Join-Accumulate Machine:
融合去信任机制与广泛协同能力的超级计算机

草案 0.6.5 - 2025 年 4 月 22 日

GAVIN WOOD 博士
波卡（Polkadot）与以太坊（Ethereum）创始人
gavin@parity.io

中文版译者：Qinwen Wang; Qinwen@Lollipop.builders

摘要：我们通过本文为 JAM 呈现一个全面且正式的定义。JAM 是一个结合了波卡（Polkadot）和以太坊（Ethereum）要素数据传输和处理机制的协议。在一个统一的、一致性的架构下，JAM 提供了一个在全局作用域中，以单例模式运行免许可、且基于对象的系统环境，这一环境与以太坊开创的智能合约环境类似，同时融合了波卡首创的，在由可扩展节点所构成的网络上进行的安全并行侧链计算。

JAM 引入了一个去中心化的混合系统，该系统围绕一个安全且可扩展的核心/链上二元结构设计，并内置了智能合约功能。尽管智能合约功能让人联想到以太坊的范式，但 JAM 所提供的服务整体模型在很大程度上受到了波卡底层架构的启发和驱动。

JAM 本质上是无需授权的，任何人都可以在 JAM 上部署代码来提供服务。服务产生的费用与代码所消耗的资源直接相关，而这些资源的费用则通过购买和分配 Coretime 来触发代码执行。Coretime 是一种衡量灵活且普遍存在的计算能力的指标，其计费模式与以太坊中用于执行操作的“gas”购买方式相类似。我们已构思出一种与波卡兼容的 CoreChains 服务方案。

1. 引言

1.1 术语说明

在本文中，我们介绍了一种去中心化的加密经济协议。该协议将在获得波卡（Polkadot）网络治理机构的批准后，作为一项重大更新被波卡网络所采用。

该协议的早期未完善版本最初在波卡 Fellowship RFC31 中提出，被称为 CoreJam。CoreJam 的名称来源于其服务提案核心的计算模型：收集、精炼、连接、累积。虽然 CoreJam RFC 提出了对波卡协议的不完整、范围有限的修改，但 JAM 指的是一个完整且状态一致的整体区块链协议。

1.2 驱动因素

在区块链及更广泛的 Web3 领域，我们首要的驱动力是打造具备高度韧性的系统。一个完善的 Web3 数字系统，应当始终坚守其声明的服务规范，理想状态下，这些规范不应被任何经济参与者（无论是个人、组织，还是其他 Web3 系统）的意愿、经济实力或权力地位所左右。这诚然是个宏大的理想。对于如何完美实现这一点，我们必须保持务实。尽管如此，Web3 系统应力求在实际应用中提供坚不可摧的保障，真正赢得“不可阻挡”的声誉。

尽管比特币可能是经济领域内这类系统的先驱，但从服务性质的通用性角度来看，它并非一个全方位适用的系统。比特币的有用程度，在基于规则的服务范畴内，严格受限于其规则设计的通用程度。比特币的原始设计，即为一个具有固定总量且所有权通过掌握特定私钥来近似验证并自动执行转移的系统，这一核心特性将在后续内容中深入展开探讨。

随后，以太坊横空出世，带来了一个前所未有的通用规则集，这个规则集是图灵完备的[1]。在我们着手构建一个旨在支持大规模多用户应用平台的 Web3 系统时，通用性成为了不可或缺的基石。因此，我们自然而然地将以太坊所提供的图灵完备性视为构建该系统的一个基本前提和既定条件。

除了韧性和通用性，事情变得更有趣了，促使我们进一步深挖背后的驱动力。就目前而言，我们确定了另外三个目标：

1. 韧性：拥有高度的系统抗停运、抗破坏和抗审查能力。
2. 通用性：能够执行图灵完备的计算。
3. 高性能：能够快速且低成本地执行计算。
4. 协同性：状态的不同元素之间可能存在的因果关系，以及单个应用程序的协作程度。
5. 可获得性：对创新几乎没有障碍；创新过程简便、快速、低成本且无需许可。

在探讨作为一种宣称的 Web3 技术时，我们隐含地预设了韧性与通用性这两个核心目标。然而，有趣的是，根据某个尚未明确命名的信息理论原则，我们发现高性能（目标3）与协同性（目标4）之间存在着一种潜在的对立关系。尽管我们坚信这一原理必定以某种形式存在着，但遗憾的是，我们目前尚无法为其提供一个确切的命名。为了便于后续的讨论与研究，我们暂且将这种对立关系命名为“规模－协同性对立”。

1.3 规模 - 协同性对立下的扩展

规模与协同性之间的对立关系是一个基本规律，它揭示了一个现象：随着信息系统状态空间的不断扩展，系统的协同性将不可避免地减弱。这一规律是基于因果关系受限于速度这一普遍原理的直接推论。在物理学中，信息的传播速度有一个上限，即真空中的光速 c。然而，在其他类型的信息系统中，这个速度上限可能会更低。例如，在生物系统中，信息的传递速度主要取决于化学反应的速率；而在电子系统中，它则受限于电子在物质中的移动速度。对于分布式软件系统来说，其速度下限往往更低，这主要取决于软件的设计、硬件的性能以及分组交换网络基础设施的可靠性和多样性。
其论点如下：

1. 系统用于数据处理的状态越多，这些状态所占据的空间就越大。
2. 使用的空间越大，状态组件之间的平均距离和距离方差就越大。
3. 随着平均距离和方差的增加，因果解决的时间（即所有正确影响被感知到的时间）在整个系统中变得分散，从而导致不协同性的问题。

[1] Gas 机制确实限制了程序的执行，对可执行的步数设置了上限。但在无权限环境中，肯定要引入某种限制以避免无限计算。

暂且不考虑整体的安全性问题，我们通过一种策略来应对不协同性，即将系统划分为多个因果独立的子系统，每个子系统都保持足够小的规模以确保其协同性。在资源充裕的条件下，细菌会选择分裂成两个个体，而不是单纯地增长到原先的两倍体积。这种生长模式实际上为我们提供了一种粗略的处理增长过程中不协同性的方法：系统内部的小规模处理能够保持较高的一致性，而系统之间的处理虽然能够支持更大的总体规模，但在协同性方面却有所欠缺。这正是波卡、Cosmos 以及扩展版以太坊（稍后我们将详细探讨）等元网络背后的核心理念。这些系统通常依赖于与“结算区域”的异步且简单的通信机制，这些区域提供了一个局限性的协同状态空间，专门用于管理特定的交互活动，例如代币的转移。

本研究致力于探索规模与协同性对立关系中的平衡之道。我们提出了一种创新的计算模型，旨在规避现有方法导致的系统状态空间过度碎片化问题。该模型将高度可扩展且基本协同的元素，通过流水线的方式传输至同步且完全协同的元素中。尽管我们并未完全消除异步性，但已将其局限在流水线的长度范围内。同时，我们采用了在多 CPU 共享内存系统中常见的“缓存亲和性”策略，以替代当前可扩展系统中普遍存在的粗粒度划分方式。

与基于零知识证明（SNARK）的二层区块链扩展技术不同，这个模型采纳了加密经济机制，不仅继承了其低成本、高性能的优势，而且有效规避了中心化的风险。

1.4 文档结构

我们在第 2 节中简要概述了当前区块链技术中的扩展方法。在第 3 节中，我们定义并阐明了将用于形式化表述的符号。

在第 4 节中，我们对该协议进行了广泛概述，勾勒出主要领域，包括波卡虚拟机（PVM）、共识协议 Safrole 和 GRANDPA、Common Clock，并为形式化表述奠定基础。

接着，我们进一步定义了完整的协议，该协议由两大核心部分组成。第一部分是正确的链上状态转换公式，这对于所有希望核实链上状态的节点而言至关重要，它们可以借此验证信息的准确性。第二部分，则在第 14 节和第 19 节中详细阐述，主要描述了持有验证器密钥的参与者在链下操作时应当遵循的诚实策略。

主体部分在第 20 节中讨论了该协议的性能特征，并在第 21 节中得出结论。

附录包含了对协议定义至关重要的各种补充材料，包括附录 A 和 B 中的 PVM、附录 C 和 D 中的序列化和默克尔化，以及附录 E、G 和 H 中的密码学内容。我们以术语索引结束，该索引包括附录 I 中工作中使用的所有简单常量项的值，并附上参考文献。

2. 过往工作与当下趋势

自以太坊黄皮书初次面世以来的数年间，区块链开发领域实现了飞跃式的发展。除了致力于提升可扩展性之外，开发工作还广泛涉及底层共识算法的优化、智能合约语言的创新、机器交互的增强以及整体状态环境的完善。尽管这些议题极具吸引力，但它们大多超出了本研究的范畴，因为它们通常并不直接影响底层的可扩展性机制。

2.1 波卡

为了构建系统化的服务体系，JAM 采纳了众多与波卡相似的博弈论原理及加密机制，这些机制在 2024 年被 Jeff Burdges、Cevallos 等人形象地称为“精灵”（ELVES）。然而，JAM 与上述机制存在着显著不同。具体而言，JAM 提供了一种全新的计算模式，更接近验证人网络本身的实际计算能力，而非那种简化或抽象化的单一模型。

波卡项目最初的核心构想是打造一个可扩展的异构多链架构，其设计理念在于通过工作负载的分区与分布式处理来提升系统性能。在此过程中，波卡明确指出了可组合性方面可能存在的妥协。从实际部署的角度来看，波卡的基石——平行链，在本质上具备高度的独立性。尽管引入了消息传递系统（XCMP），但该系统是异步运行的，且消息粒度较大，同时还依赖于一种高层次但进展相对缓慢的交互语言 XCM。

因此，相较于像以太坊这样的智能合约系统，波卡在其各组成链之间的可组合性方面稍显不足。以太坊提供了一个统一且通用的环境，为敏捷和创新的集成提供条件，这正是其成功的基石。而波卡目前更像是一个由多个独立生态系统组成的集合，其平行链之间的协作机会相当有限。尽管在用户体验上，波卡与其他通过桥相连的区块链非常相似，但在安全特性上却截然不同。有一个名为 SPREE 的技术提案，旨在利用波卡独特的共享安全性机制来增强可组合性，然而，即便如此，区块链之间仍将保持隔离状态。

实施并启动一个区块链项目不仅难度高、耗时长，而且成本高昂。根据波卡的原始设计，仅有那些既具备实施与启动区块链项目的能力，又能筹集到足够押金以在长期的插槽拍卖中胜出的团队，才有机会在波卡网络上进行部署。目前，波卡网络上长期可用的插槽数量大约只有 50 个。尽管波卡网络本身并非基于许可制，但其可访问性显然要低于类似以太坊的智能合约系统。

对于一个 Web3 应用平台而言，吸纳并促进尽可能多的创新者参与其中，实现他们之间的相互交互，以及与整个生态系统中其他用户群体的互动，无疑是衡量其成功的一个关键要素。正因如此，可访问性显得尤为重要。

2.2 以太坊

2014年，Wood 在被誉为本文精神先驱的“黄皮书”中，正式为以太坊协议确立了定义。在很大程度上源自 Buterin 在2013年所撰写的开创性概念论文。自“黄皮书”面世以来的十年间，以太坊的实际协议及其公共网络实例经历了多次重大变革，这些变革主要聚焦于通过优化交易格式、指令集，以及引入以太坊虚拟机（EVM）的“预编译”功能——即针对特定领域设计的高级附加指令，从而大幅提升了系统的灵活性。

近100万名加密经济参与者参与以太坊的验证[2]。区块扩展则是通过随机的出块节点轮换机制来实现的，该机制会在主出块节点生成区块之前公开其物理地址[3]。以太坊使用 Buterin 与 Griffith 于2019年提出的 Casper-FFG 协议来实现区块的最终确认；借助庞大的验证者网络规模，链的扩展大约每13分钟就能达成最终确认。

以太坊的直接计算性能与2015年推出时大致相似，但有一项显著的不同，它现在支持每个区块托管 1MB 的承诺数据，这些数据会被所有节点在限定时间内保存。虽然这些数据无法直接参与主状态转换过程，但借助特定函数，可以验证这些数据（或其特定部分）的有效性。根据以太坊基金会 2024b 年的报告，当前的设计方向是，在未来几年内，借助名为 Dank-sharding 的协议，将存储职责分散至广泛的验证者群体中，以进一步优化这一机制。

根据以太坊基金会 2024a 年度的报告，以太坊采取了将数据可用性与多样化的侧链计算设施（即 Roll-ups）相结合的策略来应对扩容问题。在这一策略中，基于 zk-SNARK 技术的 Roll-ups 被明确指定为首选方案。值得注意的是，市场上不同的 Roll-up 供应商在其设计、执行及运营方面均展现出独特的特性和影响力。

[2] 实际问题确实制约了真正的去中心化程度。验证者软件具备允许单个实例配置多个密钥集的功能，从个体操作者和硬件角度来看，系统性地削弱了实际的去中心化效果，远低于表面上参与者的数量。根据 Dune 和 hildobby 在 2024 年整理的关于以太坊 (Ethereum 2.0) 的数据，一个主要的节点运营商 Lido在近百万的加密经济参与者中占据了近三分之一的份额。

[3] Ethereum 开发团队正努力改善这一现状，以进一步提升平台安全性，但目前尚未确定具体的实施时间表。

人们有理由相信，通过多供应商整合来推动规模扩张的多元化市场策略，能够促使那些经过深思熟虑的解决方案蓬勃发展。然而，这一策略并非没有挑战。Sharma 在2023年发布的一份研究报告深入探讨了各类 Roll-ups 的去中心化程度，结果显示普遍存在中心化的倾向。尽管如此，报告同时也提到了正在进行的缓解这一问题的努力。至于这些努力最终能否让Roll-ups实现更高程度的去中心化，我们仍需拭目以待。

在各家 Roll-ups 供应商激烈竞争所构建的宏大的拼贴式整合中，不同领域间的异构通信特性（例如数据报文延迟、语义差异等）、安全特性（包括撤销、损坏、停滞及审查所带来的成本）以及经济特性（接收和处理特定传入消息或交易所需承担的费用）均可能呈现出显著的差异性。尽管以太坊网络未来有可能为侧链计算提供必要的底层支撑机制，但关于是否会迎来一次全面的"大整合"，目前尚不清楚。关于这种碎片化策略可能引发的种种负面效应，我们仍缺乏充分而深入的探论[4]。

2.2.1 Snark Roll-ups

虽然以太坊协议的基础对 Roll-ups 的性质没有做出重大假设，但以太坊的侧链计算策略确实聚焦于基于 SNARK 的 Roll-ups，这一战略导向促使以太坊协议逐步演进，以适应并优化此类策略的实施。SNARK 是密码学领域的一项独特技术，能够生成证明，向中立的第三方证实某些预定义计算的正确性它允许构造证明。验证这些证明所需的复杂性通常与被验证的计算规模呈亚线性关系，而且这一验证过程不会泄露任何关于计算内部细节或其依赖的见证数据的信息。

Zk-SNARKs 技术确实面临一些限制。在运用时，我们需要在证明的大小、验证过程的复杂性以及生成证明所需的计算量之间做出权衡。尤其是当涉及到包含大量二进制操作的非标准通用计算时（这正是智能合约的魅力所在），这些计算往往很难与 SNARKs 的模型相契合。

以 RISC-Zero 为例（根据 Bögli 在2024年的评估），这是一个前沿项目，它专门为基于 RISC-V 虚拟机（一种广受欢迎、开源且设计简洁的 RISC 架构，得到了广泛工具支持）的计算提供 SNARKs 生成平台。根据2024年 Polkavm 项目发布的最新基准测试报告，相比 RISC-Zero 自身的基准结果，仅生成证明就比简单地重新编译并执行（相同计算）要慢 61,000倍，即使是在 32倍的 CPU 核心、20,000倍的内存以及附加一台顶尖 GPU 的环境下运行，依然无法弥补这部分开销。而根据硬件租赁服务商 https://cloud - gpus.com 的数据，使用 RISC-Zero 进行证明的成本，是使用 Polkavm 重新编译器执行成本的 66,000,000倍[5]。

许多加密原语因成本过高而变得不切实际，不得不被专门的算法和结构所替代。然而，这些替代方案往往在其他性能指标上也不尽如人意。考虑到可能会采用 SNARKs（例如 Gabizon、Williamson 和 Ciobotaru 在2019年提出的 PLONK），以太坊项目的 Dank-sharding 可用性系统主流设计选择了一种基于大质数域上的多项式承诺的纠删码形式。这种形式使得 SNARKs 能够以可接受的性能访问数据的子部分。但与其他选项（如本文中提到的二进制域和默克尔树结构）相比，这一选择导致验证器节点的 CPU 使用率显著上升，高出了几个数量级。

除了基本成本问题，SNARKs 并不能有效地减少对去中心化和冗余的需求，这进而引发了额外的成本上升。尽管 SNARKs 的可验证性特性在一定程度上减少了对于某些权益质押去中心化优势的需求，但为了防范单个实体形成垄断（或多个实体结成联营），仍然需要激励多方进行大体相同的工作。理论上，证明错误的状态转换应当是不可能的，然而服务完整性仍可能以其他方式受损；即便是短暂的，如仅暂停几分钟的证明生成，对于实时金融应用而言，也可能导致重大的经济影响。

现实世界中不乏中心化引发垄断的实例，基于 SNARK 的交易所框架便是其中之一。该框架虽名义上服务于去中心化交易，但实质上却呈现出中心化的特征。Starkware 凭借其生成与提交交易证明的能力，在该框架中占据了垄断性的地位。这种中心化的结构带来了单点故障的风险：一旦 Starkware 的服务出现问题，整个系统的活跃度与运作能力都将受到直接影响。

---

[4] 基于对这一问题的初步思考，Solana 于2024年提出了一项提案：借助 Polkadot 技术，在侧链（Roll-up）生态系统之间实现一定程度的互操作性。

[5] 实际上，很可能性能还会高出很多，因为这是在消费级设备上使用的低端"备用"硬件，而且我们的重编译器还未经过优化。

目前，尚未有确凿证据表明基于 SNARK 的信任消除计算策略能够在成本上与多方加密经济平台相抗衡。事实上，所有已提出的基于 SNARK 的解决方案都高度依赖于加密经济系统来构建框架并解决相关问题。其中，数据可用性和排序是两个公认的需要借助加密经济方案来解决的领域。

值得注意的是，SNARK 技术正处于不断演进之中，其背后的密码学家与工程师确实对未来几年内的技术改进抱有期待。在 Thaler 于2023年发表的一篇文章中，我们可以看到一些可靠的预测：随着某些密码学技术的最新进展，证明生成的速度放缓可能仅相当于常规原生执行的 50,000倍，并且其中大部分工作可以实现并行处理。相较于当前状况，这无疑是一个显著的进步。然而，即便如此，与成熟的加密经济技术（例如 ELVES）在成本效益上的竞争相比，SNARK 技术的速度仍然慢了几个数量级。

2.3 碎片化元网络

其他项目在探索通用计算可扩展性时，主要聚焦于两种策略：碎片化方法与中心化方法。然而，我们坚信这两种途径都未能给出令人满意的解决方案。

碎片化策略得到了 Cosmos（由 Kwon 和 Buchman 于2019年提出）以及 Avalanche（由 Tanana 于同年提出）等项目的推崇。这一策略构建了一个系统，该系统由采用相同共识机制但由不同动机的验证器集运行的网络片段组成。这与波卡采用的单一验证器集模式，以及以太坊所倡导的异构 Roll-ups 策略形成了鲜明对比。在以太坊的异构 Roll-ups 策略中，部分安全保障依赖于在统一激励框架下运作的相同验证器集。碎片化策略的同质性特点，使得消息传递机制能够保持较高的一致性，这为众多相互连接的网络提供了一个相对统一的交互界面。然而，这种表面上的统一其实并不深入。这些网络的无信任特性，仅在其验证器能够正确运行的前提下才成立，而验证器的运行则依赖于由经济激励和惩罚措施共同构建及执行的加密经济安全框架。若要在相同的安全标准下实现工作量翻倍，且验证器集之间无需进行特殊协调，那么这样的系统本质上要求构建一个具有相同总体激励水平的新网络。

几个关键问题浮现出来。首要的是，波卡的孤立平行链与以太坊的孤立 Roll-up 链均存在一个共同的缺陷：它们各自持续的分片状态导致了协同性的缺失，进而使得这些链无法进行同步的组合。

更为严重的是，Cosmos 所提出的通过碎片化来实现扩展的策略，在安全性上并未达到同质化，因此也无法提供无需信任的坚实保障。各网络间的验证器集合需被视为独立选择并受各自激励的实体。这导致了一个网络中的拜占庭行为，与另一个网络中可能产生的后果之间，既不存在直接的因果关系，也难以预测其概率联系。简而言之，若验证器勾结起来破坏或篡改某一网络的状态，其连锁反应可能会波及整个生态系统内的其他网络。

这是一个普遍认知的问题，针对此，项目方提出了两种应对策略。首要策略是通过从统一的验证者群体中分配资源，来修复网络间潜在攻击的成本预期（进而增强安全性）。这种方法，即 Cosmos 项目于2023年提出的“复制安全性”大规模冗余策略，要求每位验证者必须在所有网络上进行验证，并接受相同的激励与惩罚措施。然而，从经济角度来看，这种安全提供方式效率较低，因为每个网络都需要独立地提供与其希望互操作的最安全网络相当水平的激励与惩罚，以确保验证者的经济动机不受影响，同时维持所有网络在安全性上的等价性。值得注意的是，目前复制安全性并非一种现成的、无需许可即可使用的服务。我们或许可以推测，这些惩罚性的经济机制与这一现状存在一定的关联。

由 OmniLedger 团队（Kokoris-Kogias 等人，2017年）提出的更高效的方法是：

将验证者进行非冗余化分配，按照不同的网络进行专门配置，并定期进行安全且随机的重新分配。相较于将所有验证者集中部署于单一网络，这种做法能够降低攻击成本，因为即便在整体验证者群体中恶意验证者的比例未超过安全阈值，单个网络

仍有可能偶然遭遇具有破坏性的恶意验证者。然而，这种分配方式在弱一致性要求的基础上，为系统提供了一种有效的扩展途径。通过精心设计的重新分配机制，可以在保障系统安全性的同时，实现更高效的资源利用和扩展能力。

或者，我们可以考虑 Jeff Burdges、Cevallos 等人在2024年提出的 ELVES 方案。该方案采用非冗余验证者分配策略，并将其与验证者参与的提议与审计博弈相结合。这一机制旨在清除和惩罚无效计算，同时确保一个网络的最终性状态依赖于所有因果相关的网络。在提出的三种解决方案中，ELVES 被视为最安全且经济高效的途径。它通过提供高度可靠的机制，在最终确定效应之前识别和纠正所有无效转变，从而保障了系统的安全性。然而，实施 ELVES 方案需要更为复杂的逻辑设计，并且由于网络间的因果纠缠，可能会对网络的可扩展性，即可以添加的网络数量，构成一定的限制。

2.4 高性能全同步网络

近年来，区块链领域呈现出另一种发展态势，即采取一系列“战术性”措施来优化数据吞吐量。这包括限制验证者集合的规模与多样性，以精简系统；专注于软件层面的深度优化，提升运行效率；强化验证者间的一致性要求，确保数据同步无误；对验证者所需硬件设定高标准，保障系统性能；以及限制数据可用性范围。

Solana 区块链建立在 Yakovenko 于2018年引入的技术基础之上，理论上能够每秒处理超过 70 万笔交易。然而，根据 Ng 在2024年的报告，该网络实际处理的交易量远低于这一理论上限，但仍远高于大多数区块链网络。这得益于一系列精心设计的工程优化，旨在最大限度地提升同步性能。这些努力共同塑造了一个高度一致性的智能合约环境，其应用程序编程接口（API）与以太坊黄皮书所描述的以太坊环境相似（尽管采用了不同的底层虚拟机技术）。在 Solana 上，交易一旦被纳入区块，便几乎可以立即获得确认和最终确定性。

这种方法面临两大挑战：首要的是，将协议高度优化为特定的代码库，可能会导致结构上的中心化趋势，进而削弱其整体的韧性。Jha 在2024年的报告中指出：“自2022年1月以来，已经发生了 11 次重大故障，累计导致 15 天的部分或完全停机时间。”这在主流区块链领域中显得尤为突出，因为大多数主要区块链几乎不存在停机情况。这些停机的根源复杂多样，但往往可以归结为各个子系统内被发现的漏洞。

与之形成鲜明对比的是以太坊，至少在最近之前，它展现了极高的稳健性。以太坊拥有经过严格审查的规范、对其加密经济基础的深入研究，以及多个独立开发的实现版本。因此，当其最广泛部署的实现版本中发现漏洞并被恶意利用时，正如 Hertig 在2016年所描述的那样，以太坊网络依然能够保持大部分功能的运行，这或许并不令人感到意外。

第二个核心难题在于该协议在无法跨越单台机器硬件界限时，其最终所能达到的可扩展性极限。

在大规模应用场景下，历史交易数据和状态信息会迅速增长到难以管理的地步。Solana 区块链就凸显了这一问题可能带来的严峻挑战。与传统区块链有所不同，Solana 协议并未提供针对历史数据的归档与后续审核的有效方案，而这对于第三方从基本原理出发验证当前状态准确性至关重要。尽管相关文献对 Solana 如何处理这一难题的讨论甚少，但根据 Solana 基金会 2023 年的声明，节点仅仅是将数据存储在谷歌托管的集中式数据库中[6]。

Solana 鼓励验证者配备大容量内存（Solana Labs 在2024年的建议是至少 512GB），以便在内存中存储其庞大的状态数据。然而，Solana 也揭示了一个事实：在没有采用分而治之策略的情况下，验证者所能提供的硬件水平直接决定了完全同步、一致执行模型所能达到的性能上限。这导致硬件要求成为了验证者参与的一道门槛，且在不牺牲去中心化原则以及最终透明度的前提下，这一门槛难以被有效提升。

[6] 较早版本的节点使用了 Arweave 网络（一种去中心化的数据存储），但发现它无法满足 Solana 所需的数据吞吐量要求，因此并不可靠。

3. 符号约定

与以太坊黄皮书一样，本文中使用了一些符号约定。为清晰起见，我们在此对这些约定进行定义。以下将以太坊黄皮书简称为 ***YP***。

3.1 排版

我们采用多种字体风格来区分不同类型的术语。当某个术语仅在文档的某个局部段落中使用，且仅与该段落内容相关时，我们选用小写罗马字母来表示，例如 x，y（常用于表示集合或序列中的元素）或 i，j（常用于表示数值索引）。若是在局部上下文中提及布尔术语或函数，我们倾向于使用大写罗马字母，如 A, F。而当某个术语的复杂性或多维度需要特别强调时，我们可能会使用粗体字，尤其是在涉及序列和集合的场合。

对于在本文中始终保持其特定定义的项目，我们遵循额外的排版约定。集合通常使用黑板粗体字体来表示，例如，$\mathbb{N}$ 代表包含零在内的所有自然数的集合。对于可以参数化的集合，我们会通过下标或在括号内添加参数来进行表示。在本文中使用的但未专门定义的外部导入函数，我们用花体字母来表示，例如，$\mathcal{H}$ 代表 Blake2 加密哈希函数。而对于本文中首次引入的、不依赖于特定上下文的其他函数，我们则采用大写希腊字母来表示，例如，Υ 代表状态转换函数。

在本文中，对于那些并非固定但始终保持一致含义的值，我们采用希腊小写字母来表示，例如 σ 用于表示状态标识符。当这些值异常复杂时，我们可能会使用加粗字体来突出显示。

3.2 函数与运算符

我们引入了“先于”关系（precedes relation）这一概念，用以表明一个术语是基于另一个术语来定义的。具体而言，$y < x$ 表示 y 可以完全根据 x 来定义：

$$y < x \Leftrightarrow \exists f : y = f(x) \tag{3.1}$$

该函数等价于第一个非空集（$\emptyset$）的参数，若不存在这样的参数，则结果为空集 $\emptyset$ ：

$$\mathcal{U}(\mathrm{a}_0, \ldots \mathrm{a}_n) \equiv \mathrm{a}_x : (\mathrm{a}_x \neq \emptyset \vee x = n), \bigwedge_{i=0}^{x-1} \mathrm{a}_i = \emptyset \tag{3.2}$$

因此，例如 $\mathcal{U}(\emptyset, 1, \emptyset, 2) = 1 \text{and} \mathcal{U}(\emptyset, \emptyset) = \emptyset$ 。

3.3 集合

给定一个集合 $\mathbf{s}$，其幂集（power set）和基数（cardinality）分别用通常的 $\wp(\mathbf{s})$ 和 $|\mathbf{s}|$ 来表示。在形成幂集时，我们能够通过应用数字下标来限定结果中集合的特定基数范围。
例如，$\wp\langle\{1,2,3\}\rangle_2 = \{\{1,2\},\{1,3\},\{2,3\}\}$ 。

集合可以与标量进行运算，此时结果是一个对每个元素都应用了该运算的新集合，例如，$\{1,2,3\} + 3 = \{4,5,6\}$ 。函数也可以应用于集合的所有成员以生成一个新集合，但为了清晰起见，我们用 $^\#$ 上标来表示这一点，
例如，$f^\#(\{1,2\}) \equiv \{f(1), f(2)\}$ 。

我们用关系符号 $\wr$ 来表示集合的互斥性。形式上：

$$\mathrm{A} \cap \mathrm{B} = \emptyset \Leftrightarrow \mathrm{A} \perp \mathrm{B}$$

我们通常使用 $\emptyset$ 表示某个项被合法地留空，没有具体的值。其基数定义为零。我们定义操作 ？，使得 $\mathrm{A?} \equiv \mathrm{A} \cup \{\emptyset\}$，表示在原有集合的基础上增加了 $\emptyset$ 空集。

术语 ∇ 用于表明操作的意外失败，或某个值无效或意外。（我们尽量避免在此使用更常规的 $\perp$，以免与布尔值 false 混淆，因为在特定场景中，false 可能被解释为某种成功结果）

3.4 数字

$\mathbb{N}$ 表示包括 0 在内的自然数集，而 $\mathbb{N}_n$ 则意味着对该集合加以限制，只包含小于 n 的值。$\mathbb{N} = \{0,1,\ldots\}$，而 $\mathbb{N}_n = \{x \mid x \in \mathbb{N}, x < n\}$。

$\mathbb{Z}$ 表示整数集。我们用 $\mathbb{Z}_{a\ldots b}$ 来表示区间 $[a, b)$ 内的整数集。形式上：$\mathbb{Z}_{a\ldots b} = \{x | x \in \mathbb{Z}, a \le x < b\}$。例如，$\mathbb{Z}_{2\ldots5} = \{2,3,4\}$。我们用偏移/长度形式 $\mathbb{Z}_{a\ldots+b}$ 来表示这个集合，它是 $\mathbb{Z}_{a\ldots a+b}$ 的简写形式。

有时，表示序列的长度同时限制其大小是很有用的，特别是在处理必须实际存储的八位字节序列时。通常，这些长度可以定义为集合 $\mathbb{N}_{2^{32}}$。为了提高清晰度，我们用 $\mathbb{N}_L$ 来表示八位字节序列的长度的集合，它等价于 $\mathbb{N}_{2^{32}}$。

我们将 % 运算符表示为取模运算符，例如 $5 \,\%\, 3 = 2$。此外，我们有时可能会用分隔符 R 来表示除法的商和余数，例如 $5 \div 3 = 1R2$。

3.5 字典

字典是一种可能不完整的映射，从某个定义域映射到某个陪域，方式与常规函数相似。然而，与函数不同的是，字典中所有配对的集合必然是可枚举的，并且我们用某种数据结构来表示它们，即所有（$key \mapsto value$）对的集合。（在这种数据定义的映射中，通常将定义域内的值称为 “key”，将陪域内的值称为 “$value$”，因此得名字典。）

因此，我们定义形式化表示 $\mathbb{D}\langle \mathrm{K} \to \mathrm{V} \rangle$ 来表示一个从定义域 K 映射到值域 V 的字典。我们将字典定义为所有字典集合 $\mathbb{D}$ 中的一个元素，并且将其定义为一组对 $p = (k \mapsto v)$：

$$\mathbb{D} \subset \{\{(k \mapsto v)\}\} \tag{3.3}$$

对于任何 key k，一个字典的成员最多只能关联一个唯一的值。

$$\forall \mathbf{d} \in \mathbb{D} : \forall (k \mapsto v) \in \mathbf{d} : \exists! v' : (k \mapsto v') \in \mathbf{d} \tag{3.4}$$

这一断言使我们能够明确地为字典 d 定义下标运算符和减法运算符：

$$\forall \mathbf{d} \in \mathbb{D} : \mathbf{d}[k] \equiv \begin{cases} v & \text{if} \exists k : (k \mapsto v) \in \mathbf{d} \\ \emptyset & \text{otherwise} \end{cases} \tag{3.5}$$

$$\forall \mathbf{d} \in \mathbb{D}, \mathbf{s} \subseteq \mathrm{K} : \mathbf{d} \setminus \mathbf{s} \equiv \{(k \mapsto v) : (k \mapsto v) \in \mathbf{d}, k \notin \mathbf{s}\} \tag{3.6}$$

注意，当使用下标运算符时，这隐含地断言了该 key 在字典中存在。如果该 key 不存在，则结果是未定义的，任何依赖于此的代码块都必须被视为无效。

通常，限制 keys 和 values 可以从中抽取的集合是很有用的。形式上：我们将类型化字典 $\mathbb{D}\langle K \to V\rangle$ 定义为一组形式为 $(k \mapsto v)$ 的对 p 的集合：

$$\mathbb{D}\langle K \to V\rangle \subset \mathbb{D} \tag{3.7}$$

$$\mathbb{D}\langle K \to V\rangle \equiv \{(k \mapsto v) | k \in K \wedge v \in V\} \tag{3.8}$$

为了表示字典 $\mathbf{d} \in \mathbb{D}\langle K \to V\rangle$ 的定义域（即 keys 的集合），我们使用 $\mathcal{K}(\mathbf{d}) \subseteq K$，而为了表示值域（即 values 的集合），我们使用 $\mathcal{V}(d) \subseteq V$。形式上：

$$\mathcal{K}(\mathbf{d} \in \mathbb{D}) \equiv \{k | \exists v : (k \mapsto v) \in \mathbf{d}\} \tag{3.9}$$

$$\mathcal{V}(\mathbf{d} \in \mathbb{D}) \equiv \{v | \exists k : (k \mapsto v) \in \mathbf{d}\} \tag{3.10}$$

注意，由于值域 $\mathcal{V}$ 是一个集合，如果字典中出现具有相同值的不同 keys，该集合将只包含其中一个这样的值。
字典可以通过并集运算符 $\cup$ 进行合并，在 key 冲突 (key-collision) 的情况下，优先保留右侧操作数的值：

$$\forall \mathbf{d} \in \mathbb{D}, \mathbf{e} \in \mathbb{D} : \mathbf{d} \cup \mathbf{e} \equiv (\mathbf{d} \setminus \mathcal{K}(\mathbf{e})) \cup \mathbf{e} \tag{3.11}$$

3.6 元组

元组是由一组值组成的集合，其中每个元素可能属于不同的集合。它们用括号表示，例如，自然数 3 和 5 组成的元组 t 表示为 $t = (3, 5)$，它存在于自然数对的集合中，有时表示为 $\mathbb{N} \times \mathbb{N}$，但在本文中表示为 $(\mathbb{N}, \mathbb{N})$。

我们经常需要引用元组值中的特定项，因此为每个项声明一个名称会很方便。例如，我们可以将一个具有两个命名自然数组件 a 和 b 的元组表示为 $T = (a \in \mathbb{N}, b \in \mathbb{N})$。我们会通过下标其名称来表示 $t \in T$，因此对于某个 $t = (\mathrm{a}; 3, b; 5)$ $\mathrm{t}_a = 3$ 且 $\mathrm{t}_b = 5$。

3.7 序列

序列是一系列按特定顺序排列的元素，其顺序不依赖于元素的值。所有元素均来自某个集合 T 的序列集合表示为 $[\![T]\!]$，它定义了一个从 $\mathbb{N} \to T$ 的部分映射。恰好包含 n 个元素且每个元素都属于集合 T 的序列集合可以表示为 $[\![T]\!]_n$，并相应地定义了一个从 $\mathbb{N}_n \to T$ 的完整映射。类似地，包含最多 n 个元素和至少 n 个元素的序列集合可以分别表示为 $[\![T]\!]_{:n}$ 和 $[\![T]\!]_{n:}$。

序列可以通过下标进行索引，因此序列 $\mathbf{s}$ 中索引为 i 的特定项可以表示为 $\mathbf{s}[i]$，或者在无歧义的情况下表示为 $\mathbf{s}_i$。范围可以使用省略号表示，例如：$[0,1,2,3]_{\ldots 2} = [0,1]$ 和 $[0,1,2,3]_{1\cdots+2} = [1,2]$。这样的序列的长度可以表示为 $|\mathbf{s}|$。
我们将模运算下标表示为 $\mathbf{s}[i]^{\circlearrowleft} \equiv \mathbf{s}[i \mod |\mathbf{s}|]$。我们通过函数 $\text{last}(\mathbf{s}) \equiv x$ 来表示序列 $\mathbf{s} = [\ldots, x]$ 的最后一个元素 x。

3.7.1 构造

我们可能希望根据其他值的递增下标来定义一个序列：$[\mathbf{x}_0, \mathbf{x}_1, \ldots]_n$ 表示一个从 $\mathbf{x}_0$ 开始一直延续到 $\mathbf{x}_{n-1}$ 的 n 个值的序列。此外，我们也可能希望定义一个序列，其中每个元素都是其索引 i 的函数；在这种情况下，我们用 $\left[f(i) | i \lessdot \mathbb{N}_n\right] \equiv \left[f(0), f(1), \ldots, f(n-1)\right]$。因此，当元素的顺序很重要时，我们使用 $\lessdot$ 而不是无序的表示法 $\in$。后者也可以简写为 $[f(i \lessdot \mathbb{N}_n)]$。这适用于任何具有明确顺序的集合，特别是序列，因此 $[i^2 | i \lessdot [1,2,3]] = [1,4,9]$。多个序列可以组合，因此 $[i \cdot j | i \lessdot [1,2,3], j < [2,3,4]] = [2,6,12]$。

与集合一样，我们使用显式表示法 $f^{\#}$ 来表示一个函数，该函数映射序列中的所有项。

序列可以通过序列排序表示法 $\left[i_k | i \in X\right]$ 从集合或其他序列中构造出来，其中忽略原有顺序。该表示法定义为产生其参数的集合或序列，但所有元素 i 都按照对应的值 i_k 的升序排列。

关键组件可以被省略，在这种情况下，假设直接按元素排序；即 $[i \in X] \equiv \left[i | i \in X\right].\left[i_k | i \in X\right]$ 具有相同的作用，但排除 i 的任何重复值。例如，假设 $\mathbf{s} = [1,3,2,3]$ ，那么 $[i || ; i \in \mathbf{s}] = [1,2,3]$ 且 $[-i | i \in \mathbf{s}] = [3,3,2,1]$ 。

集合可以通过常规集合构造语法从序列中构造出来，例如，假设 $\mathbf{s} = [1,2,3,1]$ ，那么 $\{a | a \in \mathbf{s}\}$ 将等价于 $\{1,2,3\}$ 。

具有定义顺序的值序列具有类似于常规字典的隐含顺序，因此 $[1,2,3] < [1,2,4]$ ，$[1,2,3] < [1,2,3,1]$ 。

3.7.2 编辑

我们定义了序列连接操作符 $\frown$ ，使得 $\left[\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{y}_0, \mathbf{y}_1, ...\right] \equiv \mathbf{x} \frown \mathbf{y}$ 。对于序列的序列，我们定义了一个一元的全连接操作符：$\hat{\mathbf{x}} \equiv \mathbf{x}_0 \frown \mathbf{x}_1 \frown ....$ 。此外，我们将元素连接表示为 $x \doteplus i \equiv x \frown [i]$ 。我们将由序列 $\mathbf{s}$ 的前 n 个元素组成的序列表示为 $\vec{\mathbf{s}}^n \equiv \left[\mathbf{s}_0, \mathbf{s}_1, ..., \mathbf{s}_{n-1}\right]$ ，而仅将最后 n 个元素表示为 $\overleftarrow{\mathbf{s}}^n$ 。

我们定义 $\mathrm{T}_{\mathbf{X}}$ 为序列之序列 $\mathbf{X}$ 的转置，其完整定义见方程 H.5。我们也可以将其应用于元组序列，以生成序列的元组。

我们用集合减法运算符的稍作修改来表示序列减法；具体来说，某个序列 $\mathbf{s}$ 中去掉最左边等于 v 的元素，可以表示为 $\mathbf{s} \setminus \{v\}$。

3.7.3 布尔值

$\mathbb{B}_s$ 表示长度为 s 的布尔字符串集合，因此 $\mathbb{B}_s = [\{\bot, \top\}]_s$ 。在处理布尔值时，我们可以假设存在一个隐式的等价映射到比特，其中 $\top = 1$且 $\bot = 0$ ，因此 $\mathbb{B}_{\square} = [\mathbb{N}_2]_{\square}$ 。我们使用函数 $\text{bits}\,(\mathbb{Y}) \in \mathbb{B}$ 来表示比特序列，按照最高有效位在前的顺序排列，该序列代表八位字节序列 $\mathbb{Y}$，因此 $\text{bits}\,([160,0]) = [1,0,1,0,0,...]$ 。

一元非运算符既适用于布尔值，也适用于布尔值序列，因此 $\neg\top = \bot$ ，并且 $\neg[\top, \bot] = [\bot, \top]$ 。

3.7.4 八位字节和二进制块

$\mathbb{Y}$ 表示任意长度的八位字节字符串（“blobs”）的集合。不出所料，$\mathbb{Y}_x$ 表示长度为 x 的此类序列的集合。 $\mathbb{Y}_{\$}$ 表示 $\mathbb{Y}$ 的子集，其中包含 ASCII 编码的字符串。需要注意的是，虽然一个八位字节与小于 256 的自然数之间具有隐含且明显的双射关系，并且我们可以在八位字节形式和自然数形式之间进行隐式转换，但我们并不将它们视为完全等价的实体。特别是在序列化的目的下，一个八位字节总是序列化为自身，而一个自然数可能会被序列化为多个八位字节的序列，这取决于其大小以及编码变体。

3.7.5 随机打乱

我们定义了一个序列打乱函数 F，该函数最初由 Fisher 和 Yates 在1938年提出。到2024年，Wikipedia 描述了一种高效的就地算法来实现这一功能。该函数接收一个序列以及一些熵作为输入。熵可以是一个无限长的自然数序列，也可以是一个哈希值。函数会返回一个与输入序列长度相同、包含相同元素但顺序由提供的熵决定的序列。关于该函数的完整定义，请参阅附录 F。

3.8 密码学

3.8.1 哈希

$\mathbb{H}$ 表示通常通过密码学函数得到的 256 位值的集合，相当于 $\mathbb{Y}_{32}$ ，其中 $\mathbb{H}^0$ 等于 $[0]_{32}$ 。我们假设有一个函数 $\mathcal{H}(m \in \mathbb{Y}) \in \mathbb{H}$ ，表示由 Saarinen 和 Aumasson 在2015年引入的 Blake2b 256 位哈希函数；另有一个函数 $\mathcal{H}_k(m \in \mathbb{Y}) \in \mathbb{H}$ ，表示由 Bertoni 等人于2013年提出的 Keccak 256 位哈希函数，该函数由 Wood 在2014年使用。

有时我们可能只希望取哈希值的前 x 个八位字节，在这种情况下，我们用 $\mathcal{H}_x(m) \in \mathbb{Y}_x$ 来表示 $\mathcal{H}(m)$ 的前 x 个八位字节。哈希函数的输入应通过我们的序列化编解码器 $\mathcal{E}$ 进行处理，以生成可以应用加密技术的八位字节序列。（注意，八位字节序列方便地保持了其原始数据的恒等性）我们可能希望用假定的解码函数 $\mathcal{E}^{-1}(x \in \mathbb{Y})$ 将一段八位字节序列解释为另一种类型的值。在这两种情况下，我们都可以在转换函数的下标中标明我们期望该八位字节序列项所具有的字节数。因此， $\mathrm{r} = \mathcal{E}_4(x \in \mathbb{N})$ 表示 $x \in \mathbb{N}_{2^{32}}$ 且 $r \in \mathbb{Y}_4$ ，而 $s = \mathcal{E}_8^{-1}(y)$ 表示 $y \in \mathbb{Y}_8$ 且 $s \in \mathbb{N}_{2^{64}}$ 。

3.8.2 签名方案

$\mathbb{E}_k\langle m\rangle \subset \mathbb{Y}_{64}$ 表示有效的 Ed25519 签名集合，该集合由 Josefsson 和 Liusvaara 在2017年定义，是通过掌握一个私钥而生成的，该私钥对应的公钥为 $k \in \mathbb{Y}_{32}$ ，且签名的消息为 m 。为便于阅读，我们将有效公钥的集合表示为 $\mathbb{H}_E$ 。

我们使用 $\mathbb{Y}_{\mathrm{BLS}} \subset \mathbb{Y}_{144}$ 来表示 BLS 签名方案的公钥集合，该方案由 Boneh、Lynn 和 Shacham 在2004年描述，基于 Hopwood 等人在2020年定义的 bls12-381 曲线。

我们将有效的 Bandersnatch 公钥集合记为 $\mathbb{H}_B$ ，其定义见附录 G。 $\mathbb{F}_{k\in\mathbb{H}_B}^{m\in\mathbb{Y}}\langle x \in \mathbb{Y}\rangle \subset \mathbb{Y}_{96}$ 表示利用公钥 k 的秘密对应项、某些上下文 x 和消息 m 生成的有效单一上下文签名集合。

同时， $\bar{\mathbb{F}}_{r\in\mathbb{Y}_R}^{m\in\mathbb{Y}}\langle x \in \mathbb{Y}\rangle \subset \mathbb{Y}_{784}$ 表示有效的 Bandersnatch RingVRF 确定性单一上下文知识证明集合，用于证明在某个秘密集合中掌握某个秘密，而该秘密集合由有效根集合 $\mathbb{Y}_{\mathrm{R}} \subset \mathbb{Y}_{144}$ 中的某个根标识。我们用 $\mathcal{O}(\mathbf{s} \in [\mathbb{H}_B]) \in \mathbb{Y}_R$ 来表示特定于公钥对应项集合 $\mathbf{s}$ 的根。一个根意味着一组特定的 Bandersnatch 密钥对，知道其中一个秘密就意味着能够生成一个唯一、有效且匿名的证明，证明在该集合中知道一个唯一的秘密。

