时，运行时会：(1) 通过反向索引将 α_{EVM} 解析为对应的 `id_com`，(2) 从统一账本读取 balance(M, `id_com`)，(3) 以兼容 ERC-20 的 ABI 编码格式返回结果。

SPL 接口。 SPL 代币账户被建模为兼容别名。关联代币地址（ATA）使用标准 Solana PDA 算法推导得出：

$$ata = \text{FindPDA}([owner_pubkey, spl_program_id, M], ata_program_id) \tag{13}$$

当 Solana 虚拟机程序查询代币账户的余额时，运行时会将 ATA 解析为所有者的 `id_com`，并从同一个统一账本读取数据。

5.3 跨虚拟机转账原子性

定理 5.1 （跨虚拟机原子性）。 从以太坊虚拟机上下文到 *Solana* 虚拟机上下文的代币转账，会作为一次状态转换在统一账本上执行，无需中间桥接状态。

证明概要。以太坊虚拟机和 Solana 虚拟机接口最终都在由 `id_com` 作为键的同一存储时隙上执行操作。从 $\texttt{id_com}_A$ 转账至 $\texttt{id_com}_B$ 恰好会修改两个时隙：

$$\text{balance}(M, \texttt{id_com}_A) \mathrel{-}= v, \qquad \text{balance}(M, \texttt{id_com}_B) \mathrel{+}= v.$$

由于两次写入都发生在同一状态树上的单笔交易中，根据公式 (3) 的交易执行语义，它们具有原子性。不存在中间的“锁定”或“进行中”状态。

□

这消除了整类桥接相关漏洞：不存在锁定铸造—销毁释放周期，也不需要多重签名委员会或乐观挑战期。

6 并行执行

6.1 写集冲突检测

为充分利用异构虚拟机工作负载（例如，针对不同账户的以太坊虚拟机合约调用和原生转账）所固有的并行性，n-Vm 调度器会从每笔交易中提取写集，并构建无冲突批次。

定义 6 (写集)。 交易的写集 $W(tx)$ 为：

$$W(tx) = \begin{cases} \{a_1, a_2, \ldots\} & \textit{if the written accounts can be statically determined,} \\ \top \textit{ (global)} & \textit{otherwise (conflicts with all transactions).} \end{cases}$$

对于写集可静态确定的交易（原生转账、SVM 转账、BVM 转账），调度器会直接从有效载荷中提取发送方和接收方账户 ID。对于具有无界副作用的交易（EVM `CALL`、EVM `CREATE`、SVM `INVOKE`），写集会被保守地标记为全局。

定义 7 (冲突)。 如果以下条件成立，则两个交易 tx_i 和 tx_j 冲突：

$$W(tx_i) = \top \ \vee \ W(tx_j) = \top \ \vee \ W(tx_i) \cap W(tx_j) \neq \emptyset.$$

算法 2 批次构建

要求： 有序交易列表 $[\text{交易}_1、\ldots\ldots、\text{交易}_m]$
确保： 有序的无冲突批次列表 $[B_1, B_2, \ldots]$

批次 $\leftarrow [\,]$；当前项 $\leftarrow [\,]$；已使用项 $\leftarrow \emptyset$
对于 $j = 1、\ldots\ldots、m$ **执行**
    $w \leftarrow W(\text{交易}j)$
    如果 $w = \top$ **或** $w \cap$已使用项 $\neq \emptyset$ **则**
        如果 $|\text{当前项}| > 0$ **则**
            追加 当前项 至批次
        结束如果
        追加 $[\text{交易}_j]$ 至 批次　　▷ 全局交易单独运行
        当前项 $\leftarrow [\,]$；已使用项 $\leftarrow \emptyset$
    else
        追加 交易j 至 当前项
        已使用项 $\leftarrow$ 已使用项 $\cup w$
    结束如果
结束循环
如果 $|\text{当前项}| > 0$ **则**
    追加 当前项 至批次
结束条件判断
返回批次

同一批次中的交易会并行执行（在实现中通过 Rayon [13] 实现）；各批次则顺序执行，以保持区块级确定性。

6.2 基于上下文的分片

为实现更高程度的并行化，n-Vm 分发器支持基于上下文的分片：交易可以携带可选的 16 字节上下文标签，用于确定其分片分配。

定义 8（分片路由）。给定 K 个分片（默认为 $K = 64$）、一个虚拟机前缀字符串vm和一个上下文标签 ctx：

$$\mathsf{shard} = \mathsf{SHA\text{-}256}\big(\mathsf{len}(\mathsf{vm}) \| \mathsf{vm} \| \mathsf{ctx}\big) \bmod K. \tag{14}$$