Bandersnatch 签名和 RingVRF 证明都严格要求成员在生成签名时将其秘密密钥与上下文 x 以及消息 m 结合使用；区别在于，前者中成员是被识别的，而后者中成员是匿名的。此外，两者都定义了一个 VRF（可验证随机函数）输出，这是一个受 x 影响但不受 m 影响的高熵哈希，正式表示为 $\mathcal{Y}(\bar{\mathbb{F}}_r^m\langle x\rangle) \subset \mathbb{H}$ 和 $\mathcal{Y}(\mathbb{F}_k^m\langle x\rangle) \subset \mathbb{H}$ 。

我们定义函数 $\mathcal{S}$ 为签名函数，使得 $\mathcal{S}_k(\mathrm{m}) \in \mathbb{F}_k^m\langle[]\rangle \cup \mathbb{E}_k\langle m\rangle$ 。我们断言，计算该函数结果的能力依赖于对密钥的掌握。

4. 概述

正如在黄皮书中所述，我们首先通过回顾来开始我们的形式化定义：区块链可以定义为某个初始状态与一个区块级状态转移函数的配对。这个状态转移函数，在给定一个先前的状态以及施加于该状态上的数据块之后，能够确定下一个状态。形式上，我们可以将其表述为：

(4.1) $$\sigma' \equiv \Upsilon(\sigma, \mathbf{B})$$

其中，σ 表示先前状态，σ' 表示后续状态，B 是某个有效的区块，而是我们的区块级状态转移函数。广义上讲，JAM（以及一般的区块链）可以通过指定 Υ 和某个创世状态 $\sigma^{0.7}$ [7]。来简单定义。我们还做出了几个额外的共同知识假设：一个普遍已知的时钟，以及与其他遵循相同共识规则的系统共享数据的实际手段。后两者都是在 ***YP*** 中默认做出的假设。

4.1 区块

为了帮助我们理解和定义本协议，我们将尽可能多的术语按其功能组件进行划分。我们从区块 $\mathbf{B}$ 开始，它可以重新表述为头部 $\mathbf{H}$ 以及一些系统外部输入的数据，因此这些数据被称为外在（extrinsic） 数据 $\mathbf{E}$ 。

$$\mathbf{B} \equiv (\mathbf{H}, \mathbf{E}) \tag{4.2}$$

$$\mathbf{E} \equiv (\mathbf{E}_T, \mathbf{E}_D, \mathbf{E}_P, \mathbf{E}_A, \mathbf{E}_G) \tag{4.3}$$

区块头是一组元数据，主要涉及指向区块链前置块的加密引用，以及当前状态转移的操作数和结果。作为一个“先验”不可变的已知量，区块头在区块转移的所有功能组件中都被假定为是可用的。此外，外部数据被分割成了多个部分：

- 票证 (tickets) ： 用于管理区块出块授权中验证者选择机制的票据。该组件记作 $\mathbf{E}_T$ 。
- 预映像 (preimages)：指当前请求使工作负载能够按需获取的静态数据。该组件记作 $\mathbf{E}_P$ 。
- 报告 (reports) ： 新完成工作负载的报告，其准确性由特定验证者保证。该组件记 $\mathbf{E}_G$ 。
- 可用性 (availability) ： 指各验证者对其正确接收并在本地存储的工作负载输入数据所做的保证。该组件记作 $\mathbf{E}_A$ 。
- 争议 (disputes) ： 指与验证者间关于报告有效性争议相关的信息。该组件记作 $\mathbf{E}_D$ 。

4.2 状态

我们的状态可以在逻辑上被划分为几个基本且相互独立的段。这种划分方法不仅有助于减少协议描述中的视觉复杂性，还为那些能够并行处理的计算元素提供了形式化的处理手段。因此，我们声明，一个完整的状态 σ 与其经过划分后形成的各个部分所组成的元组之间，存在着一种等价关系。

$$\sigma \equiv (\alpha, \beta, \gamma, \delta, \eta, \iota, \kappa, \lambda, \rho, \tau, \varphi, \chi, \psi, \pi, \vartheta, \xi) \tag{4.4}$$

总而言之，在 JAM 系统中，δ 代表与“服务”相关的状态部分，其功能类似于 ***YP*** 中所描述的（智能合约）账户，后者在以太坊黄皮书（***YP***'s Ethereum）中被定义为唯一的状态实体。至于那些拥有特权地位的服务身份，则在 χ 中被追踪和记录。

验证者（Validators）群体构成了 JAM 链的基石，他们是一群拥有特殊权限的经济行为者，致力于构建与维护 JAM 链的稳定运行。这些验证者在 κ 系统中被赋予标识，其详细信息则在 λ 系统中存档备查，同时他们还需按照 ι 系统中的规则进行排队等候处理。除此之外，所有与验证者密钥相关的状态信息均被安全地保存在 γ 系统中。值得注意的是，JAM 链中的验证者概念与 ***YP*** 中的工作量证明（proof-of-work）机制有所不同。***YP*** 的工作量证明机制几乎不依赖于任何状态信息，且其验证者集合并未被明确列出，而是由那些具备足够计算能力能在 SHA2-256 加密哈希函数中找到特定哈希碰撞的个体自然构成。另外，JAM 链上的熵池信息被妥善保存在 η 系统中。

[7] 实际上来说，区块链有时会假设有一部分参与者的行为是诚实的，既无法证明其错误，也没有经济上的反向激励使其行为不当。虽然这一假设可能是合理的，但它必须独立于状态转换规则之外来阐述。

我们持续监控每个核心的两个关键指标：首先是 α，它代表了在核心上完成的工作必须达成的授权标准。同时，我们还维护了一个队列 φ，用于管理那些已经满足 α 标准、准备进行报告的工作。另一方面，我们还关注 ρ，即每个核心当前负责生成的报告。这些报告所涉及的工作包的可用性，需要经过绝大多数验证者的严格确认。

最后，β 负责追踪最新区块的详细信息，而 τ 则记录时段索引。同时，ϑ 用于追踪准备累积的工作报告，ξ 则记录最近累积的工作包。此外，ψ 负责记录判决信息，而 π 则用于追踪验证者的统计信息。

4.2.1 状态转换依赖图

与在 ***YP*** 中的做法相似，我们将 Υ 定义为依据先前的状态以及数据块，来阐述所有后续状态项的内在逻辑关系。为了促进计算过程的并行化处理，我们尽可能简化了依赖图的层级结构。现在，我们来详细阐述整体依赖图的规范。

(4.5) $$\tau' \prec \mathbf{H}$$

(4.6) $$\beta^\dagger \prec (\mathbf{H}, \beta)$$

(4.7) $$\gamma' \prec (\mathbf{H}, \tau, \mathbf{E}_T, \gamma, \iota, \eta', \kappa', \psi')$$

(4.8) $$\eta' \prec (\mathbf{H}, \tau, \eta)$$

(4.9) $$\kappa' \prec (\mathbf{H}, \tau, \kappa, \gamma)$$

(4.10) $$\lambda' \prec (\mathbf{H}, \tau, \lambda, \kappa)$$

(4.11) $$\psi' \prec (\mathbf{E}_D, \psi)$$

(4.12) $$\rho^\dagger \prec (\mathbf{E}_D, \rho)$$

(4.13) $$\rho^\ddagger \prec (\mathbf{E}_A, \rho^\dagger)$$

(4.14) $$\rho' \prec (\mathbf{E}_G, \rho^\ddagger, \kappa, \tau')$$

(4.15) $$\boldsymbol{W}^* \prec (\mathbf{E}_A, \rho')$$

(4.16) $$(\vartheta', \xi', \delta^\ddagger, \chi', \iota', \varphi', \mathbf{C}) \prec (\mathbf{W}^*, \vartheta, \xi, \delta, \chi, \iota, \varphi, \tau, \tau')$$

(4.17) $$\beta' \prec (\mathbf{H}, \mathbf{E}_G, \beta^\dagger, \mathbf{C})$$

(4.18) $$\delta' \prec (\mathbf{E}_P, \delta^\ddagger, \tau')$$

(4.19) $$\alpha' \prec (\mathbf{H}, \mathbf{E}_G, \varphi', \alpha)$$

(4.20) $$\pi' \prec (\mathbf{E}_G, \mathbf{E}_P, \mathbf{E}_A, \mathbf{E}_T, \tau, \kappa', \pi, \mathbf{H})$$

唯一观察到的同步依赖是通过带有匕首（†）符号的中间组件来实现的，这些组件在方程 4.6、4.18 和 4.13 中有明确的定义。特别是，方程 4.18 和 4.13 在依赖图中指出了合并与连接的关键点，这表明外部可用性可能已被充分考虑，并且在进行状态更新之前，已经将原像查找的外部操作整合进来了。

4.3 哪段历史?

区块链是由一系列区块组成的序列，每个区块通过包含其头部哈希的方式加密引用前一个区块，一直追溯到第一个区块，该区块引用了创世头部。我们已经假定对这个创世头部 $\mathbf{H}^0$ 及其所代表的状态 σ^0 ，各方已达成了共识。

通过设定一个确定性函数，该函数能够针对任意（有效）的前置状态与区块组合，推导出唯一的一个后置状态，我们便能为任意给定的区块指定一个独一无二的规范状态。在常规语境下，我们将拥有最多祖先区块的那个区块称为“头区块”，而它所对应的状态则被称作“头状态”。

两个区块可能同时被认为是有效的，并且都引用了同一个前置区块，这种情况我们称之为分叉。这意味着系统中可能会并存两个不同的头区块，每个头区块都对应着各自的状态。尽管我们尚未找到完全排除这种分叉情况的方法，但为了维护系统的实用性和稳定性，我们必须竭尽全力将其发生的可能性降至最低。因此，我们致力于确保：

(1) 两个头区块的形成在一般情况下是不太可能的。

(2) 当两个头区块形成时，它们能迅速合并成一个单一的头区块。

(3) 能够确定一个距离头区块不相差太远的区块，我们有极高的信心认为这个区块将永久成为区块链历史的一部分。当一个区块被如此确定时，我们称之为“最终确定”（finalized），这一属性自然也延伸到其所有祖先区块。

这些目标是通过结合两种共识机制来实现的：一是 Safrole，它负责区块链（不一定是无分叉的）的扩展管理；二是 Grandpa，它负责将特定的扩展最终确定为规范历史。因此，Safrole 机制实现了第 1 个目标，Grandpa 机制实现了第 3 个目标，而两者对于达成第 2 个目标都起着至关重要的作用。关于这些协议部分的详细描述，我们将在第 6 节和第 19 节中分别展开。

虽然 Safrole 在很大程度上通过密码学手段、经济激励机制以及共同时间认知（下文将详细阐述）来限制分叉的产生，但在某些特定情境下，我们可能会故意选择进行分叉，比如当我们得知某个链扩展必须被撤销时。虽然在正常操作环境下，这种情况理应不会发生，但我们仍无法完全排除恶意行为或节点故障的可能性。因此，我们明确定义，任何包含被“其他任何”区块状态标记为无效数据的区块的扩展，均被视为非法扩展（关于如何实现这一点，请参见第 10 节）。我们进一步向 Grandpa 机制提出要求，不得将包含此类区块的任何扩展最终确定为规范历史。更多相关信息，请参阅第 19 节。

4.4 时间

我们假设存在一个预先达成共识的时间机制，该机制专门用于区块的生成与接收。虽然这并不是波卡最初的基本假设，但实际上存在既实用又可靠的解决方案，例如 NTP（网络时间协议）以及相关的网络机制。我们仅利用这一假设来做一件事：如果某个区块的时隙标记为未来的时间点，那么我们会暂时认为该区块是无效的。关于这一点的具体说明，将在第 6 节中详细展开。

我们正式地将时间定义为自“混乱共同纪元”（JAM Common Era）起始点——即2025年1月1日中午 12:00 UTC 以来所流逝的秒数[8]。选择 UTC 时间中午12点作为基准，旨在确保在全球各大主要时区，从共同纪元起始时刻算起的任何整24小时的倍数时刻，对应的日期都是一致的。形式上，我们使用 T 来表示这一时间值。

4.5 最佳区块

当面对多个有效的区块时，我们需要确定哪一个区块能够被誉为“最佳”区块，即这个区块在未来将会稳定地作为所有 JAM 链中的最新区块出现。为了达成这一目标，最为直接且稳妥的做法是参考 GRANDPA 终局性机制，该机制能够为我们提供一个确信无疑的区块，它将持续作为所有未来链头的祖先区块。

[8] Unix 纪元后 1,735,689,600 秒

然而，为了减小所选区块最终未能纳入规范链的风险，GRANDPA 通常会选定一个比最新产生的区块稍早一些的区块作为结果（根据现有部署的实际情况，在常规运行状态下，这通常意味着选择过去1到2个区块中的某一个）。当然，在某些场景下，我们可能会希望减少延迟，即便这样做会增加所选区块最终不被纳入未来规范链的风险。例如，当我们处于能够生成新区块的位置时，就需要决定其父区块是哪一个；或者，当我们需要推测最新状态以便为那些依赖于 JAM 状态的下游应用提供信息时，也会面临这样的选择。在这些特定场景下，我们将“最佳区块”定义为最佳链的头部，而关于最佳链的具体定义，请参见第 19 节。

4.6 经济学

本文阐述了一个融合密码学、经济学及博弈论要素的加密经济系统，旨在实现数字服务的自主管理。为了规范并驱动经济激励机制，我们引入了一种系统内置代币（在本文中简称为“代币”），作为该系统运作的核心要素。

代币的价值通常被称为“余额”，这种价值被认为是余额集合 $\mathbb{N}_B$ 中的一个成员，$\mathbb{N}_B$ 正好等价于小于 2^{64} 的自然数集合（即编程术语中的 64 位无符号整数）。形式上：

$$\mathbb{N}_B \equiv \mathbb{N}_{2^{64}} \tag{4.21}$$

尽管对本文不重要，但我们假设存在一个标准命名单位，用于表示 10^9 个代币。这与以太坊（使用 10^{18} 作为单位）、波卡（使用 10^{10} 作为单位）以及波卡的实验性分支 Kusama（使用 10^{12} 作为单位）都不同。

余额被限制为小于 2^{64} 的事实意味着，在 JAM 中，代币的总数永远不会超过大约 18×10^9 个（每个代币可细分为 10^{-9} 的部分）。我们预计，实际发行的代币总数将远小于这个数量。

我们进一步假定，以代币表示的一些固定价格是已知的。然而，具体的数值将在后续工作中确定：

- B_I：映射中单个项目所隐含的额外最低余额；
- B_L：映射中单个字节数据所隐含的额外最低余额；
- B_S：服务所隐含的最低余额。

4.7 虚拟机和 Gas

在当下的研究中，我们设定了一个名为 “PolkaVM”（PVM）的概念。这款虚拟机是以 RISC-V 指令集架构为蓝本构建的，特别是采纳了 RV64EM 这一版本，它为我们将无许可逻辑融入状态转移函数中奠定了坚实的基础。

PVM（PolkaVM）与黄皮书中阐述的以太坊虚拟机（EVM）有所相似，但设计更为简洁：它省略了复杂的加密操作指令及与环境交互的指令。总体而言，PVM 更为中立，因为它是基于现有的通用 RISC-V 设计进行定制化修改以满足我们的特定需求。这一做法使我们能够充分利用现有的出色工具支持，因为 PVM 基本上与 RISC-V 兼容，这意味着我们可以使用诸如 LLVM编译器工具包，以及 Rust 和 C++ 等语言的支持。此外，RISC-V 与PVM 共享指令集的简洁性，以及一致的寄存器大小（64位）、寄存器数量（13个）和字节序（小端序），这些特性使得它们在常见硬件架构上构建高效的重新编译器变得尤为合适。

关于 PVM（PolkaVM）的完整定义，请参见附录A。但为了便于读者理解，以下是对基本调用函数 Ψ 的简要概述。该函数负责计算一个 PVM 实例在执行过程中的最终状态，该实例由特定的寄存器组（包含 13 个寄存器，标记为 $[\![\mathbb{N}_R]\!]_{13}$）和内

存（RAM）（$\mathbb{M}$）初始化，并且在执行时受到一个最大 gas 量（$\mathbb{N}_G$）的限制。这里的 gas 量大致与执行的计算步骤所需时间成正比。

$$\Psi: \begin{pmatrix} \mathbb{Y}, & \mathbb{N}_R, & \mathbb{N}_G, \\ \left[\mathbb{N}_R\right]_{13}, & \mathbb{M} \end{pmatrix} \to \begin{pmatrix} \{\blacksquare, \lightning, \infty\} \cup \{\llcorner, \hbar\} \times \mathbb{N}_R \\ \mathbb{N}_R, \quad \mathbb{Z}_G, \quad \left[\mathbb{N}_R\right]_{13}, \quad \mathbb{M} \end{pmatrix} \tag{4.22}$$

我们将那些与时间成比例的计算步骤称为“gas”（类似于 ***YP*** 中的概念），并将其设定为一个 64 位的数值。我们可以使用 $\mathbb{N}_G$ 或 $\mathbb{Z}_G$ 来界定它，$\mathbb{N}_G$ 作为先验参数，因为它已知为正数，$\mathbb{Z}_G$ 作为结果呈现，其中若其值为负，则意味着所尝试执行的操作已超出了预设的 gas 限制。在 PVM 的运行环境中，我们通常用 $\varrho \in \mathbb{N}_G$ 来表示前的 gas。

$$\mathbb{Z}_G \equiv \mathbb{Z}_{-2^{63}\ldots 2^{63}}, \quad \mathbb{N}_G \equiv \mathbb{N}_{2^{64}}, \quad \mathbb{N}_R \equiv \mathbb{N}_{2^{64}} \tag{4.23}$$

确保在计算函数时，所花费的时间量最大计算时间大致与 $\Psi(\ldots, \varrho, \ldots)$ 的值成正比，而不受其他操作数的影响，这是一个相当重要的实现细节。

PVM 是一种非常简单的 RISC“寄存器机器”，配备了 13 个寄存器，每个寄存器都是 64 位的数值，记作 $\mathbb{N}_R$，即小于 2^{64} 的自然数[9]。在 PVM 的运行环境中，通常用 $\omega \in \left[\mathbb{N}_R\right]_{13}$ 来表示这些寄存器。

$$\mathbb{M} \equiv \left(\mathbf{V} \in \mathbb{Y}_{2^{32}}, \mathbf{A} \in [\![\{\mathrm{W}, \mathrm{R}, \emptyset\}]\!]_p\right), p = \frac{2^{32}}{\mathrm{Z}_p} \tag{4.24}$$

$$\mathrm{Z}_p = 2^{12} \tag{4.25}$$

PVM 构想了一个基础的分页式随机存取存储器（RAM），其地址为 32 位，能够定位到分布在各个页面中的字节，这些页面的标准大小为 Z_p= 4096 字节。RAM 中的每个页面被设定为三种状态：不可更改（意味着内容固定）、可更改（允许读写操作）或无法访问（可能处于未映射或受保护状态）。RAM 的定义 $\mathbb{M}$ 涵盖了两个核心部分：一部分是存储数据的值 $\mathbf{V}$，另一部分是控制访问的权限 $\mathbf{A}$。如果在执行索引操作时未明确指定访问的是哪个部分，那么通常会默认访问的是值 $\mathbf{V}$。在 PVM 的运行环境中，我们通常使用符号 $\mu \in \mathbb{M}$ 来表示当前的 RAM 状态。

$$\mathbb{V}_\mu \equiv \{i | \mu_{\mathbf{A}}[i/\mathrm{z}_p] \neq \emptyset\} \tag{4.26}$$

$$\mathbb{V}^*_\mu \equiv \{i | \mu_{\mathbf{A}}[i/\mathrm{z}_p] = \mathbf{W}\} \tag{4.27}$$

我们为 RAM μ 定义了两组索引：$\mathbb{V}_\mu$ 是可从中读取的索引集合；而 $\mathbb{V}^*_\mu$ 是可写入其中的索引集合。

PVM 的调用结果，元组中的第一个项目是退出原因。其可能为：

- 由显式停止指令引起的正常程序终止，$\blacksquare$。
- 由某些异常情况引起的非正常程序终止，$\lightning$。
- Gas 耗尽，∞。
- 页面错误（尝试访问 RAM 中不可访问的某个地址），$\llcorner$。此处包含错误页面的地址。
- 尝试进行主机调用推进，$\hbar$。这允许在常规 PVM 之外，根据上下文依赖的状态机进行推进和集成。

完整的定义见附录 A。

[9] 这比 RISC-V 的 16 个少了 3 个，但编译器输出的程序代码实际使用的是 13 个寄存器，因为其中两个被保留给操作系统使用，而第三个固定为零。

4.8 纪元与时隙

不同于依赖工作量证明共识机制的 *YP* 以太坊，JAM 系统引入了一种权威证明共识机制。在这一机制中，被授权的验证者通过一组公钥进行身份标识，并且他们的资格由 JAM 托管系统内的一个特定“质押”流程来确定。需要注意的是，本工作并未涉及质押机制的具体细节；相反，我们提供了一个 API 接口，以便在必要时更新这些公钥。我们预设，任何质押机制所需的逻辑将在后续需要时被引入，并会通过这一 API 接口进行操作。

Safrole 机制将创世后的时间划分为固定长度的周期，每个周期分为 $\mathrm{E} = 600$个时隙，每个时隙长度均匀为 $\mathrm{P} = 6$秒，因此一个周期的总时长为 $\mathrm{E} \cdot \mathrm{P} = 3600$秒，即一小时。

Safrole 机制采用了一个6秒的时隙周期作为 JAM 区块链中区块之间的最小时间间隔。我们的核心目标是通过这一机制来大幅降低因时隙内的竞争——即在同一6秒时段内可能同时生成两个有效区块的情况——以及跨时隙的竞争——即在不同时隙但拥有相同父区块的情况下也可能出现两个有效区块的现象，从而有效控制区块链的分叉情况。

在正式标识时隙索引时，我们使用小于 2^{32} 的自然数（在计算术语中，即 32 位无符号整数）来表示从 JAM 共同纪元起算的6秒时隙数量。为在此上下文中使用，我们引入集合 $\mathbb{N}_T$ ：

$$\mathbb{N}_T \equiv \mathbb{N}_{2^{32}} \tag{4.28}$$

这意味着所提议的协议的寿命将延续至 2840年8月中旬，按照人类目前的发展轨迹来看，这应该是足够长的时间。

4.9 核心模型与服务

在 *YP* 中，当我们构建描述所有网络参与者达成共识的状态机时，我们假设：所有维护并致力于扩展完整网络状态的机器（或至少意图如此的设备）都会参与并执行所有计算任务。这种“全员参与，全面计算”的模式可称为链上共识模型。然而，该模型首可扩展性局限。因为整个网络在共识过程中所能处理的逻辑复杂度，受限于任何单一节点在任意特定时间段内所能独立处理的逻辑量。

4.9.1 核心共识

在当前工作中，我们通过引入第二种计算模型实现了工作的可扩展性，我们称之为“核心共识模型”。在该模型中，正常情况下，只有网络的一个子集负责实际执行任何给定的计算，并确保它所依赖的任何输入数据对其他节点可用。通过这一策略，并结合网络验证节点间存在的计算并行性假设，我们成功实现了共识过程中计算量的增长与网络规模成正比，而非受限于任何单一节点的计算能力。在本项工作中，我们预计网络核心所能完成的计算量将远超单台机器以全速运行虚拟机时的处理能力，具体而言，可望达到 300 倍以上的提升。

由于核心共识并非由网络上的所有节点进行评估或验证，我们必须找到其他方式来足够确信计算结果是正确的，并且用于确定此结果的任何数据在一段实际时间内都是可获取的。我们通过一个包含三个阶段（称为“保证 guaranteeing 、确保 assuring 、审计 auditing ，以及可能的裁决 judging”）的加密经济游戏来实现这一点。这三个阶段分别给某些提议的计算无效性附加了实质性的经济成本；然后确保了对计算输入将在一段时间内可用的足够信心；最后，确保了对计算有效性（从而强制执行第一个保证）将由我们期望诚实的某方进行检查的足够信心。

核心中执行的操作都必须具备可由任何已同步至最终确定链部分的节点重现的能力。因此，核心中的执行被设计为尽可能无状态。执行它所需的要求仅包括服务的精炼代码、授权者的代码以及其在执行过程中进行的任何原像查找。当工作报告在链

上呈现时，会标识出一个特定的区块，称为“查找锚点”。正确的行为要求该区块必须在已最终确定的链中，并且相对较新，这两个属性都可以被证明，因此适用于共识协议中使用。

我们将在后面的相关章节中详细描述这一流程。

4.9.2 关于服务和账户

在以太坊 ***YP*** 中，存在两种类型的账户：“合约账户”（其行为基于账户关联的代码和状态确定性地定义）和“简单账户”（作为数据进入世界状态的网关，由某个私钥的控制者操控）。在 JAM 中，所有账户都是“服务账户”。这些服务账户与以太坊的合约账户有相似之处，它们有相关的余额、一定的代码以及状态信息。然而，一个显著的区别在于，服务账户并不依赖于私钥来控制，因此它们无需使用随机数（nonce）来防止重放攻击。

于是问题来了：如何将外部数据输入到 JAM 的世界状态中？进一步说，如果不通过扣除签名交易者的账户余额，那么整体支付是如何进行的？第一个问题的答案在于，我们的服务定义实际上包含了“多个”代码入口点，一个用于“精炼”，另一个用于“累积”。前者作为一种高性能的无状态处理器，能够接受任意输入数据，并能将这些数据提炼为少量输出数据。后者则更具状态性，提供了对某些链上功能的访问，比如余额转移和调用其他服务中的代码执行。由于“累积”入口点具有状态性，因此它的运作方式更接近于以太坊中的合约账户代码。

要了解 JAM 如何拆分其服务代码，就得先理解 JAM 在通用性和可扩展性方面的核心理念。所有外部输入数据都会传入某个服务的精炼（Refinement）代码，该部分并不在链上执行，而是在链下的核心（in-core）环境中执行。相比之下，累积（Accumulator）代码与以太坊智能合约类似，受到与以太坊合约账户相同的可扩展性限制；而精炼代码则在链下运行，不受这些限制。因此， JAM 服务可以在输入数据规模和计算复杂度上大幅度扩展。

虽然精炼和累积发生在不同性质的共识环境中，但它们都是由同一验证者集合的成员执行的。 JAM 协议通过其奖励和惩罚机制，确保在核心中执行的代码具有与“链上”执行代码相当水平的加密经济安全性，两者之间的主要区别仅在于可扩展性与同步性。

在支付管理方面，JAM 采纳了一种新颖的抽象机制，该机制以波卡为基础，构建了 Agile 核心时间框架。与以太坊的交易模式有所不同，以太坊将账户授权与购买区块空间的机制在一定程度上进行了融合，两者共同依赖于加密签名来确认唯一的“交易者”账户身份。然而，在 JAM 体系中，这两大机制是相互独立的，并且摒弃了“交易者”这一传统概念。

以太坊使用 gas 模型来购买和衡量区块空间，而 JAM 则引入了核心时间 (Coretime) 的概念，核心时间是预先购买并分配给授权代理的。核心时间在某种程度上类似于 gas ，因为它是使用 JAM 时不可或缺的基础资源。至于如何获取核心时间，本文暂不探讨，预计这一管理过程将由一个专设的系统平行链负责，该链配备多个核心，专门用于运行此类系统服务。权代理的设立，使得外部参与者能够向服务提供输入，而无需像以太坊交易签名那样公开其身份。有关授权代理的详尽阐述，请参阅第 8 节。

5. 区块头

我们首先需要根据其组成部分来定义头部。头部包括一个父哈希和先前状态根（ $\mathbf{H}_p$ 和 $\mathbf{H}_r$ ）、一个外部数据哈希 $\mathbf{H}_x$ 、一个时隙索引 $\mathbf{H}_t$ 、纪元、获胜票证和违规标记 $\mathbf{H}_e$ 、$\mathbf{H}_w$ 和 $\mathbf{H}_o$ 、一个 Bandersnatch 区块作者索引 $\mathbf{H}_i$ 以及两个 Bandersnatch 签名；一个是产生熵的 VRF 签名 $\mathbf{H}_v$ ，另一个是区块密封 $\mathbf{H}_s$ 。头部可以序列化为一个八位字节序列，使用 $\mathcal{E}$ 表示包含后者（密封组件）的序列，使用 $\mathcal{E}_U$ 表示不包含后者（密封组件）的序列。形式上：

(5.1) $$\mathbf{H} \equiv (\mathbf{H}_p, \mathbf{H}_r, \mathbf{H}_x, \mathbf{H}_t, \mathbf{H}_e, \mathbf{H}_w, \mathbf{H}_o, \mathbf{H}_i, \mathbf{H}_v, \mathbf{H}_s)$$

区块链是由一系列区块组成的序列，每个区块通过包含从父区块头部派生出的哈希值，以加密方式引用某个先前的区块，一直追溯到某个链的起始区块，即首个区块，该区块则引用了被称为创世头部的初始数据。我们已经默认对这个创世头部 $\mathbf{H}^0$ 及其所代表的初始状态 σ^0 达成了共识。

除创世区块的头部外，所有区块头部 $\mathbf{H}$ 都有一个相关联的父区块头部，其哈希值 $\mathbf{H}_p$ 。我们用 $\mathbf{H}^- = P(\mathbf{H})$ 来表示父头部：

(5.2) $$\mathbf{H}_p \in \mathbb{H}, \quad \mathbf{H}_p \equiv \mathcal{H}(\mathcal{E}(P(\mathbf{H})))$$

因此，P 被定义为从一个区块头部到其父区块头部的映射。有了 P ，我们就能够定义祖先区块头部集合 A：

(5.3) $$h \in \mathbf{A} \Leftrightarrow h = \mathbf{H} \vee (\exists i \in \mathbf{A}: h = P(i))$$

我们仅要求实现方案存储任何区块 $\mathbf{B}$ 希望验证的之前的 $\mathrm{L} = 24$ 小时内创建的祖先区块的头部。

外部数据哈希是对区块外部数据的默克尔承诺（Merkle commitment），特别注意要允许单独证明报告被包含的可能性。给定任何区块 $\mathbf{B} = (\mathbf{H}, \mathbf{E})$ ，那么形式上：

(5.4) $$\mathbf{H}_x \in \mathbb{H}, \quad \mathbf{H}_x \equiv \mathcal{H}(\mathcal{E}(\mathcal{H}^{\#}(\mathbf{a})))$$

(5.5) $$\text{where } \mathbf{a} = [\mathcal{E}_T(\mathbf{E}_T), \mathcal{E}_P(\mathbf{E}_P), \mathbf{g}, \mathcal{E}_A(\mathbf{E}_A), \mathcal{E}_D(\mathbf{E}_D)]$$

(5.6) $$\text{and } \mathbf{g} = \mathcal{E}(\updownarrow [\mathcal{E}(\mathcal{H}(w), \mathcal{E}_4(t), \updownarrow a) | (w, t, a) \in \boldsymbol{E}_G])$$

一个区块只有在时间槽索引 $\mathbf{H}_t$ 已成为过去时，才能被视为有效。它总是严格大于其父区块的时间槽索引。形式上：

(5.7) $$\mathbf{H}_t \in \mathbb{N}_T, \quad P(\mathbf{H})_t < \mathbf{H}_t \wedge \mathbf{H}_t \cdot \mathrm{P} \leq \mathcal{T}$$

根据这一规则被视为无效的区块，随着时间 $\mathcal{T}$ 的推进可能会变得有效。

父状态根 $\mathbf{H}_r$ ，作为由“先前”状态数据构建的默克尔树的顶端，代表了该默克尔树结构的根，依据定义，它同时也标志着父区块的后续状态。相比之下，波卡和以太坊 ***YP*** 采取了不同的策略，它们将“后”状态的默克尔根直接纳入区块头部。我们之所以选择当前的方式，主要是为了优化区块计算的流水线流程，特别是为了提升默克尔化（Merklization）的处理效率。

(5.8) $$\mathbf{H}_r \in \mathbb{H}, \quad \mathbf{H}_r \equiv \mathcal{M}_\sigma(\sigma)$$

我们设定了一个状态默克尔化函数 $\mathcal{M}_\sigma$，它能够将任意状态 σ 转换为一个 32 字节的承诺值有关这两个函数的完整定义，请参阅附录 D。

每个区块都配备了一个与之相关联的公钥，用于识别该区块的创建者。我们将此公钥视作在后续验证者集合 κ' 中的一个索引位置。同时，我们用 $\mathbf{H}_a$ 来表示区块作者的 “Bandersnatch” 密钥，但重要的是要理解，这仅仅是一个等价的表示方法，它并不会作为区块头部信息的一部分进行序列化存储。

(5.9) $$\mathbf{H}_i \in \mathbb{N}_V, \quad \mathbf{H}_\mathrm{a} \equiv \kappa'[\mathbf{H}_\mathrm{i}]_b$$

5.1 标记

如果非 $\emptyset$，则纪元标记会指定与下一个纪元相关的密钥和熵值，这一安排是为了应对选票竞争未能充分完成的极端情况（尽管这种情况发生的可能性极低）。同样地，如果获胜选票标记非 $\emptyset$，它会为下一个纪元提供一系列 600 个时段密封的“选票”（见下一节）。最后，违规者标记则用于记录新发现的、存在不当行为的验证者的 Ed25519 密钥序列，关于这一点的详细解释将在第 10 节中给出。形式上：

$$\mathbf{H}_e \in (\mathbb{H}, \mathbb{H}, \llbracket \mathbb{H}_{B,} \mathbb{H}_{E,} \rrbracket_V)?, \quad \mathbf{H}_w \in [\mathbb{C}]_E?, \quad \mathbf{H}_o \in \llbracket \mathbb{H}_E \rrbracket \tag{5.10}$$

这些术语在第 6.6 节和第 10 节中有完整定义。

6. 区块生成与链增长

如前文所述，JAM 是基于一种混合共识机制构建的，其特性与波卡所采用的 BABE/GRANDPA 混合机制有着相似之处。JAM 的区块生成部分被称为 Safrole，这一命名灵感来源于简化并创新后的 Sassafras 生成机制。相较于 *YP* 中所述的 Nakamoto 共识，Safrole 是一个更为复杂且具备状态性的系统。

区块生成共识机制的核心目标在于控制新区块的产出速度，并在最理想的状态下，排除“分叉”的可能性，也就是防止出现多个拥有相同祖先区块数量的并行区块链分支。

为了达到这一目的，Safrole 机制设定，在任何给定的六秒时段内，只有预定义验证者集合中的一个密钥持有者能够成为区块的潜在创作者。进一步地，在常规运作状态下，未来时段的密钥持有者身份将享有极高的匿名性。作为 Safrole 运行的一个附加益处，它能够生成一个高质量的熵池。这个熵池不仅可供协议的其他组成部分使用，而且对于在该协议上运行的所有服务也都是可获取的。

由于 Safrole 的角色范围被严格界定，其核心状态 γ 与协议的其他组成部分保持独立。它主要通过以下方式与协议的其他部分进行交互：通过 ι（预期的验证者密钥集合）和 κ（活动的验证者密钥集合）来识别验证者，通过 τ （表示最近区块的时段）来追踪时间进度，以及通过 η（熵累加器）与协议的其他部分进行交互。

Safrole 协议在每个纪元中会生成一个包含 E 个“密封密钥”的序列，这些密钥各自对应于整个纪元内可能产生的一个区块。每个区块头都会包含两个关键信息：时段索引 $\mathbf{H}_t$（即从 JAM 共同纪元起开始以来的六秒周期数），以及一个有效的密封签名 $\mathbf{H}_s$。这个密封签名是由与区块时段相对应的密封密钥（实际上是某个验证者的化名）签署的，它赋予了该验证者在相应时段内创建区块的特权。

为了在正常运作中生成这一密封密钥序列，同时确保不泄露它们与验证者集合之间的对应关系，我们采用了一种创新的加密构造——基于 Bandersnatch 曲线的 RingVRF（环向可验证随机函数）。Bandersnatch RingVRF 能够生成一种证明，该证明既验证了作者掌控着验证者集合中的一个密钥，又产生了一个无偏的、确定性的哈希值，这一哈希值充当了一个安全的可验证随机函数（VRF）的输出。这个既匿名又安全的随机输出被视作一张“选票”，而得分最高的验证者选票则决定了新的密封密钥，从而赋予被选中的验证者行使特权，在合适的时间点创建新的区块。

6.1 计时

在这里，τ 定义了最新区块的时段索引，我们将其过渡到区块头中定义的时段索引。

(6.1) $$\tau \in \mathbb{N}_T, \tau' \equiv \mathbf{H}_t$$

我们通过在状态中记录时段索引 τ ，来方便地辨识新的纪元，并追溯上一个区块是在哪个时段被创建的。这里，我们用 e 来标记上一个区块的纪元索引，m 则代表该纪元内上一个区块的时段相位索引。相应地，当前区块的这些值则分别用 e' 和 m' 来表示。

(6.2) $$\text{let } e\,\mathrm{R}\,m = \frac{\tau}{\mathrm{E}},\quad e'\,\mathrm{R}\,m' = \frac{\tau'}{\mathrm{E}}$$

6.2 Safrole 基本状态

我们将 γ 重新表述为若干个组成部分：

(6.3) $$\gamma \equiv (\gamma_{\mathbf{k}}, \gamma_z, \gamma_{\mathbf{s}}, \gamma_{\mathbf{a}})$$

纪元的根 γ_z 是一个 Bandersnatch 环根，它由下一个纪元中每个验证者的一个 Bandersnatch 密钥构成。该根的定义基于 $\gamma_{\mathbf{k}}$ （ $\gamma_{\mathbf{k}}$ 的具体定义将在后续章节中详细阐述）。

(6.4) $$\gamma_z \in \mathbb{Y}_{\mathcal{R}}$$

最终，$\gamma_{\mathbf{a}}$ 作为票据累积器，收纳了一系列得分最高的票据标识符，以供下一个纪元使用。而 $\gamma_{\mathbf{s}}$ 则代表了当前纪元的时段密封者序列，它可能是由 $\mathbf{E}$ 张票据组成的完整集合，或者在回退模式下，是由 $\mathbf{E}$ 个 Bandersnatch 密钥构成的序列。

(6.5) $$\gamma_{\mathbf{a}} \in \llbracket \mathbb{C} \rrbracket_{:\mathrm{E}},\quad \gamma_{\mathbf{s}} \in \llbracket \mathbb{C} \rrbracket_{\mathrm{E}} \cup \llbracket \mathbb{H}_B \rrbracket_{\mathrm{E}}$$

在此，$\mathbb{C}$ 被用来表示票据的集合，它是由一个可验证的随机票据标识 $\mathbf{y}$ 与票据的入口索引 r 共同组成的：

(6.6) $$\mathbb{C} \equiv (\mathbf{y} \in \mathbb{H}, r \in \mathbb{N}_{\mathrm{N}})$$

如我们在 6.4 节所述，Safrole 要求每个区块头 $\mathbf{H}$ 包含一个有效的密封 $\mathbf{H}_s$ ，这是利用当前纪元密封密钥系列中适当索引 m 处的公钥所生成的 Bandersnatch 签名。该密封密钥系列在状态中由 $\gamma_{\mathbf{s}}$ 表示。

6.3 密钥轮换

在 Safrole 的状态结构中，除了包含活跃验证器密钥集 κ 和暂存集 ι 之外，我们还维护了一个待处理集 $\gamma_{\mathbf{k}}$ 。活跃集 κ 用于标识当前有权创建区块并执行验证任务的节点密钥，而待处理集 $\gamma_{\mathbf{k}}$ 则在每个纪元初被重置为暂存集 ι，它代表了将在下一个纪元中活跃的密钥集合。此外，待处理集 $\gamma_{\mathbf{k}}$ 还决定了 Bandersnatch 环根，该环根是授权票据参与下一个纪元密封密钥竞争的关键。

(6.7) $$\iota \in \llbracket \mathbb{K} \rrbracket_{\mathrm{V}},\quad \gamma_{\mathbf{k}} \in \llbracket \mathbb{K} \rrbracket_{\mathrm{V}},\quad \kappa \in \llbracket \mathbb{K} \rrbracket_{\mathrm{V}},\quad \lambda \in \llbracket \mathbb{K} \rrbracket_{\mathrm{V}}$$

我们必须引入 $\mathbb{K}$ ，即验证器密钥元组集。该集合由一系列加密公钥及其对应的元数据所构成。这里的元数据是一个不透明的八位字节序列，尽管其形式简单，但扮演着至关重要的角色，它用于唯一指定验证器的实际标识符，尤其是硬件地址等关键信息。

这是一个由一组加密公钥和元数据组成的集合。元数据部分虽然呈现为不透明的八位字节序列，但其核心作用在于明确指定验证器的实际标识符，特别是硬件地址这一关键属性。

验证器密钥集本身等同于 336 个八位字节序列的集合。然而，为了提高可读性起见，我们将该序列分为四个组成部分，每个部分都易于描述。对于任意验证器密钥 κ 而言，Bandersnatch 密钥 κ_b 占据序列的前 32 个八位字节；紧接着的 32 个八位字节则代表 Ed25519 密钥表示为 κ_e ；随后的 144 个八位字节用于表示 BLS 密钥表示为 κ_{BLS} ，等同于；最后，剩余的 128 个八位字节则构成了元数据 κ_m 是。形式上，这一结构可以表述为：

(6.8) $$\mathbb{K} \equiv \mathbb{Y}_{336}$$

(6.9) $$\forall k \in \mathbb{K} : k_b \in \mathbb{H}_B \equiv k_{0\cdots+32}$$

(6.10) $$\forall k \in \mathbb{K} : k_e \in \mathbb{H}_E \equiv k_{32\cdots+32}$$

(6.11) $$\forall k \in \mathbb{K} : k_{BLS} \in \mathbb{Y}_{BLS} \equiv k_{64\cdots+144}$$

(6.12) $$\forall k \in \mathbb{K}: k_m \in \mathbb{Y}_{128} = k_{208\cdots+128}$$

在正常运作的情况下，每当一个新的纪元拉开序幕，验证器密钥会经历一次轮换过程，与此同时，该纪元的 Bandersnatch 密钥根也会相应地进行更新，新的密钥根被设定为 $\gamma_z^{'}$ ：

(6.13) $$(\gamma_{\mathbf{k}}^{'}, \kappa^{'}, \lambda^{'}, \gamma_z^{'}) \equiv \begin{cases} (\Phi(\iota), \gamma_{\mathbf{k}}, \kappa, z) & \text{if } e^{'} > e \\ (\gamma_{\mathbf{k}}, \kappa, \lambda, \gamma_z) & \text{otherwise} \end{cases}$$

$$\text{Where } z = \mathcal{O}([k_b | k \in \gamma_{\mathbf{k}}^{'}])$$

(6.14) $$\Phi(\mathbf{k}) \equiv \left[\begin{matrix} [0,0,\ldots] & \text{if } k_e \in \psi_{\mathbf{o}}^{'} \\ k & \text{otherwise} \end{matrix} \right\} \mid k \in \mathbf{k} \right]$$

值得注意的是，在纪元更迭之际，后续待处理的验证器密钥集 $\gamma_k^{'}$ 的定义会有所调整。具体而言，属于违规者 $\psi_{\mathbf{o}}^{'}$ 的传入密钥将被替换为一个特殊的空密钥，该空密钥仅由零组成。关于违规者的具体识别与判定标准，将在第 10 节中做详细阐述。

6.4 密封与熵累积

区块头必须包含一个有效的密封签名以及一个有效的 VRF（可验证随机函数）输出。这两个签名都依赖于当前时隙的密封密钥来生成。其中，密封签名的消息数据是区块头（已省略密封组件 $\mathbf{H}_s$ ）的序列化形式；而 VRF 输出则作为抗偏差的熵源，其消息内容在签名前必须已经确定。为此，我们采用由密封签名对应的 VRF 所产生的熵。形式上：

设 $\text{let } i = \gamma_{\mathbf{s}}^{'}[\mathbf{H}_t]^{\circlearrowleft}$

(6.15) $$\gamma_s^{'} \in [\![\mathbb{C}]\!] \Rightarrow \begin{cases} i_{\mathbf{y}} = \mathcal{Y}(\mathbf{H}_s), \\ \mathbf{H}_s \in \mathbb{F}_{\mathbf{H}_a}^{\mathcal{E}_U(\mathbf{H})} \langle \mathrm{X}_T \sim \eta_{3\#}^{'} \, i_r \rangle \\ \mathbf{T} = 1 \end{cases}$$

(6.16) $$\gamma_{\mathbf{s}}^{'} \in [\![\mathbb{H}_B]\!] \Rightarrow \begin{cases} i = \mathbf{H}_a, \\ \mathbf{H}_s \in \mathbb{F}_{\mathbf{H}_a}^{\mathcal{E}_U(\mathbf{H})} \langle \mathrm{X}_F \sim \eta_3^{'} \rangle \\ \mathbf{T} = 0 \end{cases}$$

(6.17) $$\mathbf{H}_v \in \mathbb{F}_{\mathbf{H}_a}^{[\,]} \langle \mathrm{X}_E \sim \mathcal{Y}(\mathbf{H}_s) \rangle$$

(6.18) $$X_E = \$jam\_entropy$$

(6.19) $$X_F = \$jam\_fallback\_seal$$

(6.20) $$X_T = \$jam\_ticket\_seal$$

使用票据进行密封具有更高的安全性，这一特性在我们决定扩展区块链的候选区块时显得尤为重要，详细内容见第 19 节。为了明确区分，我们使用布尔标记 **T** 来注明该区块是在常规安全性下密封的。这一定义主要是为了便于后续规范的说明和理解。简而言之，**T** 标记的存在帮助我们清晰地识别区块的密封安全性级别。

除了当前的熵累加器 η_0 之外，我们还额外保留了最近结束的三个纪元时点的熵累加器的历史值，分别是 η_1、η_2 和 η_3。其中，倒数第二个最旧的值 η_2 在确保未来熵的无偏性方面发挥着关键作用（参见方程 6.29），并且还为备用密封密钥的生成函数提供了必要的随机性种子（参见方程 6.24）。而最旧的值则用于在验证上述密封过程时，重新生成这种随机性，以确保验证的有效性（参见方程 6.16 和 6.15）。

(6.21) $$\eta \in \llbracket \mathbb{H} \rrbracket_4$$

η_0 表征了随机数累加器的当前状态，每当新区块生成时，都会将基于不可偏置输入所得的可验证随机函数（VRF）的可靠随机输出融入此状态之中。同时，为了追溯历史状态，我们分别保留了 η_1、η_2 和 η_3，它们依次记录了最近三个纪元结束时累加器的状态快照。

(6.22) $$\eta_0' \equiv \mathcal{H}(\eta_0 \frown \mathcal{Y}(\mathbf{H}_v))$$

在纪元转换时（以条件 $e' > e$ 为标识），我们将累加器的值轮转到历史记录 η_1、η_2 和 η_3 中：

(6.23) $$(\eta_1', \eta_2', \eta_3') \equiv \begin{cases} (\eta_0, \eta_1, \eta_2) & \text{if } e' > e \\ (\eta_1, \eta_2, \eta_3) & \text{otherwise} \end{cases}$$

6.5 时隙密钥序列

后续时隙的密钥序列 $\gamma_{\mathbf{s}}'$ 的确定，需要依据区块具体状况，可归结为以下三种情形：若当前区块并非某一纪元的首个区块，则它将维持与前一密钥序列 $\gamma_{\mathbf{s}}$ 相同。若当前区块通过纪元索引预示着新纪元的开启，并且其前一个区块的时隙恰好处于前一纪元的尾声阶段，那么它将采纳先前票据累加器 $\gamma_{\mathbf{a}}$ 的值。除此之外的其他情况，它取备用密钥序列的对应值。形式上：

(6.24) $$\gamma_{\mathbf{s}}' \equiv \begin{cases} Z(\gamma_{\mathbf{a}}) & \text{if } e' = e+1 \wedge m \geq \mathrm{Y} \wedge |\gamma_{\mathbf{a}}| = \mathrm{E} \\ \gamma_{\mathbf{s}} & \text{if } e' = e \\ F(\eta_2', \kappa') & \text{otherwise} \end{cases}$$

在这里，我们使用 Z 作为从外到内的序列函数，其定义如下：

(6.25) $$Z: \begin{cases} \llbracket \mathbb{C} \rrbracket_{\mathrm{E}} \to \llbracket \mathbb{C} \rrbracket_{\mathrm{E}} \\ \mathbf{s} \mapsto [\mathbf{s}_0, \mathbf{s}_{|\mathbf{s}|-1}, \mathbf{s}_1, \mathbf{s}_{|\mathbf{s}|-2}, \ldots] \end{cases}$$

最后，F 是备用密钥序列函数，它使用在链上收集的熵 r，从验证者密钥集 $\mathbf{k}$ 中选择一个具有纪元价值的验证者 Bandersnatch 密钥（$\llbracket \mathbb{H}_{\mathrm{B}} \rrbracket_{\mathrm{E}}$）：

(6.26) $$F: \begin{cases} (\mathbb{H}, \llbracket \mathbb{K} \rrbracket) \to \llbracket \mathbb{H}_B \rrbracket_{\mathrm{E}} \\ (r, \mathbf{k}) \mapsto \left[\mathbf{k}[\mathcal{E}^{-1}(\mathcal{H}_4(r \sim \mathcal{E}_4(i)))]_b^{\circlearrowleft} \,\middle|\, i \in \mathbb{N}_{\mathrm{E}} \right] \end{cases}$$

6.6 标记

纪元标记和获胜票据标记被嵌入在头部信息中，其设计初衷在于最大限度地减少确定与任意特定纪元相关联的验证者密钥所需的数据传输量。对于那些不同步特定区块完整状态的节点而言，这些标记尤为实用，因为它们仅凭头部链即可高效地、安全地追踪验证者密钥集的变动情况。

如前所述，区块头部的纪元标记 $\mathbf{H}_e$ 要么为空，要么该区块是新纪元的第一个区块；此时区块头部会包含一个元组（Tuple），元祖中包括：当前和下一个纪元的随机数，以及一个验证者密钥对序列；该序列中包含 Bandersnatch 密钥和 Ed25519 密钥，这些密钥用于定义下个纪元 开始时的验证者集（即他们指定了那些验证者将要负责验证和出块）。形式上：

(6.27)
$$\mathbf{H}_e \equiv \begin{cases} (\eta_0, \eta_1, [\,(k_b, k_e) | k \in \gamma_k^{'}]) & \text{if } e^{'} > e \\ \emptyset & \text{otherwise} \end{cases}$$

获胜票据标记 $\mathbf{H}_w$ 要么为空，要么（如果该区块是票据提交期结束后的第一个区块，且票据累加器已饱和）为票据标识符的最终序列。形式上：

(6.28)
$$\mathbf{H}_w \equiv \begin{cases} Z(\gamma_a) & \text{if } e^{'} = e \wedge m < \mathrm{Y} \leq m^{'} \wedge |\gamma_a| = \mathrm{E} \\ \emptyset & \text{otherwise} \end{cases}$$

6.7 外部交易与票据

外部数据 $\mathbf{E}_T$ 构成了一组有效票据的验证集合，这些票据代表了我们纪元“竞赛”中的参赛项目，旨在决定哪些验证者将享有在下一个纪元的各个时隙中创建区块的特权。每张票据都明确标注了一个唯一的条目索引，并附有验证其有效性的确凿证据。这一证据还隐含着一个票据标识符，它是一个具有高熵且无偏见的 32 字节序列，既作为竞赛中的评分依据，又作为链上可验证随机函数（VRF）运算的输入参数。

在纪元接近尾声时（即从开始算起的 Y 个时隙后），这场竞赛便宣告落幕，意味着同一纪元内的连续区块必须具有空的票据外部数据。此时，下一个纪元所使用的密封密钥序列也随之确定下来，不再发生变化。

我们将外部数据界定为一系列有效票据的验证信息集合，每一项验证信息均构成了一个包含条目索引（该索引为小于 N 的自然数）及票据有效性证明的元组。形式上：

(6.29)
$$\mathrm{E_T} \in \left\llbracket \left(\mathrm{r} \in \mathbb{N}_{\mathrm{N}}, \mathrm{p} \in \bar{\mathbb{F}}^{[]}_{\gamma_z} \langle \mathrm{X_T} \frown \eta_2^{'} \shortparallel \mathrm{r} \rangle \right) \right\rrbracket$$

(6.30)
$$|\mathbf{E}_T| \leq \begin{cases} \mathrm{K} & \text{if } m^{'} < \mathrm{Y} \\ 0 & \text{otherwise} \end{cases}$$

我们将 $\mathbf{n}$ 定义为新票据的集合，其中每张票据都拥有一个独特的标识符，这个标识符是一个哈希值，具体地，它是 Bandersnatch RingVRF 证明的输出部分。

(6.31)
$$\mathbf{n} \equiv \left[(\mathbf{y} : \mathcal{Y}(i_p), r : i_r) | i \in \mathbf{E}_T \right]$$

通过外部数据提交的票据必须依据其内含的标识符进行排序。系统严格禁止提交具有重复标识符的票据，以此确保验证者无法重复利用同一张票据进行多次提交：

(6.32)
$$\mathbf{n} = \left[x_{\mathbf{y}} \,\backslash\backslash\, x \in \mathbf{n} \right]$$

(6.33) $$\{x_{\mathbf{y}} \mid x \in \mathbf{n}\} \backslash\!\backslash \{x_{\mathbf{y}} \mid x \in \gamma_{\mathbf{a}}\}$$

新的票据累加器 $\gamma_{\mathbf{a}}'$ 是通过将新票据合并到之前的累加器值（如果是新纪元，则为空序列）中构建的：

(6.34) $$\gamma_{\mathbf{a}}' \equiv \left[x_{\mathbf{y}} \wr x \in \mathbf{n} \cup \begin{cases} \vec{\emptyset} & \text{if } e' > e \\ \gamma_{\mathbf{a}} & \text{otherwise} \end{cases}\right]^{\mathrm{E}}$$

票据累加器的最大容量被设定为 E。在每个区块的生成过程中，累加器会更新为先前累加器 $\gamma_{\mathbf{a}}$ 与所提交票据的排序并集（去除重复项后）中的最小元素集合。将无效或重复的票据包含在外部数据中是被禁止的，这意味着所有提交的票据都必须在后续的票据累加器中得到体现。形式上：

(6.35) $$\mathbf{n} \subseteq \gamma_{\mathbf{a}}'$$

注意，可以证明，在外部数据为空（即 $\mathbf{E}_T = [\,]$，由 $m' \geq \mathrm{Y}$ 暗示）且纪元未改变 $(e' = e)$ 的情况下，$\gamma_{\mathbf{a}}' = \gamma_{\mathbf{a}}$。

7. 近期历史

我们在状态中保留有关最近 H 个区块的信息。这用于排除提交重复或过时的工作报告的可能性。

(7.1) $$\beta \in \llbracket (h \in \mathbb{H}, \mathbf{b} \in \llbracket \mathbb{H}? \rrbracket, s \in \mathbb{H}, \mathbf{p} \in \mathbb{D}\langle \mathbb{H} \to \mathbb{H} \rangle) \rrbracket_{:\mathrm{H}}$$

对于每个最近的区块，我们保留其头哈希、状态根、累积结果 MMR（默克尔多根树）以及每个报告项对应的工作包哈希（数量不超过总核心数，即 $\mathbf{C} = 341$）。

在累积阶段，会提供一个包含此状态部分转换的值，其中包含对父区块新已知根的更新：

(7.2) $$\beta^\dagger \equiv \beta \text{ except } \beta^\dagger[\,|\beta| - 1]_s = \mathbf{H}_r$$

我们定义一个项目 n，它包括新区块的头哈希、其累积结果的默克尔树根以及其中所做的工作报告集合（对此我们使用保证外部数据 $\mathbf{E}_G$）。请注意，累积结果树根 r 是从 $\mathbf{C}$（在第 12 节中定义）使用基本二进制默克尔化函数 $\mathcal{M}_B$（在附录 $\mathbf{E}$ 中定义）推导出来的，并使用 MMR 追加函数 $\mathcal{A}$（在附录 $\mathbf{E.2}$ 中定义）进行追加，以形成默克尔山脉。

(7.3)
$$\text{Let } r = M_B\left(\left[s \wr \mathcal{E}_4(s) \frown \mathcal{E}(h) \mid (s,h) \in \mathbf{C}\right], \mathcal{H}_K\right)$$
$$\text{Let } \mathbf{b} = \mathcal{A}(\text{last}([[]] \frown [x_{\mathbf{b}} | x \lessdot \beta]), r, \mathcal{H}_K)$$
$$\text{Let } \mathbf{p} = \{((g_w)_s)_h \mapsto ((g_w)_s)_e | g \in \mathbf{E}_G\}$$
$$\text{Let } n = \left(\mathbf{p}, h : \mathcal{H}(\mathbf{H}), \mathbf{b}, s : \mathbb{H}^0\right)$$

状态字典树的根被视为零哈希 $\mathbb{H}^0$，虽然在区块 β' 的结束状态时这不准确，但它是安全的，因为 β' 仅用于定义下一个区块的 $\beta^\dagger$，而 $\beta^\dagger$ 中包含对此的更正值。

最终的状态转换如下：

(7.4) $$\beta' \equiv \beta^\dagger \overleftarrow{\shortparallel}^{\mathrm{H}} n$$

8. 授权

我们之前已经讨论过工作包和服务的模型（见第 4.9 节），但尚未就如何将核心时间资源分配给特定工作包及其相关服务制定具体规则。在 ***YP*** 以太坊模型中，底层资源（即 gas ）是在链上引入时获取的，且购买者始终是描述待执行工作（即交易）的数据创建者。相反，在波卡中，底层资源（即平行链插槽）通常是通过一次性大额押金获取，租期通常为 24 个月，而获取者（通常是平行链团队）往往与待执行工作（即平行链区块）的创建者没有直接关系。

为了提升系统的灵活性，我们期望 JAM 能够兼容并包，既吸纳以太坊式的交互特色，也融入波卡式的交互风格。为了达到这一愿景，我们设计了一个授权机制，它实现了核心时间使用目的与具体工作任务规范及执行计划（即在所分配的核心时段内计划完成的具体事务）之间的有效分离。因此，我们能够将核心时间的购买和分配与具体待执行工作的确定相分离，从而能够支持以太坊风格和波卡风格的交互模式。

8.1 授权者和授权

授权系统包含三个核心要素：授权者、Token 和追踪信息。Token 是一段不可解析的数据，需随工作包一并提交，以此证明该工作包应获得授权。追踪信息同样是一段不可解析的数据，用于记录或描述某次成功授权的相关特征。授权者则是一段逻辑代码，它在既定且明确的计算资源范围内运行，其职责是判断携带 Token 的工作包能否在特定核心上执行授权操作，若授权成功，便生成相应的追踪信息。

授权者的标识是通过将其 PVM 代码哈希值与配置数据块（Configuration blob）拼接组合生成的。这里的配置数据块，和 Token、追踪信息类似，都是对 PVM 代码具有特定意义且不公开的数据。判断工作包是否应被授权（或拒绝）这一流程，并非由链上逻辑负责，而是完全在核心内部执行，相关具体细节会在 14.3 节展开说明。不过，链上逻辑必须能够识别分配给每个核心的授权者集合，以此验证工作包是否有权合法调用该资源。接下来，我们将对这个子系统进行定义。

8.2 池和队列

我们为特定核心 c 定义了一组允许的“授权者”，即授权者池 $\alpha[c]$ 。为了维护这个值，我们为每个核心跟踪了状态的另一部分：核心的当前授权者队列 $\varphi[c]$ ，我们将从中抽取值来填充池。形式上：

(8.1) $$\alpha \in \left\llbracket \left\llbracket \mathbb{H} \right\rrbracket_{:O} \right\rrbracket_C \quad , \quad \varphi \in \left\llbracket \left\llbracket \mathbb{H} \right\rrbracket_{:Q} \right\rrbracket_C$$

注意：状态 φ 的部分只能通过来自具有适当权限的服务的累积逻辑的外部调用来修改。

区块的状态转换涉及从队列中放置一个新的授权到池中：

(8.2) $$\forall c \in \mathbb{N}_C : \alpha'[c] \equiv \overleftarrow{F(c) \eqcirc \varphi'[c]}[\mathbf{H}_t]^{\circlearrowleft}$$

(8.3) $$F(c) \equiv \begin{cases} \alpha[c] \setminus \{(g_w)_a\} & \text{if } \exists g \in \mathbf{E}_G : (g_w)_c = c \\ \alpha[c] & \text{otherwise} \end{cases}$$

由于 α' 依赖于 φ' ，实际上，这一步必须在累积阶段之后计算，累积阶段是定义 φ' 的阶段。注意，我们使用保证外部数据 $\mathbf{E}_G$ 来移除已在当前区块中为已保证工作包证明使用的最旧授权者。这将在方程 11.23 中进一步定义。

9. 服务账户

如我们之前所述，JAM 中的服务在某种程度上类似于以太坊中的智能合约，因为它包括代码组件、存储组件和余额等。与以太坊不同的是，服务的代码被分成两个独立的入口点，每个入口点都有其自己的环境条件：一个是精炼（refinement），它基本上是无状态的，并在核心中执行；另一个是累积（accumulation），它是有状态的，并在链上执行。我们现在将关注后者。

服务账户以状态 δ 保存，其中 δ 是一个部分映射，将服务标识符 $\mathbb{N}_S$ 映射到一个命名元素元组，该元组指定了与 JAM 协议相关的服务属性。形式上：

$$\mathbb{N}_S \equiv \mathbb{N}_{2^{32}} \tag{9.1}$$

$$\delta \in \mathbb{D}\langle \mathbb{N}_S \to \mathbb{A} \rangle \tag{9.2}$$

服务账户被定义为包含存储字典 $\mathbf{s}$ 、预镜像查找字典 $\mathbf{p}$ 和 $\mathbf{l}$ 、代码哈希 c 、余额 b 以及两个代码 gas 限制 g & m 的元组。形式上：

$$\mathbb{A} \equiv \begin{pmatrix} \mathbf{s} \in \mathbb{D}\langle \mathbb{H} \to \mathbb{Y} \rangle, & \mathbf{p} \in \mathbb{D}\langle \mathbb{H} \to \mathbb{Y} \rangle, \\ \mathbf{l} \in \mathbb{D}\langle (\mathbb{H}, \mathbb{N}_L) \to \llbracket \mathbb{N}_T \rrbracket_{:3} \rangle, & \\ c \in \mathbb{H},\ b \in \mathbb{N}_B,\ g \in \mathbb{N}_G,\ m \in \mathbb{N}_G & \end{pmatrix} \tag{9.3}$$

因此，服务索引 s 的余额将被表示为 $\delta[s]_b$ ，而该服务的存储项，key 为 $k \in \mathbb{H}$ ，将被表示为 $\delta[s]_{\mathbf{s}}[k]$ 。

9.1 代码和 Gas

服务账户的代码及其相关元数据由哈希值标识，如果服务要正常运行，则必须在其预镜像查找中存在（见第 9.2 节）。因此，我们定义实际的代码 c 和元数据 m ：

$$\forall \mathbf{a} \in \mathbb{A} : \mathcal{E}(\updownarrow \mathbf{a}_{\mathbf{m}}, \mathbf{a}_{\mathbf{c}}) \equiv \begin{cases} \mathbf{a}_{\mathbf{p}}[\mathbf{a}_c] & \text{if} \mathbf{a}_{\mathbf{c}} \in \mathbf{a}_{\mathbf{p}} \\ \emptyset & \text{otherwise} \end{cases} \tag{9.4}$$

代码中有三个入口点：

0. refine：精炼，在核心中执行且无状态[10]。
1. accumulate：累积，在链上执行且有状态。
2. on_transfer：转账处理器，在链上执行且有状态。

虽然第一个入口点（在核心中执行）将在第 14.3 节中详细描述，但后两个入口点将在本节中定义。

如附录 **A** 所述，JAM 虚拟机中的执行时间以 gas 来确定性地计量，其表示为一个小于 2^{64} 的自然数，并正式记作 $\mathbb{N}_G$。如果该数值可能为负数，我们也可以用 $\mathbb{Z}_G$ 表示集合 $\mathbb{Z}_{-2^{63}\ldots 2^{63}}$ 。账户中规定了两个限制：g ，即执行服务代码中 Accumulate 入口点所需的最小 gas；以及 m ，即在转移过程中 (On Transfer) 入口点所需的最小 gas。

[10] 从技术角度看，这里存在一个微小的状态假设，即每个服务的原像都有一个相对近期的实例。所有验证区块的验证者必然具备的该状态的具体细节将在第 14.3 节中讨论。

9.2 预镜像查找

除了以任意 key/value 对的形式在链上存储数据外，账户还可以请求数据也在核心中可用，从而可在服务的精炼逻辑中查询这些数据。与这一功能相关的状态保存在服务的 $\mathbf{p}$ 和 $\mathbf{l}$ 组件下。

预镜像查找与存储之间存在几个差异。首先，预镜像查找是从哈希到其预镜像的映射，而一般存储是将任意 keys 映射到 values。其次，预镜像数据是外在提供的，而存储数据是作为服务累积的一部分产生的。第三，一旦提供预镜像数据，就不能随意移除；相反，它经历一个被标记为不可用的过程，并且只有在一段时间后才能从状态中移除。这确保了其历史存在的记录得以保留。最后一点尤为重要，因为预镜像数据被设计为在服务代码的精炼逻辑中查询，因此知道预镜像的历史可用性是很重要的。

我们开始重新制定与数据查找系统相关的状态部分。该系统的目的是提供一种在链上存储静态数据的方法，以便稍后可以在执行任何服务代码时将其作为仅接受数据哈希和长度的函数来访问。

在服务代码的累积函数在链上执行时，实现这一点是微不足道的，因为所有验证区块的验证器都固有地知道该状态，即 σ 。然而，对于在核心中执行的精炼来说，没有这样的状态固有地可供所有验证器使用；因此，我们命名一个历史状态，即"查找锚点"，它必须在工作报告的结果可以累积之前被认为是最近已最终确定的，从而为这种保证提供基础。

通过保留其可用性的历史信息，我们能够确信，任何具有链的最近最终确定视图的验证器，都能够确定在任何给定时间段内（在此期间可能发生审计），任何给定的预镜像是否可用。这确保了判断即使在没有链状态共识的情况下也具有确定性的。

重申一下，我们必须能够定义一个"历史"查找函数 Λ ，该函数确定某个服务账户 $\mathbf{a}$ 在某个时间段 t 的某个哈希 h 的预镜像是否可用，如果是，则提供其预镜像：

$$\Lambda: \begin{cases} (\mathbb{A}, \mathbb{N}_{\mathbf{H}_t - C_D \cdots \mathbf{H}_{t'}}, \mathbb{H}) \to \mathbb{Y}? \\ (\mathbf{a}, t, \mathcal{H}(\mathbf{p})) \mapsto v : v \in \{\mathbf{p}, \emptyset\} \end{cases} \tag{9.5}$$

此函数将在下面的方程 9.7 中定义。

服务索引 s 的预镜像查找用 $\delta[s]_\mathbf{p}$ 表示，它是一个将哈希映射到其对应预镜像的字典。此外，还有一个与查找相关的元数据，用 $\delta[s]_\mathbf{l}$ 表示，它是一个将哈希和假定长度映射到历史信息的字典。

9.2.1 不变性

查找系统的状态自然遵循一些不变性条件。首先，任何预镜像值都必须与其对应的哈希值相匹配，这一点在方程 9.6 中有所体现。其次，当存在预镜像值时，它与其哈希值以及长度对之间必然存在着某种关联状态，这种关联同样在方程 9.6 中得到了描述。从形式上：

$$\forall a \in \mathbb{A}, (h \mapsto \mathbf{p}) \in a_\mathbf{p} \Rightarrow h = \mathcal{H}(\mathbf{p}) \wedge (h, |\mathbf{p}|) \in \mathcal{K}(a_\mathbf{l}) \tag{9.6}$$

9.2.2 语义

历史状态组件 $h \in \left[\mathbb{N}_T\right]_{:3}$ ，其序列形式最多可容纳三个时间段。这一序列依据其包含时间段的数量，展现了四种可能的模式：

- $h = [\,]$: 已请求预镜像，但尚未提供。

- $h \in \left[\mathbb{N}_T\right]_1$ ：预镜像“可用”，并且自时间 h_0 起已存在。
- $h \in \left[\mathbb{N}_T\right]_2$ ：之前可用的预镜像现在自时间 h_1 起“不可用”。它曾在时间 h_0 到 h_1 期间可用。
- $h \in \left[\mathbb{N}_T\right]_3$ ：预镜像“可用”，并且自时间 h_2 起已存在。它曾在时间 h_0 到 h_1 期间可用，然后在 h_1 到 h_2 期间 不可用。

现在可以定义历史查找函数 Λ ：

$$\Lambda\colon (\mathbb{A}, \mathbb{N}_T, \mathbb{H}) \to \mathbb{Y}?$$

$$\Lambda(\mathbf{a}, t, h) \equiv \begin{cases} \mathbf{a_p}[h] & \text{if } h \in \mathcal{K}(\mathbf{a_p}) \wedge I(\mathbf{a_l}[h, |\mathbf{a_p}[h]|], t) \\ \emptyset & \text{otherwise} \end{cases} \tag{9.7}$$

$$\text{where} I(1, t) = \begin{cases} \bot & \text{if } [\,] = \mathbf{1} \\ x \le t & \text{if} [x] = \mathbf{1} \\ x \le t < y & \text{if} [x, y] = \mathbf{1} \\ x \le t < y \vee z \le t & \text{if} [x, y, z] = \mathbf{1} \end{cases}$$

9.3 账户占用空间和阈值余额

我们定义服务的存储占用空间为依赖值 i（项数）和 o（总八位字节数），它们仅基于服务的存储映射来确定。服务存储一旦变化，这两个值也会随之更新。

在账户序列化函数（见第 C 节）中，我们将看到这些值会在默克尔化的状态数据中明确体现，因此我们对此进行了明确界定。

接着，我们可以根据服务的存储占用空间，来定义给定服务账户的最低余额阈值 t 。

$$\forall a \in \mathcal{V}(\delta)\colon \begin{cases} a_i \in \mathbb{N}_{2^{32}} & \equiv 2 \cdot |a_{\mathbf{l}}| + |a_{\mathbf{s}}| \\ a_o \in \mathbb{N}_{2^{64}} & \equiv \sum_{(h,z) \in \mathcal{K}(a_1)} 81 + z \\ & \quad + \sum_{x \in \mathcal{V}(a_{\mathbf{s}})} 32 + |x| \\ a_t \in \mathbb{N}_B & \equiv \mathbb{B}_S + \mathbb{B}_I \cdot a_i + \mathbb{B}_L \cdot a_o \end{cases} \tag{9.8}$$

9.4 服务特权

最多可识别三个特权服务，这些信息由状态中的 χ 部分保存。χ 包含三个服务索引和一个 gas 限制：χ_m 是管理服务的索引，能逐块更改 χ ；χ_a 和 χ_v 分别是能逐块更改 φ 和 ι 的服务索引。最后，$\chi_{\mathbf{g}}$ 是一个小字典，记录了每个区块中自动累积的服务索引及累积所需的基本 gas 量。形式上：

$$\chi \equiv \left(\chi_m \in \mathbb{N}_S, \chi_a \in \mathbb{N}_S, \chi_v \in \mathbb{N}_S, \chi_{\mathbf{g}} \in \mathbb{D}\langle \mathbb{N}_S \to \mathbb{N}_G \rangle\right) \tag{9.9}$$

10. 争议、裁决和判断

JAM 提供了一种判断方法：通过多数验证者对工作报告（ JAM 中的工作单元，详见第 11 节）有效性的一致投票来形成判决。同时，JAM 还允许注册异议、判断和保证，这与已确立的判决可能相悖，共同构成了争议系统。

在实践中，判决登记虽不常发生，但它是重要的安全保障。它能移除和禁止无效的工作报告，能在验证者密钥被认为有问题时将其移除，还能协调节点撤销包含无效报告的链扩展，并为集中处理违规验证者提供便利。

判断声明作为审计过程的一部分自然产生，通常为积极判断，确认担保人关于报告有效性的断言。若出现负面判断，所有验证者将审计该报告，并预期会达成判决。审计和担保是链下过程，分别见第 14 节和第 17 节。

对报告的判断意味着其所属链可能在积累前分叉。第 19 节详细描述了链选择策略。包含非正面判决的区块会取消其即将发生的积累（如方程 10.15 所示）。

判决登记还会在链上永久记录该事件，并允许立即或在未来区块中永久记录任何违规密钥。

持久链上记录不当行为在多方面都有帮助，它提供了一种简单方法来识别需对验证者采取措施的情况。若 JAM 用于如 Polkadot 的公共网络，这意味着削减违规验证者在质押平行链上的份额。

记录被认为高度无效的报告很重要，以防止其重新提交。而记录有效报告则能确保在未来链回滚到积累前时，不再提出争议。

10.1 状态

争议状态包含四项内容，其中三项与判决紧密相关：良好集（ $\psi_{\mathbf{g}}$ ）、不良集（ $\psi_{\mathbf{b}}$ ）和异常集（ $\psi_{\mathbf{w}}$ ），分别汇总了被判为正确、错误或无法判定的工作报告的哈希值。第四项是惩罚集（ $\psi_{\mathbf{o}}$ ），它由一组 Ed25519 密钥构成，代表了对工作报告判断错误的验证者。

$$\psi \equiv (\psi_{\mathbf{g}}, \psi_{\mathbf{b}}, \psi_{\mathbf{w}}, \psi_{\mathbf{o}}) \tag{10.1}$$

10.2 外部的

争议的外在属性（ $\mathbf{E}_D$ ）可能包含一项或多项判决，这些判决由当前或前一个时代的验证者集合中至少三分之二加一的成员汇编而成，即基于 k 或 λ 的 Ed25519 密钥。同时，它还可能包含验证者违规的证据，要么是提交的工作报告被证实无效（肇事者 culprits ， $\mathbf{c}$ ），要么是签署了与工作报告有效性冲突的判决（错误 faults ， $\mathbf{f}$ ）。这两种情况均视为违规。形式上：

$$\begin{aligned} \mathbf{E}_D &\equiv (\mathbf{v}, \mathbf{c}, \mathbf{f}) \\ \text{Where } \mathbf{v} &\in \left\llbracket \left(\mathbb{H}, \left\lfloor \frac{\tau}{\mathbf{E}} \right\rfloor - \mathbb{N}_2, \left[(\{\mathrm{T}, \bot\}, \mathbf{N}\mathbf{v}, \mathbb{E}) \right]_{\lfloor 2/3\mathbf{V} \rfloor + 1} \right) \right\rrbracket \\ \text{and } \mathbf{c} &\in \left\llbracket \mathbb{H}, \mathbb{H}_E, \mathbb{E} \right\rrbracket, \mathbf{f} \in \left\llbracket \mathbb{H}, \{\mathrm{T}, \bot\}, \mathbb{H}_E, \mathbb{E} \right\rrbracket \end{aligned} \tag{10.2}$$

所有判决的签名需用两个允许的验证者密钥集之一验证，且判决中应明确第二术语，为前一状态的时代索引或其减一，以标识所用密钥集。形式上：

$$\text{Where } \mathbf{k} = \begin{cases} \kappa & \text{if } a = \left\lfloor \frac{\tau}{\mathrm{E}} \right\rfloor \\ \lambda & \text{otherwise} \end{cases} \tag{10.3}$$

$$\mathrm{X}_{\mathrm{T}} \equiv \$\mathrm{jam\_valid}, \mathrm{X}_{\bot} \equiv \$\mathrm{jam\_invalid} \tag{10.4}$$

违规者签名需有效，且必须附上含判决的工作报告，不得提及已受罚的密钥：

(10.5) $$\forall (r,k,s) \in \mathbf{c}: \bigwedge \begin{cases} r \in \psi'_{\mathbf{b}}, \\ k \in \mathbf{k}, \\ s \in \mathbb{E}_k\langle \mathsf{X}_G \frown r \rangle \end{cases}$$

(10.6) $$\forall (r,v,k,s) \in \mathbf{f}: \bigwedge \begin{cases} r \in \psi'_{\mathbf{b}} \Leftrightarrow r \notin \psi'_{\mathbf{g}} \Leftrightarrow v, \\ k \in \mathbf{k}, \\ s \in \mathbb{E}_k\langle \mathbf{X}_v \frown r \rangle \end{cases}$$

$$\text{Where } \mathbf{k} = \{k_e | k \in \lambda \cup \kappa\} \setminus \psi_{\mathbf{o}}$$

判决 $\mathbf{v}$ 需依报告哈希排序，违规者签名 $\mathbf{c}$ 和 $\mathbf{f}$ 也需分别按验证者 Ed25519 密钥排序。外部信息及过往报告哈希中，均不得有重复哈希。形式上：

(10.7) $$\mathbf{v} = \left[r \;\; \text{⫴} \;\; (r,a,\mathbf{j}) \in \mathbf{v}\right]$$

(10.8) $$\mathbf{c} = \left[k \text{⫴} (r,k,s) \in \mathbf{c}\right], \mathbf{f} = \left[k \text{⫴} (r,v,k,s) \in \mathbf{f}\right]$$

(10.9) $$\{\mathrm{r} | (\mathrm{r},\mathrm{a},\mathrm{j}) \in \mathrm{v}\} \backepsilon \psi_{\mathrm{g}} \cup \psi_{\mathrm{b}} \cup \psi_{\mathrm{w}}$$

所有判决结果需按验证者索引排序，且确保无重复。

(10.10) $$\forall (r,a,\mathbf{j}) \in \mathbf{v}: \mathbf{j} = \left[i \text{⁞} (v,\ i,\ s\}) \in \mathbf{j}\right]$$

我们定义 $\mathbf{V}$ 为从区块外部信息中引入的判决序列，它仅包含报告哈希和正面判决的总和。这个总和必须精确等于验证者集合的三分之二加一（表示报告有效）、零（表示报告无效）或三分之一（表示报告存在问题）[11]。形式上：

(10.11) $$\mathbf{V} \in [(\mathbb{H}, \{0, \lfloor 1/3\mathbf{V} \rfloor, \lfloor 2/3\mathbf{V} \rfloor + 1\})]$$

(10.12) $$\mathbf{V} = \left[\left(r, \sum_{(v,i,s) \in \mathbf{j}} v\right) \middle| (r,a,\mathbf{j}) \lessdot \mathbf{v}\right]$$

对于此外部信息的构成，有以下约束：若判决全为有效，则报告在故障序列 $\mathbf{f}$ 中至少有一有效条目；若判决全为无效，则报告在违规者序列 $\mathbf{c}$ 中至少有两有效条目。 形式上：

(10.13) $$\forall (r, \lfloor 2/3\mathrm{V} \rfloor + 1) \in \mathbf{V}: \exists (r, \ldots) \in \mathbf{f}$$

(10.14) $$\forall (r,0) \in \mathbf{V}: |\{(r, \ldots) \in \mathbf{c}\}| \geq 2$$

我们会剔除所有认为不确定或无效的工作报告核心内容。

(10.15) $$\forall c \in \mathbb{N}_{\mathbb{C}}: \rho^{\dagger}[c] = \begin{cases} \emptyset & \text{if} \{(\mathcal{H}(\rho[c]_w), t) \in \mathbf{V}, t < \lfloor 2/3\mathrm{V} \rfloor\} \\ \rho[c] & \text{otherwise} \end{cases}$$

状态中的良好、不良和问题集合各自吸收每个判决的哈希值。最终，惩罚集合汇总所有违规验证者的密钥。形式上：

(10.16) $$\psi'_{\mathbf{g}} \equiv \psi_{\mathbf{g}} \cup \{r | (r, \lfloor 2/3\mathbf{V} \rfloor + 1) \in \mathbf{V}\}$$

(10.17) $$\psi'_{\mathbf{b}} \equiv \psi_{\mathbf{b}} \cup \{r | (r,0) \in \mathbf{V}\}$$

[11] 这一要求看似随意，实则是我们为三种可能行动设定的决策门槛。在安全假设中，只要至少三分之二加一的验证者活跃，这些门槛就是合理的。Jeff Burdges, Cevallos 等人在2024年的论文中对此的安全影响进行了深入探讨。

(10.18) $$\psi'_{\mathbf{w}} \equiv \psi_{\mathbf{w}} \cup \{r | (r, \lfloor 1/3\mathbf{V} \rfloor) \in \mathbf{V}\}$$

(10.19) $$\psi'_{\mathbf{o}} \equiv \psi_{\mathbf{o}} \cup \{k | (r, k, s) \in \mathbf{c}\} \cup \{k | (r, v, k, s) \in \mathbf{f}\}$$

10.3 头部

违规者标记需准确涵盖所有新违规者的关键信息。形式上：

(10.20) $$\mathbf{H}_o \equiv \left[k | (r, k, s) \in \mathbf{c}\right] \frown \left[k | (r, v, k, s) \in \mathbf{f}\right]$$

11. 报告与保证

报告与保证是我们链上执行的两个关键步骤，旨在使核心计算的结果纳入服务状态单例 δ 。工作包含多个工作项，由验证者转化为工作报告，报告同样包含多个工作输出，并通过链上的保证外在数据（guarantees extrinsic）展示。工作包随后被编码分片，分发给验证者，他们通过在链上提交保证来证明片段的可用性。当获得足够的保证后，工作报告即被视为可用，工作输出通过累积方式更新服务状态，此过程详述于第 12 节。若工作报告超时，则会被另一不累积的报告替代。

从工作报告角度看，先经历保证过程，再是确认过程。但从区块状态转换角度看，优先处理验证者的确认更为合适，因为每个核心同时只能有一个待处理的工作报告等待其包可用。因此，我们先介绍处理可用性保证后的状态转换，再阐述工作报告的初始保证过程。这种同步性通过中间状态 $\rho^\ddagger$ 的正式要求体现，该要求在方程 11.29 中被应用。

11.1 状态

协议中，报告和可用性状态主要通过 ρ 来体现。ρ 追踪那些已报告但尚未获大多数验证者确认可用的工作报告，并记录每个报告的提交时间。需注意的是，任一时刻，一个核心仅能被分配一个报告任务。形式上：

(11.1) $$\rho \in \left[\!\left[(\mathrm{w} \in \mathbb{W}, \mathrm{t} \in \mathbb{N}_\mathrm{T})? \right]\!\right]_\mathrm{C}$$

如常，中间值及后续值（$\rho^\dagger$, $\rho^\ddagger$, ρ'）与先前值遵循相同约束条件。

11.1.1 工作报告

工作报告属于集合 $\mathbb{W}$ ，定义为包含以下元素的元组：工作包规范 s 、精炼上下文 x 、核心索引（指明工作完成的核心）、授权者哈希 a 、输出 $\mathbf{o}$ 、段根查找字典 $\mathbf{I}$ ，以及至少 1 项、最多 $\mathbf{I}$ 项的评估结果 $\mathbf{r}$ 。形式上：

(11.2) $$\mathbb{W} \equiv \begin{pmatrix} s \in \mathbb{S}, & x \in \mathbb{K}, & c \in \mathbb{N}_\mathbb{C}, & a \in \mathbb{H}, \\ \mathbf{o} \in \mathbb{Y}, & \mathbf{I} \in \mathbb{D}\langle \mathbb{H} \to \mathbb{H} \rangle, & \mathbf{r} \in [\mathbb{L}]_{1:\mathrm{I}}, & g \in \mathbb{N}_G \end{pmatrix}$$

我们将段根查找字典的项数及先决条件总数限定 $\mathbf{J} = 8$ 。

(11.3) $$\forall w \in \mathbb{W}: |w_1| + |(w_x)_\mathbf{p}| \leq \mathbf{J}$$

11.1.2 细化上下文

精炼上下文，集合 $\mathbb{X}$ 以示，描绘了工作包在评估时的链上环境。它确定了两个关键历史区块：一是锚点区块，含头部哈希 a 及相应的后继状态根 s 和 “BEEFY” 根 b ；二是查找锚点，即特定时隙 t 的头部哈希 l 。此外，还明确了任何先决条件工作包的哈希值 $\mathbf{p}$ 。形式上：

$$\mathbb{X} \equiv \begin{pmatrix} a \in \mathbb{H}, & s \in \mathbb{H}, & b \in \mathbb{H}, \\ l \in \mathbb{H}, & t \in \mathbb{N}_T, & \mathbf{p} \in \{\mathbb{H}\} \end{pmatrix} \tag{11.4}$$

11.1.3 可用性

我们定义可用性规范集合 $\mathbb{S}$ 为一个元组，其中包括工作包的哈希 h 、可审计的工作包长度 l（详见第 14.4.1 节）、擦除根 u 、段根 e 以及段计数 n 。工作结果中将包含这一规范，以确保能够准确重构并审核报告中所声称的工作包影响。形式上：

$$\mathbb{S} \equiv (h \in \mathbb{H}, l \in \mathbb{N}_L, u \in \mathbb{H}, e \in \mathbb{H}, n \in \mathbb{N}) \tag{11.5}$$

擦除根（u）是二叉默克尔树的根，它承诺了报告审计及后续工作包检索所需的所有数据。保证者用它来验证数据的正确性，审计者随后也会进行验证，具体详见第 14 节。

段根（e）则是固定深度、左偏且用零哈希填充的二叉默克尔树的根，它对每个工作项的导出段哈希做出承诺。保证者利用段根验证重建段的正确性，以评估后续工作包。这一内容也在第 14 节中有所讨论。

11.1.4 可用性

我们最终确定了工作结果 $\mathbb{L}$ ，它作为数据通道，使得服务状态能够通过工作包中的计算得以更新。

$$\mathbb{L} \equiv \begin{pmatrix} s \in \mathbb{N}_S, c \in \mathbb{H}, y \in \mathbb{H}, g \in \mathbb{N}_G, \mathbf{d} \in \mathbb{Y} \cup \mathbb{J}, \\ u \in \mathbb{N}_G, i \in \mathbb{N}, x \in \mathbb{N}, z \in \mathbb{N}, e \in \mathbb{N} \end{pmatrix} \tag{11.6}$$

工作结果是一个由多个项目组成的元组。首先，s 是要更新状态的服务索引，其精炼代码已执行。元组中包含报告时服务代码的哈希值 c ，根据方程 11.42，该哈希值在工作报告中必须准确预测。

其次，元组中有执行阶段中工作项有效载荷（y）的哈希值，该工作项产生了此结果。此哈希值目前虽无直接相关性，但供服务的累积逻辑使用。

接着是气体优先级比率 g ，用于决定在执行此项目累积时应分配的气体量。

最后是代码执行的实际输出数据或错误 $\mathbf{d}$ 。执行成功时，$\mathbf{d}$ 为八位字节序列；执行失败时，$\mathbf{d}$ 为集合 $\mathbb{J}$ 中的某个成员，集合 $\mathbb{J}$ 定义为可能错误的集合。

$$\mathbb{J} \in \{\infty, \lightning, \circledcirc, \text{BAD}, \text{BIG}\} \tag{11.7}$$

最后，我们定义了五个字段来描述此工作负载在核心上执行输出数据时所承担的计算活动水平。（这些字段包括）我们用 $\mathbf{u}$ 表示在优化（refinement）过程中实际消耗的 gas 量；$\mathbf{i}$ 和 $\mathbf{e}$：分别表示从 **Segments DA**（数据可用性层）导入和导出的数据段数量；$\mathbf{x}$ 和 $\mathbf{z}$ 分别表示用于计算该工作负载的**交易**（extrinsics，注：在 substrate 架构中指“用户签名或未签名的交易，及基础设施内部生成的交易）数量，以及其以字节为单位的总大小。关于这些值的具体含义，请参考第 14 章。

前两个错误码是虚拟机执行中的特殊情况，∞ 代表气体不足，$\lightning$ 代表程序异常终止。其余三个错误中，第一个指导出数量报告无效，第二个指服务代码无法用于查找锚定区块的后续状态，第三个指代码虽可用但超出了最大允许大小 W_C 。

为确保区块外部空间公平使用，工作报告对成功输出数据块和授权者输出数据块的总大小设置了限制，从而有效控制了整体大小。

$$\forall w \in \mathbb{W}: |w_{\mathbf{o}}| + \sum_{r \in w_{\mathbf{r}}} |r_{\mathbf{d}}| \leq \mathsf{W}_R \tag{11.8}$$

$$\mathsf{W}_R \equiv 48 \cdot 2^{10} \tag{11.9}$$

11.2 软件包可用性保证

我们首先定义中间状态 $\rho^{\ddagger}$（将在第 11.4 节中使用），以及可用工作报告的集合 $\mathbf{W}$（将在第 12 节中用到）。这两者的定义都需要融入保证数据（extrinsic $\mathbf{E}_A$）的相关信息。

11.2.1 外部保证

保证数据是由一系列保证值组成的，每个验证者最多提供一个。每个保证值包含一个针对每个核心的二进制值序列（即位字符串），同时附有签名和提供该保证的验证者索引。如果某个索引处的值为 1（或布尔值表示为 T ），则意味着该验证者保证其正在为相应核心的可用性做出贡献[12]。形式上：

$$\mathbf{E}_A \in \llbracket \left(a \in \mathbb{H}, f \in \mathbb{B}_C, v \in \mathbb{N}_\mathrm{V}, s \in \mathbb{E}\right) \rrbracket_{:\mathrm{V}} \tag{11.10}$$

所有保证都必须与父区块锚定，并按验证者索引的顺序进行排序。

$$\forall a \in \mathbf{E}_A: a_a = \mathbf{H}_p \tag{11.11}$$

$$\forall i \in \{1 \ldots |\mathbf{E}_\mathrm{A}|\}: \mathbf{E}_A[i-1]_\mathrm{V} < \mathbf{E}_A[i]_v \tag{11.12}$$

签名必须采用保证验证者的公钥来生成，且签名消息应包含父区块的哈希 $\mathbf{H}_p$ 及上述位字符串的序列化形式：

$$\forall a \in \mathbf{E}_A: a_s \in \mathbb{E}_{\kappa[a_v]_e} \langle \mathsf{X}_A \frown \mathcal{H}(\mathcal{E}(\mathbf{H}_p, a_f)) \rangle \tag{11.13}$$

$$X_A \equiv \$\mathrm{jam\_available} \tag{11.14}$$

只有当对应的核心存在未处理的可用性报告时，该位才能被设置：

$$\forall a \in \mathbf{E}_A, c \in \mathbb{N}_C: a_f[c] \Rightarrow \rho^{\dagger}[c] \neq \emptyset \tag{11.15}$$

11.2.2 可用报告

一个工作报告被视为“可用”的条件是，当且仅当有超过 2/3 的验证者在其区块的保证数据中将对应核心标记为已设置。形式上，我们将新可用工作报告的集合 $\mathbf{W}$ 定义为：

[12] 这是一种“非强制性”的含义，因为在链上不诚实地报告并不会带来实际后果。更多相关信息，请参见第 16 节。

$$\mathbf{W} \equiv \left[\rho^{\dagger}[c]_{w} \middle| c \lessdot \mathbb{N}_{\mathbb{C}}, \sum_{a \in \mathbf{E}_{A}} a_{f}[c] > 2/3\mathbf{V}\right] \tag{11.16}$$

这个值在 $\delta^{'}$ 和 $\rho^{\ddagger}$ 的定义中有所应用，我们接下来将定义 $\rho^{\ddagger}$ ，它与 $\rho^{\dagger}$ 是相同的，但会去除那些现已可用或已超时的项目：

$$\forall c \in \mathbb{N}_{\mathbb{C}}: \rho^{\ddagger}[c] \equiv \begin{cases} \emptyset & \text{if} \rho[c]_{w} \in \mathbf{W} \vee \mathbf{H}_{t} \geq \rho^{\dagger}[c]_{t} + \mathbf{U} \\ \rho^{\dagger}[c] & \text{otherwise} \end{cases} \tag{11.17}$$

11.3 担保人分配

每个区块中，每个核心都由三个唯一指定的验证者来负责保证其工作报告。在验证者总数 $\mathbf{V}$=1,023 ，核心总数 $\mathbf{C} = 341$ 的情况下，这一安排得以确保，因为验证者与核心的比例 $\mathbf{V}/\mathbf{C} = 3$。验证者被分配的核心索引，以及其 Ed25519 密钥，用 $\mathbf{G}$ 来表示。

$$\mathbf{G} \in \left(\left[\!\left[\mathbb{N}_{\mathrm{C}} \right]\!\right]_{\mathbb{N}\mathrm{V}}, \left[\!\left[\mathbb{H}_{k} \right]\!\right]_{\mathbb{N}\mathrm{V}}\right) \tag{11.18}$$

我们通过周期熵和周期性轮换的方式来洗牌，以决定每个验证者被分配到哪个核心，这样做能增强网络的安全性和活性。为避免分叉放大的风险（即周期结束时，链状态的不确定性可能导致出现两个已确立的分叉，且需时间自然解决），我们选择了 η_2 作为周期熵，而非 η_1。

接下来，我们定义置换函数 P 、旋转函数 R ，以及最终的验证者分配函数 $\mathbf{G}$ 如下：

$$R(\mathbf{c}, n) \equiv \left[(x + n) \bmod C \middle| x \lessdot \mathbf{c}\right] \tag{11.19}$$

$$P(e, t) \equiv R\left(\mathcal{F}\left(\left[\left[\frac{\mathrm{C} \cdot i}{\mathrm{V}}\right] \middle| i \lessdot \mathbb{N}_{V}\right], e\right), \left[\frac{t \bmod \mathrm{E}}{\mathrm{R}}\right]\right) \tag{11.20}$$

$$\mathbf{G} \equiv (P(\eta_{2}^{'}, \tau^{'}), \Phi(\kappa^{'})) \tag{11.21}$$

我们还定义了 $\mathbf{G}^{*}$ ，它表示的是在前一轮轮换中，$\mathbf{G}$ 本应该呈现的值：

$$\begin{aligned} \mathrm{let}(e, \mathbf{k}) &= \begin{cases} (\eta_{2}^{'}, \kappa^{'}) & \text{if} \left\lfloor \frac{\tau^{'} - \mathbf{R}}{\mathbf{E}} \right\rfloor = \left\lfloor \frac{\tau^{'}}{\mathbf{E}} \right\rfloor \\ (\eta_{3}^{'}, \lambda^{'}) & \text{otherwise} \end{cases} \\ \mathbf{G}^{*} &\equiv (P(e, \tau^{'} - \mathbf{R}), \Phi(\mathbf{k})) \end{aligned} \tag{11.22}$$