长度前缀可避免例如（*"evm", "foo:bar"*）与（*"evm:foo", "bar"*）之间的碰撞歧义。

面向不同分片的交易可独立执行。一个保留的共享分片处理涉及多个分片中账户的跨上下文交易，并按串行方式执行，以确保原子性。

6.3 吞吐量分析

我们针对三种执行策略对吞吐量进行建模。

表 3：不同执行策略下的预期吞吐量。

策略	描述	预期每秒交易数（TPS）
顺序执行	单线程，所有虚拟机	$\sim$5，000
并行批处理	写集冲突检测，Rayon	$\sim$16，000
上下文分片	64 个分片 + 跨上下文检测	$\sim$33，000–66，000

吞吐量预测假设：

- 400 毫秒区块时间（与 Ace Runtime 流水线 [3] 一致）。
- 每笔交易进行 $\sim$1–5 μ 秒证明检查。
- 每笔交易执行耗时 $\sim$10–50 μ 秒（取决于虚拟机）。
- 16 核商用服务器硬件。
- 70% 的交易具有可静态确定的写集（原生交易和简单转账）。

$\sim$66，000 每秒交易数的上限假设分片局部性较高（大多数交易都在分片内），且跨上下文流量极少。

7 跨虚拟机通信

7.1 EVM 预编译合约

为使以太坊虚拟机智能合约能够与 n-Vm 身份和跨虚拟机层交互，运行时在保留地址处提供了一组预编译合约：

表 4：ACE EVM 预编译合约。

地址名称		函数
0x0100	id_com_verify	验证身份承诺证明
0x0101	context_derive	根据输入推导上下文
0x0102	admin_factor_check	检查管理员因子凭证
0x0103	zkace_batch_verify	批量验证 ZK-ACE 证明
0x0104	multisig_derive	派生多重签名地址
0x0105	multisig_verify	验证多重签名证明
0x0106	cross_vm_call	对源自以太坊虚拟机的跨虚拟机调用请求进行编码
0x0107	resolve_svm_addr	解析 id _com → SVM 地址

cross_vm_call 预编译合约（0x0106）提供了一个面向以太坊虚拟机的接口，可通过提供程序 ID 和指令数据来表达面向 Solana 虚拟机的调用请求。在当前设计中，该预编译合约定义了跨虚拟机请求的调用封装和分发器路由钩子。支持共享燃料计量的通用同步返回值路径留待未来实现。

7.2 SVM 系统调用

同样地，SVM 程序可以通过自定义系统调用访问身份层和互补的跨虚拟机路由钩子：

- ace_attest：在 SVM 程序中验证证明凭证。
- ace_id_com：查询当前交易发送方的 id_com 。
- ace_cross_vm_call：从 SVM 程序发起面向 EVM 的跨虚拟机调用请求。

结合 EVM 预编译合约，这些系统调用为身份访问和跨虚拟机请求路由定义了双向接口层。它们规定了多虚拟机交互的编码与分派方式，而完全通用的同步跨虚拟机执行模型仍属于未来工作。

8 安全分析

8.1 虚拟机隔离

命题 8.1 (执行隔离)。 在虚拟机 V_i 执行交易 *tx* 期间发生的故障（回滚、燃料耗尽、恐慌），不会影响任何其他虚拟机 V_j （$j \neq i$）的状态，也不会影响之后执行的任何交易。

证明概要。 根据算法 1，调度器会在委派给任何引擎之前创建状态快照。若执行失败，状态将回滚至该快照。由于状态树是唯一的共享资源，且回滚会将其恢复到执行前状态，因此失败交易产生的副作用不会保留。 □

8.2 身份绑定的正确性

命题 8.2 (原始链绑定)。 对于来自链 c 且具有恢复的地址 $addr$ 的原生链交易，其与 $id_com = \text{SHA-256}(\texttt{"legacy_}c\texttt{:"} \| addr)$ 的绑定为：

1. 确定性：相同的地址始终映射到相同的 id_com。*2.* 抗碰撞：两个不同的地址以极高概率映射到不同的 id_com 值。*3.* 域分离：来自不同链的地址映射到不同的 id_com 值，即使原始字节完全相同也是如此。