11.4 工作报告担保

我们首先定义保证数据 $\mathbf{E}_{G}$ ，它包含一系列保证，每个核心最多对应一个。每个保证都是一个组合，包括一个工作报告、一个凭证 a 以及对应的时间槽 t 。这些保证的核心索引必须是独一无二的，并且需要按照索引的升序进行排列。形式上：

$$\mathbf{E}_{G} \in \left[\!\left[(w \in \mathbb{W}, t \in \mathbb{N}_{T}, a \in \left[\!\left[(\mathbb{N}_{V}, \mathbb{E}) \right]\!\right]_{2:3}) \right]\!\right]_{\mathrm{c}} \tag{11.23}$$

$$\mathbf{E}_{G} = \left[(g_{w})_{c} \wr g \in \mathbf{E}_{G}\right] \tag{11.24}$$

凭证是由两个或三个元组构成的序列，每个元组均包含一个独一无二的验证者索引及一个签名。这些凭证需按照验证者索引的升序进行排列：

$$\forall_{g} \in \mathbf{E}_{G}: g_{a} = [\nu \wr\wr (\nu, s) \in g_{a}] \tag{11.25}$$

签名必须采用凭证中对应验证者的公钥来生成，且签名消息应为工作报告哈希的序列化形式。若保证的时间槽与当前区块时间槽在同一轮次内，则签署验证者必须被分配到当前区块 $\mathbf{G}$ 所指定的核心；若保证的时间槽属于前一轮次，则签署验证者必须被分配到最近一组分配 $\mathbf{G}^*$ 所指定的核心。

$$\forall (w,t,a) \in \mathbf{E}_{G'}, \forall (v,s) \in a : \begin{cases} s \in \mathbb{E}_{(\mathbf{k}_v)_e}\langle \mathrm{X}_G \frown \mathcal{H}(\mathcal{E}(w)) \rangle \\ \mathbf{c}_v = w_c \wedge \mathrm{R}(\lfloor \tau' / \mathrm{R} \rfloor - 1) \le t \le \tau' \end{cases}$$

(11.26)
$$k \in \mathbf{R} \Leftrightarrow \exists (w,t,a) \in \mathbf{E}_G, \exists (v,s) \in a : k = (\mathbf{k}_v)_e$$

$$\text{where}(\mathbf{c},\mathbf{k}) = \begin{cases} \mathbf{G} & \text{if} \left\lfloor \frac{\tau'}{\mathbf{R}} \right\rfloor = \left\lfloor \frac{t}{\mathbf{R}} \right\rfloor \\ \mathbf{G}^* & \text{otherwise} \end{cases}$$

(11.27)
$$\mathrm{X}_G \equiv \$\text{jam\_guarantee}$$

我们注意到，凭证中每个验证者的 Ed25519 密钥（其签名包含在凭证中）都被纳入了报告者集合 R 中，这一信息由验证者活动统计簿记系统（详见第 13 节）所使用。

此外，我们用 $\mathbf{w}$ 来表示当前外部数据 $\mathbf{E}$ 中的工作报告集合：

(11.28)
$$\text{let } \mathbf{w} = \{ g_w \mid g \in \mathbf{E}_G \}$$

不能在已有未处理的可用性报告的核心上提交新报告。报告仅当其授权者的哈希值存在于该报告所属核心的授权者池中时，才被视为有效。形式上：

(11.29)
$$\forall w \in \mathbf{w} : \rho^{\ddagger}[w_c] = \emptyset \wedge w_a \in [w_c]$$

我们规定，每个工作报告为各个工作结果分配的 gas 量，必须达到完成服务所需的最低 gas 标准。同时，所有工作报告分配的 gas 总量，不得超过设定的总体 gas 上限 G_A ：

(11.30)
$$\forall w \in \mathbf{w} : \sum_{r \in w_{\mathbf{r}}} (r_g) \le \mathrm{G}_A \wedge \forall r \in w_{\mathbf{r}} : r_g \ge \delta[r_s]_g$$

11.4.1 报告的上下文有效性

为了简化表述，我们定义两个等价关系：$\mathbf{x}$ 代表外部数据中的所有上下文集合，$\mathbf{p}$ 代表工作包哈希集合。

(11.31)
$$\text{let } \mathbf{x} \equiv \{ w_x \mid w \in \mathbf{w} \}, \mathbf{p} \equiv \{ (w_s)_h \mid w \in \mathbf{w} \}$$

不允许存在重复的工作包哈希（即两个工作报告不能对应同一个工作包）。因此，我们要求工作包哈希集合 $\mathbf{p}$ 的元素数量等于工作报告序列 $\mathbf{w}$ 的长度。

(11.32)
$$|\mathbf{p}| = |\mathbf{w}|$$

我们要求锚定区块必须在最近的 $\mathbf{H}$ 个区块范围内，且其详细信息必须准确无误，确保它包含在我们最近的区块序列 $\beta^{\dagger}$ 中。

(11.33)
$$\forall x \in \mathbf{x} : \exists y \in \beta^{\dagger} : \; x_a = y_h \wedge x_s = y_s \wedge x_b = \mathcal{M}_R\left(y_{\mathbf{b}}\right)$$

我们要求查找的锚定区块必须在最近 $\mathbf{L}$ 个时间槽内。

(11.34) $$\forall x \in \mathbf{x}\colon\ x_t \geq \mathbf{H}_t - \mathbf{L}$$

我们也要求对此进行记录；这是少数几个不能仅通过链上状态来验证的条件之一，必须通过保留最后 $\mathbf{L}$ 个区块头作为祖先集合 $\mathbf{A}$ 来进行验证。由于它是基于区块头链来确定的，因此仍具有确定性和可计算性。形式上：

(11.35) $$\forall x \in \mathbf{x}\colon\ \exists h \in \mathbf{A}\colon\ h_t = x_t \wedge \mathcal{H}(h) = x_l$$

我们要求本报告的工作包必须独一无二，不得与以往任何报告重复，并确保在我们的流程中无重复出现。形式上：

(11.36) $$\text{Let } \mathbf{q} = \{(w_x)_p | \mathbf{q} \in \vartheta, (w, \mathbf{d}) \in \mathbf{q}\}$$

(11.37) $$\text{Let } \mathbf{a} = \{((i_w)_x)_p | i \in \rho, i \neq \emptyset\}$$

(11.38) $$\forall p \in \mathbf{p}, p \notin \bigcup_{x \in \beta} \mathcal{K}(x_{\mathbf{p}}) \cup \bigcup_{x \in \xi} x \cup \mathbf{q} \cup \mathbf{a}$$

我们要求，先决工作包及段根查找中提及的所有工作包，必须源自外部或来自我们近期的历史记录。

(11.39) $$\forall w \in \mathbf{w}, \forall p \in (w_x)_{\mathbf{p}} \cup \mathcal{K}(w_1)\colon$$
$$p \in \mathbf{p} \cup \{x | x \in \mathcal{K}(b_{\mathbf{p}}), b \in \beta\}$$

我们要求，根据最近的工作包历史和当前区块信息，验证段根查找中提及的每个段根，确保其准确无误：

(11.40) $$\text{Let } \mathbf{p} = \{((g_w)_s)_h \mapsto ((g_w)_s)_e | \ g \in \mathbf{E}_G\}$$

(11.41) $$\forall w \in \mathbf{w}\colon w_1 \subseteq \mathbf{p} \cup \bigcup_{b \in \beta} b_{\mathbf{p}}$$

（请注意，这些检查可能会允许存在明显依赖循环的工作报告被接受。但我们并不认为这是个问题：因为在预积累阶段，系统已确保在这些情况下不会发生积累，这些报告仅会被忽略。）

最终，我们要求外部数据中的所有工作结果，都能准确预测其对应服务的代码哈希值：

(11.42) $$\forall w \in \mathbf{w}, \forall r \in w_r\colon r_c = \delta[r_s]_c$$

11.5 报告转换

我们规定，除非外部数据对某个条目进行了替换，否则 ρ' 与 $\rho^{\ddagger}$ 视为等价。若条目被替换，新值中将包含当前时间 τ'，这意味着在足够的时间过去后，可以无需考虑其可用性即可替换该值（见方程 11.29）。

(11.43) $$\forall c \in \mathbb{N}_C\colon \rho'[c] \equiv \begin{cases} (w, t\colon \tau') & \text{if } \exists (c, w, a) \in \mathbf{E}_G \\ \rho^{\ddagger}[c] & \text{otherwise} \end{cases}$$

关于报告和保证的论述至此结束。现在，我们已经完整定义了 ρ'，以及将在第 12 节中使用的 $\mathbf{W}$。第 12 节将详细阐述工作报告在得到保证且可用后，状态转换过程中所发生的变化。

12. 累积

积累可以看作是一个函数，它接收 $\mathbf{W}$ 、δ 以及（有时是部分转换的）状态中的特定部分作为输入，并输出后续的服务状态 δ' ，以及额外的状态元素 ι' 、φ' 和 χ'。

积累的概念其实相当直观：我们的目标仅仅是执行那些至少有一个工作输出的服务的 Accumulate 逻辑，将工作输出和必要的上下文信息传递给它们。然而，在实施过程中，我们面临三个主要挑战。首先，需要明确这个逻辑的执行环境，特别是确定哪些主机函数可供其使用。其次，要设定每个服务在执行时所能消耗的气体量的上限。最后，要明确 Accumulate 中的转账特性，这会导致我们需要引入第二个入口点，即 on-transfer 。

12.1 历史记录和排队

若工作包或工作报告有未满足的依赖关系，其累积将延迟；若依赖关系无效，则相关工作包或报告会被直接取消。依赖关系通过工作包的哈希值确定，同时，为追踪已累积的工作包，我们维护了一个记录，即 ξ ，其容量足够记录一个时代（或周期）内的工作报告。形式上：

(12.1) $$\xi \in \llbracket \{\mathbb{H}\} \rrbracket_{\mathbf{E}}$$

(12.2) $$\hat{\xi} \equiv \bigcup_{x \in \xi} (x)$$

我们在状态项 ϑ 中记录了已准备好（包括可用和 / 或已审核）但尚未累积的工作报告信息。这些报告至少在一个时代（或周期）前就已可用，但因存在或曾存在未满足的依赖关系而未能累积。除了报告本身，我们还保存了其未累积的依赖关系，即一组相关工作包的哈希值。形式上：

(12.3) $$\vartheta \in \left\llbracket \llbracket (\mathbb{W}, \{\mathbb{H}\}) \rrbracket \right\rrbracket_{\mathbf{E}}$$

新出炉的工作报告 $\mathbf{W}$ 依据是否无需前置工作报告被分为两类。符合条件的 $\mathbf{W}^!$ 会立即累积，而不符合条件的 $\mathbf{W}^Q$ 则需排队等候处理。形式上：

(12.4) $$\mathbf{W}^! \equiv \left[w \middle| w \in \mathbf{W}, |(w_x)_{\mathbf{p}}| = 0 \wedge w_{\mathbf{l}} = \{\} \right]$$

(12.5) $$\mathbf{W}^Q \equiv E([D(w) | w \in \mathbf{W}, |(w_x)_{\mathbf{p}}| > 0 \vee w_1 \neq \{\}], \hat{\xi})$$

(12.6) $$D(w) \equiv (w, \{(w_x)_{\mathbf{p}}\} \cup \mathcal{K}(w_{\mathbf{l}}))$$

我们设计了一个队列编辑函数 $\mathbf{E}$ ，它本质上是一个变异函数，用于修改如 ϑ 中的项目。此函数以当前已累积的工作包哈希集合（即 ξ 中的哈希值）为参数，旨在在工作报告累积时更新工作报告队列。具体来说，它会删除所有工作报告哈希值在给定集合中的条目，并移除与该集合中哈希值相关的任何依赖关系。形式上：

(12.7) $$E: \begin{cases} (\llbracket (\mathbb{W}, \{\mathbb{H}\}) \rrbracket, \{\mathbb{H}\}) \to \llbracket (\mathbb{W}, \{\mathbb{H}\}) \rrbracket \\ (\mathbf{r}, \mathbf{x}) \mapsto \left[(w, \mathbf{d} \setminus \mathbf{x}) \middle| \begin{cases} (w, \mathbf{d}) \in \mathbf{r}, \\ (w_s)_h \notin \mathbf{x} \end{cases} \right] \end{cases}$$

我们进一步定义了累积优先级队列函数 Q ，该函数能基于一组尚未累积的工作报告及其依赖关系，生成一个可累积的工作报告序列。

(12.8)
$$Q: \begin{cases} [\![(\mathbb{W}, \{\mathbb{H}\})]\!] \to [\![\mathbb{W}]\!] \\ \mathbf{r} \mapsto \begin{cases} [] \quad \text{if } \mathrm{g} = [] \\ \mathbf{g} \frown Q\big(E(\mathbf{r}, P(\mathbf{g}))\big) \quad \text{otherwise} \\ \text{where } \mathrm{g} = [w \vert (w, \{\}) \in \mathbf{r}] \end{cases} \end{cases}$$

最终，我们定义了映射函数 P ，用于从一组工作报告中提取对应的工作包哈希值：

(12.9)
$$P: \begin{cases} \{\mathbb{W}\} \to \{\mathbb{H}\} \\ \mathbf{w} \mapsto \{(w_s)_h \vert w \in \mathbf{w}\} \end{cases}$$

我们现在把这个区块里可以累积的工作报告序列命名为 $\mathbf{W}^*$ 。

(12.10)
$$\text{Let } m = \mathbf{H}_t \bmod \mathbf{E}$$

(12.11)
$$\mathbf{W}^* \equiv \mathbf{W}^! \frown Q(\mathbf{q})$$

(12.12)
$$\text{where } \mathbf{q} = E(\overbrace{\vartheta_{m\ldots}} \frown \overbrace{\vartheta_{\ldots m}} \frown \mathbf{W}^Q, P(\mathbf{W}^!))$$

12.2 执行

每个区块可用的 gas 量有限，因此可能无法在一个区块内处理完 $\mathbf{W}^*$ 中的所有项目。为了优化单个区块中累积的工作项（及其工作报告）数量，我们面临两个相互制约的因素：

一方面，虽然每个待累积的工作项都有预设的 gas 上限，但累积过程可能会减少实际使用的 gas 量。只有在工作项累积完成后，我们才能确定其使用的 gas 是否低于预设上限。这就要求我们采用顺序执行模式。

另一方面，考虑到虚拟机（PVM）的设置成本不为零，我们希望将这些成本分摊到尽可能多的工作项上。这可以通过将同一服务相关的工作项整合到同一次 PVM 调用中来实现，即采用非顺序执行模式。

为解决这一问题，我们定义了一个函数 Δ_+ ，该函数按顺序累积工作报告，同时利用另一个函数 Δ_* ，该函数则以非顺序、服务聚合的方式累积工作报告。

在所有累积操作完成后，我们整合结果，并据此定义相关的后验状态项。在此过程中，我们还会考虑累积操作可能带来的任何延迟转账影响。

我们的形式化方法首先定义了 $\mathbb{U}$ ，用于表征状态组件（即能够表示状态组件的值），这些状态组件既是累积过程所需的，也可能被累积过程修改。这包括服务账户状态（如 δ 中的）、即将到期的验证者密钥 ι、授权者队列 φ 和权限状态 χ 。形式上：

(12.13)
$$\mathbb{U} \equiv \begin{pmatrix} \mathbf{d} \in \mathbb{D}\langle \mathbb{N}_S \to \mathbb{A} \rangle, \mathbf{i} \in [\![\mathbb{K}]\!]_V, \mathbf{q} \in [\![[\![\mathbb{H}]\!]_Q]\!]_{C'} \\ \mathbf{x} \in (\mathbb{N}_S, \mathbb{N}_S, \mathbb{N}_S, \mathbb{D}\langle \mathbb{N}_S \to \mathbb{N}_G \rangle) \end{pmatrix}$$

我们将延迟转账的集合标记为 $\mathbb{T}$ ，并指出这些转账包含以下要素：一个 128 字节长的备忘录 m ，发送方服务索引 s ，接收方服务索引 d ，转账金额 a ，以及转账的 gas 上限 g 。形式上：

(12.14)
$$\mathbb{T} \equiv (s \in \mathbb{N}_S, d \in \mathbb{N}_S, a \in \mathbb{N}_B, m \in \mathbb{Y}_{\mathbb{W}_T}, g \in \mathbb{N}_G)$$

最终，我们将服务/哈希对集合定义为 B ，该集合用作对累积输出按服务索引进行承诺的依据：

(12.15) $$B \equiv \left\{\left(\mathbb{N}_S, \mathbb{H}\right)\right\} \qquad \mathrm{U} \equiv \left[\left[\left(\mathbb{N}_S, \mathbb{N}_G\right)\right]\right]$$

我们定义了外部累积函数 Δ_+ ，该函数接收以下输入：gas 上限、工作报告序列、初始部分状态以及一个记录享受免费累积服务的字典。该函数输出一个元组，该元组包含累积的工作结果数量、后续状态上下文、生成的延迟转账以及累积输出配对：

(12.16) $$\Delta_+: \begin{cases} \left(\mathbb{N}_G, [\![\mathbb{W}]\!], \mathbb{U}, \mathbb{D}\left\langle \mathbb{N}_S \to \mathbb{N}_G \right\rangle\right) \mapsto \left(, \mathbb{U}, [\![\mathbb{T}]\!], B, U\right) \\ \left(g, \mathbf{o}, \mathbf{w}, \mathbf{f}\right) \mapsto \begin{cases} (0, \mathbf{o}, [], \{\}) & \text{if } i = 0 \\ (i + j,\ \mathbf{t}^* \frown \mathrm{t},\ \ \mathbf{b}^* \cup \mathrm{b},\ \ \mathbf{u}^* \frown \mathrm{u})\ \mathrm{o/w} \end{cases} \\ \text{where } i = \max(\mathbb{N}_{|\mathbf{w}|+1}) : \sum_{w \in \mathbf{w}_{\ldots_i}} \sum_{r \in \mathbf{w}_\mathbf{r}} (r_g) \le g \\ \text{and } (\mathbf{u}^*,\ \mathbf{o}^*, \mathbf{t}^*,\ \ \mathbf{b}^*) = \Delta_*(\mathbf{o}, \mathbf{w}_{\ldots_i}, \mathbf{f}) \\ \text{and } (j, \mathbf{o}', \mathbf{t}, \mathbf{b},\ \ \mathbf{u}) = \Delta_+(\ g - \sum_{(s,u) \in \mathbf{u}^*} u, \mathbf{w}_{\ldots_j}, \mathbf{o}^*, \{\}) \end{cases}$$

我们定义并行累积函数 Δ_* ，它利用单服务累积函数 Δ_1 来处理数据。该函数接收一个初始状态上下文、一系列工作报告以及一个特权始终累积服务字典作为输入，并输出一个元组。该元组包含以下信息：PVM（虚拟机）执行期间消耗的总 gas 量 u 、更新后的状态上下文$(\mathbf{x}', \mathbf{d}', \mathbf{i}', \mathbf{q}')$、生成的累积输出配对 $\mathbf{b}$ ，以及产生的延迟转账 $\hat{\mathbf{t}}$ 。

(12.17) $$\Delta_*: \begin{cases} \left(\mathbb{U}, [\![\mathbb{W}]\!], \mathbb{D}\langle \mathbb{N}_S \to \mathbb{N}_G \rangle\right) \to \left(\mathbb{U}, [\![\mathbb{T}]\!], \mathrm{B}, \mathrm{U}\right) \\ (\mathrm{o}, \mathrm{w}, \mathrm{f}) \mapsto \left(\left((\mathrm{d} \cup \mathrm{n}) \setminus \mathrm{m}, \mathrm{i}', \mathrm{q}', \mathrm{x}'\right), \hat{\mathrm{t}}, \mathrm{b}, \mathrm{u}\right) \\ \text{where:} \\ \mathrm{s} = \{\mathrm{r_s} | \mathrm{w} \in \mathrm{w}, \mathrm{r} \in \mathrm{w_r}\} \cup \mathcal{K}(\mathrm{f}) \\ \mathrm{u} = [(\mathrm{s}, \Delta_1(\mathrm{o}, \mathrm{w}, \mathrm{f}, \mathrm{s})_\mathrm{u}) \mid \mathrm{s} \lessdot \mathrm{s}] \\ \mathrm{b} = \{(\mathrm{s}, \mathrm{b}) | \mathrm{s} \in \mathrm{s}, \mathrm{b} = \Delta_1(\mathrm{o}, \mathrm{w}, \mathrm{f}, \mathrm{s})_\mathrm{b}, \mathrm{b} \neq \emptyset\} \\ \mathrm{t} = [\Delta_1(\mathrm{o}, \mathrm{w}, \mathrm{f}, \mathrm{s})_\mathrm{t} | \mathrm{s} \lessdot \mathrm{s}] \\ (\mathrm{d}, \mathrm{i}, \mathrm{q}, (\mathrm{m}, \mathrm{a}, \mathrm{v}, \mathrm{z})) = \mathrm{o} \\ \mathrm{x}' = (\Delta_1(\mathrm{o}, \mathrm{w}, \mathrm{f}, \mathrm{m})_\mathrm{o})_\mathrm{x} \\ \mathrm{i}' = (\Delta_1(\mathrm{o}, \mathrm{w}, \mathrm{f}, \mathrm{v})_\mathrm{o})_\mathrm{i} \\ \mathrm{q}' = (\Delta_1(\mathrm{o}, \mathrm{w}, \mathrm{f}, \mathrm{a})\mathrm{o})\mathrm{q} \\ \mathrm{n} = \bigcup_{\mathrm{s} \in \mathrm{s}} (\{(\Delta_1(\mathrm{o}, \mathrm{w}, \mathrm{f}, \mathrm{s})_\mathrm{o})_\mathrm{d} \setminus \mathcal{K}(\mathrm{d} \setminus \{\mathrm{s}\})\}) \\ \mathrm{m} = \bigcup_{\mathrm{s} \in \mathrm{s}} (\mathcal{K}(\mathrm{d}) \setminus \mathcal{K}((\Delta_1(\mathrm{o}, \mathrm{w}, \mathrm{f}, \mathrm{s})_\mathrm{o})_\mathrm{d})) \end{cases}$$

我们观察到，在合并所有已更改、新添加和已移除服务的索引（即 $\mathcal{K}(\mathbf{n}) \cup \mathbf{m}$ ）时，不同服务可能不会为这些变动贡献相同的索引。对于已移除和已更改的服务，这种情况不可能发生，因为已移除服务的代码哈希没有已知的原像，无法自行更改。新服务的情况也应如此，因为新索引是精心挑选的，以避免冲突。若这种极端情况真的出现，则该区块将被视为无效。

单服务累积函数 Δ_1 的作用是将一个初始状态上下文、一组工作报告和一个服务索引作为输入，输出一个更新后的状态上下文、一系列转账、可能的累积输出以及 PVM（虚拟机）实际消耗的 gas。该函数会从工作报告中提取特定服务的工作项，并利用这些数据驱动 PVM 执行：

(12.18) $$\mathbb{O} \equiv (h \in \mathbb{H}, e \in \mathbb{H}, a \in \mathbb{H}, o \in \mathbb{Y}, y \in \mathbb{H}, \mathbf{d} \in \mathbb{Y} \cup \mathbb{J})$$

$$\Delta_1: \begin{cases} (\mathbb{U}, [\![\mathbb{W}]\!], \mathbb{D}\langle \mathbb{N}_S \to \mathbb{N}_G \rangle, \mathbb{N}_S) \to \begin{pmatrix} \mathbf{o} \in \mathbb{U}, & \mathbf{t} \in [\![\mathbb{T}]\!], \\ \mathbf{b} \in \mathbb{H}?, & \mathbf{u} \in \mathbb{N}_G \end{pmatrix} \\ (\mathbf{o}, \mathbf{w}, \mathbf{f}, s) \mapsto \Psi_A(\mathbf{o}, \tau', s, g, \mathbf{p}) \\ \text{where:} \\ g = \mathcal{U}(\mathbf{f}_s, 0) + \sum_{w \in \mathbf{w}, \mathbf{r} \in \mathbf{w}_r, \mathbf{r}_s = s} (\mathbf{r}_g) \\ \mathbf{p} = \left[\begin{pmatrix} \mathbf{d}: \mathbf{r}_\mathbf{d}, e: (w_s)_e, \mathbf{o}: w_\mathbf{o}, \\ y: \mathbf{r}_y, r: (w_s)_h, a: w_a \end{pmatrix} \middle| w \in \mathbf{w}, \mathbf{r} \in w_\mathbf{r}, \mathbf{r}_s = s \right] \end{cases} \tag{12.19}$$

这引入了整理后的操作数元组 $\mathbb{O}$，作为 PVM 累积函数 Ψ_A 的输入操作数。同时，它还采用了 gas 上限 g，这一上限由工作报告以及服务 s 和特权 $\mathbf{p}$ 的 gas 权限共同决定，它将 $\mathbf{w}$ 中 s 的工作项重新表述为一系列操作数组合 $\mathbb{O}$。

12.3 延期转账与状态整合

根据顶级 Δ_+ 函数的输出结果，我们可以确定后续状态 χ'、φ' 和 ι'，以及服务账户的首个中间状态 $\delta^\dagger$，还有 BEEFY 承诺的映射关系 $\mathbf{C}$：

$$\text{Let } g = max\left(\mathbf{G}_T, \mathbf{G}_A \cdot \mathbf{C} + \sum_{x \in \mathcal{V}(\chi_\mathbf{g})} (x)\right) \tag{12.20}$$

$$\text{Let } (n, \mathbf{o}, \mathbf{t}, \mathbf{C}, \mathbf{u}) = \Delta_+(g, \mathbf{W}^*, (\chi, \delta, \iota, \varphi), \chi_\mathbf{g}) \tag{12.21}$$

$$(\chi', \delta^\dagger, \iota', \varphi') \equiv \mathbf{o} \tag{12.22}$$

$\mathbf{I}$ 是我们的累积统计数据，它是一个映射关系，用于跟踪**区块链系统中累积过程**的关键数据，其中包括在累积过程中消耗的 gas 总量以及累积的工作报告数量。其正式定义如下：

$$\mathbf{I} \in \mathrm{D}\langle \mathbb{N}_S \to (\mathbb{N}_G, \mathbb{N}) \rangle \tag{12.23}$$

$$\mathbf{I} \equiv \{s \mapsto (\sum_{(s,u) \in u} (u), |N(s)|) | N(s) \neq []\} \tag{12.24}$$

$$\text{where } N(s) \equiv [r | w \in \mathbf{W}^*_{\ldots n}, r \in w_s, r_s = s] \tag{12.25}$$

注意，累积承诺映射 $\mathbf{C}$ 是一个集合，包含了输出产生型累积服务的索引及其对应的累积结果配对。这一映射在方程 7.3 中发挥着关键作用，用于确定当前区块的累积结果树根，对 BEEFY 协议而言至关重要。

我们将隐含的转账序列记为 $\mathbf{t}$，该序列内部按照源服务的执行顺序进行了排序。此外，我们定义了一个选择函数 R，该函数能够将一个延迟转账序列与目标服务索引对应起来，生成针对该服务的转账序列。这个转账序列首先根据源服务索引进行排序，其次再依据它们在 $\mathbf{t}$ 中的顺序进行排序。形式上：

$$R: \begin{cases} ([\![\mathbb{T}]\!], \mathbb{N}_S) \to [\![\mathbb{T}]\!] \\ (\mathbf{t}, d) \mapsto [t | s \in \mathbb{N}_S, \quad t \in \mathbf{t}, \quad t_s = s, \quad t_d = d] \end{cases} \tag{12.26}$$

接着，我们可以定义第二个中间状态 $\delta^\ddagger$，该状态反映了所有延迟转账效果应用后的系统状况：

$$\mathbf{x} = \{s \mapsto \Psi_T(\delta^\dagger, \tau', s, R(\mathbf{t}, s)) | (s \mapsto a) \in \delta^\dagger\} \tag{12.27}$$

$$\delta^\ddagger \equiv \{s \mapsto a | (s \mapsto (a, u)) \in \mathbf{x}\} \tag{12.28}$$

此外，我们构造延迟转账统计值 $\mathbf{X}$，为每个目标服务索引所对应的两项数据：转账数量，以及处理这些转账所消耗的 gas 总量。其正式定义如下：

$$\mathbf{X} \in \mathbb{D}\langle \mathbb{N}_S \to \left(\mathbb{N}, \mathbb{N}_G\right)\rangle \tag{12.29}$$

$$\mathbf{X} \equiv \left\{ \left\{ d \mapsto (|R(\mathbf{t}, d)|, u) \;\middle|\; \begin{matrix} R(\mathbf{t}, d) \neq [\,], \\ \exists_a : \mathbf{x}[d\,] = (a, u) \end{matrix} \right\} \right\} \tag{12.30}$$

注意：Ψ_T 在附录 **B.5** 中有明确定义。如果 $R(d)=[\,]$，即表示该账户未收到任何转账，那么其结果就是 $\delta^\dagger[d]$，意味着账户的中间状态未发生变化。

我们根据本区块累积的工作报告，并结合先前状态中的项目转移情况（最旧的项目将被移除），来定义就绪队列和累积映射的最终状态：

$$\xi'_{\mathrm{E}-1} = P(\mathbf{W}^*_{\ldots n}) \tag{12.31}$$

$$\forall i \in \mathbb{N}_{\mathrm{E}-1} : \xi'_i \equiv \xi_{i+1} \tag{12.32}$$

$$\forall i \in \mathbb{N}_{\mathrm{E}} : \vartheta'^{\circlearrowleft}_{m-i} \equiv \begin{cases} E\left(\mathbf{W}^Q, \xi'_{\mathrm{E}-1}\right) & \text{if } i = 0 \\ [\,] & \text{if } 1 \le i < \tau' - \tau \\ E(\vartheta^{\circlearrowleft}_{m-i}, \xi'_{\mathrm{E}-1}) & \text{if } i \ge \tau' - \tau \end{cases} \tag{12.33}$$

12.4 原像整合

累积完成后，我们需要整合所有从外部数据中查找到的原像，以得出后续的账户状态。这些外部数据由一系列有序且不重复的服务索引和数据对组成（如方程 12.29 所示）。这些数据必须是某个服务请求的，但在先前的状态中尚未提供。形式上：

$$\mathbf{E}_P \in \left[\!\left[(\mathbb{N}_S, Y) \right]\!\right] \tag{12.34}$$

$$\mathbf{E}_P = [i \,\backslash\!\backslash\, i \in \mathbf{E}_P] \tag{12.35}$$

$$R(\mathbf{d}, s, h, l) \Leftrightarrow h \notin \mathbf{d}[s]_{\mathbf{p}} \wedge \mathbf{d}[s]_{\mathbf{l}}[(h, l)] = [] \tag{12.36}$$

$$\forall (s, \mathbf{p}) \in \mathbf{E}_P : R(\delta, s, \mathcal{H}(\mathbf{p}), |\mathbf{p}|) \tag{12.37}$$

在不考虑其他因素的情况下，我们忽略了那些因累积效应而变得无用的原像。随后，我们将 δ' 定义为整合了所有仍然具有相关性的原像后的状态：

$$\text{Let } \mathbf{P} = \{(s, \mathbf{p}) | (s, \mathbf{p}) \in \mathbf{E}_P, R(\delta^\ddagger, s, \mathcal{H}(\mathbf{p}), |\mathbf{p}|)\} \tag{12.38}$$

$$\delta' = \delta^\ddagger \text{ex.} \forall (s, \mathbf{p}) \in \mathbf{P} : \begin{cases} \delta'[s]_{\mathbf{p}}[\mathcal{H}(\mathbf{p})] = \mathbf{p} \\ \delta'[s]_{\mathbf{l}}[\mathcal{H}(\mathbf{p}), |\mathbf{p}|] = [\tau'] \end{cases} \tag{12.39}$$

13. 验证者活动统计

JAM 链不直接负责奖励发放——这项工作由质押子系统来完成（以波卡为例，设想为一个系统平行链，在当前的公共 JAM 网络中免费运行）。然而，与验证者惩罚信息一样，JAM 链必须确保验证者活动信息能够顺利传递给质押子系统，以便子系统据此采取行动，这一点至关重要。

需要注意的是，此类性能信息无法全面反映验证者的所有活动；虽然区块生产、担保人报告和可用性保证等可以在链上轻松追踪，但 GRANDPA、BEEFY 和审计活动则无法做到这一点。在后者的情况下，我们通过验证者的投票活动来追踪：验证者就彼此的工作表现进行投票，对于任何给定的验证者，我们可以将中位数视为其真实情况的反映。在假设有 50% 诚实验证者的前提下，这为获取此类信息提供了一种有效的方法。

验证者统计数据按每个时代进行编制，我们保留了两份记录：一份是已完成统计数据的记录，另一份是当前时代累积器的记录。这两者在 π 中进行追踪，因此 π 是一个包含两个元素的序列：第一个元素是累积器，第二个元素是上一个时代的统计数据。对于每个时代，我们都会为每个验证者追踪一份性能记录：

(13.1) $$\pi \equiv (\pi\nu, \pi L, \pi c, \pi s)$$

(13.2) $$(\pi\nu, \pi L) \in \llbracket (b \in \mathbb{N}, t \in \mathbb{N}, p \in \mathbb{N}, d \in \mathbb{N}, g \in \mathbb{N}, a \in \mathbb{N}) \rrbracket^{2}_{V}$$

我们追踪以下六项关键统计数据：

- b ： 验证者所生产的区块总数。
- t ： 验证者所提交的票证数量。
- p ： 验证者所引入的原像数量。
- d ： 验证者所引入的所有原像中，八位字节的总数。
- g ： 验证者所保证的报告数量。
- a ： 验证者所提供的可用性保证数量。

这些目标统计数据会根据实际情况进行实时更新，形式上：

(13.3) $$let\ e = \left\lfloor \frac{\tau}{\mathbf{E}} \right\rfloor, e' = \left\lfloor \frac{\tau'}{\mathbf{E}} \right\rfloor$$

(13.4) $$(\mathbf{a}, \pi'_{L}) \equiv \begin{cases} (\pi_0, \pi_1) & \text{if } e' = e \\ ([(0, \ldots, [0, \ldots]), \ldots], \pi_0) & \text{otherwise} \end{cases}$$

(13.5) $$\forall v \in \mathbb{N}_{V}: \begin{cases} \pi'_{\mathrm{v}}[v]_b \equiv \mathbf{a}[v]_b + (v = \mathbf{H}_i) \\ \pi'_{V}[v]_t \equiv \mathbf{a}[v]_t + \begin{cases} |\mathbf{E}_T| & \text{if } v = \mathbf{H}_i \\ 0 & \text{otherwise} \end{cases} \\ \pi'_{V}[v]_n \equiv \mathbf{a}[v]_n + \begin{cases} |\mathbf{E}_P| & \text{if } v = \mathbf{H}_i \\ 0 & \text{otherwise} \end{cases} \\ \pi'_{V}[v]_d \equiv \mathbf{a}[v]_d + \begin{cases} \sum_{d \in \mathbf{E}_P} |d| & \text{if} v = \mathbf{H}_i \\ 0 & \text{otherwise} \end{cases} \\ \pi'_{V}[v]_g \equiv \mathbf{a}[v]_g + (\kappa'_v \in \mathbf{R}) \\ \pi'_{V}[v]_a \equiv \mathbf{a}[v]_a + (\exists a \in \mathbf{E}_A : a_v = v) \end{cases}$$

注意，$\mathbf{R}$ 是方程 11.26 中定义的报告者集合。

其他两个统计组件是 核心活动统计 和 服务活动统计。与 验证者统计 在整个 纪元内跟踪不同，这两项统计仅在每个区块级别进行跟踪。

$$\pi c \in \left[\left(\begin{matrix} d \in \mathbb{N}, & p \in \mathbb{N}, & i \in \mathbb{N}, & e \in \mathbb{N}, \\ z \in \mathbb{N}, & x \in \mathbb{N}, & b \in \mathbb{N}, & u \in \mathbb{N}_G \end{matrix}\right)\right]_{\mathrm{C}} \tag{13.6}$$

$$\pi s \in \mathbb{D}\left\langle \mathbb{N}_S \to \left(\begin{matrix} p \in (\mathbb{N},\mathbb{N}), & r \in (\mathbb{N},\mathbb{N}_G) \\ i \in \mathbb{N}, & e \in \mathbb{N}, \quad z \in \mathbb{N}, \quad x \in \mathbb{N}, \\ & a \in (\mathbb{N},\mathbb{N}_G), \quad t \in (\mathbb{N},\mathbb{N}_G) \end{matrix}\right) \right\rangle \tag{13.7}$$

核心统计数据是通过整体状态转换函数中的多个中间值更新的; **w**: 表示传入的工作报告，定义在 公式 11.28; **W**: 表示新可用的工作报告，定义在 公式 11.16。我们将这些统计数据定义如下:

$$\forall v \in \mathbb{N}_C\colon \pi_C'[\mathrm{C}] \equiv \left(\begin{matrix} i\colon R(c)_i\,, x\colon R(c)_x\,, \mathrm{z}\colon R(c)_z\,, \\ e\colon R(c)_e\,, y\colon R(c)_u\,, b\colon R(c)_b\,, \\ d\colon D(c)\,, p\colon \sum_{a \in E_{\mathrm{A}}} a_f[c] \end{matrix}\right) \tag{13.8}$$

$$\text{where } R(c \in \mathbb{N}_C) \equiv \sum_{r \in w_r, w \in W, w_c = c} \left(r_i\,, r_x\,, r_z\,, r_e\,, r_u\,, b : (w_x)t\right) \tag{13.9}$$

$$\text{and } D(c \in \mathbb{N}_C) \equiv \sum_{w \in W, w_c = c} (w_s)_l + \lceil \mathrm{W}_G\ (w_s)_n 65/64 \rceil \tag{13.10}$$

最后，服务统计数据使用与核心统计数据相同的中间值进行更新，但采用不同的计算方式:

$$\forall s \in \mathbf{S}\colon \pi_S'[s]\colon \begin{cases} i\colon R(s)_i\,, x\colon R(s)_x\,, \mathrm{z}\colon R(s)_z\,, \\ e\colon R(s)_e\,, r\colon (R(s)_n\,, R(s)_u)\,, \\ p\colon \sum_{(s,\mathrm{p}) \in \mathbf{E}_P} (1,\ |p|)\,, \\ \quad a\colon \mathcal{U}(\mathbf{I}[s], (0,0))\,, \\ \quad t\colon \mathcal{U}(\mathbf{X}[s], (0,0)) \end{cases} \tag{13.11}$$

$$\text{where } \mathbf{s} = \mathbf{r} \cup \mathbf{p} \cup \mathcal{K}(\mathbf{I}) \cup \mathcal{K}(\mathbf{X}) \tag{13.12}$$

$$\text{and } \mathbf{r} = \{r_s \mid r \in w_r, w \in W\} \tag{13.13}$$

$$\text{and } \mathbf{p} = \{s \mid \exists x\colon (s,x) \in \mathbf{E}_P\} \tag{13.14}$$

$$\text{and } R(s \in \mathbb{N}_S) \equiv \sum_{r \in w_r, w \in W, r_s = s} \left(n : 1, r_u\,, r_i\,, r_x\,, r_z\,, r_e\right) \tag{13.15}$$

14. 工作包和工作报告

14.1 诚实行为

到目前为止，我们已经明确了如何识别 JAM 区块链中正确过渡的区块。通过定义状态转移函数和状态默克尔化函数，我们也确立了有效区块头的识别标准。虽然对于能够控制生成区块头中两个签名所需密钥的节点而言，创建新区块并填写其他区块头字段并不构成太大挑战，但读者可能会注意到，关于外在内容的具体细节尚未明确。

我们不仅通过创建正确的区块来界定正确行为，还定义了诚实行为，这涵盖了节点参与的多种链下活动。在 ***YP*** 以太坊中，虽然相关方面并未被明确提及，但确实存在类似的做法: 创建区块、在这些区块中传播和包含交易等链下活动，都被视为有助于诚实行为的表现。而在 JAM 中，诚实行为有着明确的定义，并且预期至少有 $^2/_3$ 的验证者会遵守这一行为准则。

除了生产区块之外，受激励的诚实行为还包括以下方面：

- 保证和报告工作包，以及工作包和数据块的分块与分发，相关内容已在第 15 节中讨论；
- 在接收到工作包数据后，确保其可用性；
- 确定要审核的工作报告，获取并审核这些报告，然后根据审核结果恰当地创建并分发判断；
- 提交其他验证者所完成的正确数量的审核工作，相关内容已在第13节中讨论。

14.2 分段与清单

我们的基本纠删编码段大小设定为 $\mathbf{W}_E = 684$，这一设定基于以下考量：即便在 1023 名参与者中有近三分之二表现恶意或失效，我们仍能实现数据重构；纠删编码技术基于16位伽罗瓦 (Galois) 域；同时，我们致力于高效支持至少 4KB 大小的数据编码。

工作包设计得相对较小，旨在减轻担保人的带宽负担，便于他们验证工作报告以获取报酬。工作包并不直接携带大量数据，而是通过承诺机制引用数据，其中最简单的是外部数据承诺。

外部数据是由工作包构建者引入系统的数据块，与工作包一同提交，并作为参数供精炼逻辑使用。我们通过在工作包中纳入各数据块的哈希值来做出承诺。

此外，工作包还与两种外部数据相关联：一是对每个导入段的加密承诺，二是导出的段数量。

14.2.1 段、导入和导出

在 JAM 可用性系统中，实现从一个工作包向后续工作包传输大量数据的功能至关重要。导出段，被定义为集合 $\mathbb{G}$，是一个长度为 $\mathbf{W}_G = 4104$ 的固定序列。它是工作包在精炼过程中，能够从长期导入数据存储区（DA）独立导入或导出的最小数据单元。导出段的长度恰好是纠删编码片段大小的整数倍，这保证了工作包的数据段能够高效地在 DA 系统中进行存储和读取。

$$\mathbb{G} \equiv \mathbb{Y}_{\mathbf{W}_G} \tag{14.1}$$

导出段是精炼逻辑执行过程中产生的数据，可以视为工作包转化为工作报告时的附加产物。这些数据依据精炼逻辑自动生成，无需在工作包中特别标注，只需记录每次精炼时产生的导出段数量，以便后续准确索引。

相反，导入段则是先前工作包导出的数据段。为了方便获取和验证这些段，我们不采用哈希引用，而是结合当时引入的其他所有段的默克尔树根及该段在序列中的索引进行引用。这种方式能生成正确性证明，便于存储，并能与获取的数据一同进行验证。关于这一点，下一节将详细阐述。

14.2.2 数据收集与论证

保证人的职责是从众多独特验证者处收集纠删码块，以完整重建所有导入段。但仅仅重建并不足够，因为一旦有验证者提供了错误块，数据就会受损。因此，我们采用导入段规范（包含默克尔树根 Merkle root 及树中索引）作为加密承诺，通过证明来验证特定段的准确性。

在可能需要某些段时，任何节点都应能获取到相应的证明数据。然而，由于验证单个段正确性所需的数据量约为 350 字节，验证者难以存储所有数据。为此，我们采用与数据本身相同的总体可用性框架来存储证明元数据。

保证人可利用这些证明来确保自己没有在无谓的错误上浪费时间。而审计员则无需经历同样的验证过程。相反，保证人构建一个包含原始工作包、外部数据、导入数据及其正确性简洁证明的可审计工作包 （Auditable Work Package），并将其放入审计数据 （Audit DA）系统。虽然这种做法会在导入数据 (Imports DA) 系统和审计数据 系统之间产生数据重复，但这是可以接受的，因为它降低了审计员的带宽成本。毕竟，每个工作包平均会被审计 30 次，而保证过程通常只进行两到三次。

14.3 工作包和项目

我们首先定义工作包为集合 $\mathbb{P}$，其包含的工作项则为集合 $\mathbb{I}$。工作包 $\mathbf{p}$ 具体由一个授权代币数据块 $\mathbf{j}$、服务索引 h （用于托管授权代码）、授权代码哈希u、参数化数据块 $\mathbf{p}$、上下文 $\mathbf{x}$ 以及一系列工作项 $\mathbf{w}$ 组成。

(14.2) $$\mathbb{P} \equiv (\mathbf{j} \in \mathbb{Y}, h \in \mathbb{N}_S, u \in \mathbb{H}, \mathbf{p} \in \mathbb{Y}, \mathbf{x} \in \mathbb{X}, \mathbf{w} \in [\mathbb{I}]_{1:!})$$

一个工作项涵盖以下内容:

s ： 相关服务的唯一标识符;

c ： 服务在报告时的代码哈希（需确保能从查找锚定块中还原其原像）;

$\mathbf{y}$ ： 有效载荷数据块;

g & a ： 分别代表精炼和累积的气体限制;

清单部分包含三个元素:

$\mathbf{i}$ ： 一系列导入的数据段，通过索引和导出工作包的标识来关联先前导出的数据段;

$\mathbf{x}$ ： 本块中将引入的数据块哈希和长度列表（假定验证者已知这些信息）;

e ： 本工作项导出的数据段数量。

(14.3) $$\mathbb{I} \equiv \begin{pmatrix} s \in \mathbb{N}_S, c \in \mathbb{H}, \mathbf{y} \in \mathbb{Y}, g \in \mathbb{N}_G, a \in \mathbb{N}_G, e \in \mathbb{N}, \\ \mathbf{i} \in [\![(\mathbb{H} \cup (\mathbb{H}^{\boxplus}), \mathbb{N})]\!], \mathbf{x} \in [\![(\mathbb{H}, \mathbb{N})]\!] \end{pmatrix}$$

注意，工作包的导入数据段通过集合 $\mathbb{H}$ 及其扩展变体 $\mathbb{H}^{\boxplus}$ 的合并来识别。若值取自常规集合 $\mathbb{H}$，则表示该哈希值包含导出的段根哈希；若取自 $\mathbb{H}^{\boxplus}$，则哈希值代表导出工作包的哈希。在后一种情形中，保证人需将其转换为段根，并在工作报告中报告此转换结果，以供链上验证。

我们规定，导出项与导入项的总数均不得超过 3072（$\mathsf{W}_\mathsf{M} = 3072$），同时外部调用的总数上限为 128（$\mathsf{T} = 128$）。

(14.4) $$\forall p \in \mathbb{P}: \quad \sum_{w \in p_\mathbf{w}} w_e \leq \mathsf{W}_M \wedge \sum_{w \in p_\mathbf{w}} |w_\mathbf{i}| \leq \mathsf{W}_M \wedge \sum_{w \in p_\mathbf{w}} |w_\mathbf{x}| \leq \mathsf{T}$$

我们假定保证人已经掌握每个工作项中所有外部调用哈希的原像，这些原像数据一般会随工作包一同提供给保证人。通常，这些数据将与工作包一起传递给担保人。

为了实现大约 2MB /秒/核心的数据吞吐量，我们将隐含导入项与外部调用项的总大小，以及所有有效载荷、授权者参数和授权代币的总大小，均限制在 12MB 以内。

$$\forall p \in \mathbb{P}: \left(|p_{\mathbf{j}}| + |p_{\mathbf{p}}| + \sum_{w \in p_{\mathbf{w}}} S(w) \right) \leq \mathsf{W}_B \tag{14.5}$$

$$\text{where } S(w \in \mathbb{I}) \equiv |w_{\mathbf{y}}| + |w_{\mathbf{i}}| \cdot \mathsf{W}_G + \sum_{(h,l) \in w_{\mathbf{x}}} l$$

$$\mathsf{W}_B \equiv 12 \cdot 2^{20} \tag{14.6}$$

我们将这两种 gas 限制的总和，分别限定为单个核心在执行对应操作时所能分配的最大 gas 量。

$$\forall p \in \mathbb{P}: \sum_{w \in p_{\mathbf{w}}} (w_a) < \mathsf{G}_A \wedge \sum_{w \in p_{\mathbf{w}}} (w_r) < \mathsf{G}_R \tag{14.7}$$

已知某个工作项(work-item)执行后得到的结果为 $\mathbf{d}$ ，且执行该工作项所消耗的燃气量为 u ，我们将工作项到摘要(digest)的映射关系用函数 C 来定义，具体如下:

$$C: \begin{cases} (\mathbb{I}, \mathbb{Y} \cup \mathbb{J}, \mathbb{N}_G) \to \mathbb{L} \\ \left(\begin{pmatrix} s, c, \mathrm{y}, \\ a, e, i, x \end{pmatrix}, \mathbf{d}, u \right) \mapsto \begin{pmatrix} s, c, \mathrm{y}\colon \mathcal{H}(\mathbf{y}), g\colon a, d, u, \\ i\colon |\mathbf{i}|, e, x\colon |\mathbf{x}|, \mathrm{z}\colon \sum_{(h,z) \in x} z \end{pmatrix} \end{cases} \tag{14.8}$$

我们定义工作包隐含的授权者为 p_a，该授权者是通过将授权代码哈希与参数化拼接后得到的哈希值。同时，我们定义授权代码为 p_c，并规定在从历史查找服务 p_h 中检索锚定区块时，该授权代码必须处于可用状态。形式上:

$$\forall p \in \mathbb{P}: \begin{cases} p_{\boldsymbol{a}} \equiv \mathcal{H}(p_{\boldsymbol{u}} \frown p_{\boldsymbol{p}}) \\ \mathcal{E}(\updownarrow p_{\boldsymbol{m}}, p_{\boldsymbol{c}}) \equiv \lambda(\delta\left[p_{\boldsymbol{h}}\right], (p_{\boldsymbol{x}})_t, p_{\boldsymbol{u}}) \\ (p_{\boldsymbol{m}}, p_{\boldsymbol{c}}) \in (\mathbb{Y}, \mathbb{Y}) \end{cases} \tag{14.9}$$

（历史查找函数 $\boldsymbol{\Lambda}$ 在方程 9.7 中定义。）

14.3.1 导出

任何工作包中的工作项均能导出数据段，这些数据段的根会按照导出它们的工作项顺序排列在工作报告中，并根据固定深度二叉默克尔树（定义于方程 **E.4** ）生成根。

保证人需对数据进行擦除编码，并分发两组数据。第一组是数据块，即可审计的工作包，包含编码后的工作包、外部数据及自证明的导入数据段，这些数据存储在短期审计数据存储中。第二组则是导出的数据段数据及分页证明元数据。第一组数据为短期存储，保证人仅需保留至确保工作结果的工作包可用性区块达成最终性即可。而第二组数据则为长期存储，预计在工作报告提交后至少保留 28 天（即 672 个完整周期）。

我们定义了分页证明函数 P, 该函数接收一系列导出的数据段 $\mathbf{s}$, 并确定了一系列通过擦除编码和分发后放入导入数据(DA)存储的附加数据段。该函数会计算出一系列哈希页及子树证明，以便基于数据段根验证其正确性:

$$P: \begin{cases} [\mathbb{G}] \to [\mathbb{G}] \\ \mathbf{s} \mapsto \left[\mathcal{P}_l(\mathcal{E}(\uparrow \mathcal{J}_0(\mathbf{s}, i), \uparrow \mathcal{L}_6(\mathbf{s}, i))) | i < \mathbb{N}_{\lceil |\mathbf{s}|/64 \rceil} \right] \\ \text{where } l = \mathsf{W}_G \end{cases} \tag{14.10}$$

14.4 工作成果的计算

接下来，我们介绍工作结果计算函数 Ξ ，它是 JAM 平台上所有核心应用的基础。该函数接收为特定核心 c 准备的工作包 p 作为输入，输出结果为错误 ∇ ，或者输出工作结果及一系列导出的数据段。这个函数具有确定性，并且得益于其历史查找功能，只需在最近最终确定区块后的八个周期内进行评估。因此，即便在实际网络环境中可能存在同步不完全的情况，任何节点在审计期内都能轻松完成对该函数的评估。形式上：

(14.11)

$$\Xi: \begin{cases} (\mathbb{P}, \mathbb{N}_C) \to \mathbb{W} \\ (p, c) \mapsto \begin{cases} \nabla & \text{if } \mathbf{o} \notin \mathbb{Y} \\ (s, x{:}\, p_{\mathbf{x}}, c, a{:}\, p_a, \mathbf{o}, \mathbf{l}, \mathbf{r}, g) & \text{otherwise} \end{cases} \end{cases}$$

where:

$$\mathcal{K}(\mathbf{l}) \equiv \{h | w \in p_{\mathbf{w}}, (h^{\boxplus}, n) \in w_{\mathbf{i}}\}, |\mathbf{l}| \leq 8$$

$$(\mathbf{o}, g) = \Psi_I(p, c)$$

$$(\mathbf{r}, \bar{\mathbf{e}}) = {}^{\mathsf{T}}[(C(p_{\mathbf{w}}[j], r, u), \mathbf{e}) | (r, u, \mathbf{e}) = I(p, j), j \lll \mathbb{N}_{|p_{\mathbf{w}}|}]$$

$$I(p,j) \equiv \begin{cases} (r, u, \mathbf{e}) & \text{if} |\mathbf{e}| = w_e \\ (r, u, [\mathbb{C}_0, \mathbb{C}_0, \ldots]_{\ldots w_e}) & \text{otherwise if } r \notin \mathbb{Y} \\ (\odot, u, [\mathbb{G}_0, \mathbb{G}_0, \ldots]_{\ldots w_e}) & \text{otherwise} \\ \text{where} (r, \mathbf{e}, u) = \Psi_R(j, p, \mathbf{o}, S(w, l), \ell) \\ \text{and } h = \mathcal{H}(p), w = p_{\mathbf{w}}[j], \ell = \sum_{k<j} p_{\mathbf{w}}[k]_e \end{cases}$$

请注意，我们妥善处理了以下情况：当工作项的 Refine 操作实际导出的数据段数量与报告中声明的导出数据段数量不符时，整个工作包依然有效，但该工作项所导出的数据段将被替换为与导出数据段计数相同数量的零数据段序列。

起初，我们对段根字典 l 设定了限制：它必须包含所有无法通过段根直接识别，而需通过工作包哈希来识别的导入数据段所对应的工作包哈希的唯一条目。

随后，我们定义了依赖于该字典的段根查找函数 L，该函数利用字典将段根与工作包哈希的并集整合为单一的段根：

(14.12)

$$L(r \in \mathbb{H} \cup \mathbb{H}^{\boxplus}) \equiv \begin{cases} r & \text{if } r \in \mathbb{H} \\ \mathbf{1}[h] & \text{if } \exists h \in \mathbb{H}: r = h^{\boxplus} \end{cases}$$

为保证所构建的工作报告能够获得应有的补偿，保证人需要为l设定一个值，这个值不仅要满足前述条件，还需额外确保所有工作包哈希与段根之间的对应关系准确无误：

(14.13)

$$\forall (h \mapsto e) \in \mathbf{l}: \exists p, c \in \mathbb{P}, \mathbb{N}_{\mathbb{C}}: \mathcal{H}(p) = h \wedge (\Xi(p, c)_s)_e = e$$

如果保证人无法满足上述约束条件，则应认为该工作包无效。审计者无需为该值进行填充，而应直接使用他们正在审计的工作报告中的值。

接下来，我们介绍几个术语：$\mathbf{o}$ 代表授权输出，是 Is-Authorized 函数的结果；$(\mathbf{r}, \bar{\mathbf{e}})$ 代表工作包中每个工作项的结果序列及其导出的所有数据段；我们还定义了一个执行有序累积 I（即计数）的机制，以确保 Refine 函数能够获取到从工作包开始到当前工作项为止导出的数据段总数。

上述内容依赖于两个函数：S 和 X，它们分别为工作项参数 w 定义导入数据段数据和外部数据。此外，我们还定义了 J 函数，用于编译数据段数据的证明：

(14.14)
$$X(w \in \mathbb{I}) \equiv \left[\mathbf{d} | (\mathcal{H}(\mathbf{d}), |\mathbf{d}|) \in w_{\mathbf{x}}\right]$$
$$S(w \in \mathbb{I}) \equiv \left[\mathbf{s}[n] | \mathcal{M}(\mathbf{s}) = L(r), (r, n) \in w_{\mathbf{i}}\right]$$
$$J(w \in \mathbb{I}) \equiv [\updownarrow \mathcal{J}_0(\mathbf{s}, n) | \mathcal{M}(\mathbf{s}) = L(r), (r, n) \in w_{\mathbf{i}}]$$

接着，我们可以借助这两个函数以及将在第 14.4.1 节中定义的可用性指定函数 A，来明确 s 作为该数据包的可用性规范：

(14.15)
$$s = A(\mathcal{H}(p), \mathcal{E}(p, X^{\#}(p_{\mathbf{w}}), S^{\#}(p_{\mathbf{w}}), J^{\#}(p_{\mathbf{w}})), \hat{\mathbf{e}})$$

注意，尽管 S 和 J 函数在描述时都使用了术语 s（代表工作包导出的所有用于导入的数据段），但实际上并不需要如此庞大的数据量。因为可以通过单个分页证明来推导证明，这极大地减少了保证人在最坏情况下需要获取的数据量，即每导入一个数据段只需获取两个数据段。在导入连续导出的数据段（这往往是一种常见情况）时，一个分页证明就足以验证多个导入的数据段。

另外值得注意的是，除了证明的默克尔路径外，其他序列都不包含长度前缀。工作包本身就能确定所有其他序列的长度。在执行 Is-Authorized 逻辑之前，必须确保其已被正确执行，以确保工作包满足所需的核心时间要求。接下来，保证人应确认所有构成导入数据段承诺的数据段树根都是已知且未过期的。最后，保证人还需确保能够获取所有作为外部数据段承诺引用的原像数据。

完成上述步骤后，需要重建导入的数据段。这个过程可以是惰性的，因为 Refine 函数在获取主机调用之前不会使用这些数据。获取数据通常意味着从足够多的唯一验证者（包括保证人，共 342 个）中检索擦除编码的数据块，这在附录 H 中有详细描述。由于我们使用了系统性擦除编码，因此在 342 个验证者都响应的情况下，数据重建变得相对简单。同时，必须获取数据本身和证明元数据的数据块，以确保数据的准确性。

作为可用性保证者的验证者，应根据其促进重建的数据段树的索引来索引这些数据块。由于数据段数据的大小仅为 12 字节，因此固定通信成本得以保持在最低水平。一个优秀的网络协议（目前不在讨论范围内）将允许保证人仅通过指定数据段树根、索引以及一个布尔值（指示是否需要提供证明数据块）来简化操作。鉴于我们假设至少有 341 个其他验证者在线且行为良好，因此保证人可以基于对数据可用性 $\mathbf{s}^{\clubsuit}$（下文将详细说明）的一般信任，有信心地计算上述 S 和 J 函数。

14.4.1 可用性说明符

我们定义了可用性指定函数A，它基于包哈希、一个便于审计的工作包捆绑包的八位字节序列（该序列涵盖了工作包本身、外部数据以及串联起来的导入数据段和它们的正确性证明），以及导出的数据段序列，来生成一个可用性指定符。

(14.16)
$$A: \begin{cases} (\mathbb{H}, \mathbb{Y}, [\![\mathbb{G}]\!]) \to \mathbb{S} \\ (h, \mathbf{b}, \mathbf{s}) \mapsto (h, l: |\mathbf{b}|, u, e: \mathcal{M}(\mathbf{s}), n: |\mathbf{s}|) \end{cases}$$
$$\text{where } u = \mathcal{M}_B(\hat{[}\mathbf{x} | \mathbf{x} \in {}^{\mathrm{T}}[\mathbf{b}^{\clubsuit}, \mathbf{s}^{\clubsuit}]])$$
$$\text{and } \mathbf{b}^{\clubsuit} = \mathcal{H}^{\#}(\mathcal{C}_{\lceil |\mathbf{b}| / \mathbf{w}_E \rceil}(\mathcal{P}_{\mathrm{W}_E}(\mathbf{b})))$$
$$\text{and } \mathbf{s}^{\clubsuit} = \mathcal{M}_B^{\#}({}^{\mathrm{T}}\mathcal{C}_6^{\#}(\mathbf{s} \sim P(\mathbf{s})))$$

之前在方程 14.10 中定义的分页证明函数 P，能够接收一个数据段序列作为输入，并输出一个分页证明序列。这个序列足以验证每个数据段的正确性。生成的分页证明数据段数量恰好是向上取整到最近的整数个（即$\lceil 1/64 \rceil$），每个分页证明包含一页，页内含有 64 个数据段哈希，以及从根到包含这 64 个数据段的子树根的默克尔证明。
函数 $\mathcal{M}$ 和 $\mathcal{M}_B$ 分别是固定深度和简单二叉默克尔根函数，已在方程 E.4 和 E.3 中定义。函数$\mathcal{C}$是擦除编码函数，具体定义参见附录 H。

另外，$\mathcal{P}$（此处可能是一个笔误或特定上下文中的符号，假设其代表某个功能）是一个零填充函数，用于将八位字节数组的长度调整为 n 的倍数：

$$\mathcal{P}_{n\in\mathbb{N}_{1\dots}} : \begin{cases} \mathbb{Y} \to \mathbb{Y}_{k\cdot n} \\ \mathbf{x} \mapsto \mathbf{x} \sim [0,0,\dots]_{((|x|+n-1) \bmod n)+1\dots n} \end{cases} \tag{14.17}$$

验证者有动力将每个新擦除编码的数据块分发给相关验证者，因为他们只有在超级多数验证者确认工作报告可用时，才能获得提供保证的报酬。针对我们的工作包 $\mathbf{p}$，我们应向那些密钥与导入、外部及导出数据段数据的相应数据块相匹配的验证者，发送对应的工作包捆绑包数据块和导出数据段数据块。这样，每个验证者都能根据工作报告的擦除根来验证数据的完整性。在即将到来的时代更迭时，他们还能通过向新的验证者集合分发数据，来最大化预期收益。

后续章节将展示这个函数在保证、审计和评判方面的应用。

15. 保证

保证工作包需要创建并分发工作报告，同时满足特定条件，并附带验证者对其正确性承诺的签名。一旦获得两名保证人的签名，工作报告即可分发给即将生成 JAM Chain 区块的节点，用于执行保证 $\mathbf{E}_G$，从而确保保证人获得奖励。

在公开系统中，我们假定验证者若提交未能准确反映对工作包应用 Ξ 结果的报告，将受到严厉处罚。整体流程如下：

（1）评估工作包的授权状态，并与 JAM Chain 最新状态中的授权池进行比对。

（2）创建并发布工作包报告。

（3）根据擦除编码，将工作包及其外部数据和导出数据分块。

（4）将这些数据块分发给验证者集合。

（5）根据需要，向其他验证者提供工作包、外部数据和导出数据，以优化网络性能。

对于接收到的任何工作包 $\mathbf{p}$，我们可以确认其是否与我们负责的核心 c 的某个工作报告（如存在）相对应。在需要 JAM Chain 状态时，我们始终参考最新区块的链状态。

对于分配给核心 c 和工作包 p 的、索引为 v 的保证人，我们将工作报告 r 定义为：

$$r = \Xi(p, c) \tag{15.1}$$

这样的保证人能够安全地生成并传播有效载荷 (s, v)。组件 s 是按照方程 11.26 生成的，具体而言，它是利用验证者注册的 Ed25519 密钥对有效载荷 l 进行签名而得到的：

$$l = \mathcal{H}(\mathcal{E}(c, r)) \tag{15.2}$$

为了提升利润，保证人需确保工作结果符合第 11.4 节所述保证外部数据中规定的所有预期条件，包括上下文有效性和授权池中的授权。虽然不满足这些条件不会受到惩罚，但会导致区块创建者拒绝包含该工作包，进而影响奖励。

高级节点可通过预测报告送达区块创建者时链的状态，来提高报告被纳入区块的可能性。相比之下，简单节点在验证工作报告时可能仅参考当前的链头。为减轻工作量，节点应在评估报告工作结果的 Ψ_R 函数之前，完成所有必要的评估。

一旦确认工作包具有保证价值，保证人应通过寻求核心共识来避免工作浪费。他们应将工作包发送给同一核心上可能尚不了解该工作包的其他保证人。

为了减轻区块创建者的工作量并提升预期利润，保证人应根据工作报告、核心索引以及包括自己和尽可能多其他人的证明集，来构建其核心的下一个保证外部数据。

为降低因反垃圾邮件措施而被区块创建者忽视的风险，保证人应控制签署工作报告的频率，平均每个时段不超过两份。

16. 可用性保障

当验证者拥有处于待可用状态的工作报告的全部对应擦除编码数据块时，应发布保证书。要获取任何工作报告的保证书，验证者需具备两类数据：

首先，他们需获取该报告数据包的擦除编码数据块。此数据块的有效性可通过工作报告的工作包擦除根及相应的默克尔包含证明来验证，证明其位置正确。保证书提供者需包含此证明。该数据块用于确认工作报告的有效性和完整性，一旦工作报告审核通过，即可不再保留。在此之前，验证者可按需申请获取。

其次，验证者还需持有每个由段根引用的导出段所对应的擦除编码数据块，这些数据块需保留 28 天，并根据验证者的需求随时提供。

17. 审计与评判

我们的审计与评判系统在理论上与 Jeff Burdges 及 Cevallos 等人在 2024 年提出的 ELVES 系统中的机制具有等价性。关于该机制的全面安全性分析，请参见他们的论文。在术语使用上，两者有所区别： ELVES 系统中的“支持（backing）”对应我们这里的“保证（guaranteeing）”，“批准（approval）”对应“审计（auditing）”，“包含（inclusion）”则对应“累积（accumulation）”。

17.1 概述

审计流程要求每个节点从包含可用工作报告的每个 JAM Chain 区块（即 **W** 区块）中选取一组随机但固定的工作报告进行获取、评估，并发布判断。在进行评估前，节点需声明其需求并提供证明。之后，在特定的共同时间点，若声明的意图在合理时间内未获得肯定判断，或出现了否定判断，节点可能需要从更多 **W** 区块中评估工作报告，这些额外评估的报告集合称为“份额（tranches）”。

若在某时间节点，一个工作报告的所有声明意图均获得肯定判断，则该工作报告即视为已审计。当某个区块内所有新可用的工作报告均被审计后，该区块即被认为已审计。节点最终确认一个区块的前提是该区块已被其认定为已审计。值得注意的是，虽然最终会对区块是否已审计达成共识，但在区块被最终确认时，可能尚未形成全面共识，但这并不影响系统的加密经济保障。

在常规操作中，对工作报告的否定判断极为罕见，且审计阶段通常不产生直接后果。若出现否定判断，可能产生以下情况之一：若仍有超过 $^{2}/_{3}\,\mathrm{V}$ 的肯定判断，发布否定判断的验证者可能因无效操作而受到惩罚；若超过 $^{1}/_{3}\,\mathrm{V}$ 的验证者给出否定判断，则包含该工作报告的区块将被列入禁止列表，该区块及其所有后续区块都将被忽略，不得在其上构建。在所有情况下，一旦获得足够投票，区块作者可构建一个判断外在（judgment extrinsic），并将其上链以表示最终结果。详情请参阅第 10 节。

所有公告和判断将与描述签名材料的元数据一同发布给所有验证者。验证者在收到确定数据后，应更新其视角（后文定义为 J 和 A）。

17.2 数据获取

对于每份待审计的工作报告，我们会利用其擦除根（erasure-root），从足够数量的验证者（assurers）处请求擦除编码块（erasure-coded chunks）。通过高效的网络协议，我们会从每个保证者那里一次性获取与工作包超级块（work-package super-chunks）、自证明导入超级块（self-justifying imports super-chunks）及外在段超级块（extrinsic segments super-chunks）相对应的三项内容。

我们可以通过核对哈希值来验证工作包的重建，确保其与工作报告中规范部分的哈希值一致。同样地，我们也可以验证各外部段，只需核对它们的哈希值是否分别与相关工作项中的哈希值相匹配。

最后，我们验证每个导入段时，会依据其必须紧随的连接段，以此核实每个段的哈希值是否均列在对应工作项的引用默克尔根及索引之中。

导出的段无需进行重构，其确定方式与保证过程一致，都是通过执行精炼逻辑来完成的。

工作报告中的工作包规范字段项目，需基于已确认的准确数据重新计算并验证，这相当于复核保证者的步骤，以确保数据的正确性。

17.3 报告选择

每位验证者都应履行对接收到的每个有效区块的审计职责。随着我们即将深入链下逻辑，且无法预设共识存在，我们将自身设定为索引为 v 的特定验证者，并聚焦于最近的某个区块 $\mathbf{B}$，以及与之相关的状态转换项。在此，ρ 代表该区块的前一个核心分配，k 代表其前一个验证者集合，$\mathbf{H}$ 代表其区块头等信息。实际上，这些考量因素需在所有区块中重复进行，且可能同时涉及多个区块的考量。

我们设定一个待审计的工作报告序列 $\mathbf{Q}$，其长度与核心数量相等。$\mathbf{Q}$ 作为核心索引的映射，指向刚变得可用且待审计的工作报告。若某核心上无报告可用，则对应位置标记为 $\emptyset$（空）。形式上：

$$\mathbf{Q} \in \left[\!\left[\mathbb{W}?\right]\!\right]_{\mathrm{C}} \tag{17.1}$$

$$\mathbf{Q} \equiv \left[\begin{cases} \rho[c]_w & \text{if } \rho[c]_w \in \mathbf{W} \\ \emptyset & \text{otherwise} \end{cases} \middle|\, c \lessdot \mathbb{N}c \right] \tag{17.2}$$

我们将其定义为基于专为初始审计批次产生的可验证随机量 s_0 来确定：

$$s_0 \in \mathbb{F}^{[]}_{\kappa[v]_b}\langle \mathcal{X}_U \frown \mathcal{Y}(\mathbf{H}_v)\rangle \tag{17.3}$$

(17.4) $$\mathrm{X}_U = \$\mathrm{jam\_audit}$$

接着，我们可以把 $\mathbf{a}_0$ 界定为，通过可验证的随机挑选方式，从十个核心中选出的一个待审计的非空项目：

(17.5) $$\mathbf{a}_0 = \{(c, w) | (c, w) \in \mathbf{p}_{\ldots+10}, w \neq \emptyset\}$$

(17.6) $$\text{where } \mathbf{p} = \mathcal{F}([(c, \mathbf{Q}_c) | c \leqslant \mathbb{N}_\mathrm{C}], r)$$

(17.7) $$\text{and } r = \mathcal{Y}(s_0)$$

每 8 秒（$\mathsf{A} = 8$）一个新的时间槽结束后，就会启动一个新的审计批次。在这个批次中，我们可能会确定需要审计更多的核心，这些被选中的核心项目被称为当前批次 n 中的 $\mathbf{a}_n$。形式上：

(17.8) $$\text{let } n = \left\lfloor \frac{\mathcal{T} - \mathsf{P} \cdot \mathbf{H}_t}{\mathsf{A}} \right\rfloor$$

新批次可能包含 $\mathbf{Q}$ 中的项目，原因有两个：一是收到了负面评价；二是前一批次的评价数量未达到该批次公告的要求。在第一种情况下，验证者必须对工作报告进行评价。而在第二种情况下，则需要使用一个新的专用可验证随机函数（VRF）来决定是否由我们来进行审计并给出评价。

在所有情况下，我们都会发布一份签名公告，明确指出我们认为需要审计的核心，并附上采用的可验证随机函数（VRF）签名的证明。同时，公告中还会包含前一批次中其他验证者发布但尚未匹配到评价的公告。这样，其他所有验证者都能对这份公告进行验证。发布公告即视为达成了一项协议，意味着我们承诺无论未来获取到何种信息，都将完成审计工作。

形式上，对于每一个批次 n，我们都会确保公告声明得到发布并传达给所有其他验证者，同时附带我们的验证者索引 v、证据 s_n 以及全部签名数据。验证者发布的公告声明必须满足集合 S 的要求：

(17.9) $$S \equiv \mathbb{E}_{\kappa[v]_e}\langle \mathbf{X}_I \doteqdot n \frown \mathbf{x}_n \frown \mathcal{H}(\mathbf{H}) \rangle$$

(17.10) $$\text{where } \mathbf{x}_n = \mathcal{E}([\mathcal{E}_2(c) \frown \mathcal{H}(w) | (c, w) \in \mathbf{a}_n])$$

(17.11) $$\mathrm{X}_I = \$\mathrm{jam\_announce}$$

我们将 A_n 定义为对批次 n 中需审计的工作报告（通过其对应的核心识别）所涉及的验证者的认知，这一认知基于其他验证者的公告（如前文所述）。在批次 n 成为当前处理对象之前，我们无法对其进行精确评估。但对自己需审计的内容，我们有着明确的认识。

(17.12) $$A_n : \mathbb{W} \to \wp\langle \mathbb{N}_\mathrm{V} \rangle$$

(17.13) $$\forall (c, w) \in \mathbf{a}_0 : v \in q_0(w)$$

我们进一步将 $J_\top$ 和 $J_\bot$ 分别定义为已知的对每个工作报告核心给出正面和负面判断的验证者索引集合，而不考虑这些判断是在哪个批次中作出的。

(17.14) $$J_{\{\bot,\top\}} : \mathbb{W} \to \wp\langle \mathbb{N}_\mathbb{V} \rangle$$

我们可以根据尚未给出判断但已知需审计的验证者数量，为首个批次后的各批次定义$\mathbf{a}_n$。若信息延迟到达导致 $\mathbf{a}_n$ 发生变化，节点应重新评估并据此采取相应措施。

因此，为了定义首个批次后的 $\mathbf{a}_n$ ，我们可以采用一个新的可验证随机函数（VRF），该函数将作用于未出席验证者的集合上。

(17.15) $$\forall n > 0:$$

$$s_n(w) \in \mathbb{F}^{[]}_{\kappa[v]_b}\langle \mathrm{X}_U \frown \mathcal{Y}(\mathbf{H}_v) \frown \mathcal{H}(w) \shuffle n \rangle$$

(17.16) $$\mathbf{a}_n \equiv \{\frac{\mathrm{V}}{256\mathrm{F}}\mathcal{Y}(s_n(w))_0 < m_n | w \in \mathbf{Q}, w \neq \emptyset\}$$

$$\text{where } m_n = |A_{n-1}(w) \setminus J_\top(w)|$$

我们设定偏差因子 $\mathrm{F} = 2$，意味着若前一批次有验证者缺席，则预期需有两个验证者对一份工作报告作出判断。Jeff Burdges、Cevallos 等人 2024 年的模型表明，这是最佳选择。

后续的审计公布方式需与首次审计保持一致。若因新信息的获取导致审计需求减少（即原先缺席的验证者补回了正面判断），则已公布的审计应继续执行并公布其结果。若新信息的获取使得审计需求增加（即发现了新的公布但未伴随判断），我们将公开宣布将开展额外的审计。

随着时间的推移，n 逐渐增大，$\mathbf{a}_n$ 变得明确，进而界定了我们的审计职责。我们需努力重建所有工作包及其对应的、我们必须审计的每份工作报告所必需的数据。这可以通过向三分之一的验证者请求擦除编码的数据块来实现。或者，为了更快捷地获取数据，我们也可以向合作的第三方（如原始担保人）请求原像。

因此，对于任意工作报告 w ，我们确信能够获取其对应的候选工作包编码 $F(w)$ 。这个编码要么源自通过擦除编码的默克尔根验证后重建的擦除编码数据块，要么直接来自工作包哈希的原像。随后，我们将对这个候选数据块进行解码，以恢复出一个完整的工作包。

除了工作包，我们还假定能够以同样的方式，即从三分之一的验证者处请求并重建擦除编码的数据块，来获取与工作包相关联的所有清单数据。

接着，我们尝试复现核心报告，以获取e_n ，即实现从核心到评估结果的映射：

(17.17) $$\forall (c, w) \in \mathbf{a}_n : e_n(w) \Leftrightarrow \begin{cases} w = \Xi(p, c) & \text{if } \exists p \in \mathbb{P} : \mathcal{E}(p) = F(w) \\ \bot & \text{otherwise} \end{cases}$$

请注意，若解码失败，则表明工作报告无效。

基于上述映射，验证者会做出一组判断，记为 $\mathbf{j}_n$ 。

(17.18) $$\mathbf{j}_n = \{\mathcal{S}_{\kappa[v]_e}(\mathrm{X}_{e(w)} \frown \mathcal{H}(w)) | (c, w) \in \mathbf{a}_n\}$$

所有判断 $\mathbf{j}_*$ 都应公之于众，供其他验证者参考，以便他们构建自己对 J 的认知。在出现负面判断的情况下，这些判断还可以作为外部证据，用于执行争议（$\mathbf{E}_D$）的处理。

我们认为，工作报告在以下两种情形下视为已审计：一是该报告没有负面判断，并且存在一个批次，其中包含了我们认为需审计的所有验证者的正面判断；二是超过三分之二的验证者对该工作报告给出了正面判断。

(17.19) $$U(w) \Leftrightarrow \bigvee \begin{cases} J_\bot(w) = \emptyset \wedge \exists n: A_n(w) \subset J_\intercal(w) \\ |J_\intercal(w)| > 2/3\mathrm{V} \end{cases}$$

当所有提供的工作报告均被视为已审计时，我们的区块 $\mathbf{B}$ 即可视为已审计，这一条件我们称之为 $\mathbf{U}$ 。形式上：

(17.20) $$\mathbf{U} \Leftrightarrow \forall w \in \mathbf{W}: U(w)$$

对于任意区块，在确认其已审计（即 $\mathbf{U} = \intercal$ ）之后，我们方可在 GRANDPA 中进行投票，以使其最终确定。更多详情，请参阅第 19 节。

此外，我们明确排除那些至少有 $1/3$ 验证者判断为无效的报告所累积的链。任何包含这类区块的链均不具备继续构建新区块的资格。我们定义最佳区块为：在构建新区块时所选的区块，它所在的链包含最多的常规 Safrole 区块，且不含任何被排除的区块。在实际操作中，这可能意味着需要回溯到更早的链头或选择其他分支链。

作为区块创作者，我们在区块中纳入了一项判断外在数据，该数据将收集的判断签名整合并在链上公布。当遇到非有效判断（即确认有效的判断未达到三分之二加一的要求）时，我们将在即将包含这些非有效工作报告的区块中引入这一外在数据。这项非有效判断的外在数据会将它们从待处理的工作报告列表中移除，即 ρ 。有关此内容的更多细节，请参阅第 10 节。

18. BEEFY 分发

对于每个由验证者导入并最终确定的区块 $\mathbf{B}$，该验证者需遵循 Hopwood 等人在2020年提出的方法，在 BLS12-381 曲线上生成一个 BLS 签名。此签名用于确认该区块最新 BEEFY MMR（默克尔多路复用树）的 Keccak 哈希值。签名及其相关材料应公开发布并自由分发。这些签名可以进一步聚合，从而为第三方系统提供一个简洁且有效的最终性证明。签名及聚合机制的具体细节由 Jeff Burdges、Ciobotaru 等人在 2022年进行了完整阐述。

形式上，我们设定 $\mathbf{F}_v$ 为验证者索引 v 的签名承诺，并将此承诺进行公布。

(18.1) $$\mathbf{F}_v \equiv \mathcal{S}_{\kappa'_v}\left(\mathcal{X}_B \frown \mathcal{H}_K(\mathcal{E}_M(\text{last}(\beta)_\mathbf{b}))\right)$$

(18.2) $$\mathrm{X}_B = \$\text{jam\_ beefy}$$

19. GRANDPA 与最佳链

节点依据 Stewart 和 Kokoris-Kogia 在 2020年提出的 GRANDPA 协议参与运作。

我们将最新且最终确定的区块称作 $\mathbf{B}^\natural$，类似地，所有与区块和状态相关的术语都会用上标进行标识。我们认为最佳区块 $\mathbf{B}^\flat$ 是从满足以下条件的可接受区块集合中挑选出来的：

- 以最终确定的区块为其祖先。
- 在我们观测的范围内，不包含任何存在分歧（即同一时间槽内出现两个有效区块）且尚未最终确定的区块。
- 被视为已经过验证的。

形式上：

(19.1) $$\mathbf{A}(\mathbf{H}^\flat) \ni \mathbf{H}^\natural$$

(19.2) $$\mathbf{U}^{\flat} \equiv \mathbf{T}$$

(19.3) $$\nexists \mathbf{H}^A, \mathbf{H}^B : \wedge \begin{cases} \mathbf{H}^A \neq \mathbf{H}^B \\ \mathbf{H}^A_T = \mathbf{H}^B_T \\ \mathbf{H}^A \in \mathbf{A}(\mathbf{H}^{\flat}) \\ \mathbf{H}^A \notin \mathbf{A}(\mathbf{H}^{\natural}) \end{cases}$$

在这些可接受的区块里，我们应选择包含最多使用密封密钥票证（而非备用密钥）的作者祖先区块的那个作为最佳头区块，并让参与者在这条链上进行 GRANDPA 投票。

具体而言，我们的目标是选定一个 $\mathbf{B}^{\flat}$，以使其对应的 m 值最大化，其中 m 值是指：

(19.4) $$m = \sum_{\mathbf{H}^A \in \mathbf{A}^{\flat}} \mathbf{T}^A$$

20. 讨论

20.1 技术特性

总体而言，我们的目标是配置 1,023 名验证者，每个核心分配 3 名，且要求每位验证者在一个时段内平均完成 10 次审计。据此，每个工作报告将涵盖 30 次审计结果，使得 JAM 能够无信任地处理并整合每个时段的 341 个工作包。

我们假定节点硬件为现代配置，具体为：16 核CPU、64GB 内存、8TB 辅助存储以及 0.5GbE 网络。

我们的性能模型预估 CPU 时间的分配大致如下：

	比例
审计	$^{10}/_{16}$
默克尔化	$^{1}/_{16}$
区块执行	$^{2}/_{16}$
GRANDPA 和 BEEFY	$^{1}/_{16}$
纠删编码	$^{1}/_{16}$
网络及其他	$^{1}/_{16}$

网络带宽需求预估如下：

吞吐量，兆字节/时隙	*Tx*	*Rx*
保证	106	48

确保	144	13
审计	0	133
创建	53	87
GRANDPA 和 BEEFY	4	4
总计	304	281
隐含带宽，兆字节/秒	387	357

因此，为确保 500MB/s 连接稳定，网络需预留足够的冗余空间，以应对其他验证节点及公共连接的需求。尽管区块发布具有突发性，验证节点仍需确保峰值带宽更高。

在这些条件下，我们预测网络的数据可用性总容量将达到 2PB ，每个节点最多可分配 6TB 用于存储。内存使用预估如下：

	GB	
审计	20	2 × 10个 PVM 实例
区块执行	2	1个 PVM 实例
状态缓存	40	
其他	2	
总计	64	

大致估算，Polkadot 中继链上每条平行链平均占用空间约为 2MB；一个 40GB 的状态能存储大约 20,000 条平行链的信息。

JAM core 的“虚拟硬件”实际上是一个普通的 CPU 内核，它在整个六秒周期内以正常速度的 25% 至 50% 运行。该虚拟硬件通常能提供和处理平均 2MB/s 的通用 输入/输出（I/O），同时使用最多 2GB 的 RAM。这里的 输入/输出包括从 JAM Chain 状态中进行的任何无需信任的历史读取操作。此外，该虚拟硬件还支持对半静态原像查找数据库进行无限次读取。每个工作包可占用硬件资源，在六秒内执行任意代码，生成最多 48KB 的结果。随后，该结果有权在同一机器上获得 10 毫秒的处理时间，期间无“外部”输入/输出（除该结果外），但可即时访问完整的 JAM Chain 状态，并可能更新所属服务的状态。

20.2 性能展示

就纯处理能力而言，JAM 机器架构能提供极高水平的同质化无信任计算。但需注意，JAM 的核心模型属于经典并行计算架构，为充分发挥其效能，设计方案需在一定程度上兼顾这一特点。因此，在 JAM 上出现与现有用例具有相似语义的新用例之前，直接将其与现有系统进行比较颇具难度。不过，在做出一些合理假设的前提下，我们仍可进行一些大致的比较。

20.2.1 与 Polkadot 的比较

Polkadot 目前最多能验证 80 条平行链，每条链在每六秒的周期内，有一秒用于原生计算，并能处理 5MB 的输入/输出。这大约相当于 13 倍原 生 CPU 的总计算能力，且每日总分布式可用性约为 67MB/s。而 Accumulation（累积）的功能超出了 Polkadot 的能力范围，因此无法进行直接比较。

相比之下，在我们的基础模型中，JAM 预计能达到单个原生 CPU 内核计算负载的大约 85 倍，并且分布式可用性高达 682MB/s 。

20.2.2 简单转账

我们尝试对每秒能处理的事务数量进行建模, 其中每个事务均涉及签名验证及两个账户余额的修改。在明确具体设计细节前，需先做一些基本假设。我们的初步模型设想是将 JAM 内核（即精简版）仅用于事务验证和账户查找，而 JAM Chain 则负责在状态中保存并更新余额。由于大部分所需的输入/输出操作都将同步进行，这种方法的性能可能并不出众，但它能为我们提供一个坚实的基础。

一个 12MB 的工作包能容纳约 96,000 个事务，每个事务大小为 128 字节。然而，48KB 的工作结果仅能编码大约 6,000 个账户更新，每项更新包含 4 字节的账户索引和 4 字节的余额，因此每个工作包实际能处理的事务上限为 3,000 个，总 TPS（每秒事务数）可达 171,000 。通常，这 8 个字节的账户和余额信息可能可压缩至一到两个字节，从而略微提升最大吞吐量。我们预估，通过高度并行的 Merkle 化处理，状态更新的读写速度可达每秒 50 万至 100 万次，这意味着根据系统瓶颈，TPS 大致在 25 万到 35 万之间。

一个更精细的模型是，让 JAM 内核同时负责余额更新和事务验证。我们需预设，能在工作包间以一定效率对状态及操作这些状态的事务进行划分，且 12MB 的工作包将在事务数据与状态见证数据间进行分配。基于我们的基础模型预测，若将 32 位账户系统分页，每页含 210 个账户，每个事务大小为 128 字节，并假定仅约 1% 的预言机账户有效，那么根据划分方式和使用特点，平均 TPS 有望突破 140 万。分区可通过固定分片（即状态分片）、旋转分区模式或动态分区（需特定排序）来实现。

值得注意的是，我们预测这两种模型在计算方面均不会构成瓶颈，因此事务设计可以更加复杂，甚至可以采用更灵活的加密技术或智能合约功能，而无需担心对性能造成显著影响。

20.2.3 吞吐量计算

每秒事务数（TPS）作为衡量分布式系统计算能力的指标并不恰当，因此我们转向另一个更侧重于计算的基准测试：以太坊虚拟机（EVM）。即将迎来十周年的以太坊网络，作为通用去中心化计算的典范，是一个合理的衡量标准。它能维持125 万 gas/秒的计算及输入/输出速率，峰值吞吐量可达这一数字的两倍。EVM 的 gas 指标旨在成为与时间成正比的指标，用于预测和限制程序执行。鉴于两平台间存在字长、字节序、栈/寄（stack/register）存器架构及内存模型等差异，要准确比较 PVM（假设为某种虚拟机）与 EVM 的吞吐量颇具挑战，且难免带有主观性。不过，我们仍会尝试给出一个合理的数值范围。

EVM 的 gas 并不直接对应于原生执行，因为它还涵盖了状态读/写以及事务输入数据的处理，这意味着 EVM 每秒能处理的最大操作组合为：595 次存储读、57 次存储写、125 万计算 gas 以及78KB 输入数据，这些操作之间可以相互平衡。[13]

[13] 最新的"准分片"技术改进后，能接受每秒 87.3 千字节的已提交数据，但这些数据在内部状态中不可直接访问，因此图中未展示，且即使计入，对结果影响也甚微。

由于我们未找到关于存储（I/O）与纯计算之间典型分配的具体分析，为了做出一个极为保守的估算，我们假设 EVM 同时执行这四种操作。但实际上，我们预计它平均能执行每种操作的四分之一左右。

我们的实验[14]显示，在现代高端消费级硬件上，采用高质量的 EVM 实现来处理纯计算任务时，吞吐量预计在 100 到 500 gas/微秒之间（实验中特别采用了 Odd-Product、Triangle-Number 算法及几种 Fibonacci 计算的实现）。为了与 PVM（假设为另一种虚拟机）进行保守的比较，我们提议将 EVM 代码转换为 PVM 代码，并在 Polkavm 原型上重新执行测试[15]。

为了估算 EVM gas/微秒在内存和输入/输出密集型工作负载中的合理下限，我们参考了 EVM 在无许可环境下的实际应用情况。以 Moonbeam 网络为例，在考虑到其在相对保守的 Polkadot 硬件平台上执行重新编译的 WebAssembly 平台时的速度下降后，其吞吐量约为 100 gas/微秒。因此，我们断定，在计算性能方面，现代高端消费级硬件上，1 微秒大约能处理 100 到 500 个 EVM gas[16] 。

基准测试和回归测试的结果显示, 原型 PVM（假设为某种虚拟机）引擎对程序代码有大约每字节 5 纳秒的固定预处理开销。并且，至少对于算术密集型任务来说，与 EVM（以太坊虚拟机）执行相比，PVM 的边际系数仅为 1.6-2%，这意味着其渐近加速比高达约 50-60 倍。对于预期计算时间约为 1 秒的 1MB 大小的机器代码，编译成本所占时间比例仅为 0.5% 。[17] 对于那些与 256 位 EVM 指令集架构不兼容的代码，我们预计在 PVM 上的执行时间会有显著提升。但要确信这些加速比具有广泛适用性，还需进一步的研究工作。

若我们接受预处理在执行中所耗时间与边际成本相当（例如，程序庞大但运行迅速的情况），并假定 PVM 的计量方式意味着有 2 倍的安全速度开销，那么一个 JAM 核心预计能处理的工作量大约相当于 1,500 EVM gas/微秒。鉴于分析的粗略性，我们合理推测其实际值可能在此范围的上下三倍之内波动，即介于 500 至 5,000 EVM gas/微秒之间。

每个 JAM 核心能提供 2MB/s 的带宽，这涵盖了状态输入/输出以及新数据（如交易）的传输。写入操作对核心而言成本较低，仅需通过哈希运算确定最终的更新默克尔根。然而，读取操作需进行见证，若采用含有一百万个条目的二叉默克尔树，每次读取大约需 640 字节的见证数据。据此保守估算，每个核心每秒最多能执行约 3000 次读取操作，具体次数取决于带宽中分配给新输入数据的部分。

综合考虑 JAM 网络的所有因素（排除可能提升吞吐量的累积操作），相关数值可提高至 341 倍（但前提是每个计算过程必须独立进行，除非通过状态预言或累积操作与其他过程交互）。与 Polkadot 和以太坊等 Rollup 链设计不同，JAM 网络无需维持状态的持续分片。只要所有更新均通过每个核心每秒 8KB 的处理结果来实施，智能合约状态就能在 JAM Chain 上以统一格式保存，而这些处理结果仅需包含已变更合约状态根的哈希值。

基于我们的建模假设，我们得出以下总结:

	以太坊 L1	JAM 核心	JAM
计算（EVM Gas/微秒）	1.25†	500-5,000	0.15-1.5M

[14] 如需了解更多详情，请访问 https://hackmd.io/@XXX9CM1uSSCWVNFRYaSB5g/HJarTUhJA，我们会根据信息更新情况对此链接内容进行相应更新。

[15] 这个估计是偏保守的，原因在于未考虑源代码最初是为 EVM（以太坊虚拟机）编译的。因此，PVM（虚拟机）机器代码会复制 EVM 的架构特性，可能导致估计结果过于悲观。例如，EVM 中所有算术操作均为 256 位，而原生 64 位的 PVM 却不得不遵循这一规则，即便源代码实际只需 64 位的值。

[16] 我们推测，EVM 指令集架构与现代典型硬件之间的主要架构差异，可能是造成这一巨大差异的部分原因。

[17] 例如，在奇数乘积基准测试中，这是一项纯粹的计算密集型算术任务。在 EVM 上执行需要 58 秒，而在我们的 PVM 原型上执行，包括所有预处理步骤在内，仅需 1.04 秒。

状态写入（每秒）	57†	n/a	n/a
状态读取（每秒）	595†	4K‡	1.4M‡
输入数据（每秒）	78KB†	2MB‡	682MB‡

我们观察到，JAM 的整体预测性能表现出色，其性能可能相当于数千条基本的以太坊 L1 链。这一显著差异主要源于三大因素：一是空间并行性，JAM 在其安全框架下能够托管数百个核心；二是时间并行性，JAM 旨在为核心实现连续执行，并对区块间的大部分计算进行流水线处理，以确保稳定且最优的工作负载；三是平台优化，通过采用与现代硬件架构高度匹配的虚拟机（VM）和 gas 模型来实现。

然而，需要明确的是，这只是一个临时且大致的估算，仅用于以具体数字形式展 JAM 的性能。特别注意的是，该估算未涵盖以下因素：

- 这些数字是根据以太坊的实际性能以及我们对 JAM 性能的建模分析得出的（我们的模型以组件在现实世界的表现为基础）。
- 可能在 JAM 或以太坊上实现的任何第二层（L2）扩展；
- JAM 所隐含的状态分区技术；
- PVM 的尚未固定的 gas 模型；
- PVM 与EVM 之间的比较存在一定的不精确性；
- (†) 以太坊 L1 的所有数据均源自同一渠道，平均而言，每个数据值仅为最大值的四分之一；
- (‡) JAM 的状态读取和输入数据均来自同一渠道，平均来说，这些数据的大小仅为最大可能值的一半。