证明。 性质 (1) 和 (2) 直接源于 SHA-256 是一种确定性、抗碰撞的哈希函数。性质 (3) 源于针对特定链的前缀：如果 $c_1 \neq c_2$，那么 $\texttt{"legacy_}c_1\texttt{:"} \neq \texttt{"legacy_}c_2\texttt{:"}$，因此即使地址 $_1 =$ 地址$_2$，SHA-256 的输入也各不相同。 □

8.3 确定性执行

命题 8.3 (区块级确定性)。 给定初始状态 S_0 和区块 B，所有验证者的最终状态 S_m 和收据列表 $[R_1, \ldots, R_m]$ 均相同。

证明概要。操作码路由函数（定义1）具有确定性。每个虚拟机引擎的 `execute` 函数都具有确定性（上海模式下的 revm；使用固定内置程序的 SVM；Script 求值具有确定性的 BVM）。状态快照和回滚也具有确定性。由于同一批次中的交易按设计互不冲突（算法 2），批次内的并行执行与顺序执行结果相同。上下文分片路由（定义 8）具有确定性。 □

9 实现

9.1 系统概述

n-Vm 分发器以 Rust 编写，作为 `ace-n-vm` crate 实现，包含约 5,000 行 Rust 代码和 3,000 行测试代码。表 5 总结了具体实例中的五个虚拟机引擎。

表 5：已实现的虚拟机引擎（$n = 5$）。

VM	操作码	后端	地址	功能
原生	0x01–0x0F	ace_engine	32 字节 id_com	转账、账户创建、授权密钥管理
EVM	0x10–0x1F	revm v19（上海）	20 字节	完整的以太坊虚拟机：Solidity, Vyper、ERC-20/721/1155/4626
SVM	0x20–0x2F	内置程序	32 字节	SPL 代币，系统程序，PDAs
BVM	0x30–0x3F	脚本解释器	32 字节	比特币脚本、未花费交易输出管理，P2WPKH/Taproot
TVM	0x40–0x4F	revm（已重新映射）	20 字节	通过以下方式实现波场兼容性：操作码重新映射

9.2 TVM 作为以太坊虚拟机委托执行

TVM 引擎展示了 n-Vm 架构的可扩展性：TVM 并未实现独立的执行引擎，而是通过操作码重新映射复用了以太坊虚拟机引擎：

$$\mathsf{TVM.execute}(tx) = \mathsf{EVM.execute}\big(\mathsf{remap}(tx)\big) \tag{15}$$

其中，重新映射会转换操作码 0x40 → 0x10、 0x41 → 0x11、 0x42 → 0x12，并将地址命名空间从波场调整为以太坊虚拟机。这种方法只需少量代码即可添加一个新的虚拟机，同时为重新映射的操作码子集保留与波场兼容的执行模型。

9.3 RawChain 验证

每种原始链入口路径都会实现特定于链的签名恢复和载荷规范化：

- **EVM**：通过 ecrecover 恢复 secp256k1 签名者，验证链 ID 与证明域匹配，并根据原始 RLP 编码交易重建规范载荷。

- **Solana**：验证 Ed25519 签名，提取转账参数，并计算发送方的传统 id_com。Solana 的重放保护通过按时隙签名去重来实现。

- **比特币**：根据输出类型（P2PKH、P2WPKH、P2WSH、Taproot）验证 ECDSA 或 Schnorr 签名，并根据序列化交易重建规范载荷。

- **波场**：与以太坊虚拟机类似，但采用波场专用的 Protocol Buffers 编码和域时隙验证。

10 相关工作

多虚拟机第一层链。 Sei v2 [5] 在 Cosmos SDK 链中结合了以太坊虚拟机和 CosmWasm 执行环境，并使用指针合约在不同虚拟机之间桥接状态。然而，指针合约引入了间接层，无法实现统一标识：以太坊虚拟机和 CosmWasm 地址仍然彼此独立，虚拟机之间的代币转账需要显式的指针交互。Artela [6] 通过 WebAssembly 扩展为以太坊虚拟机添加了自定义运行时模块，但 WASM 层从属于以太坊虚拟机，而非与其平等的执行环境。

多虚拟机第二层系统。 Movement Labs [7]在基于 Move 的结算链之上部署了兼容以太坊虚拟机的执行层。Eclipse [8]将 Solana 的 Solana 虚拟机作为以太坊二层汇总运行。这些方案继承了各自第一层／二层结算机制的信任假设与延迟，而身份仍然分散在不同层之间。