我们计划将性能实证分析，以及 JAM 与假想的以太坊生态系统（涵盖最大数量的 L2 部署、完整的 Dank 分片及其所需的其他共识机制）的对比研究，作为后续工作来开展。但这部分内容已超出本文的讨论范畴。

21. 结论

我们设计了一种创新的计算模型，该模型能够有效运用现有的加密经济机制，在大幅提升可扩展性的同时，防止状态碎片化问题长期存在，从而确保整体一致性不受损害。我们将这一整体框架命名为“收集-精炼-连接-累积”（collect-refine-join-accumulate）模式。此外，我们还明确界定了该逻辑的在链执行部分，即“连接-累积”（join-accumulate）环节，并将其协议称为 JAM Chain。

我们认为， JAM 模型开创性地找到了一个“最佳平衡点”。与完全同步模型相比，它能在确保安全与弹性的共识环境下执行大量计算。同时，与那些可能导致持续碎片化的模型不同， JAM 模型严格保证了计算的时机，并能将这些计算整合到一个统一的状态机中。

21.1 后续工作

尽管我们可以依据一些基本假设来估算理论计算能力，并与现有系统做出大致对比，但真实的数据依然具有不可替代的价值。我们认为，该模型极有必要开展进一步的实证研究，以便更深入地理解这些理论极限如何在实际中转化为性能表现。同时，对现有协议进行成本分析和比较也将成为我们未来工作的一个重要研究方向。

我们有理由相信， JAM 的设计足以支持托管一项服务，用于验证波卡的平行链。然而，为了深入了解由 PVM 驱动的计量系统所能支持的吞吐量，还需进一步开展原型设计工作。我们将这部分的研究报告留作后续工作来完成。同时，我们也有意略去了关于加密货币、核心时段销售、质押以及常规智能合约功能等高级协议元素的详细阐述。

为了提升协议的实际应用性，我们正考虑对本文所述的协议做出一些可能的调整。这些调整涵盖：

- 在“累积”阶段，实现服务间的同步调用；
- 限制转账功能，以提升“累积”过程中的并行处理效率；
- 在特定条件下，“累积”阶段可能会预留出额外的强大计算能力；
- 在工作包格式中加入默克尔树，从而无需下载整个包即可评估其授权情况。

在此阶段，网络协议尚未明确定义，其详细描述将在后续的提案中补充完成。

当前，验证者的性能尚未实现在链上的追踪。我们计划在 JAM 协议的最终修订版本中，加入对验证者性能的链上追踪功能，但由于其具体格式尚未确定，因此暂时未将其纳入。

22. 致谢

本项工作的很大部分均建立在他人研究成果的基础之上。Web3 基金会的研究团队，特别是 Alistair Stewart 和 Jeff Burdges，开发出了 ELVES，这是波卡的安全机制，为 JAM 实现核心内计算提供了关键支持。该团队还开发了 Sassafras、GRANDPA 和 BEEFY 等其他项目。

Safrole 是 Sassafras 的一个简化版本，其开发过程得到了 Davide Galassi 和 Alistair Stewart 的细致审核。

最初的 CoreJam RFC 在 Bastian Köcher 和 Robert Habermeier 的审阅下得以完善，该提案的核心要素大部分已被采纳并融入到了当前的工作中。

PVM 是对 Jan Bujak 所开发的、经过一定程度简化的 PolkaVM 软件原型的一种正式描述。Cyrill Leutwiler 为本文中所述的 PVM 实证分析部分提供了贡献。

PolkaJam 团队，特别是 Arkadiy Paronyan、Emeric Chevalier 和 Dave Emett，在 JAM 协议底层设计方面发挥了举足轻重的作用，特别是在默克尔化处理和输入/输出设计上。

自发布以来，众多代码的贡献者帮助我们纠正了不少错误，对此我们深表感谢。

此外，我们还要特别感谢杰出的 Lemon Jelly（ 即 Fred Deakin 和 Nick Franglen ），他们创作的三张精美专辑中的首张，其封面艺术为我们的背景设计提供了灵感。

附 录 A. PolkaVM 波卡虚拟机

A.1. 基本定义

我们定义通用的波卡虚拟机（PVM）函数为 Ψ。假设存在单步调用函数 Ψ_1，PVM 则递归地由一系列此类单步状态变更构成，直至满足停止条件。同时，定义 deblob 函数，用于从程序二进制大对象（blob）中抽取指令数据、操作码位掩码及动态跳转表。

$$\Psi: \begin{cases} (\mathbb{Y}, \mathbb{N}_R, \mathbb{N}_G, [\mathbb{N}_R]_{13}, \mathbb{Y}) \to (\{\blacksquare, \lightning, \infty\} \cup \{\Lightsteel, \hbar\} \times \mathbb{N}_R, \mathbb{N}_R, \mathbb{Z}_G, [\![\mathbb{N}_R]\!]_{13}, \mathbb{M}) \\ (\mathbf{p}, i, \varrho, \omega, \mu) \mapsto \begin{cases} \Psi(\mathbf{p}, i', \varrho', \omega', \mu') & \text{if } \varepsilon = \blacktriangleright \\ (\infty, i', \varrho', \omega', \mu') & \text{if } \varrho' < 0 \\ (\varepsilon, 0, \varrho', \omega', \mu') & \text{if } \varepsilon \in \{\lightning, \blacksquare\} \\ (\varepsilon, i', \varrho', \omega', \mu') & \text{otherwise} \end{cases} \\ \text{where} (\varepsilon, i', \varrho', \omega', \mu') = \begin{cases} \Psi_1(\mathbf{c}, \mathbf{k}, \mathbf{j}, \iota, \varrho, \omega, \mu) & \text{if} (\mathbf{c}, \mathbf{k}, \mathbf{j}) = \text{deblob}(\mathbf{p}) \\ (t, \iota, \varrho, \omega, \mu) & \text{otherwise} \end{cases} \end{cases} \tag{A.1}$$

$$\text{deblob}: \begin{cases} Y \to (Y, B, [N_R]) \cup \nabla \\ \mathbf{p} \mapsto \begin{cases} (\mathbf{c}, \mathbf{k}, \mathbf{j}) & \text{if } \exists! \, \mathbf{c}, \mathbf{k}, \mathbf{j} : \mathbf{p} = \mathcal{E}(\mathbf{j})) \sim \mathcal{E}_1(z) \sim \mathcal{E}(|\mathbf{c}|) \sim \mathcal{E}_z(\mathbf{j}) \sim \mathcal{E}(\mathbf{c}) \sim \mathcal{E}(\mathbf{k}), |\mathbf{k}| = |\mathbf{c}| \\ \nabla & \text{otherwise} \end{cases} \end{cases} \tag{A.2}$$

波卡 PVM 的退出原因 $\varepsilon \in \{\blacksquare, \lightning, \infty\} \cup \{\Lightsteel, \hbar\} \times \mathbb{N}_R$，$\varepsilon$ 包括：正常停止 $\blacksquare$、恐慌 $\lightning$、Gas 不足 ∞，或者是主机调用 h（附带主机调用标识符），亦或是页面错误（附带内存地址）。其中，主机调用和页面错误分别用特定符号和内存地址来标识。

A.2. 指令、操作码和跳转距离

程序块 $\mathbf{p}$ 被切分成多个八位字节，这些字节组成了指令数据 $\mathbf{c}$ 和操作码掩码 $\mathbf{k}$，以及动态跳转表 $\mathbf{j}$ 。两者隐含了一条指令序列共同隐含了一条指令序列，进而也确定了一个由终止指令后指令索引构成的基本块序列。

后者，即动态跳转表，是指令数据块中的一系列索引，用于执行动态跳转时的查找。它被编码为一串自然数（即非负整数）序列，每个数都采用相同长度的八位字节表示。这个长度，我们称之为 z，是预先编码好的。

PVM 以八位字节为计算单位来处理指令（而非以整条指令为单位）。因此，我们可以方便地界定哪些八位字节是指令的起始（即操作码八位字节），哪些不是。这就是指令-操作码位掩码 $\mathbf{k}$ 的作用。我们确定，位掩码的长度与指令数据块的长度是一致的。

我们定义了 Skip 函数 skip，它根据指令操作码在 $\mathbf{c}$ 及扩展到 $\mathbf{k}$ 中）的索引位置，返回到达下一个指令操作码所需的八位字节数减一的值。

$$\text{skip}: \begin{cases} \mathbb{N} \to \mathbb{N} \\ i \mapsto \min(24, j \in \mathbb{N} : (\mathbf{k} \curvearrowright [1,1,\ldots])_{i+1+j} = 1) \end{cases} \tag{A.3}$$

Skip 函数在 $\mathbf{k}$ 中添加了一系列设定好的位，以确保对最终指令 $\text{skip}(|\mathbf{c}| - 1)$ 能得出一个确定的结果。

给定指令索引 i，其操作码可简便表示为 c_i，而要前进到下一条指令，所需的八位字节距离是 $1 + \text{skip}(i)$。不过，每条指令的“长度”（即从操作码起算，连续所需的八位字节总数，以完整表达指令含义）是隐含的，且最长不超过 16 个八位字节。

我们将 ζ 定义为与指令 $\mathbf{c}$ 等同，但在其后添加了一个长度不定的零序列，以防止发生越界访问。这样做实际上为最终指令中任何未明确的参数提供了默认值，并确保一旦程序计数器超出程序代码的范围，就会触发陷阱机制。形式上：

(A.4) $$\zeta \equiv \mathbf{c} \frown [0,0,\ldots]$$

A.3. 基本块与终止指令

以下操作码的指令被视为基本块的终止指令；除了陷阱（trap）和顺序执行（fallthrough）情况外，它们指的是那些可能将指令计数器设置为不同于其原值加上该指令 skip 值的指令：

- 陷阱和顺序执行：trap, fallthrough
- 跳转：jump , jump_ind
- 加载并跳转：load_imm_jump, load_imm_jump_ind
- 分支：branch_eq, branch_ne, branch_ge_u , branch_ge_s , branch_lt_u , branch_lt_s , branch_eq_imm , branch_ne_imm
- 立即分支：branch_lt_u_imm , branch_lt_s_imm , branch_le_u_imm , branch_le_s_imm , branch_ge_u_imm , branch_ge_s_imm , branch_gt_u_imm , branch_gt_s_imm

我们将此集合用操作码索引（而非名称）来标识，记为 T。定义基本块起始指令的操作码索引为 ϖ。

(A.5) $$\varpi \equiv [0] \frown [n + 1 + \text{skip}(n) | n < \mathbb{N}_{|\mathbf{c}|} \wedge \mathbf{k}_n = 1 \wedge \mathbf{c}_n \in T]$$

A.4. 单步状态转移

现在我们来定义单步虚拟机的状态转移函数 Ψ_1。

(A.6) $$\Psi_1 \colon \begin{cases} (\mathbb{Y}, \mathbb{B}, [\mathbb{N}_R], \mathbb{N}_R, \mathbb{N}_G, [\mathbb{N}_R]_{13}, \mathbb{M}) \to (\{\text{i}, \blacksquare, \blacktriangleright\} \cup \{\lightning, \hbar\} \times \mathbb{N}_R, \mathbb{N}_R, \mathbb{Z}_G, [\mathbb{N}_R]_{13}, \mathbb{M}) \\ (\boldsymbol{c}, \boldsymbol{k}, \boldsymbol{j}, \imath, \varrho, \omega, \mu) \mapsto (\varepsilon, \imath', \varrho', \omega', \mu') \end{cases}$$

我们将 ε 与机器状态中各项的后验值（以撇号标记）一并定义，具体如下表所示。

通常来说，指令引起机器状态转换时，会满足一系列条件，而指令的定义主要是基于它们对这些常规规则的例外情况。具体来讲，机器会持续运行，指令计数器增加一，剩余 gas 量会根据指令类型相应减少，同时内存（RAM）和寄存器的状态保持不变。形式上：

(A.7) $$\varepsilon = \blacktriangleright, \imath' = \imath + 1 + \text{skip}(\imath), \varrho' = \varrho - \varrho_\Delta, \omega' = \omega, \mu' = \mu \text{ except as indicated}$$

在执行指令时，可能会需要访问内存。若访问的内存索引小于 2^{16}，机器会立即崩溃，状态不再变化，无论该索引看似是否可访问。若给定的内存索引不可访问，则机器状态保持不变，退出原因为读取故障，故障地址为最低不可访问的页地址。同样，若需修改内存却无法进行可变访问，机器状态也保持不变，退出原因为写入故障，故障地址为最低不可写入的页地址。

形式上，设 $\mathbf{r}$ 和 $\mathbf{w}$ 分别为计算 Ψ_1 结果时需检查（读取）和修改（写入）μ 的索引集合。我们定义内存访问异常状态 ε^{μ}，若其不为 ▸（即存在异常），则将独立影响 Ψ_1 的返回结果，具体情形如下：

$$\text{let } \mathbf{x} = \{x : x \in \mathbf{r} \wedge x \bmod 2^{32} \notin \mathbb{V}_{\mu} \vee x \in \mathbf{w} \wedge x \bmod 2^{32} \notin \mathbb{V}^{*}_{\mu}\} \tag{A.8}$$

$$\varepsilon^{\mu} = \begin{cases} \blacktriangleright & \text{if } \mathbf{x} = \{\} \\ \lightning & \text{if } \min(\mathbf{x}) \bmod 2^{32} < 2^{16} \\ \lightning \times \mathbb{Z}_{P}\left[\min(\mathbf{x}) \bmod 2^{32} \div \mathbb{Z}_{P}\right] & \text{otherwise} \end{cases} \tag{A.9}$$

我们为不同的字节宽度，明确了有符号与无符号的转换规则：

$$\mathcal{Z}_{n \in \mathbb{N}} : \begin{cases} \mathbb{N}_{2^{8n}} \to \mathbb{Z}_{-2^{8n-1} \ldots 2^{8n-1}} \\ a \mapsto \begin{cases} a & \text{if } a < 2^{8n-1} \\ a - 2^{8n} & \text{otherwise} \end{cases} \end{cases} \tag{A.10}$$

$$\mathcal{Z}^{-1}_{n \in \mathbb{N}} : \begin{cases} \mathbb{Z}_{-2^{8n-1} \ldots 2^{8n-1}} \to \mathbb{N}_{2^{8n}} \\ a \mapsto (2^{8n} + a) \bmod 2^{8n} \end{cases} \tag{A.11}$$

$$\mathcal{B}_{n \in \mathbb{N}} : \begin{cases} \mathbb{N}_{2^{8n}} \to \mathbb{B}_{8n} \\ x \mapsto \mathbf{y} : \forall i \in \mathbb{N}_{8n} : \mathbf{y}[i] \Leftrightarrow \left\lfloor \frac{x}{2^{i}} \right\rfloor \bmod 2 \end{cases} \tag{A.12}$$

$$\mathcal{B}^{-1}_{n \in \mathbb{N}} : \begin{cases} \mathbb{B}_{8n} \to \mathbb{N}_{2^{8n}} \\ \mathbf{x} \mapsto y : \sum_{i \in \mathbb{N}_{8n}} \mathbf{x}_{i} \cdot 2^{i} \end{cases} \tag{A.13}$$

$$\overleftarrow{\mathrm{B}}_{n \in \mathbb{N}} : \begin{cases} \mathbb{N}_{2^{8n}} \to \mathbb{B}_{8n} \\ x \mapsto \mathbf{y} : \forall i \in \mathbb{N}_{8n} : \mathbf{y}[8n - 1 - i] \Leftrightarrow \left\lfloor \frac{x}{2^{i}} \right\rfloor \bmod 2 \end{cases} \tag{A.14}$$

$$\overleftarrow{\mathrm{B}}^{-1}_{n \in \mathbb{N}} : \begin{cases} \mathbb{B}_{8n} \to \mathbb{N}_{28n} \\ \mathbf{x} \mapsto y : \sum_{i \in \mathbb{N}_{8n}} \mathbf{x}_{8n-1-i} \cdot 2^{i} \end{cases} \tag{A.15}$$

立即数参数采用小端格式编码，最高位作为符号位。为紧凑编码，可省略更高位字节。若值的最高有效位（MSB）为零，则省略字节视为零；若为非零，则视为 255 。此方式可对正负数进行紧凑表示。因此，我们定义了有符号扩展函数 $\mathcal{X}_{n}:$，作用于 n 个字节的输入：

$$\mathcal{X}_{n \in \{0,1,2,3,4,8\}} : \begin{cases} \mathbb{N}_{2^{8n}} \to \mathbb{N}_{R} \\ x \mapsto x + \left\lfloor \frac{x}{2^{8n-1}} \right\rfloor (2^{64} - 2^{8n}) \end{cases} \tag{A.16}$$

程序计数器因静态跳转、调用或分支而变更时，必须指向基本块的开头，否则程序将崩溃。此规则无例外。形式上：

$$\text{branch}(b, C) \Longrightarrow (\varepsilon, \iota') = \begin{cases} (\blacktriangleright, \iota) & \text{if } \neg C \\ (\lightning, \iota) & \text{otherwise if } b \notin \varpi \\ (\blacktriangleright, b) & \text{otherwise} \end{cases} \tag{A.17}$$

对于动态计算的跳转指令，必须使用跳转表j的索引地址。由于工具的一个特性[18]，我们将指令所需的动态地址设定为：跳转表索引加1后，再乘以跳转对齐因子 $Z_A = 2$ 。

[18] LLVM作为广泛应用的代码生成后端，在代码生成时要求并假定动态跳转的地址总是内存对齐的。由于我们的工具依赖此特性，因此我们必须接受这一假设。

对于程序计数器的任何非常规修改，其目标代码索引必须指向基本块的开头，否则程序将崩溃。形式上：

$$\text{djump}(a) \Longrightarrow (\varepsilon, \iota') = \begin{cases} (\blacksquare, \iota) & \text{if } a = 2^{32} - 2^{16} \\ (\lightning, \iota) & \text{otherwise if } a = 0 \vee a > |\mathbf{j}| \cdot Z_A \vee a \bmod Z_A \neq 0 \vee \mathbf{j}_{(a/Z_A)-1} \notin \varpi \\ (\blacktriangleright, \mathbf{j}_{(a/Z_A)-1}) & \text{otherwise} \end{cases} \tag{A.18}$$

A.5. 指令表

注意，若操作码未在下表中定义，则该指令视为无效，并将导致崩溃，$\varepsilon = \lightning$。

我们假定跳过长度 ℓ 已有明确定义：

$$\ell \equiv \text{skip}(\iota) \tag{A.19}$$

A.5.1. 无参数

ζ_i	名称（Name）	$\varrho\Delta$	状态变更（Mutations）
0	trap	0	$\varepsilon = \lightning$
1	fallthrough	0	

A.5.2. 带有一个立即数参数的指令

$$\text{let } l_X = \min(4, \ell), \nu_X \equiv \mathcal{X}_{l_X}(\mathcal{E}^{-1}_{l_X}(\zeta_{i+1 \cdots + l_X})) \tag{A.20}$$

ζ_i	名称（Name）	$\varrho\Delta$	状态变更（Mutations）
10	ecalli	0	$\varepsilon = \hbar \times \nu_X$

A.5.3. 带有一个寄存器和一个扩展宽度立即数参数的指令

$$\text{let } r_A = min(12, \zeta_{t+1} \bmod 16), \omega'_A \equiv \omega'_{r_A}, \nu_X \equiv \mathcal{E}^{-1}_8(\zeta_{t+2 \cdots + 8}) \tag{A.21}$$

ζ_i	名称（Name）	$\varrho\Delta$	状态变更（Mutations）
20	load_imm_64	0	$\omega'_A = \nu_X$

A.5.4. 带有两个立即数参数的指令

$$\begin{gathered} \text{let } l_X = min\left(4, \zeta_{s+1} mod\ 8\right), \nu_X = \mathcal{X}_{l_X}\left(\mathcal{E}^{-1}_{l_X}(\zeta_{s+2 \cdots + l_X})\right) \\ \text{let } l_Y = min(4, max(0, \ell - l_X - 1)), \nu_Y \equiv \mathcal{X}_{l_Y}(\mathcal{E}^{-1}_{l_Y}(\zeta_{s+2+l_X \cdots + l_Y})) \end{gathered} \tag{A.22}$$

ζ_i	名称（Name）	$\varrho\Delta$	状态变更（Mutations）

30	store_imm_u8	0	$\mu^{\circlearrowleft}_{\nu_X} = \nu_Y \bmod 2^8$
31	store_imm_u16	0	$\mu^{\circlearrowleft}_{\nu_X \cdots +2} = \mathcal{E}_2(\nu_Y \bmod 2^{16})$
32	store_imm_u32	0	$\mu^{\circlearrowleft}_{\nu_X \cdots +4} = \mathcal{E}_4(\nu_Y \bmod 2^{32})$
33	store_imm_u64	0	$\mu^{\circlearrowleft}_{\nu_X \cdots +8} = \mathcal{E}_8(\nu_Y)$

A.5.5. 带有两个立即数参数的指令

(A.23)
$$\text{let } l_X = \min(4, \imath), \nu_X \equiv \imath + \mathcal{Z}_{l_X}(\mathcal{E}^{-1}_{l_X}(\zeta_{\imath+1 \cdots +l_X}))$$

ζ_i	名称（**Name**）	$\varrho\Delta$	状态变更（**Mutations**）
40	jump	0	branch (ν_X, T)

A.5.6. 带有一个寄存器和一个立即数参数的指令

(A.24)
$$\text{let } r_A = \min(12, \zeta_{t+1} \bmod 16), \omega_A \equiv \omega_{r_A}, \omega'_A \equiv \omega'_{r_A}$$
$$\text{let } l_X = \min(4, \max(0, \imath - 1)), \nu_X \equiv \mathcal{X}_{l_X}(\mathcal{E}^{-1}_{l_X}(\zeta_{l+2 \cdots +l_X}))$$

ζ_i	名称（**Name**）	$\varrho\Delta$	状态变更（**Mutations**）
50	jump_in	0	$\text{djump}((\omega_A + \nu_X) \bmod 2^{32})$
51	load_imm	0	$\omega'_A = \nu_X$
52	load_u8	0	$\omega'_A = \mu^{\circlearrowleft}_{\nu_X}$
53	load_i8	0	$\omega'_A = \mathcal{X}_1(\mu^{\circlearrowleft}_{\nu_X})$
54	load_u16	0	$\omega'_A = \mathcal{E}^{-1}_2(\mu^{\circlearrowleft}_{\nu_X \cdots +2})$
55	load_i16	0	$\omega'_A = \mathcal{X}_2(\mathcal{E}^{-1}_2(\mu^{\circlearrowleft}_{\nu_X \cdots 2}))$
56	load_u32	0	$\omega'_A = \mathcal{E}^{-1}_4(\mu^{\circlearrowleft}_{\nu_X \cdots +4})$
57	load_i32	0	$\omega'_A = \mathcal{X}_4(\mathcal{E}^{-1}_4(\mu^{\circlearrowleft}_{\nu_X \cdots +4}))$
58	load_u64	0	$\omega'_A = \mathcal{E}^{-1}_8(\mu^{\circlearrowleft}_{\nu_X \cdots +8})$
59	store_u8	0	$\mu'^{O}_{\nu_X} = \omega_A \bmod 2^8$
60	store_u16	0	$\mu^{\circlearrowleft}_{\nu_X \cdots +2} = \mathcal{E}_2(\omega_A \bmod 2^{16})$
61	store_u32	0	$\mu^{\circlearrowleft}_{\nu_X \cdots +4} = \mathcal{E}_4(\omega_A \bmod 2^{32})$
62	store_u64	0	$\mu^{\circlearrowleft}_{\nu_X \cdots +8} = \mathcal{E}_8(\omega_A)$

A.5.7. 带有一个寄存器和两个立即数参数的指令

$$\text{let } r_A = min\left(12, \zeta_{i+1} \bmod 16\right), \omega_A \equiv \omega_{r_A}, \omega'_A \equiv \omega'_{r_A}$$

$$\text{let } l_X = min\left(4, \left\lfloor \frac{\zeta_{i+1}}{16} \right\rfloor \bmod 8\right), \nu_X = \mathcal{X}_{l_X}\left(\mathcal{E}^{-1}_{l_X}(\zeta_{i+2\cdots+l_X})\right) \tag{A.25}$$

$$\text{let } l_Y = min(4, \max(0, \iota - l_X - 1)), \nu_Y = \mathcal{X}_{l_Y}(\mathcal{E}^{-1}_{l_Y}(\zeta_{i+2+l_X\cdots+l_Y}))$$

ζ_i	名称（**Name**）	$\varrho\Delta$	状态变更（**Mutations**）
70	store_imm_ind_u8	0	$\mu^{\circlearrowleft}_{\omega_A+\nu_X} = \nu_Y \bmod 2^8$
71	store_imm_ind_u16	0	$\mu^{\circlearrowleft}_{\omega_A+\nu_X\cdots+2} = \mathcal{E}_2(\nu_Y \bmod 2^{16})$
72	store_imm_ind_u32	0	$\mu^{\circlearrowleft}_{\omega_A+\nu_X\cdots+4} = \mathcal{E}_4(\nu_Y \bmod 2^{32})$
73	store_imm_ind_u64	0	$\mu^{\circlearrowleft}_{\omega_A+\nu_X\cdots+8} = \mathcal{E}_8(\nu_Y)$

A.5.8. 带有一个寄存器、一个立即数和一个偏移量的指令

$$\text{let } r_A = \min\left(12, \zeta_{i+1} \bmod 16\right), \omega_A \equiv \omega_{r_A}, \omega'_A \equiv \omega'_{r_A}$$

$$\text{let } l_X = \min\left(4, \left\lfloor \frac{\zeta_{i+1}}{16} \right\rfloor \bmod 8\right), \nu_X = \mathcal{X}_{l_X}\left(\mathcal{E}^{-1}_{l_X}(\zeta_{i+2\cdots+l_X})\right) \tag{A.26}$$

$$\text{let } l_Y = \min(4, \max(0, \iota - l_X - 1)), \nu_Y = \imath + \mathcal{Z}_{l_Y}(\mathcal{E}^{-1}_{l_Y}(\zeta_{\imath+2+l_X\cdots+l_Y}))$$

ζ_i	名称（**Name**）	$\varrho\Delta$	状态变更（**Mutations**）
80	load_imm_jump	0	$\text{branch}(\nu_Y, \mathsf{T}), \omega'_A = \nu_X$
81	branch_eq_imm	0	$\text{branch}(\nu_Y, \omega_A = \nu_X)$
82	branch_ne_imm	0	$\text{branch}(\nu_Y, \omega_A \neq \nu_X)$
83	branch_lt_u_imm	0	$\text{branch}(\nu_Y, \omega_A < \nu_X)$
84	branch_le_u_imm	0	$\text{branch}(\nu_Y, \omega_A \leq \nu_X)$
85	branch_ge_u_imm	0	$\text{branch}(\nu_Y, \omega_A \geq \nu_X)$
86	branch_gt_u_imm	0	$\text{branch}(\nu_Y, \omega_A > \nu_X)$
87	branch_lt_s_imm	0	$\text{branch}(\nu_Y, \mathcal{Z}_8(\omega_A) < \mathcal{Z}_8(\nu_X))$
88	branch_le_s_imm	0	$\text{branch}(\nu_Y, \mathcal{Z}_8(\omega_A) \leq \mathcal{Z}_8(\nu_X))$
89	branch_ge_s_imm	0	$\text{branch}(\nu_Y, \mathcal{Z}_8(\omega_A) \geq \mathcal{Z}_8(\nu_X))$

90	branch_gt_s_imm	0	$\text{branch}(\nu_Y, \mathcal{Z}_8(\omega_A) > \mathcal{Z}_8(\nu_X))$

A.5.9. 带有两个寄存器参数的指令

$$\text{let } r_D = \min(12, (\zeta_{i+1}) \bmod 16), \omega_D \equiv \omega_{r_D}, \omega'_D \equiv \omega'_{r_D}$$
$$\text{let } r_A = \min(12, \left\lfloor \frac{\zeta_{i+1}}{16} \right\rfloor), \omega_A \equiv \omega_{r_A}, \omega'_A \equiv \omega'_{r_A} \tag{A.27}$$

ζ_i	名称（**Name**）	$\varrho\Delta$	状态变更（**Mutations**）
100	move_reg	0	$\omega'_D = \omega_A$
101	sbrk	0	$\omega'_D \equiv \min(x \in \mathbb{N}_R): x \geq h \wedge \mathbb{N}_{x \cdots + \omega_A} \notin \mathbb{V}_\mu \wedge \mathbb{N}_{x \cdots + \omega_A} \subseteq \mathbb{V}^*_{\mu'}$
102	count_set_bits_64	0	$\omega'_D = \sum_{i=0}^{63} \mathcal{B}_8(\omega_A)_i$
103	count_set_bits_32	0	$\omega'_D = \sum_{i=0}^{31} \mathcal{B}_4(\omega_A \bmod 2^{32})_i$
104	leading_zero_bits_64	0	$\omega'_D = \max(n \in \mathbb{N}_{65}) \text{where} \sum_{i=0}^{i<n} \mathcal{B}_8(\omega_A)_i = 0$
105	leading_zero_bits_32	0	$\omega'_D = \max(n \in \mathbb{N}_{33}) \text{where} \sum_{i=0}^{i<n} \mathcal{B}_4(\omega_A \bmod 2^{32})_i = 0$
106	trailing_zero_bits_64	0	$\omega'_D = \max(n \in \mathbb{N}_{65}) \text{where} \sum_{i=0}^{i<n} \mathcal{B}_8(\omega_A)_{63-i} = 0$
107	trailing_zero_bits_32	0	$\omega'_D = \max(n \in \mathbb{N}_{33}) \text{where} \sum_{i=0}^{i<n} \mathcal{B}_4(\omega_A \bmod 2^{32})_{31-i} = 0$
108	sign_extend_8	0	$\omega'_D = \mathcal{Z}_8^{-1}(\mathcal{Z}_1(\omega_A \bmod 2^8))$
109	sign_extend_16	0	$\omega'_D = \mathcal{Z}_8^{-1}(\mathcal{Z}_2(\omega_A \bmod 2^{16}))$
110	zero_extend_16	0	$\omega'_D = \omega_A \bmod 2^{16}$
111	reverse_bytes	0	$\forall i \in \mathbb{N}_8: \mathcal{E}_8(\omega'_D)_i = \mathcal{E}_8(\omega_A)_{7-i}$

注意，上述的 h 代表堆的起始位置，即内存第二个主要部分的开始，其位置根据方程 A.41 定义为 $2Z_Z + Z(|\mathbf{o}|)$。若在不支持此内存布局的 PVM 实例上执行 sbrk 指令，则 $h = 0$。

A.5.10. 带有两个寄存器和一个立即数参数的指令

$$\text{let } r_A = \min(12, (\zeta_{i+1}) \bmod 16), \omega_A \equiv \omega_{r_A}, \omega'_A \equiv \omega'_{r_A}$$

(A.28) $$\text{let } r_B = \min\left(12, \left\lfloor \frac{\zeta_{i+1}}{16} \right\rfloor\right), \omega_B \equiv \omega_{r_B}, \omega'_B \equiv \omega'_{r_B}$$

$$\text{let } l_X = \min(4, \max(0, \iota - 1)), \nu_X \equiv \mathcal{X}_{l_X}(\mathcal{E}^{-1}_{l_X}(\zeta_{\iota+2\cdots+l_X}))$$

ζ_i	名称（**Name**）	$\varrho\Delta$	状态变更（**Mutations**）
120	store_ind_u8	0	$\mu^{\circlearrowleft}_{\omega_B+\nu_X} = \omega_A \bmod 2^8$
121	store_ind_u16	0	$\mu^{\circlearrowleft}_{\omega_B+\nu_X\cdots+2} = \mathcal{E}_2(\omega_A \bmod 2^{16})$
122	store_ind_u32	0	$\mu^{\circlearrowleft}_{\omega_B+\nu_X\cdots+4} = \mathcal{E}_4(\omega_A \bmod 2^{32})$
123	store_ind_u64	0	$\mu^{\circlearrowleft}_{\omega_B+\nu_X\cdots+8} = \mathcal{E}_8(\omega_A)$
124	load_ind_u8	0	$\omega'_A = \mu^{\circlearrowleft}_{\omega_B+\nu_X}$
125	load_ind_i8	0	$\omega'_A = \mathcal{Z}^{-1}_8(\mathcal{Z}_1(\mu^{\circlearrowleft}_{\omega_B+\nu_X}))$
126	load_ind_u16	0	$\omega'_A = \mathcal{E}^{-1}_2(\mu^{\circlearrowleft}_{\omega_B+\nu_X\cdots+2})$
127	load_ind_i16	0	$\omega'_A = \mathcal{Z}^{-1}_8(\mathcal{Z}_2(\mathcal{E}^{-1}_2(\mu^{\circlearrowleft}_{\omega_B+\nu_X\cdots+2})))$
128	load_ind_u32	0	$\omega'_A = \mathcal{E}^{-1}_4(\mu^{\circlearrowleft}_{\omega_B+\nu_X\cdots+4})$
129	load_ind_i32	0	$\omega'_A = \mathcal{Z}^{-1}_8(\mathcal{Z}_4(\mathcal{E}^{-1}_4(\mu^{\circlearrowleft}_{\omega_B+\nu_X\cdots+4})))$
130	load_ind_u6	0	$\omega'_A = \mathcal{E}^{-1}_8(\mu^{\circlearrowleft}_{\omega_B+\nu_X\cdots+8})$
131	add_imm_32	0	$\omega'_A = \mathcal{X}_4((\omega_B + \nu_X) \bmod 2^{32})$
132	and_imm	0	$\forall i \in \mathbb{N}_{64}: \mathcal{B}_8(\omega'_A)_i = \mathcal{B}_8(\omega_B)_i \wedge \mathcal{B}_8(\nu_X)_i$
133	xor_imm	0	$\forall i \in \mathbb{N}_{64}: \mathcal{B}_8(\omega'_A)_i = \mathcal{B}_8(\omega_B)_i \oplus \mathcal{B}_8(\nu_X)_i$
134	or_imm	0	$\forall i \in \mathbb{N}_{64}: \mathcal{B}_8(\omega'_A)_i = \mathcal{B}_8(\omega_B)_i \vee \mathcal{B}_8(\nu_X)_i$
135	mul_imm_32	0	$\omega'_A = \mathcal{X}_4((\omega_B \cdot \nu_X) \bmod 2^{32})$
136	set_lt_u_imm	0	$\omega'_A = \omega_B < \nu_X$
137	set_lt_s_imm	0	$\omega'_A = \mathcal{Z}_8(\omega_B) < \mathcal{Z}_8(\nu_X)$
138	shlo_l_imm_32	0	$\omega'_A = \mathcal{X}_4((\omega_B \cdot 2^{\nu_X \bmod 32}) \bmod 2^{32})$
139	shlo_r_imm_32	0	$\omega'_A = \mathcal{X}_4(\lfloor \omega_B \bmod 2^{32} \div 2^{\nu_X \bmod 32} \rfloor)$
140	shar_r_imm_32	0	$\omega'_A = \mathcal{Z}^{-1}_8(\lfloor \mathcal{Z}_4(\omega_B \bmod 2^{32}) \div 2^{\nu_X \bmod 32} \rfloor)$
141	neg_add_imm_32	0	$\omega'_A = \mathcal{X}_4((\nu_X + 2^{32} - \omega_B) \bmod 2^{32})$
142	set_gt_u_imm	0	$\omega'_A = \omega_B > \nu_X$
143	set_gt_s_imm	0	$\omega'_A = \mathcal{Z}_8(\omega_B) > \mathcal{Z}_8(\nu_X)$

144	shlo_l_imm_alt_32	0	$\omega'_A = \mathcal{X}_4((\nu_X \cdot 2^{\omega_B \bmod 32}) \bmod 2^{32})$
145	shlo_r_imm_alt_32	0	$\omega'_A = \mathcal{X}_4(\lfloor \nu_X \bmod 2^{32} \div 2^{\omega_B \bmod 32} \rfloor)$
146	shar_r_imm_alt_32	0	$\omega'_A = \mathcal{Z}_8^{-1}(\lfloor \mathcal{Z}_4(\nu_X \bmod 2^{32}) \div 2^{\omega_B \bmod 32} \rfloor)$
147	cmov_iz_imm	0	$\omega'_A = \begin{cases} \nu_X & \text{if} \omega_B = 0 \\ \omega_A & \text{otherwise} \end{cases}$
148	cmov_nz_imm	0	$\omega'_A = \begin{cases} \nu_X & \text{if} \omega_B \neq 0 \\ \omega_A & \text{otherwise} \end{cases}$
149	add_imm_64	0	$\omega'_A = (\omega_B + \nu_X) \bmod 2^{64}$
150	mul_imm_64	0	$\omega'_A = (\omega_B \cdot \nu_X) \bmod 2^{64}$
151	shlo_l_imm_64	0	$\omega'_A = \mathcal{X}_8((\omega_B \cdot 2^{\nu} X \bmod 64) \bmod 2^{64})$
152	shlo_r_imm_64	0	$\omega'_A = \mathcal{X}_8(\lfloor \omega_B \div 2^{\nu_X \bmod 64} \rfloor)$
153	shar_r_imm_64	0	$\omega'_A = \mathcal{Z}_8^{-1}(\lfloor \mathcal{Z}_8(\omega_B) \div 2^{\nu_X \bmod 64} \rfloor)$
154	neg_add_imm_64	0	$\omega'_A = (\nu_X + 2^{64} - \omega_B) \bmod 2^{64}$
155	shlo_l_imm_alt_64	0	$\omega'_A = (\nu_X \cdot 2^{\omega_B \bmod 64}) \bmod 2^{64}$
156	shlo_r_imm_alt_64	0	$\omega'_A = \lfloor \nu_X \div 2^{\omega_B \bmod 64} \rfloor$
157	shar_r_imm_alt_64	0	$\omega'_A = \mathcal{Z}_8^{-1}(\lfloor \mathcal{Z}_8(\nu_X) \div 2^{\omega_B \bmod 64} \rfloor)$
158	rot_r_64_imm	0	$\forall i \in \mathbb{N}_{64}: \mathcal{B}_8(\omega'_A)_i = \mathcal{B}_8(\omega_B)_{(i+\nu_X) \bmod 64}$
159	rot_r_64_imm_alt	0	$\forall i \in \mathbb{N}_{64}: \mathcal{B}_8(\omega'_A)_i = \mathcal{B}_8(\nu_X)_{(i+\omega_B) \bmod 64}$
160	rot_r_32_imm	0	$\omega'_A = \mathcal{X}_4(x) \text{where} x \in \mathbb{N}_{2^{32}}, \forall i \in \mathbb{N}_{32}: \mathcal{B}_4(x)_i = \mathcal{B}_4(\omega_B)_{(i+\nu_X) \bmod 32}$
161	rot_r_32_imm_alt	0	$\omega'_A = \mathcal{X}_4(x) \text{where} x \in \mathbb{N}_{232}, \forall i \in \mathbb{N}_{32}: \mathcal{B}_4(x)_i = \mathcal{B}_4(\nu_X)_{(i+\omega_B) \bmod 32}$

A.5.11. 带有两个寄存器和一个偏移量的指令

$$\begin{aligned} &\text{let } r_A = \min(12, (\zeta_{i+1}) \bmod 16), \omega_A = \omega_{r_A}, \omega'_A = \omega'_{r_A} \\ &\text{let } r_B = \min\left(12, \left\lfloor \frac{\zeta_{i+1}}{16} \right\rfloor\right), \omega_B \equiv \omega_{r_B}, \omega'_B \equiv \omega'_{r_B} \\ &\text{let } l_X = \min(4, \max(0, l - 1)), \nu_X \equiv \imath + \mathcal{Z}_{l_X}(\mathcal{E}_{l_X}^{-1}(\zeta_{\imath+2 \cdots + l_X})) \end{aligned} \tag{A.29}$$

ζ_i	名称（**Name**）	$\varrho\Delta$	状态变更（**Mutations**）
170	branch_eq	0	$\text{branch}(\nu_X, \omega_A = \omega_B)$
171	branch_ne	0	$\text{branch}(\nu_X, \omega_A \neq \omega_B)$

172	branch_lt_u	0	$\text{branch}(\nu_X, \omega_A < \omega_B)$
173	branch_lt_s	0	$\text{branch}(\nu_X, \mathcal{Z}_8(\omega_A) < \mathcal{Z}_8(\omega_B))$
174	branch_ge_u	0	$\text{branch}(\nu_X, \omega_A \geq \omega_B)$
175	branch_ge_s	0	$\text{branch}(\nu_X, \mathcal{Z}_8(\omega_A) \geq \mathcal{Z}_8(\omega_B))$

A.5.12. 带有两个寄存器和两个立即数参数的指令

$$\begin{aligned}
&\text{let } r_A = \min(12, (\zeta_{i+1}) \bmod 16), \omega_A \equiv \omega_{r_A}, \omega'_A \equiv \omega'_{r_A} \\
&\text{let } r_B = \min\left(12, \left\lfloor \tfrac{\zeta_{i+1}}{16} \right\rfloor\right), \omega_B \equiv \omega_{r_B}, \omega'_B \equiv \omega'_{r_B} \\
&\text{let } l_X = \min(4, \zeta_{l+2} \bmod 8), \nu_X = \mathcal{X}_{l_X}\left(\mathcal{E}^{-1}_{l_X}(\zeta_{l+3\cdots+l_X})\right) \\
&\text{let } l_Y = \min(4, \max(0, \iota - l_X - 2)), \nu_Y = \mathcal{X}_{l_Y}(\mathcal{E}^{-1}_{l_Y}(\zeta_{l+3+l_X\cdots+l_Y}))
\end{aligned} \tag{A.30}$$

ζ_i	名称（**Name**）	$\varrho\Delta$	状态变更（**Mutations**）
180	load_imm_jump_ind	0	$\text{djump}((\omega_B + \nu_Y) \bmod 2^{32}), \omega'_A = \nu_X$

A.5.13. 带有三个寄存器参数的指令

$$\begin{aligned}
&\text{let } r_A = \min(12, (\zeta_{i+1}) \bmod 16), \omega_A \equiv \omega_{r_A}, \omega'_A \equiv \omega'_{r_A} \\
&\text{let } r_B = \min(12, \left\lfloor \tfrac{\zeta_{i+1}}{16} \right\rfloor), \omega_B \equiv \omega_{r_B}, \omega'_B \equiv \omega'_{r_B} \\
&\text{let } r_D = \min(12, \zeta_{i+2}), \omega_D \equiv \omega_{r_D}, \omega'_D \equiv \omega'_{r_D}
\end{aligned} \tag{A.31}$$

ζ_i	名称（**Name**）	$\varrho\Delta$	状态变更（**Mutations**）
190	add_32	0	$\omega'_D = \mathcal{X}_4((\omega_A + \omega_B) \bmod 2^{32})$
191	sub_32	0	$\omega'_D = \mathcal{X}_4((\omega_A + 2^{32} - (\omega_B \bmod 2^{32})) \bmod 2^{32})$
192	mul_32	0	$\omega'_D = \mathcal{X}_4((\omega_A \cdot \omega_B) \bmod 2^{32})$
193	div_u_32	0	$\omega'_D = \begin{cases} 2^{64} - 1 & \text{if} \omega_B \bmod 2^{32} = 0 \\ \mathcal{X}_4(\lfloor (\omega_A \bmod 2^{32}) \div (\omega_B \bmod 2^{32}) \rfloor) & \text{otherwise} \end{cases}$
194	div_s_32	0	$\omega'_D = \begin{cases} 2^{64} - 1 & \text{if } b = 0 \\ \mathcal{Z}^{-1}_8(a) & \text{if a} = -2^{31} \wedge \text{b} = -1 \\ \mathcal{Z}^{-1}_8(rtz(a \div b)) & \text{otherwise} \\ \text{wherea} = \mathcal{Z}_4(\omega_A \bmod 2^{32}), b = \mathcal{Z}_4(\omega_B \bmod 2^{32}) \end{cases}$
195	rem_u_32	0	$\omega'_D = \begin{cases} \mathcal{X}_4(\omega_A \bmod 2^{32}) & \text{if} \omega_B \bmod 2^{32} = 0 \\ \mathcal{X}_4((\omega_A \bmod 2^{32}) \bmod (\omega_B \bmod 2^{32})) & \text{otherwise} \end{cases}$

196	rem_s_32	0	$\omega'_D = \begin{cases} 0 & \text{if} a = -2^{31} \wedge b = -1 \\ \mathcal{Z}_8^{-1}(\text{smod}(a, b)) & \text{otherwise} \\ \text{where} a = \mathcal{Z}_4(\omega_A \bmod 2^{32}), b = \mathcal{Z}_4(\omega_B \bmod 2^{32}) \end{cases}$
197	shlo_l_32	0	$\omega'_D = \mathcal{X}_4((\omega_A \cdot 2^{\omega_B \bmod 32}) \bmod 2^{32})$
198	shlo_r_32	0	$\omega'_D = \mathcal{X}_4(\lfloor (\omega_A \bmod 2^{32}) \div 2^{\omega_B \bmod 32} \rfloor)$
199	shar_r_32	0	$\omega'_D = \mathcal{Z}_8^{-1}(\lfloor \mathcal{Z}_4(\omega_A \bmod 2^{32}) \div 2^{\omega_B \bmod 32} \rfloor)$
200	add_64	0	$\omega'_D = (\omega_A + \omega_B) \bmod 2^{64}$
201	sub_64	0	$\omega'_D = (\omega_A + 2^{64} - \omega_B) \bmod 2^{64}$
202	mul_64	0	$\omega'_D = (\omega_A \cdot \omega_B) \bmod 2^{64}$
203	div_u_64	0	$\omega'_D = \begin{cases} 2^{64} - 1 & \text{if} \omega_B = 0 \\ \lfloor \omega_A \div \omega_B \rfloor & \text{otherwise} \end{cases}$
204	div_s_64	0	$\omega'_D = \begin{cases} 2^{64} - 1 & \text{if} \omega_B = 0 \\ \omega_A & \text{if} \mathcal{Z}_8(\omega_A) = -2^{63} \wedge \mathcal{Z}_8(\omega_B) = -1 \\ \mathcal{Z}_8^{-1}(\text{rtz}(\mathcal{Z}_8(\omega_A) \div \mathcal{Z}_8(\omega_B))) & \text{otherwise} \end{cases}$
205	rem_u_64	0	$\omega'_D = \begin{cases} \omega_A & \text{if} \omega_B = 0 \\ \omega_A \mod \omega_B & \text{otherwise} \end{cases}$
206	rem_s_64	0	$\omega'_D = \begin{cases} 0 & \text{if} \mathcal{Z}_8(\omega_A) = -2^{63} \wedge \mathcal{Z}_8(\omega_B) = -1 \\ \mathcal{Z}_8^{-1}(\text{smod}(\mathcal{Z}_8(\omega_A), \mathcal{Z}_8(\omega_B))) & \text{otherwise} \end{cases}$
207	shlo_l_64	0	$\omega'_D = \begin{cases} 0 & \text{if} \mathcal{Z}_8(\omega_A) = -2^{63} \wedge \mathcal{Z}_8(\omega_B) = -1 \\ \mathcal{Z}_8^{-1}(\text{smod}(\mathcal{Z}_8(\omega_A), \mathcal{Z}_8(\omega_B))) & \text{otherwise} \end{cases}$
208	shlo_r_64	0	$\omega'_D = (\omega_A \cdot 2^{\omega_B \bmod 64}) \bmod 2^{64}$
209	shar_r_64	0	$\omega'_D = \lfloor \omega_A \div 2^{\omega_B \bmod 64} \rfloor$
210	and	0	$\omega'_D = \mathcal{Z}_8^{-1}(\lfloor \mathcal{Z}_8(\omega_A) \div 2^{\omega_B \bmod 64} \rfloor)$
211	xor	0	$\forall i \in \mathbb{N}_{64}: \mathcal{B}_8(\omega'_D)_i = \mathcal{B}_8(\omega_A)_i \wedge \mathcal{B}_8(\omega_B)_i$
212	or	0	$\forall i \in \mathbb{N}_{64}: \mathcal{B}_8(\omega'_D)_i = \mathcal{B}_8(\omega_A)_i \oplus \mathcal{B}_8(\omega_B)_i$
213	mul_upper_s_s	0	$\forall i \in \mathbb{N}_{64}: \mathcal{B}_8(\omega'_D)_i = \mathcal{B}_8(\omega_A)_i \vee \mathcal{B}_8(\omega_B)_i$
214	mul_upper_u_u	0	$\omega'_D = \mathcal{Z}_8^{-1}(\lfloor (\mathcal{Z}_8(\omega_A) \cdot \mathcal{Z}_8(\omega_B)) \div 2^{64} \rfloor)$
215	mul_upper_s_u	0	$\omega'_D = \lfloor (\omega_A \cdot \omega_B) \div 2^{64} \rfloor$
216	set_lt_u	0	$\omega'_D = \mathcal{Z}_8^{-1}(\lfloor (\mathcal{Z}_8(\omega_A) \cdot \omega_B) \div 2^{64} \rfloor)$
217	set_lt_s	0	$\omega'_D = \omega_A < \omega_B$
218	cmov_iz	0	$\omega'_D = \mathcal{Z}_8(\omega_A) < \mathcal{Z}_8(\omega_B)$

219	cmov_nz	0	$\omega'_D = \begin{cases} \omega_A & \text{if} \omega_B \neq 0 \\ \omega_D & \text{otherwise} \end{cases}$
220	rot_l_64	0	$\forall i \in \mathbb{N}_{64} : \mathcal{B}_8(\omega'_D)_{(i+\omega_B) \bmod 64} = \mathcal{B}_8(\omega_A)_i$
221	rot_l_32	0	$\omega'_D = \mathcal{X}_4(x) \text{where} x \in \mathbb{N}_2 s2, \forall i \in \mathbb{N}_{32} : \mathcal{B}_4(x)_{(i+\omega_B) \bmod 32} = \mathcal{B}_4(\omega_A)_i$
222	rot_r_64	0	$\forall i \in \mathbb{N}_{64} : \mathcal{B}_8(\omega'_D)_i = \mathcal{B}_8(\omega_A)_{(i+\omega_B) \bmod 64}$
223	rot_r_32	0	$\omega'_D = \mathcal{X}_4(x) \text{where} x \in \mathbb{N}_{2^{32}}, \forall i \in \mathbb{N}_{32} : \mathcal{B}_4(x)_i = \mathcal{B}_4(\omega_A)_{(i+\omega_B) \bmod 32}$
224	and_inv	0	$\forall i \in \mathbb{N}_{64} : \mathcal{B}_8(\omega'_D)_i = \mathcal{B}_8(\omega_A)_i \wedge \neg \mathcal{B}_8(\omega_B)_i$
225	or_inv	0	$\forall i \in \mathbb{N}_{64} : \mathcal{B}_8(\omega'_D)_i = \mathcal{B}_8(\omega_A)_i \vee \neg \mathcal{B}_8(\omega_B)_i$
226	xnor	0	$\forall i \in \mathbb{N}_{64} : \mathcal{B}_8(\omega'_D)_i = \neg(\mathcal{B}_8(\omega_A)_i \oplus \mathcal{B}_8(\omega_B)_i)$
227	max	0	$\omega'_D = \mathcal{Z}_8^{-1}(\max(\mathcal{Z}_8(\omega_A), \mathcal{Z}_8(\omega_B)))$
228	max_u	0	$\omega'_D = \max(\omega_A, \omega_B)$
229	min	0	$\omega'_D = \mathcal{Z}_8^{-1}(\min(\mathcal{Z}_8(\omega_A), \mathcal{Z}_8(\omega_B)))$
230	min_u	0	$\omega'_D = \min(\omega_A, \omega_B)$

注意，这两个带符号的模运算有特别定义：先对绝对值模运算，结果符号与分子一致。形式上：

(A.32)
$$\text{smod}: \begin{cases} (\mathbb{Z}, \mathbb{Z}) \to \mathbb{Z} \\ (a, b) \mapsto \begin{cases} a & \text{if } b = 0 \\ \text{sgn(a)}(|a| \bmod |b|) & \text{otherwise} \end{cases} \end{cases}$$

改写后的内容：除法运算结果总是向零取整，形式上：

(A.33)
$$rtz: \begin{cases} \mathbb{Z} \to \mathbb{Z} \\ x \mapsto \begin{cases} \lceil x \rceil & \text{if } x < 0 \\ \lfloor x \rfloor & \text{otherwise} \end{cases} \end{cases}$$

A.6. 主机调用定义

改写后：扩展版 PVM 调用定义，能在遇到主机调用停止条件时，推动内部主机调用状态机前进，记为 Ψ_H：

(A.34)

$$\Psi_H: \begin{cases} \begin{pmatrix} \mathbb{Y}, \mathbb{N}_R, \mathbb{N}_G, [\mathbb{N}_R]_{13}, \\ \mathbb{M}, \Omega\langle X\rangle, X \end{pmatrix} \to (\{i, \infty, \blacksquare\} \cup \{\exists\} \times \mathbb{N}_R, \mathbb{N}_R, \mathbb{Z}_G, [\mathbb{N}_R]_{13}, \mathbb{M}, X)\mathbb{Z} \\ (\mathbf{c}, t, \varrho, \omega, \mu, f, \mathbf{x}) \mapsto \begin{cases} \text{let}(\varepsilon', t', \phi', \omega', \mu') = \Psi(\boldsymbol{c}, \iota, \varrho, \omega, \mu): \\ \quad (\varepsilon', t', \boldsymbol{\phi}', \omega', \mu', \boldsymbol{x}) & \text{if } \varepsilon' \in \{\blacksquare, \lightning, \infty\} \cup \{\lightning\} \times \mathbb{N}_R \\ \quad (\lightning \times a, t', \varrho', \omega', \mu', \boldsymbol{x}) & \text{if } \wedge \begin{cases} \varepsilon' = \hbar \times h \\ \lightning \times a = f(h, \varrho', \omega', \mu', \boldsymbol{x}) \end{cases} \\ \quad \Psi_H(\boldsymbol{c}, t', \varrho'', \omega'', \mu'', f, x'') & \text{if } \Lambda \begin{cases} \varepsilon' = \hbar \times h \\ (\quad, \varrho'', \omega'', \mu'', \boldsymbol{x}'') = f(h, \varrho', \omega', \mu', \boldsymbol{x}) \end{cases} \\ \quad (\varepsilon'', i', \varrho'', \omega'', \mu'', \boldsymbol{x}'') & \text{if } \wedge \begin{cases} \varepsilon' = \hbar \times h \\ (\varepsilon'', \varrho'', \omega'', \mu'', \boldsymbol{x}'') = f(h, \varrho', \omega', \mu', \boldsymbol{x}) \\ \varepsilon'' \in \{i, \blacksquare, \infty\} \end{cases} \end{cases} \end{cases}$$

(A.35)
$$\Omega\langle X\rangle \equiv (\mathbb{N}, \mathbb{N}_G, [\mathbb{N}_R]_{13}, \mathbb{M}, X) \to (\{\blacktriangleright, \blacksquare, \lightning, \infty\}, \mathbb{N}_G, [\mathbb{N}_R]_{13}, \mathbb{M}, X) \cup \{\lightning\} \times \mathbb{N}_R$$

退出时，指令计数器 i'指向引发退出的指令。若用此计数器和代码再次调用机器，将重复执行该指令。在指令需重新执行（如燃料不足或页面错误）时，此做法合理。

然而，当 Ψ 因主机调用 $\hbar$ 退出时，指令计数器的值将变为主机调用指令的值。若继续以此状态执行，将立即以相同结果退出。因此，重新调用时，既需获取主机调用后的机器状态，也需获取主机调用退出后下一条指令的计数器值。此值在原值基础上加一，并加上相应的参数跳过距离。使用此指令计数器继续执行，将跳过主机调用指令，继续后续执行。

在定义 Ψ_H 时，我们采用了两个指令计数器的值。这是因为，如果主机调用引发页面错误，我们需要让外部环境有机会解决错误并重新执行主机调用。而如果主机调用成功并导致了状态转换，那么继续执行时，我们应从主机调用之后的下一条指令开始。

A.7. 标准程序初始化

在主文档中，我们运行了四个 PVM 实例上的软件程序，这些程序遵循了典型的设置模式，这是编译器和链接器输出的特点。程序中，RAM 被划分为特定只读数据区、读写（堆）数据区和栈区。此外，根据我们的使用习惯，还会通过一个额外的只读区域来传递调用时的特定数据（即参数）。因此，我们需要在单独的初始化函数中明确设定这些区域。这些区域被划分为主要区域，并且为了降低意外溢出的风险，各主要区域之间都留出了未分配的空间。同时，各区域会被零填充至最近的 PVM 内存页边界。

因此，我们设定了标准程序代码格式 $\mathbf{p}$，它涵盖了指令、跳转表（原用术语 $\mathbf{c}$ 表示）以及程序启动时的 RAM 状态信息。给定一个正确编码的 $\mathbf{p}$ 和相应的参数数据 $\mathbf{a}$，我们可以利用标准初始化解码函数 Y 来确定程序代码 $\mathbf{c}$、寄存器 ω 以及 RAMμ。

(A.36)
$$Y: \begin{cases} \mathbb{Y} \to (\mathbb{Y}, [\mathbb{N}_R]_{13}, \mathbb{M})? \\ \mathbf{p} \mapsto \begin{cases} (\boldsymbol{c}, \omega, \mu) & \text{if} \exists! \, (\boldsymbol{c}, \boldsymbol{o}, \boldsymbol{w}, z, s) \text{ which satisfy equation A. 37} \\ \emptyset & \text{otherwise} \end{cases} \end{cases}$$

条件为：

(A.37) $\text{let } \mathsf{E}_3(|\mathbf{o}|) \sim \mathsf{E}_3(|\mathbf{w}|) \sim \mathsf{E}_2(z) \sim \mathsf{E}_3(s) \sim \mathbf{o} \sim \mathbf{w} \sim \mathsf{E}_4(|\mathbf{c}|) \sim \mathbf{c} = \mathbf{p}$

(A.38) $\mathbb{Z}_Z = 2^{16}, \mathbb{Z}_I = 2^{24}$

(A.39) $\text{let } P(x \in \mathbb{N}) \equiv \mathbb{Z}_P\lceil \frac{x}{\mathbb{Z}_P} \rceil, Z(x \in \mathbb{N}) \equiv \mathbb{Z}_Z\lceil \frac{x}{\mathbb{Z}_Z} \rceil$

(A.40) $5\mathbb{Z}_Z + Z(|\mathbf{o}|) + Z(|\mathbf{w}| + z\mathbb{Z}_P) + Z(s) + \mathbb{Z}_I \le 2^{32}$

因此，若上述条件无法由唯一值满足，则结果为空集 $\emptyset$；否则，结果为包含 $\mathbf{c}$、μ 及 ω 的元组，且满足：

(A.41)
$$\hat{V\mathrm{I}} \in \Lambda_2\rho_2 \circ \left((\mu\nabla)_1, (\mu\Delta)_1\rho_2\right) = \begin{cases} (\mathbf{V}, \mathbf{o}_z, z_Z, \mathbf{A} : \mathrm{R}) & \text{if } Z_Z \le i < \mathbb{Z}_Z + |\mathbf{o}| \\ (0, \mathrm{R}) & \text{if } \mathbf{Z} + |\mathbf{o}| \le i < \mathbb{Z}_Z + \mathrm{P}(|\mathbf{o}|) \\ (\mathbf{w}_{z\in(QZ_Z+Z(0))}, \mathrm{W}) & \text{if } Z_Z + Z(|\mathbf{o}|)) \le i < 2\mathbb{Z}_Z + Z(|\mathbf{o}|) + |\mathbf{w}| \\ (0, \mathrm{W}) & \text{if } Z_Z + Z(|\mathbf{o}|) + |\mathbf{w}| \le i < 2\mathbb{Z}_Z + Z(|\mathbf{o}|) + F(|\mathbf{w}|) + z\mathbb{Z}_Z \\ (0, \mathrm{R}) & \text{if } Z^2 - 2\mathbb{Z}_Z - Z_I - \mathrm{P}(\mathbf{c}) \le i < 2^{22} - 2\mathbb{Z}_Z - Z_I \\ (0, \mathrm{W}) & \text{if } Z^{22} - Z_Z - Z_I \le i < 2^{32} - Z_Z - Z_I + |\mathbf{a}| \\ (0, \mathrm{R}) & \text{if } Z^{22} - Z_Z - Z_I + |\mathbf{a}| \le i < 2^{32} - Z_Z - Z_I + \mathrm{P}(|\mathbf{a}|) \\ (0, \sigma) & \text{otherwise} \end{cases}$$

(A.42)
$$\forall i \in \mathbb{N}_{13} : \omega_i = \begin{cases} 2^{32} - 2^{16} & \text{if } i = 0 \\ 2^{32} - 2\mathbb{Z}_Z - \mathbb{Z}_I & \text{if } i = 1 \\ 2^{32} - \mathbb{Z}_Z - \mathbb{Z}_I & \text{if } i = 7 \\ |\mathbf{a}| & \text{if } i = 8 \\ 0 & \text{otherwise} \end{cases}$$

A.8. 参数调用定义

在四个使用 PVM （虚拟机）的实例中，每个实例都需传入参数数据并接收返回数据。为此，我们定义了通用的 PVM 程序参数调用函数 Ψ_M。

(A.43)
$$\Psi_M : \begin{cases} (\mathbb{Y}, \mathbb{N}_R, \mathbb{N}_G, \mathbb{Y}_{:\mathbb{Z}_I}, \Omega\langle X\rangle, X) \to (\mathbb{N}_G, \mathbb{Y} \cup \{\lightning, \infty\}, X) \\ (\boldsymbol{p}, \iota, \varrho, \boldsymbol{a}, f, \boldsymbol{x}) \mapsto \begin{cases} (\lightning, t, \mathbf{x}) & \text{if } Y(\mathbf{p}) = \emptyset \\ R(\Psi_H(\boldsymbol{c}, \iota, \varrho, \omega, \mu, f, \boldsymbol{x})) & \text{if } Y(\mathbf{p}) = (\mathbf{c}, \omega, \mu) \\ \text{where } R : (\varrho, \begin{pmatrix} \mathcal{E}, \iota', \varrho', \\ \omega', \mu', \boldsymbol{x}', \end{pmatrix}) \mapsto \begin{cases} (u, \infty, \mathbf{x}') & \text{if } \mathcal{E} = \infty \\ (u, \mu'_{\omega'_7 \ldots + \omega'_8}, \mathbf{x}') & \text{if } \mathcal{E} = \blacksquare \wedge \mathbb{N}_{\omega'_7 \ldots + \omega'_8} \subset \mathbb{V}_{\mu'} \\ (u, [\,], \mathbf{x}') & \text{if } \mathcal{E} = \blacksquare \wedge \mathbb{N}_{\omega'_7 \ldots + \omega'_8} \notin \mathbb{V}_{\mu'} \\ (u, \lightning, \mathbf{x}') & \text{otherwise} \end{cases} \\ & \text{where } u = \varrho - \max(\varrho', 0) \end{cases} \end{cases}$$

注意，元组的第一个元素表示操作消耗的 Gas 量，且不会超过提供的 Gas 量。

附 录 B. 虚拟机调用

B.1. 主机调用结果常量

NONE = 2^{64}**-1**：返回值，表示项目不存在。

WHAT = 2^{64}**-2**：名称未知。

OOB = 2^{64}**-3**：提供的用于读/写的内部 PVM 内存索引无法访问。

WHO = 2^{64}**-4**：索引未知。

FULL = 2^{64}**-5**：存储空间已满。

CORE = 2^{64}**-6**：核心索引未知。

CASH = 2^{64}**-7**：资金不足。

LOW = 2^{64}**-8**：Gas 限额过低。

HUH = 2^{64}**-9**：项目已被请求或无法被遗忘。

OK = **0**：返回值，表示一般成功。

内部 PVM（过程虚拟机）调用有专属的结果代码体系：

HALT = 0：调用正常完成并停止。

PANIC = 1：调用以恐慌状态完成。

FAULT = 2：调用以页面错误完成。

HOST = 3：调用以主机调用错误完成。

OOG = 4：调用因耗尽资源（如"gas"用尽）而完成。

注意，主机调用请求退出的返回码为小于 2^{64}**-13** 的任意非零值。

B.2. 授权调用

"Is-Authorized" 调用是四种调用中最简单且完全无状态的一种。它仅提供一个主机调用函数 Ω_G，用来查看剩余的 gas 量。这个函数接受两个参数：整个工作包 $\mathbf{p}$ 和负责执行该工作包的核心 c 。形式上，我们可以将其定义为 Ψ_I。

$$\Psi_I: \begin{cases} (\mathbb{P}, \mathbb{N}_C) \to (\mathbb{Y} \cup \mathbb{J}, \mathbb{N}_G) \\ (\mathbf{p}, c) \mapsto (\mathbf{r}, u) \text{ where } (u, \mathbf{r}, \emptyset) = \Psi_M(\mathbf{p}_\mathbf{c}, 0, \mathbf{G}_I, \mathcal{E}(\mathbf{p}, c), F, \emptyset) \end{cases} \tag{B.1}$$

$$F \in \Omega\langle\{\}\rangle: (n, \varrho, \omega, \mu) \mapsto \begin{cases} \Omega_G(\varrho, \omega, \mu) & \text{if } n = \text{gas} \\ (\blacktriangleright, \varrho - 10, [\omega_0, ..., \omega_6, \text{WHAT}, \omega_8, ...], \mu) & \text{otherwise} \end{cases} \tag{B.2}$$

对于方程 B.2 中的 "Is-Authorized" 主机调用分发函数 F，我们未包含主机调用上下文，因其基本无状态，故上下文总为空（$\emptyset$）。

B.3. 细化调用

我们将 Refine 服务账户调用函数命名为 Ψ_R。它通常无法获取 JAM Chain 的当前状态，仅能进行历史查询这一例外操作。此外，该函数还能创建过程 PVM 的内部实例，并指定需要导出的数据片段。

历史查询主机调用函数 Ω_H 旨在确保，在任意可审计的时间点（我们规定为数据累积后的两个周期内），都能得出一致的结果。查询可追溯至最近历史记录前的最多 L 个时段，从而拓宽了审计的潜在时间范围。因此，我们将值设定为 $\mathrm{D} = 4800$，作为一个稳妥的时间跨度。

(B.3) $$\mathrm{D} \equiv \mathrm{L} + 4{,}800 = 19{,}200$$

同时，内部过程 PVM 调用的主机功能依赖于一个综合型的 PVM 类型，我们将其标记为 $\mathbf{M}$。它包含了程序代码、指令计数器以及 RAM 等关键组件。

(B.4) $$\mathbf{M} \equiv (\mathbf{p} \in \mathbb{Y}, \mathbf{u} \in \mathbb{M}, i \in \mathbb{N}_R)$$

Export 主机调用需要两个上下文信息：一是可追加的段序列（每个数据块长度为 $\mathbf{W}_\mathrm{G}$），二是调用函数时传入的参数，用以指明之前可能已追加的段数。这个参数能确保为调用者提供准确的段索引。

与其他调用函数相比，Refine 调用函数隐式地依赖于某些最近的服务账户状态项 δ，其具体来自哪个区块并不重要，只需确保不早于工作包的查询锚点区块即可。该函数明确接收工作包 p 和待细化的工作项索引 $\bar{\mathbf{i}}$ 作为输入。同时，还提供授权者输出 $\mathbf{o}$、所有工作项的导入段 i 以及一个导出段偏移量 ς。其输出结果要么是某个错误 $\mathbb{J}$，要么是一对包含细化输出数据块和导出序列的值。形式上：

(B.5) $$\Psi_R \begin{cases} \left(\mathbb{N}, \mathbb{P}, \mathbb{Y}, [[\mathbb{G}]], \mathbb{N}\right) \to \left(\mathbb{Y} \cup \mathbb{J}, [\mathbb{G}], \mathbb{N}_G\right) \\ (i, p, \bar{\mathbf{o}}, \bar{\mathbf{i}}, \mathbf{c}) \mapsto \begin{cases} (\mathrm{BAD}, [], 0) & \text{if } w_s \notin K(\delta) \vee \Lambda(\delta[w_s], (p_x)_t, w_c) = \emptyset \\ (\mathrm{BIG}, [], 0) & \text{otherwise if } |\Lambda(\delta[w_s], (p_x)_t, w_c)| > \mathrm{W}_C \\ & \text{otherwise:} \\ & \text{let } a = \mathcal{E}(w_s, w_y, \mathcal{H}(p), p_x, p_u), \mathcal{E}(\updownarrow \mathbf{m}, \mathbf{c}) = \Lambda(\delta[w_s], (p_x)_t, w_c) \\ & \text{and } (u, \mathbf{r}, (\mathbf{m}, \mathbf{e})) = \Psi_M(\mathbf{c}, 0, w_g, a, F, (\emptyset, [])): \\ & (\mathbf{r}, [], u) \quad \text{if } \mathbf{r} \in \{\infty, \lightning\} \\ & (\mathbf{r}, \mathbf{e}, u) \quad \text{otherwise} \\ & \text{where } w = p_w[i] \end{cases} \end{cases}$$

$$\text{(B.6)}\quad F \in \Omega\langle(\mathbb{D}\langle\mathbb{N}\to\mathbf{M}\rangle, [\![\mathbb{G}]\!])\rangle : (n, \varrho, \omega, \mu, (\mathbf{m}, \mathbf{e})) \mapsto \begin{cases} \Omega_H(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}), w_s, \delta, (p_x)_t) & \text{if } n = \text{historical\_lookup} \\ \Omega_Y(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}), i, p, \mathbf{o}, \bar{\mathbf{i}}) & \text{if } n = \text{fetch} \\ \Omega_E(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}), \varsigma) & \text{if } n = \text{export} \\ \Omega_G(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e})) & \text{if } n = \text{gas} \\ \Omega_M(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e})) & \text{if } n = \text{machine} \\ \Omega_P(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e})) & \text{if } n = \text{peek} \\ \Omega_Z(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e})) & \text{if } n = \text{zero} \\ \Omega_O(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e})) & \text{if } n = \text{poke} \\ \Omega_V(\varrho, \omega, \omega, \mu, (\mathbf{m}, \mathbf{e})) & \text{if } n = \text{void} \\ \Omega_K(\varrho, \omega, \mu, \mu, (\mathbf{m}, \mathbf{e})) & \text{if } n = \text{invoke} \\ \Omega_X(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e})) & \text{if } n = \text{expunge} \\ (\blacktriangleright, \varrho - 10, \omega', \mu) & \text{otherwise} \\ \text{where } \omega' = \omega \text{ except } \omega'_7 = \text{WHAT} \end{cases}$$

B.4. 累积调用

由于这个调用能直接影响大量链上状态的转换，所以调用上下文变得相当复杂。它是一个元组，包含了此调用可能更改的所有状态元素，除了服务账户外，还涵盖延迟转账列表、原像查找状态修改字典、核心分配字典、验证器密钥分配字典、新创建账户及账户权限级别的更改字典。

形式上，我们将结果上下文记为 $\mathbf{X}$，而调用上下文则定义为这两个上下文的组合，即 $\mathbf{X} \times \mathbf{X}$。其中，一个维度是常规维度，通常我们称之为 $\mathbf{x}$；另一个维度是异常维度，我们将其命名为 $\mathbf{y}$ 。唯一实际会影响到这个异常维度的函数是检查点函数 Ω_C，因此这个维度在平时很少会被用到。

$$\text{(B.7)}\quad \mathbf{X} \equiv (s \in \mathbb{N}_S, \mathbf{u} \in \mathbb{U}, i \in \mathbb{N}_S, \mathbf{t} \in [\![\mathbb{T}]\!], y \in \mathbb{H}?, \mathbf{p} \in \{(\mathbb{N}_S, \mathbb{Y})\})$$

$$\text{(B.8)}\quad \forall \mathbf{x} \in \mathbf{X} : \mathbf{x_s} \equiv (\mathbf{x_u})\mathbf{d}[\mathbf{x}_s]$$

对于所有这些上下文，我们定义了一个简便的等价表示 $\mathbf{x_s}$，它指的是存储在 $(\mathbf{x_u})\mathbf{d}$ 字典中、索引为 $\mathbf{x}_s$ 的累积服务账户。

在我们的上下文变更机制中，我们同时监控常规维度和异常维度，并根据终止情况是常规的还是异常的（比如 gas 耗尽或发生恐慌），将调用结果归并到相应的维度中。

我们将积累调用函数 Ψ_A 定义为：

$$\text{(B.9)}\quad \Psi_A : \begin{cases} (\mathbb{U}, \mathbb{N}_T, \mathbb{N}_S, \mathbb{N}_G, [\![\mathbb{O}]\!]) \to (\mathbb{U}, [\![\mathbb{T}]\!], \mathbb{H}?, \mathbb{N}_G) \\ (\mathbf{u}, t, s, g, \mathbf{o}) \mapsto \begin{cases} (\mathbf{u}, [], \emptyset, 0) & \text{if } \mathbf{u_d}[s]_\mathbf{c} = \emptyset \\ C(\Psi_M(\mathbf{u_d}[s]_\mathbf{c}, 5, g, \mathcal{E}(t, s, \updownarrow \mathbf{o}), F, (I(\mathbf{u}, s), I(\mathbf{u}, s)))) & \text{otherwise} \end{cases} \end{cases}$$

$$\text{(B.10)}\quad I : \begin{cases} (\mathbb{U}, \mathbb{N}_S) \to \mathbf{X} \\ (\mathbf{u}, s) \mapsto (s, \mathbf{u}, i, \mathbf{t}; [], y: \emptyset) \\ \text{where } i = \text{check}((\mathcal{E}_4^{-1}(\mathcal{H}(\mathcal{E}(s, \eta'_0, \mathbf{H}_t))) \bmod (2^{32} - 2^9)) + 2^8) \end{cases}$$

(B.11)
$$F \in \Omega\langle(\mathbf{X},\mathbf{X})\rangle\colon (n,\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) \mapsto \begin{cases} G(\Omega_R(\varrho,\omega,\mu,s,\mathbf{x}_s,\mathbf{d}),(\mathbf{x},\mathbf{y})) & \text{if } n = \text{read} \\ G(\Omega_W(\varrho,\omega,\mu,s,\mathbf{x}_s),(\mathbf{x},\mathbf{y})) & \text{if } n = \text{write} \\ G(\Omega_L(\varrho,\omega,\mu,s,\mathbf{x}_s,\mathbf{d}),(\mathbf{x},\mathbf{y})) & \text{if } n = \text{lookup} \\ \Omega_G(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{gas} \\ G(\Omega_I(\varrho,\omega,\mu,\mathbf{x}_s,\mathbf{d}),(\mathbf{x},\mathbf{y})) & \text{if } n = \text{info} \\ \Omega_B(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{bless} \\ \Omega_A(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{assign} \\ \Omega_D(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{designate} \\ \Omega_C(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{checkpoint} \\ \Omega_N(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{new} \\ \Omega_U(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{upgrade} \\ \Omega_T(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{transfer} \\ \Omega_J(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y}),\boldsymbol{H}_t) & \text{if } n = \text{eject} \\ \Omega_Q(a,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{query} \\ \Omega_S(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y}),\boldsymbol{H}_t) & \text{if } n = \text{solicit} \\ \Omega_F(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y}),\boldsymbol{H}_t) & \text{if } n = \text{forget} \\ \Omega_8(\varrho,\omega,\mu,(\mathbf{x},\mathbf{y})) & \text{if } n = \text{yield} \\ \\ (\blacktriangleright, \varrho - 10, [\omega_0, \ldots, \omega_6, \text{WHAT}, \omega_8, \ldots], \mu, \mathbf{x}) & \text{otherwise} \\ \text{where } \mathbf{d} = (\mathbf{x_u})_{\mathbf{d}}, \mathbf{s} = (\mathbf{x_u})_{\mathbf{d}}[\mathbf{x}_s] \end{cases}$$

(B.12)
$$G\colon \begin{cases} (\{\blacktriangleright, \blacksquare, \lightning, \infty\}, \mathbb{N}_G, \llbracket \mathbb{N}_R \rrbracket_{13}, \mathbb{M}, \mathbb{A}), (\mathbf{X},\mathbf{X})) \to (\{\blacktriangleright, \blacksquare, \lightning, \infty\}, \mathbb{N}_G, \llbracket \mathbb{N}_R \rrbracket_{13}, \mathbb{M}, (\mathbf{X},\mathbf{X})) \\ ((\varepsilon,\varrho,\omega,\mu,\mathbf{s}),(\mathbf{x},\mathbf{y})) \mapsto (\varepsilon,\varrho,\omega,\mu,(\mathbf{x}^*,\mathbf{y})) \\ \text{where} \mathbf{x}^* = \mathbf{x} \text{except} (\mathbf{x}^*_{\mathbf{u}})_{\mathbf{d}}[\mathbf{x}^*_s] = \mathbf{s} \end{cases}$$

(B.13)
$$C\colon \begin{cases} (\mathbb{N}_G, \mathbb{Y} \cup \{\infty, \lightning\}, (\mathbf{X},\mathbf{X})) \to (\mathbb{U}, \llbracket \mathbb{T} \rrbracket, \mathbb{H}?, \mathbb{N}_G) \\ (u, \mathbf{o}, (\mathbf{x},\mathbf{y})) \mapsto \begin{cases} (\mathbf{y_u}, \mathbf{y_t}, \mathbf{y}_y, u) & \text{if } \mathbf{o} \in \{\infty, \lightning\} \\ (\mathbf{x_u}, \mathbf{x_t}, \mathbf{o}, u) & \text{otherwise if } \mathbf{o} \in \mathbb{H} \\ (\mathbf{x_u}, \mathbf{x_t}, \mathbf{x}_y, u) & \text{otherwise} \end{cases} \end{cases}$$

变更器 F 决定了在给定参数的情况下，上下文将如何发生变化。而归并函数 C 则根据虚拟机停止的原因（常规或异常），从上下文的两个维度中选择一个进行归并。

初始化函数 I 将服务账户 $\mathbf{s}$ 及其索引 s 映射为一个变更器上下文，确保在任何退出情况下，都不会对状态产生额外改变（除 $\mathbf{s}$ 本身固有的改变外）。该组件利用随机累积器 η_0 和区块时间槽 $\mathbf{H}_t$，生成一系列极有可能唯一的确定性标识符。

具体来说，我们通过结合服务账户的标识符、当前随机累积器 η_0 和区块时间槽的 Blake2 哈希来生成一个唯一标识符。在单个服务的累积过程中，这个标识符几乎肯定是独一无二的，但在所有服务中或历史记录中，它并不一定唯一。为此，我们使用一个检查函数来找出此序列中第一个尚未被任何服务使用的索引：

(B.14)
$$\text{check}(i \in \mathbb{N}_S) \equiv \begin{cases} i & \text{if } i \notin \mathcal{K}(\mathbf{u_d}) \\ \text{check}((i - 2^8 + 1) \bmod (2^{32} - 2^9) + 2^8) & \text{otherwise} \end{cases}$$

NB 在极端罕见的情况下，如果一个区块执行后发现同一个服务索引被意外地分配给了两个不同的服务，那么该区块将被判定为无效。由于服务无法提前预知标识符序列，因此它们无法故意对区块作者造成不利影响。

B.5. 转账时调用

我们定义了“转账时调用”服务账户的调用函数 Ψ_T，它与“累积调用”相似，但有所不同。Ψ_T 仅用于促进对目标账户存储的基本更改，不能进行额外的转账操作，无法执行特权指令，不能创建新账户，也不能对目标账户执行其他任何操作。该函数的定义如下：

$$\Psi_T\colon \begin{cases} (\mathbb{D}\langle\mathbb{N}_S \to \mathbb{A}\rangle, \mathbb{N}_T, \mathbb{N}_S, [\![\mathbb{T}]\!]) \to (\mathbb{A}, \mathbb{N}_G) \\ (\mathbf{d}, t, s, \mathbf{t}) \mapsto \begin{cases} (\mathbf{s}, 0) & \text{if } \mathbf{s}_c = \emptyset \vee \mathbf{t} = [] \\ (\mathbf{s}', u) & \text{otherwise} \end{cases} \\ \text{where} (u, \mathbf{r}, \mathbf{s}') = \Psi_M(\mathbf{s}_c, 10, \sum_{r\in\mathbf{t}}(r_g), \mathcal{E}(t, s, \updownarrow \mathbf{t}), F, \mathbf{s}) \\ \text{and } \mathbf{s} = \mathbf{d}[s] \text{except } \mathbf{s}_b = \mathbf{d}[s]_b + \sum_{r\in\mathbf{t}} r_a \end{cases} \tag{B.15}$$

$$\text{and } F \in \Omega\langle\mathbb{A}\rangle\colon (n, \varrho, \omega, \mu, \mathbf{s}) = \begin{cases} \Omega_L(\varrho, \omega, \mu, \mathbf{s}, s, \mathbf{d}) & \text{if } n = \text{lookup} \\ \Omega_R(\varrho, \omega, \mu, \mathbf{s}, s, \mathbf{d}) & \text{if } n = \text{read} \\ \Omega_W(\varrho, \omega, \mu, \mathbf{s}, s) & \text{if } n = \text{write} \\ \Omega_G(\varrho, \omega, \mu) & \text{if } n = \text{gas} \\ \Omega_I(\varrho, \omega, \mu, s, \mathbf{d}) & \text{if } n = \text{info} \\ (\blacktriangleright, \varrho - 10, [\omega_0, \ldots, \omega_6, \text{WHAT}, \omega_8, \ldots], \mu, \mathbf{s}) & \text{otherwise} \end{cases} \tag{B.16}$$

B.6. 通用函数

现在，我们来定义 PVM 调用时所使用的宿主函数。这些函数通常会将 PVM 的当前状态（包括调用上下文）以及可能的其他附加参数，映射并更新到一个新的 PVM 状态。

通用函数的大致形式为：$(\varrho' \in \mathbb{Z}_G, \omega' \in [\![\mathbb{N}_R]\!]_{13}, \mu', \mathbf{s}') = \Omega\ (\varrho \in \mathbb{N}_G, \omega \in [\![\mathbb{N}_R]\!]_{13}, \mu \in \mathbb{M}, \mathbf{s} \in \mathbb{A}, \ldots)$。若函数的某个结果分量与对应参数分量相同，则在描述中可能会予以省略。此外，函数还可能依赖于其他特定的附加参数。

与附录 **B.7** 中的“累积”函数相比，这些函数并不会影响累积上下文，而只是单纯地修改服务账户 $\mathbf{s}$ 。

Gas 函数 Ω_G 的参数列表末尾带有省略号，意味着它可以接受任何额外的参数，并且这些参数会原封不动地传递到其结果中。这一特性使得该函数在多个虚拟机调用场景中能够轻松应用。

除非另有明确说明，否则在宿主调用期间，PVM 状态的各个元素（除明确定义的 gas 计数器外）均保持不变。

$$\varrho' \equiv \varrho - g \tag{B.17}$$

$$(\varepsilon', \omega', \mu', \mathbf{s}') = \begin{cases} (\infty, \omega, \mu, \mathbf{s}) & \text{if } \varrho < g \\ (\blacktriangleright, \omega, \mu, \mathbf{s}) & \text{except as indicated below} \quad \text{otherwise} \end{cases} \tag{B.18}$$

函数标识符 gas 使用量	状态变更
$\Omega_G(\varrho, \omega, \ldots)$ $\text{gas} = 0$ $g = 10$	$\omega_7' \equiv \varrho'$

$$\text{let } \mathbf{a} = \begin{cases} \mathbf{s} & \text{if } \omega_7 \in \{s, 2^{64}-1\} \\ \mathbf{d}[\omega_7] & \text{otherwise if } \omega_7 \in \mathcal{K}(\mathbf{d}) \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{let } [h, o] = \omega_{8\cdots+2}$$

$\Omega_L(\varrho, \omega, \mu, \mathbf{s}, s, \mathbf{d})$

$\text{lookup} = 1$

$g = 10$

$$\text{let } \mathbf{v} = \begin{cases} \nabla & \text{if } \mathbb{N}_{h\cdots+32} \notin \mathbb{V}_\mu \\ \emptyset & \text{otherwise if } \mathbf{a} = \emptyset \vee \mu_{h\cdots+32} \notin \mathcal{K}(\mathbf{a_p}) \\ \mathbf{a_p}[\mu_{h\cdots+32}] & \text{otherwise} \end{cases}$$

$$\text{let } f = min(\omega_{10}, |\mathbf{v}|)$$

$$\text{let } l = min(\omega_{11}, |\mathbf{v}| - f)$$

$$(\varepsilon', \omega_7', \mu_{o\cdots+l}') \equiv \begin{cases} (\lightning, \omega_7, \mu_{o\cdots+l}) & \text{if } \mathbf{v} = \nabla \vee \mathbb{N}_{o\cdots+l} \notin \mathbb{V}_\mu^* \\ (\blacktriangleright, \text{NONE}, \mu_{o\cdots+l}) & \text{otherwise if } \mathbf{v} = \emptyset \\ (\blacktriangleright, |\mathbf{v}|, \mathbf{v}_{f\cdots+l}) & \text{otherwise} \end{cases}$$

$$\text{let } s^* = \begin{cases} s & \text{if } \omega_7 = 2^{64}-1 \\ \omega_7 & \text{otherwise} \end{cases}$$

$$\text{let } \mathbf{a} = \begin{cases} \mathbf{s} & \text{if } s^* = s \\ \mathbf{d}[s^*] & \text{otherwise if } s^* \in \mathcal{K}(\mathbf{d}) \\ \emptyset & \text{otherwise} \end{cases}$$

$\Omega_R(\varrho, \omega, \mu, \mathbf{s}, s, \mathbf{d})$

$\text{read} = 2$

$g = 10$

$$\text{let } [k_o, k_z, o] = \omega_{8\cdots+3}$$

$$\text{let } \mathbf{v} = \begin{cases} \nabla & \text{if } \mathbb{N}_{k_o\cdots+k_z} \notin \mathbb{N}_\mu \\ \mathbf{a_s}[k] & \text{otherwise if a} \neq \emptyset \wedge k \in \mathcal{K}(\mathbf{a_s}), \text{where } k = \mathcal{H}(\mathcal{E}_4(s^*) \quad \mu_{k_o\cdots+k_z}) \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{let } f = min(\omega_{11}, |\mathbf{v}|)$$

$$\text{let } l = min(\omega_{12}, |\mathbf{v}| - f)$$

$$(\varepsilon', \omega_7', \mu_{o\cdots+l}') \equiv \begin{cases} (\lightning, \omega_7, \mu_{o\cdots+l}) & \text{if } \mathbf{v} = \nabla \vee \mathbb{N}_{o\cdots+l} \notin \mathbb{N}_\mu^* \\ (\blacktriangleright, \text{NOME}, \mu_{o\cdots+l}) & \text{otherwise if } \mathbf{v} = \emptyset \\ (\blacktriangleright, |\mathbf{v}|, \mathbf{v}_{f\cdots+l}) & \text{otherwise} \end{cases}$$

$$\text{let } \left[k_o, k_z, v_o, v_z\right] = \omega_{7\cdots+4}$$

$$\text{let } k = \begin{cases} \mathcal{H}(\mathcal{E}_4(s) \sim \mu_{k_o\cdots+k_z}) & \text{if } \mathbb{N}_{k_o\cdots+k_z} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

$\Omega_W(\varrho, \omega, \mu, \mathbf{s}, s)$

$\text{write} = 3$

$g = 10$

$$\text{let } \mathbf{a} = \begin{cases} \mathbf{s}, & \text{except } \mathcal{K}(\mathbf{a_s}) = \mathcal{K}(\mathbf{a_s}) \setminus \{k\} \quad \text{if } v_z = 0 \\ \mathbf{s}, & \text{except } \mathbf{a_s}[k] = \mu_{v_o\cdots+v_z} \quad \text{otherwise if } \mathbb{N}_{v_o\cdots+v_z} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } l = \begin{cases} |\mathbf{s_s}[k]| & \text{if } k \in \mathcal{K}(\mathbf{s_s}) \\ \textbf{NONE} & \text{otherwise} \end{cases}$$

	$$(\varepsilon', \omega_7', s') \equiv \begin{cases} (\lightning, \omega_7, \mathbf{s}) & \text{if } k = \nabla \vee \mathbf{a} = \nabla \\ (\blacktriangleright, \text{FULL}, \mathbf{s}) & \text{otherwise if } \mathbf{a}_t > \mathbf{a}_b \\ (\blacktriangleright, l, \mathbf{a}) & \text{otherwise} \end{cases}$$
$\Omega_I(\varrho, \omega, \mu, s, \mathbf{d})$ $\text{info} = 4$ $g = 10$	$$\text{let } \mathbf{t} = \begin{cases} \mathbf{d}[s] & \text{if } \omega_7 = 2^{64} - 1 \\ \mathbf{d}[\omega_7] & \text{otherwise} \end{cases}$$ $$\text{let } o = \omega_8$$ $$\text{let } \mathbf{m} = \begin{cases} \mathcal{E}(\mathbf{t}_c, \mathbf{t}_b, \mathbf{t}_t, \mathbf{t}_g, \mathbf{t}_m, \mathbf{t}_o, \mathbf{t}_i) & \text{if } \mathbf{t} \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$ $$\forall i \in \mathbb{N}_{\lvert\mathbf{m}\rvert} : \mu'_{o+i} \equiv \begin{cases} \mathbf{m}_i & \text{if } \mathbf{m} \neq \emptyset \wedge \mathbb{N}_{o\cdots+\lvert\mathbf{m}\rvert} \subset \mathbb{V}^*_\mu \\ \mu_{o+i} & \text{otherwise} \end{cases}$$ $$(\varepsilon', \omega_7') \equiv \begin{cases} (\lightning, \omega_7) & \text{if } \mathbb{N}_{o\cdots+\lvert\mathbf{m}\rvert} \notin \mathbb{V}^*_\mu \\ (\blacktriangleright, \text{NONE}) & \text{otherwise if } \mathbf{m} = \emptyset \\ (\blacktriangleright, \text{OK}) & \text{otherwise} \end{cases}$$

B.7. 累积函数

这定义了一系列函数，其形式大致为：$(\varrho' \in \mathbb{Z}_G, \omega' \in \llbracket \mathbb{N}_R \rrbracket_{13}, \mu', (\mathbf{x}', \mathbf{y}')) = \Omega\ (\varrho \in \mathbb{N}_G, \omega \in \llbracket \mathbb{N}_R \rrbracket_{13}, \mu \in \mathbb{M}, (\mathbf{x} \in \mathbf{X}, \mathbf{y} \in \mathbf{X}), \ldots)$。若函数的某个结果分量与对应的参数分量相同，则在描述中可能会省略该分量。此外，这些函数还可能依赖于其他特定的附加参数。

在宿主调用期间，除非另有明确说明，否则 PVM 状态的各个元素（除明确定义的 gas 计数器外）均保持不变。

(B.19) $$\varrho' \equiv \varrho - g$$

(B.20) $$(\varepsilon', \omega', \mu', \mathbf{x}', \mathbf{y}') = \begin{cases} (\infty, \omega, \mu, \mathbf{x}, \mathbf{y}) & \text{if } \varrho < g \\ (\blacktriangleright, \omega, \mu, \mathbf{x}, \mathbf{y}) & \text{except as indicated below} \quad \text{otherwise} \end{cases}$$

函数标识符 gas 使用量	状态变更
$\Omega_B(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}))$ $\text{bless} = 5$ $g = 10$	$$\text{let } [m, a, v, o, n] = \omega_{7\cdots+5}$$ $$\text{let } \mathbf{g} = \begin{cases} \{(s \mapsto g) \text{where } \mathcal{E}_4(s) \sim \mathcal{E}_8(g) = \mu_{o+12i\cdots+12} \lvert i \in \mathbb{N}_n\} & \text{if } \mathbb{N}_{o\cdots+12n} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$ $$(\varepsilon', \omega_7', (\mathbf{x}'_\mathbf{u})_\mathbf{x}) = \begin{cases} (t, \omega_7, (\mathbf{x}_\mathbf{u})_\mathbf{x}) & \text{if } g = \nabla \\ (\lightning, \text{WHO}, (\mathbf{x}_\mathbf{u})_\mathbf{x}) & \text{otherwise if } (m, a, v) \notin (\mathbb{N}_\mathbf{S}, \mathbb{N}_\mathbf{S}, \mathbb{N}_\mathbf{S}) \\ (\blacktriangleright, \text{OK}, (m, a, v, \mathbf{g})) & \text{otherwise} \end{cases}$$
$\Omega_A(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}))$ $\text{assign} = 6$ $g = 10$	$$\text{let } o = \omega_8$$ $$\text{let } \mathbf{c} = \begin{cases} \left[\mu_{o+32i\cdots+32} \lvert i \lessdot \mathbb{N}_\mathbb{Q}\right] & \text{if } \mathbb{N}_{o\cdots+32\mathbb{Q}} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