跨链互操作性。 波卡的跨共识消息格式 [9] 和 Cosmos 的区块链间通信协议 [10] 支持独立链之间的消息传递，但无法提供统一身份或共享状态。每条链都维护自己的账户模型，而代币转账则需要锁定铸造或销毁释放桥接机制。

并行执行。 Solana 的 Sealevel [11] 通过要求交易预先声明所访问的账户，开创了账户级并行。Aptos 的 Block-STM [12] 采用乐观并发控制，并在发生冲突时重新执行。n-Vm 调度器将写集提取与基于上下文的分片相结合，旨在实现跨虚拟机并行，而非单一虚拟机中的账户级并行。

11 结论

我们介绍了 n-Vm，这是一种通用架构，可在单个一层区块链上托管 n 异构虚拟机，并将它们作为地位平等的执行引擎。该设计由三大支柱构成——基于操作码的分派、通过 Ace-Gf 承诺实现的统一身份，以及支持多接口访问的单账本代币运行时——三者结合，消除了在不同执行环境间操作时对桥、封装代币或多个钱包的需求。

该架构已通过 $n = 5$ 种虚拟机（原生、以太坊虚拟机、Solana 虚拟机、BVM、TVM）实例化，但该框架在 n方面具有参数化特性：添加新的虚拟机只需实现引擎接口并注册一个操作码范围，无需更改身份层、代币账本或现有引擎。TVM 引擎通过操作码重新映射和命名空间转换复用现有的 EVM 后端，实际展示了这种可扩展性。

在第 6 节的分析假设下，写集冲突检测和基于上下文的分片表明，在商用硬件上吞吐量可能达到 $\sim$16，000–66，000 每秒交易数。

展望未来，有几项自然的扩展方向：

- **WASM 虚拟机**。将 WebAssembly 执行引擎（例如用于 CosmWasm 或 ink! 合约）作为虚拟机[6] 引入，可扩展对 Cosmos 和波卡生态系统的兼容性。

- **Move 虚拟机**。 集成 Move 执行引擎将提供与Aptos和Sui智能合约的兼容性。

- **推测性并行执行**。 用乐观并发控制（Block-STM 风格）替代保守的写集调度器，可以提高以太坊虚拟机和 Solana 虚拟机交易的并行度，因为这些交易的写集无法进行静态确定。

- **跨虚拟机可组合性**。 将当前的预编译/系统调用请求钩子泛化为带有共享燃料计量的同步跨虚拟机调用，将支持原子性的多虚拟机交易。

参考文献

[1] J. S. Wang。《ACE-GF：原子密码实体生成框架》。arXiv 预印本 arXiv:2511.20505，2025 年。[2]J. S. Wang。《VA-DAR：一种支持后量子密码、厂商无关的确定性制品解析方案，用于无服务器的抗枚举钱包恢复》。arXiv 预印本 arXiv:2603.02690，2026 年。[3] J. S. Wang。《ACE Runtime：一种原生支持零知识证明且具备亚秒级密码学最终性的区块链运行时》。arXiv 预印本 arXiv:2603.10242，2026 年。

[4] Chainalysis。跨链桥攻击事件。Chainalysis 报告，2024年。

[5] J. Gorzny 等。Sei v2：首个并行化以太坊虚拟机。Sei Labs 白皮书，2024年。[6] Artela Network。Artela：采用切面编程的以太坊虚拟机。Artela 白皮书，2024年。[7] Movement Labs。Movement：用于构建和部署基于 Move 的基础设施、应用程序和区块链的模块化框架。Movement 白皮书，2024年。[8] Eclipse。Eclipse：以太坊上的首个 Solana 虚拟机二层网络。Eclipse 文档，2024年。[9] G. Wood。波卡：异构多链框架构想。波卡白皮书，2016年。[10] C. Goes 等。区块链间通信协议概述。arXiv 预印本 arXiv:2006.15918，2020年。[11] A. Yakovenko。Solana：高性能区块链的新架构。Solana 白皮书，2018年。[12] Aptos Labs。Block-STM：通过将排序诅咒转化为性能优势来扩展区块链执行能力。arXiv 预印本 arXiv:2203.06871，2022年。

[13] N. Matsakis 和 A. Turon。Rayon：Rust 的数据并行库。https://github.com/rayon-rs/rayon，2015–2026。