	$(\varepsilon', \omega_7', (\mathbf{x}_\mathbf{u}')_\mathbf{q}[\omega_7]) = \begin{cases} (\lightning, \omega_7, (\mathbf{x}_\mathbf{u})_\mathbf{q}[\omega_7]) & \text{if } \mathbf{c} = \nabla \\ (\blacktriangleright, \text{CORE}, (\mathbf{x}_\mathbf{u})_\mathbf{q}[\omega_7]) & \text{otherwise if } \omega_7 \geq \mathbf{C} \\ (\blacktriangleright, \text{OK}, \mathbf{c}) & \text{otherwise} \end{cases}$
$\Omega_D(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}))$ designate $= 7$ $g = 10$	let $O = \omega_7$ let $\mathbf{v} = \begin{cases} [\mu_{o+336i\cdots+336} \vert i \in \mathbb{N}_\mathbb{V}] & \text{if } \mathbb{N}_{o\cdots+336\mathbb{V}} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$ $(\varepsilon', \omega_7', (\mathbf{x}_\mathbf{u}')_\mathbf{i}) = \begin{cases} (\lightning, \omega_7, (\mathbf{x}_\mathbf{u})_\mathbf{i}) & \text{if } \mathbf{v} = \nabla \\ (\blacktriangleright, \text{OK}, \mathbf{v}) & \text{otherwise} \end{cases}$
$\Omega_C(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}))$ checkpoint $= 8$ $g = 10$	$\mathbf{y}' \equiv \mathbf{x}$ $\omega_7' \equiv \varrho'$
$\Omega_N(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}))$ new $= 9$ $g = 10$	let $[o, l, g, m] = \omega_{7\cdots+4}$ let $c = \begin{cases} \mu_{o\cdots+32} & \text{if } \mathbb{N}_{o\cdots+32} \subset \mathbb{V}_\mu \wedge l \in \mathbb{N}_{2^{32}} \\ \nabla & \text{otherwise} \end{cases}$ let $\mathbf{a} \in \mathbb{A} \cup \{\nabla\} = \begin{cases} (c, \mathbf{s}: \{\}, \mathbf{l}: \{(c, l) \mapsto []\}, b: \mathbf{a}_t, g, m, \mathbf{p}: \{\}) & \text{if } c \neq \nabla \\ \nabla & \text{otherwise} \end{cases}$ let $\mathbf{s} = \mathbf{x}_\mathbf{s}$ except $\mathbf{s}_b = (\mathbf{x}_\mathbf{s})_b - \mathbf{a}_t$ $(\varepsilon', \omega_7', \mathbf{x}_i', (\mathbf{x}_\mathbf{u}')_\mathbf{d}) = \begin{cases} (\lightning, \omega_7, \mathbf{x}_i, (\mathbf{x}_\mathbf{u})_\mathbf{d}) & \text{if } c = \nabla \\ (\blacktriangleright, \text{CASH}, \mathbf{x}_i, (\mathbf{x}_\mathbf{u})_\mathbf{d}) & \text{otherwise if } \mathbf{s}_b < (\mathbf{x}_\mathbf{s})_t \\ (\blacktriangleright, \mathbf{x}_i, \text{check}(i), (\mathbf{x}_\mathbf{u})_\mathbf{d} \cup \{\mathbf{x}_i \mapsto \mathbf{a}, \mathbf{x}_\mathbf{s} \mapsto \mathbf{s}\}) & \text{otherwise} \\ \text{where } i = 2^8 + (\mathbf{x}_i - 2^8 + 42) \bmod (2^{32} - 2^9) \end{cases}$
$\Omega_U(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}))$ upgrade $= 10$ $g = 10$	let $[o, g, m] = \omega_{7\cdots+3}$ let $c = \begin{cases} \mu_{o\cdots+32} & \text{if } \mathbb{N}_{o\cdots+32} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$ $(\varepsilon', \omega_7', (\mathbf{x}_\mathbf{s}')_c, (\mathbf{x}_\mathbf{s}')_g, (\mathbf{x}_\mathbf{s}')_m) \equiv \begin{cases} (\lightning, \omega_7, (\mathbf{x}_\mathbf{s})_c, (\mathbf{x}_\mathbf{s})_g, (\mathbf{x}_\mathbf{s})_m) & \text{if } c = \nabla \\ (\blacktriangleright, 0\text{K}, c, g, m) & \text{otherwise} \end{cases}$
$\Omega_T(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}))$ transfer $= 11$ $g = 10 + \omega_9$	let $[d, a, l, o] = \omega_7 \ldots + 4$, let $\mathbf{d} = (\mathbf{x}_\mathbf{u})_\mathbf{d}$ let $\mathbf{t} \in \mathbb{T} \cup \{\nabla\} = \begin{cases} (s: \mathbf{x}_s, d, a, m: \mu_{o\ldots+\mathcal{W}_T}, g: l) & \text{if } \mathbb{N}_{o\ldots+\mathcal{W}_T} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$ let $b = (\mathbf{x}_\mathbf{s})_b - a$

$$(\varepsilon', \omega_7', \mathbf{x}_{\mathbf{t}}', (\mathbf{x}_{\mathbf{s}}')_b) = \begin{cases} (\lightning, \omega_7, \mathbf{x}_{\mathbf{t}}, (\mathbf{x}_{\mathbf{s}})_b) & \text{if } \mathbf{t} = \nabla \\ (\blacktriangleright, \text{WHO}, \mathbf{x}_{\mathbf{t}}, (\mathbf{x}_{\mathbf{s}})_b) & \text{otherwise if } d \notin \mathcal{K}(\mathbf{d}) \\ (\blacktriangleright, \text{LWW}, \mathbf{x}_{\mathbf{t}}, (\mathbf{x}_{\mathbf{s}})_b) & \text{otherwise if } l < \mathbf{d}[d]_m \\ (\blacktriangleright, \text{CASH}, \mathbf{x}_{\mathbf{t}}, (\mathbf{x}_{\mathbf{s}})_b) & \text{otherwise if } b < (\mathbf{x}_{\mathbf{s}})_t \\ (\blacktriangleright, \text{OK}, \mathbf{x}_{\mathbf{t}} + \text{t}, b) & \text{otherwise} \end{cases}$$

$\Omega_J(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}), t)$

$\text{eject} = 12$

$g = 10$

$$\text{let } [d, o] = \omega_{7,8}$$

$$\text{let } h = \begin{cases} \mu_{o\cdots+32} & \text{if } \mathbb{N}_{o\cdots+32} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } \mathbf{d} = \begin{cases} ((\mathbf{x}_{\mathbf{u}})_{\mathbf{d}})[d] & \text{if } d \neq \mathbf{x}_{\mathbf{s}} \wedge d \in \mathcal{K}((\mathbf{x}_{\mathbf{u}})_{\mathbf{d}}) \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } l = max(81, \mathbf{d}_o) - 81$$

$$\text{let } \mathbf{s}' = ((\mathbf{x}_{\mathbf{u}})_{\mathbf{d}})[\mathbf{x}_s] \text{except } \mathbf{s}_b' = ((\mathbf{x}_{\mathbf{u}})_{\mathbf{d}})[\mathbf{x}_s]_b + \mathbf{d}_b$$

$$(\varepsilon', \omega_7', (\mathbf{x}_{\mathbf{u}}')_{\mathbf{d}}) = \begin{cases} (\lightning, \omega_7, (\mathbf{x}_{\mathbf{u}})_{\mathbf{d}}) & \text{if } h = \nabla \\ (\blacktriangleright, \text{WHU}, (\mathbf{x}_{\mathbf{u}})_{\mathbf{d}}) & \text{otherwise if } \mathbf{d} = \nabla \vee \mathbf{d}_c \neq \mathcal{E}_{32}(\mathbf{x}_{\mathbf{s}}) \\ (\blacktriangleright, \text{HUH}, (\mathbf{x}_{\mathbf{u}})_{\mathbf{d}}) & \text{otherwise if } \mathbf{d}_i \neq 2 \vee (h, l) \notin \mathbf{d}_1 \\ (\blacktriangleright, \text{OK}, (\mathbf{x}_{\mathbf{u}})_{\mathbf{d}} \setminus \{d\} \cup \{\mathbf{x}_{\mathbf{s}} \mapsto \mathbf{s}'\}) & \text{otherwise if } \mathbf{d}_1[h, l] = [x, y], y < t - \text{D} \\ (\blacktriangleright, \text{HUH}, (\mathbf{x}_{\mathbf{u}})_{\mathbf{d}}) & \text{otherwise} \end{cases}$$

$\Omega_Q(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}))$

$\text{query} = 13$

$g = 10$

$$\text{let } [o, z] = \omega_{7,8}$$

$$\text{let } h = \begin{cases} \mu_{o\cdots+32} & \text{if } \mathbb{N}_{o\cdots+32} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } \mathbf{a} = \begin{cases} (\mathbf{x}_{\mathbf{s}})_{\mathbf{l}}[h, z] & \text{if}(h, z) \in \mathcal{K}((\mathbf{x}_{\mathbf{s}})_{\mathbf{l}}) \\ \nabla & \text{otherwise} \end{cases}$$

$$(e', \omega_7', \omega_8') = \begin{cases} (\lightning, \omega_7, \omega_8) & \text{if } h = \nabla \\ (\blacktriangleright, \text{MOME}, 0) & \text{otherwise if } \mathbf{a} = \nabla \\ (\blacktriangleright, 0, 0) & \text{otherwise if } \mathbf{a} = [] \\ (\blacktriangleright, 1 + 2^{32}x, 0) & \text{otherwise if } \mathbf{a} = [x] \\ (\blacktriangleright, 2 + 2^{32}x, y) & \text{otherwise if } \mathbf{a} = [x, y] \\ (\blacktriangleright, 3 + 2^{32}x, y + 2^{32}z) & \text{otherwise if } \mathbf{a} = [x, y, z] \end{cases}$$

$\Omega_S(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}), t)$

$\text{solicit} = 14$

$g = 10$

$$\text{let } [o, z] = \omega_{7,8}$$

$$\text{let } h = \begin{cases} \mu_{o\cdots+32} & \text{if } \mathbb{N}_{o\cdots+32} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } \mathbf{a} = \begin{cases} \mathbf{x}_{\mathbf{s}} \textbf{ except}: & \\ \mathbf{a}_{\mathbf{l}}[(h, z)] = [] & \text{if } h \neq \nabla \wedge (h, z) \notin (\mathbf{x}_{\mathbf{s}})_1 \\ \mathbf{a}_{\mathbf{l}}[(h, z)] = (\mathbf{x}_{\mathbf{s}})_1[(h, z)] + t & \text{if } (\mathbf{x}_{\mathbf{s}})_1[(h, z)] = [x, y] \\ \nabla & \text{otherwise} \end{cases}$$

	$(\varepsilon', \omega_7', \mathbf{x}_\mathbf{s}') = \begin{cases} (\lightning, \omega_7, \mathbf{x}_\mathbf{s}) & \text{if } h = \nabla \\ (\blacktriangleright, \text{HUH}, \mathbf{x}_\mathbf{s}) & \text{otherwise if } \mathbf{a} = \nabla \\ (\blacktriangleright, \text{FULL}, \mathbf{x}_\mathbf{s}) & \text{otherwise if } \mathbf{a}_b < \mathbf{a}_t \\ (\blacktriangleright, \text{OK}, \mathbf{a}) & \text{otherwise} \end{cases}$
$\Omega_F(\varrho, \omega, \mu, (\mathbf{x}, \mathbf{y}), t)$ $\text{forget} = 15$ $g = 10$	$\text{let } [o, z] = \omega_{7,8}$ $\text{let } h = \begin{cases} \mu_{o\cdots+32} & \text{if } \mathbb{N}_{o\cdots+32} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$ $\text{let } \mathbf{a} = \begin{cases} \mathbf{x}_\mathbf{s} \text{ excepu:} & \\ \mathcal{K}(\mathbf{a_1}) = \mathcal{K}((\mathbf{x}_\mathbf{s})_1) \setminus \{(h, z)\}, & \text{if}(\mathbf{x}_\mathbf{s})_1[h, z] \in \{[], [x, y]\}, y < t - \mathbf{D} \\ \mathcal{K}(\mathbf{a_p}) = \mathcal{K}((\mathbf{x}_\mathbf{s})_p) \setminus \{h\} & \\ \mathbf{a_1}[h, z] = [x, t] & \text{if } (\mathbf{x}_\mathbf{s})_1[h, z] = [x] \\ \mathbf{a_1}[h, z] = [w, t] & \text{if}(\mathbf{x}_\mathbf{s})_1[h, z] = [x, y, w], y < t - \mathbf{D} \\ \nabla & \text{otherwise} \end{cases}$ $(\varepsilon', \omega_7', \mathbf{x}_\mathbf{s}') \equiv \begin{cases} (\lightning, \omega_7, \mathbf{x}_\mathbf{s}) & \text{if } h = \nabla \\ (\blacktriangleright, \text{HUH}, \mathbf{x}_\mathbf{s}) & \text{otherwise if } \mathbf{a} = \nabla \\ (\blacktriangleright, \text{OK}, \mathbf{a}) & \text{otherwise} \end{cases}$
$\Omega_\beta(\varrho, \omega, \mu, (\text{x}, \text{y}))$ $\text{yield} = 16$ $g = 10$	$\text{let } O = \omega_7$ $\text{let } h = \begin{cases} \mu_{o\cdots+32} & \text{if } \mathbb{N}_{o\cdots+32} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$ $(\varepsilon', \omega_7', \mathbf{x}_y') \equiv \begin{cases} (\lightning, \omega_7, \mathbf{x}_y) & \text{if } h = \nabla \\ (\blacktriangleright, \text{OK}, h) & \text{otherwise} \end{cases}$

B.8. 优化函数

假设有一个精炼的上下文对$(\mathbf{m}, \mathbf{e}) \in (\mathbb{D}\langle\mathbb{N} \to \mathbf{M}\rangle, [\![\mathbb{G}]\!])$，两者初始均为空。在主机调用期间，除非另有说明，PVM 状态中的元素（除明确定义的 gas 计数器外）均保持不变。

(B.21) $$\varrho' \equiv \varrho - g$$

(B.22) $$(\varepsilon', \omega', \mu') = \begin{cases} (\infty, \omega, \mu) & \text{if } \varrho < g \\ (\blacktriangleright, \omega, \mu) & \text{except as indicated below} \quad \text{otherwise} \end{cases}$$

函数标识符 gas 使用量	状态变更
$\Omega_H(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}), s, \mathbf{d}, t)$ $\text{historical lookup} = 17$ $g = 10$	$\text{let } \mathbf{a} = \begin{cases} \mathbf{d}[s] & \text{if } \omega_7 = 2^{64} - 1 \wedge s \in \mathcal{K}(\mathbf{d}) \\ \mathbf{d}[\omega_7] & \text{if } \omega_7 \in \mathcal{K}(\mathbf{d}) \\ \emptyset & \text{otherwise} \end{cases}$ $\text{let } [h, o] = \omega_{8\cdots+2}$ $\text{let } \mathbf{v} = \begin{cases} \nabla & \text{if } \mathbb{N}_{h\cdots+32} \notin \mathbb{N}_\mu \\ \emptyset & \text{otherwise if } \mathbf{a} = \emptyset \\ \Lambda(\mathbf{a}, t, \mu_{h\cdots+32}) & \text{otherwise} \end{cases}$

$$\text{let } f = min(\omega_{10}, |\mathbf{v}|)$$

$$\text{let } l = min(\omega_{11}, |\mathbf{v}| - f)$$

$$(\varepsilon', \omega_7', \mu'_{o\cdots+l}) \equiv \begin{cases} (\lightning, \omega_7, \mu_{o\cdots+l}) & \text{if } \mathbf{v} = \nabla \vee \mathbb{N}_{o\cdots+l} \notin \mathbb{N}^*_\mu \\ (\blacktriangleright, \text{NONE}, \mu_{o\cdots+l}) & \text{otherwise if } \mathbf{v} = \emptyset \\ (\blacktriangleright, |\mathbf{v}|, \mathbf{v}_{f\cdots+l}) & \text{otherwise} \end{cases}$$

$\Omega_Y(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}), i, p, \mathbf{o}, \bar{\mathbf{i}})$

$\text{fetch} = 18$

$g = 10$

$$\text{let } \mathbf{V} = \begin{cases} \mathcal{E}(p) & \text{if } \omega_{10} = 0 \\ \mathbf{o} & \text{if } \omega_{10} = 1 \\ p_{\mathbf{w}}[\omega_{11}]_{\mathbf{y}} & \text{if } \omega_{10} = 2 \wedge \omega_{11} < |p_{\mathbf{w}}| \\ \mathbf{x} & \text{if } \omega_{10} = 3 \wedge \omega_{11} < |p_{\mathbf{w}}| \wedge \omega_{12} < |p_{\mathbf{w}}[\omega_{11}]_{\mathbf{x}}| \wedge (\mathcal{H}(\mathbf{x}), |\mathbf{x}|) = p_{\mathbf{w}}[\omega_{11}]_{\mathbf{x}}[\omega_{12}] \\ \mathbf{x} & \text{if } \omega_{10} = 4 \wedge \omega_{11} < |p_{\mathbf{w}}[i]|_{\mathbf{x}}| \wedge (\mathcal{H}(\mathbf{x}), |\mathbf{x}|) = p_{\mathbf{w}}[i]_{\mathbf{x}}[\omega_{11}] \\ \bar{\mathbf{i}}[\omega_{11}]_{\mathbf{x}_{12}} & \text{if } \omega_{10} = 5 \wedge \omega_{11} < |\bar{\mathbf{i}}| \wedge \omega_{12} < |\bar{\mathbf{i}}[\omega_{11}]| \\ \bar{\mathbf{i}}[i]_{\mathbf{x}_{11}} & \text{if } \omega_{10} = 6 \wedge \omega_{11} < |\bar{\mathbf{i}}[i]| \\ p_{\mathbf{p}} & \text{if } \omega_{10} = 7 \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{let } O = \omega_7$$

$$\text{let } f = min(\omega_8, |\mathbf{v}|)$$

$$\text{let } l = min(\omega_9, |\mathbf{v}| - f)$$

$$(\varepsilon', \omega_7', \mu'_{o\ldots+l}) \equiv \begin{cases} (\lightning, \omega_7, \mu_{o\ldots+l}) & \text{if } \mathbb{N}_{o\ldots+l} \notin \mathbb{V}^*_\mu \\ (\blacktriangleright, \text{NONE}, \mu_{o\ldots+l}) & \text{otherwise if } \mathbf{v} = \emptyset \\ (\blacktriangleright, |\mathbf{v}|, \mathbf{v}_{f\ldots+l}) & \text{otherwise} \end{cases}$$

$\Omega_E(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}), \varsigma)$

$\text{export} = 19$

$g = 10$

$$\text{let } p = \omega_7$$

$$\text{let } z = min(\omega_8, \mathsf{W}_G)$$

$$\text{let } \mathbf{x} = \begin{cases} \mathcal{P}_{\mathsf{W}_G}(\mu_{p\cdots+z}) & \text{if } \mathbb{N}_{p\cdots+z} \subseteq \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

$$(\varepsilon', \omega_7', \mathbf{e}') \equiv \begin{cases} (\lightning, \omega_7, \mathbf{e}) & \text{if } \mathbf{x} = \nabla \\ (\blacktriangleright, \text{FULL}, \mathbf{e}) & \text{otherwise if } \varsigma + |\mathbf{e}| \geq \mathcal{W}_X \\ (\blacktriangleright, \varsigma + |\mathbf{e}|, \mathbf{e} \doteleq \mathbf{x}) & \text{otherwise} \end{cases}$$

$\Omega_M(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}))$

$\text{machine} = 20$

$g = 10$

$$\text{let } [p_o, p_z, i] = \omega_{7\cdots+3}$$

$$\text{let } \mathbf{p} = \begin{cases} \mu_{p_o\cdots+p_z} & \text{if } \mathbb{N}_{p_o\cdots+p_z} \subseteq \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } n = \min(n \in \mathbb{N}, n \notin \mathcal{K}(\mathbf{m}))$$

$$\text{let } \mathbf{u} = (\mathbf{V}\colon [0,0,\ldots], \mathbf{A}\colon [\emptyset, \emptyset, \ldots])$$

$$(\varepsilon', \omega_7', \mathbf{m}) = \begin{cases} (\lightning, \omega_7, \mathbf{m}) & \text{if } \mathbf{p} = \nabla \\ (\blacktriangleright, \text{HUH}, \mathbf{m}) & \text{otherwise if } \text{deblob}(\mathbf{p}) = \nabla \\ (\blacktriangleright, n, \mathbf{m} \cup \{n \mapsto (\mathbf{p}, \mathbf{u}, i)\}) & \text{otherwise} \end{cases}$$

$\Omega_P(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}))$

$$\text{let } [n, o, s, z] = \omega_{7\ldots+4}$$

---

peek $= 21$

$g = 10$

$$(\varepsilon', \omega_7', \mu') \equiv \begin{cases} (\lightning, \omega_7, \mu) & \text{if } \mathbb{N}_{o\cdots+z} \notin \mathbb{N}^*_\mu \\ (\blacktriangleright, \text{WHO}, \mu) & \text{otherwise if } n \notin \mathcal{K}(\mathbf{m}) \\ (\blacktriangleright, \text{OOB}, \mu) & \text{otherwise if } \mathbb{N}_{s\cdots+z} \notin \mathbb{V}_{\mathbf{m}[n]_\mathbf{u}} \\ (\blacktriangleright, \text{OK}, \mu') & \text{otherwise} \\ \text{where } \mu' = \mu & \text{except } \mu_{o\cdots+z} = (\mathbf{m}[n]_\mathbf{u})_{s\cdots+z} \end{cases}$$

$\Omega_Q(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}))$

poke $= 22$

$g = 10$

$$\text{let } [n, s, o, z] = \omega_{7\cdots+4}$$

$$(\varepsilon', \omega_7', \mathbf{m}') = \begin{cases} (\lightning, \omega_7, \mathbf{m}) & \text{if } \mathbb{N}_{s\cdots+z} \notin \mathbb{N}_\mu \\ (\blacktriangleright, \text{WH0}, \mathbf{m}) & \text{otherwise if } n \notin \mathcal{K}(\mathbf{m}) \\ (\blacktriangleright, \text{OOB}, \mathbf{m}) & \text{otherwise if } \mathbb{N}_{o\cdots+z} \notin \mathbb{V}^*_{\mathbf{m}[n]_\mathbf{u}} \\ (\blacktriangleright, \text{OK}, \mathbf{m}') & \text{otherwise} \\ \text{where } \mathbf{m}' = \mathbf{m} & \text{except } (\mathbf{m}'[n]_\mathbf{u})_{o\cdots+z} = \mu_{s\cdots+z} \end{cases}$$

$\Omega_Z(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}))$

zero $= 23$

$g = 10$

$$\text{let } [n, p, c] = \omega_{7\cdots+3}$$

$$\text{let } \mathbf{u} = \begin{cases} \mathbf{m}[n]_\mathbf{u} & \text{if } n \in \mathcal{K}(\mathbf{m}) \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } \mathbf{u}' = \mathbf{u} \text{ except } \begin{cases} (\mathbf{u}'_\mathbf{V})_{p\mathbf{z}_p\cdots+c\mathbf{z}_p} = [0,0,\ldots] \\ (\mathbf{u}'_\mathbf{A})_{p\cdots+c} = [\mathbf{W}, \mathbf{W}, \ldots] \end{cases}$$

$$(\omega_7', \mathbf{m}') \equiv \begin{cases} (\text{HUH}, \mathbf{m}) & \text{if } p < 16 \vee p + c \geq 2^{32}/\mathbf{z}_P \\ (\text{WHO}, \mathbf{m}) & \text{if } \mathbf{u} = \nabla \\ (\text{OK}, \mathbf{m}') & \text{, where } \mathbf{m}' = \mathbf{m} \text{ except } \mathbf{m}'[n]_\mathbf{u} = \mathbf{u}' \quad \text{otherwise} \end{cases}$$

$\Omega_V(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}))$

void $= 24$

$g = 10$

$$\text{let } [n, p, c] = \omega_{7\cdots+3}$$

$$\text{let } \mathbf{u} = \begin{cases} \mathbf{m}[n]_\mathbf{u} & \text{if } n \in \mathcal{K}(\mathbf{m}) \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } \mathbf{u}' = \mathbf{u} \text{ except } \begin{cases} (\mathbf{u}'_\mathbf{V})_{p\mathbf{z}_p\cdots+c\mathbf{z}_p} = [0,0,\ldots] \\ (\mathbf{u}'_\mathbf{A})_{p\cdots+c} = [\emptyset, \emptyset, \ldots] \end{cases}$$

$$(\omega_7', \mathbf{m}') = \begin{cases} (\text{WH0}, \mathbf{m}) & \text{if} \mathbf{u} = \nabla \\ (\text{HUH}, \mathbf{m}) & \text{otherwise if } p < 16 \vee p + c \geq 2^{22}/\mathbf{z}_P \vee \exists i \in \mathbb{N}_{p-+c} : (\mathbf{u}_\mathbf{A})_i = \emptyset \\ (\text{OK}, \mathbf{m}') & \text{otherwise} \\ \text{where } \mathbf{m}' = \mathbf{m} & \text{except } \mathbf{m}'[n]_\mathbf{u} = \mathbf{u}' \end{cases}$$

$\Omega_K(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}))$

invoke $= 25$

$g = 10$

$$\text{let } [n, o] = \omega_{7,8}$$

$$\text{let } (g, \mathbf{w}) = \begin{cases} (g, \mathbf{w}) : \mathcal{E}_8(g) \sim \overline{\mathcal{E}_8^\#(\mathbf{w})} = \mu_{o\cdots+112} & \text{if } \mathbb{N}_{o\cdots+112} \subset \mathbb{V}^*_\mu \\ (\nabla, \nabla) & \text{otherwise} \end{cases}$$

$$\text{let } (c, i', g', \mathbf{w}', \mathbf{u}') = \Psi(\mathbf{m}[n]_\mathbf{p}, \mathbf{m}[n]_i, g, \mathbf{w}, \mathbf{m}[n]_\mathbf{u})$$

$$\text{let } \mu^* = \mu \text{ except } \mu^*_{o\cdots+112} = \mathcal{E}_8(g') \quad \overline{\mathcal{E}_8^\#(\mathbf{w}')}$$

$$\text{let } \mathbf{m}^* = \mathbf{m} \text{ except } \begin{cases} \mathbf{m}^*[n]_\mathbf{u} = \mathbf{u}' \\ \mathbf{m}^*[n]_i = \begin{cases} i' + 1 & \text{if } c \in \{\hbar\} \times \mathbb{N}_R \\ i' & \text{otherwise} \end{cases} \end{cases}$$

$$(\varepsilon', \omega_7', \omega_8', \mu', \mathbf{m}') = \begin{cases} (\lightning, \omega_7, \omega_8, \mu, \mathbf{m}) & \text{if } g = \nabla \\ (\blacktriangleright, \text{WH0}, \omega_8, \mu, \mathbf{m}) & \text{otherwise if } n \notin \mathbf{m} \\ (\blacktriangleright, \text{HOST}, h, \mu^*, \mathbf{m}^*) & \text{otherwise if } c = \hbar \times h \\ (\blacktriangleright, \text{FAULT}, x, \mu^*, \mathbf{m}^*) & \text{otherwise if } c = \lightning \times x \\ (\blacktriangleright, \text{OOG}, \omega_8, \mu^*, \mathbf{m}^*) & \text{otherwise if } c = \infty \\ (\blacktriangleright, \text{PANTO}, \omega_8, \mu^*, \mathbf{m}^*) & \text{otherwise if } c = \lightning \\ (\blacktriangleright, \text{HALT}, \omega_8, \mu^*, \mathbf{m}^*) & \text{otherwise if } c = \blacksquare \end{cases}$$

$\Omega_X(\varrho, \omega, \mu, (\mathbf{m}, \mathbf{e}))$

expunge $= 26$

$g = 10$

$$\text{let } n = \omega_7$$

$$(\omega_7', \mathbf{m}') \equiv \begin{cases} (\text{wH0}, \mathbf{m}) & \text{if } n \neq \mathcal{K}(\mathbf{m}) \\ (\mathbf{m}[n]_i, \mathbf{m} \setminus n) & \text{otherwise} \end{cases}$$

$\Omega_H(\varrho, \omega, \mu, (\text{m}, \text{e}), \text{s}, \text{d}, \text{t})$

historical lookup=17

$\text{g} = 10$

$$\text{let a} = \begin{cases} \text{d}[\text{s}] & \text{if } \omega_7 = 2^{64} - 1 \wedge \text{s} \in \mathcal{K}(\text{d}) \\ \text{d}[\omega_7] & \text{if } \omega_7 \in \mathcal{K}(\text{d}) \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{let } [h, \text{o}] = \omega_{8\cdots+2}$$

$$\text{let v} = \begin{cases} \nabla & \text{if } \mathbb{N}_{h\cdots+32} \notin \mathbb{N}_\mu \\ \emptyset & \text{otherwise if a} = \emptyset \\ \Lambda(\text{a}, \text{t}, \mu_{h\cdots+32}) & \text{otherwise} \end{cases}$$

$$\text{let f} = \min(\omega_{10}, |\text{v}|)$$

$$\text{let l} = \min(\omega_{11}, |\text{v}| - \text{f})$$

$$(\varepsilon', \omega_7', \mu_{\text{o}\cdots+\text{l}}') \equiv \begin{cases} (\lightning, \omega_7, \mu_{\text{o}\cdots+\text{l}}) & \text{if v} = \nabla \vee \mathbb{N}_{\text{o}\cdots+\text{l}} \notin \mathbb{N}_\mu^* \\ (\blacktriangleright, \text{NONE}, \mu_{\text{o}\cdots+\text{l}}) & \text{otherwise if v} = \emptyset \\ (\blacktriangleright, |\text{v}|, \text{v}_{\text{f}\cdots+\text{l}}) & \text{otherwise} \end{cases}$$

$\Omega_Y(\varrho, \omega, \mu, (\text{m}, \text{e}), \text{i}, \text{p}, \text{o}, \bar{\text{i}})$

fetch $= 18$

$\text{g} = 10$

$$\text{let V} = \begin{cases} \mathcal{E}(\text{p}) & \text{if } \omega_{10} = 0 \\ \text{o} & \text{if } \omega_{10} = 1 \\ \text{p}_\text{w}[\omega_{11}]_\text{y} & \text{if } \omega_{10} = 2 \wedge \omega_{11} < |\text{p}_\text{w}| \\ \text{x} & \text{if } \omega_{10} = 3 \wedge \omega_{11} < |\text{p}_\text{w}| \wedge \omega_{12} < |\text{p}_\text{w}[\omega_{11}]_\text{x}| \wedge (2\text{t}(\text{x}), |\text{x}|) = \text{p}_\text{w}[\omega_{11}]_\text{x}[\omega_{12}] \\ \text{x} & \text{if} \omega_{10} = 4 \wedge \omega_{11} < |\text{p}_\text{w}[\text{i}]|_\text{x}| \times (\mathcal{H}(\text{x}), |\text{x}|) = \text{p}_\text{w}[\text{i}]_\text{x}[\omega_{11}] \\ \bar{\text{i}}[\omega_{11}]_{\text{x}_{12}} & \text{if} \omega_{10} = 5 \wedge \omega_{11} < |\bar{\text{i}}| \wedge \omega_{12} < |\bar{\text{i}}[\omega_{11}]| \\ \bar{\text{i}}[\text{i}]_{\text{x}_{11}} & \text{if} \omega_{11} = 6 \times \omega_{11} < |\bar{\text{i}}[\text{i}]| \\ \text{p}_\text{p} & \text{if} \omega_{10} = 7 \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{let O} = \omega_7$$

$$\text{let f} = \min(\omega_8, |\text{v}|)$$

$$\text{let l} = \min(\omega_9, |\text{v}| - \text{f})$$

$$(\varepsilon', \omega_7', \mu_{\text{o}\ldots+\text{l}}') \equiv \begin{cases} (\lightning, \omega_7, \mu_{\text{o}\ldots+\text{l}}) & \text{if } \mathbb{N}_{\text{o}\ldots+\text{l}} \notin \mathbb{V}_\mu^* \\ (\blacktriangleright, \text{NONE}, \mu_{\text{o}\ldots+\text{l}}) & \text{otherwise if v} = \emptyset \\ (\blacktriangleright, |\text{v}|, \text{v}_{\text{f}\ldots+\text{l}}) & \text{otherwise} \end{cases}$$

$\Omega_E(\varrho, \omega, \mu, (\text{m}, \text{e}), \varsigma)$

export $= 19$

$\text{g} = 10$

$$\text{let p} = \omega_7$$

$$\text{let z} = \min(\omega_8, \text{W}_\text{G})$$

$$\text{let } x = \begin{cases} \mathcal{P}_{W_G}(\mu_{p\cdots+z}) & \text{if } \mathbb{N}_{p\cdots+z} \subseteq \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

$$(\varepsilon', \omega_7', e') \equiv \begin{cases} (\lightning, \omega_7, e) & \text{if } x = \nabla \\ (\blacktriangleright, \text{FULL}, e) & \text{otherwise if } \varsigma + |e| \geq \mathcal{W}_M \\ (\blacktriangleright, \varsigma + |e|, e \hash x) & \text{otherwise} \end{cases}$$

$\Omega_M(\varrho, \omega, \mu, (m, e))$

$\text{machine} = 20$

$g = 10$

$$\text{let } [p_o, p_z, i] = \omega_{7\cdots+3}$$

$$\text{let } p = \begin{cases} \mu_{p_o\cdots+p_z} & \text{if} \mathbb{N}_{p_o\cdots+p_z} \subset \mathbb{V}_\mu \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } n = \min(n \in \mathbb{N}, n \notin \mathcal{K}(m))$$

$$\text{let } u = (V{:}\,[0,0,\ldots], A{:}\,[\emptyset, \emptyset, \ldots])$$

$$(\varepsilon', \omega_7', m) = \begin{cases} (\lightning, \omega_7, m) & \text{if } p = \nabla \\ (\blacktriangleright, \text{HUH}, m) & \text{otherwise if deblob}(p) \\ (\blacktriangleright, n, m \cup \{n \mapsto (p, u, i)\}) & \text{otherwise} \end{cases}$$

$\Omega_P(\varrho, \omega, \mu, (m, e))$

$\text{peek} = 21$

$g = 10$

$$\text{let } [n, o, s, z] = \omega_{7\cdots+4}$$

$$(\varepsilon', \omega_7', \mu') \equiv \begin{cases} (\lightning, \omega_7, \mu) & \text{if} \mathbb{N}_{o\cdots+z} \notin \mathbb{N}^*_\mu \\ (\blacktriangleright, \text{WHO}, \mu) & \text{otherwise if } n \notin \mathcal{K}(m) \\ (\blacktriangleright, \text{OOB}, \mu) & \text{otherwise if } \mathbb{N}_{s\cdots+z} \notin \mathbb{V}_{m[n]_u} \\ (\blacktriangleright, \text{OK}, \mu') & \text{otherwise} \\ \text{where } \mu' = \mu & \text{except } \mu_{o\cdots+z} = (m[n]_u)_{s\cdots+z} \end{cases}$$

$\Omega_O(\varrho, \omega, \mu, (m, e))$

$\text{poke} = 22$

$g = 10$

$$\text{let } [n, s, o, z] = \omega_{7\cdots+4}$$

$$(\varepsilon', \omega_7', m') = \begin{cases} (\lightning, \omega_7, m) & \text{if } \mathbb{N}_{s\cdots+z} \notin \mathbb{N}_\mu \\ (\blacktriangleright, \text{WHO}, m) & \text{otherwise if } n \notin \mathcal{K}(m) \\ (\blacktriangleright, \text{OOB}, m) & \text{otherwise if } \mathbb{N}_{o\cdots+z} \notin \mathbb{V}^*_{m[n]_u} \\ (\blacktriangleright, \text{OK}, m') & \text{otherwise} \\ \text{where } m' = m \text{ except } (m'[n]_u)_{o\cdots+z} = \mu_{s\cdots+z} & \end{cases}$$

$\Omega_Z(\varrho, \omega, \mu, (m, e))$

$\text{zero} = 23$

$g = 10$

$$\text{let } [n, p, c] = \omega_{7\cdots+3}$$

$$\text{let } u = \begin{cases} m[n]_u & \text{if } n \in \mathcal{K}(m) \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } u' = u \text{ except } \begin{cases} (u'_V)_{pZ_p\cdots+cZ_p} = [0,0,\ldots] \\ (u'_A)_{p\cdots+c} = [W, W, \ldots] \end{cases}$$

$$(\omega_7', m') \equiv \begin{cases} (\text{HUH}, m) & \text{if } p < 16 \vee p + c \geq 2^{32}/z_P \\ (\text{WHO}, m) & \text{if } u = \nabla \\ (\text{OK}, m') & \text{where } m' = m \text{ except } m'[n]_u = u' \quad \text{otherwise} \end{cases}$$

$\Omega_V(\varrho, \omega, \mu, (m, e))$

$\text{void} = 24$

$g = 10$

$$\text{let } [n, p, c] = \omega_{7\cdots+3}$$

$$\text{let } u = \begin{cases} m[n]_u & \text{if } n \in \mathcal{K}(m) \\ \nabla & \text{otherwise} \end{cases}$$

$$\text{let } u' = u \text{ except } \begin{cases} (u'_V)_{pZ_p\cdots+cZ_p} = [0,0,\ldots] \\ (u'_A)_{p\cdots+c} = [\emptyset, \emptyset, \ldots] \end{cases}$$

$$(\omega_7', m') = \begin{cases} (\text{WHO}, m) & \text{if } u = \nabla \\ (\text{HUH}, m) & \text{otherwise if } p < 16 \vee p + c \geq 2^{22}/z_P \vee \exists i \in \mathbb{N}_{p-+c} : (u_A)_i = \emptyset \\ (\text{OK}, m') & \text{otherwise} \\ \text{where } m' = m \text{ except } m'[n]_u = u' & \end{cases}$$

$\Omega_K(\varrho, \omega, \mu, (m, e))$
invoke $= 25$
$g = 10$

$$\text{let } [n, o] = \omega_{7,8}$$
$$\text{let } (g, w) = \begin{cases} (g, w) : \mathcal{E}_8(g) \sim \overline{\mathcal{E}_8^\#(w)} = \mu_{o\cdots+112} & \text{if } \mathbb{N}_{o\cdots+112} \subset \mathbb{V}_\mu^* \\ (\nabla, \nabla) & \text{otherwise} \end{cases}$$
$$\text{let } (c, i', g', w', u') = \Psi(m[n]_p, m[n]_i, g, w, m[n]_u)$$
$$\text{let } \mu^* = \mu \text{ except } \mu^*_{o\cdots+112} = \mathcal{E}_8(g') \quad \overline{\mathcal{E}_8^\#(w')}$$
$$\text{let } m^* = m \text{ except } \begin{cases} m^*[n]_u = u' \\ m^*[n]_i = \begin{cases} i' + 1 & \text{if } c \in \{\hbar\} \times \mathbb{N}_R \\ i' & \text{otherwise} \end{cases} \end{cases}$$
$$(\varepsilon', \omega_7', \omega_8', \mu', m') = \begin{cases} (\lightning, \omega_7, \omega_8, \mu, m) & \text{if } g = \nabla \\ (\blacktriangleright, \text{WHO}, \omega_8, \mu, m) & \text{otherwise if } n \notin m \\ (\blacktriangleright, \text{HOST}, h, \mu^*, m^*) & \text{otherwise if } c = \hbar \times h \\ (\blacktriangleright, \text{FAULT}, x, \mu^*, m^*) & \text{otherwise if } c = \lightning \times x \\ (\blacktriangleright, \text{OOG}, \omega_8, \mu^*, m^*) & \text{otherwise if } c = \infty \\ (\blacktriangleright, \text{PANTO}, \omega_8, \mu^*, m^*) & \text{otherwise if } c = \lightning \\ (\blacktriangleright, \text{HALT}, \omega_8, \mu^*, m^*) & \text{otherwise if } c = \blacksquare \end{cases}$$

$\Omega_X(\varrho, \omega, \mu, (m, e))$
expunge $= 26$
$g = 10$

$$\text{let } n = \omega_7$$
$$(\omega_7', m') \equiv \begin{cases} (\text{WHO}, m) & \text{if } n \neq \mathcal{K}(m) \\ (m[n]_i, m \setminus n) & \text{otherwise} \end{cases}$$

附 录 C. 序列化编解码器

C.1. 常用术语

我们的编解码器函数 $\mathcal{E}$ 将某个项序列化为八位字节序列。我们定义了 $\mathcal{E}$ 的逆函数 $\mathcal{E}^{-1}$，用于将序列解码回原始值。该编解码器设计确保每个八位字节序列都唯一对应一个编码值，特殊情况下会使用特定的编解码器函数。

C.1.1. 简单编码

我们将 $\emptyset$ 的序列化定义为空序列：

$$\mathcal{E}(\emptyset) \equiv [] \tag{C.1}$$

我们将空集$\emptyset$序列化为空序列：

$$\mathcal{E}(x \in \mathbb{Y}) \equiv x \tag{C.2}$$

我们将匿名元组定义为其编码元素的连接结果：

$$\mathcal{E}((a, b, ...)) \equiv \mathcal{E}(a) \frown \mathcal{E}(b) \frown ... \tag{C.3}$$

向序列化函数传递多个参数，等同于传递一个包含这些参数的元组。形式上：

$$\mathcal{E}(a, b, ...) \equiv \mathcal{E}((a, b, ...)) \tag{C.4}$$

C.1.2. 整数编码

我们首先定义基本的自然数序列化函数，这些函数以下述最终序列的八位字节长度作为索引。数值采用常见的小端序编码方式，这一规则在协议的几乎所有整数编码中都得到了应用。形式上：

$$\mathcal{E}_{l \in \mathbb{N}} \cdot \begin{cases} \mathbb{N}_{2^{8l}} \to \mathbb{Y}_l \\ x \mapsto \begin{cases} [] & \text{if } l = 0 \\ [x \bmod 256] \frown \mathcal{E}_{l-1}(\lfloor \frac{x}{256} \rfloor) & \text{otherwise} \end{cases} \end{cases} \tag{C.5}$$

我们定义了一种自然数的通用序列化方法，能够编码最大至 2^{64} 的自然数，如下：

$$\mathcal{E} \cdot \begin{cases} \mathbb{N}_{2^{64}} \to \mathbb{Y}_{1:9} \\ x \mapsto \begin{cases} [0] & \text{if } x = 0 \\ [2^8 - 2^{8-l} + \lfloor \frac{x}{2^{8l}} \rfloor] \frown \mathcal{E}_l(x \bmod 2^{8l}) & \text{if } \exists l \in \mathbb{N}_8 \cdot 2^{7l} \le x < 2^{7(l+1)} \\ [2^8 - 1] \frown \mathcal{E}_8(x) & \textbf{otherwise if } x < 2^{64} \end{cases} \end{cases} \tag{C.6}$$

目前，这种方法仅用于对可变长度序列的长度前缀进行编码。

C.1.3. 序列编码

对于 $\mathcal{E}$ 定义域中的任意子集 T，我们定义了序列序列化函数 $\mathcal{E}(\llbracket \mathrm{T} \rrbracket)$。该函数只需将序列中每个元素的序列化结果依次连接起来即可。

$$\mathcal{E}(\left[i_0, i_1, \ldots\right]) \equiv \mathcal{E}(i_0) \frown \mathcal{E}(i_1) \frown \ldots \tag{C.7}$$

因此，固定长度的八位字节序列（如哈希值 $\mathbb{H}$ 及其各种变体）具有一致的序列化形式。

C.1.4. 判别符编码

当我们处理一组包含不同类型元组或不同长度序列的异构项时，需要一个判别符来明确每个项的性质，以确保能够成功反序列化。这个判别符被编码为自然数，并紧贴在相应项的前面进行编码。

在序列化长度可变的序列项（如一般的二进制大对象 $\mathbb{Y}$ 或无界限的数字序列 $\llbracket \mathbb{N} \rrbracket$）时，我们通常会使用长度判别符（但对于固定长度项，如哈希 $\mathbb{H}$，则会省略此步骤）。[19]在这种情况下，我们只需在编码前为该项添加一个长度前缀。因此，对于属于集合$(x \in \mathbb{Y}, \ldots)$ 的某个项 y，我们通常将其序列化形式定义为包含长度信息的 x，以表明这是一个大小可变的项，需要长度判别符。我们可以表示为：$\mathcal{E}(|x|) \frown \mathcal{E}(x) \frown \ldots$（其中 $|x|$ 表示 x 的长度）。为避免在这种情况下重复书写该项，形式上：

$$\updownarrow x \equiv (|x|, x) \text{thus} \mathcal{E}(\updownarrow x) \equiv \mathcal{E}(|x|) \frown \mathcal{E}(x) \tag{C.8}$$

我们还特别定义了一个便捷的判别符操作符 $\updownarrow x$，用于标识与空集 $\emptyset$ 并集中由某个可序列化集合所定义的项（通常对于某个集合 S，表示为 $S?$）。

$$¿x \equiv \begin{cases} 0 & \text{if} x = \emptyset \\ (1, x) & \text{otherwise} \end{cases} \tag{C.9}$$

C.1.5. 位序列编码

位序列 $b \in \mathbb{B}$，是一种特殊情况。为了避免浪费，我们不将每个位单独编码为八位字节，而是按从低到高的顺序，将位打包成八位字节，并形成字节流。对于可变长度序列，我们像处理一般情况那样，在序列前面加上长度前缀。

$$\mathcal{E}(b \in \mathbb{B}) \equiv \begin{cases} [] & \text{if} b = [] \\ \left[\sum_{i=0}^{min(8,|b|)} b_i \cdot 2^i\right] \frown \mathcal{E}(b_{8\ldots}) & \text{otherwise} \end{cases} \tag{C.10}$$

C.1.6. 字典编码

一般来说，字典通常会被直接放入默克尔树（详见附录 $\mathbf{E}$）。不过，对于小型字典，我们可以将其合理地编码为一个按键排序的键值对序列。形式上：

$$\forall K, V: \mathcal{E}(d \in \mathbb{D}\langle K \to V \rangle) \equiv \mathcal{E}(\updownarrow [k \between (\mathcal{E}(k), \mathcal{E}(d[k])) | k \in \mathcal{K}(d)]) \tag{C.11}$$

[19] 注意，由于某些特定值可能同时属于需要判别符和不需要判别符的集合，因此我们无法简单地引入一个统一函数来根据项的限制进行序列化。为此，我们需要构建一个比基本集合论更复杂的体系，这个体系不仅要关注值本身，还要考虑值所属的项或来源，以便能够更简洁地处理这些情况。

C.1.7. 集合编码

对于任何尚未定义特定编码的集合值，我们将其序列化定义为集合元素按一定顺序进行序列化。形式上：

$$\mathcal{E}(\{a, b, c, ...\}) \equiv \mathcal{E}(a) \frown \mathcal{E}(b) \frown \mathcal{E}(c) \frown ... \text{where } a < b < c < ... \tag{C.12}$$

C.2. 区块序列化

一个区块 $\mathbf{B}$ 按照其元素的常规顺序被序列化为一个元组，具体如方程 4.2、4.3 和 5.1 所描述。对于区块头部分，我们特别定义了常规序列化和无符号序列化两种方式，分别记为 $\mathcal{E}_U$ 。形式上：

$$\mathcal{E}(\mathbf{B}) = \mathcal{E}\left(\mathbf{H}, \mathcal{E}_T(\mathbf{E}_T), \mathcal{E}_P(\mathbf{E}_P), \mathcal{E}_G(\mathbf{E}_G), \mathcal{E}_A(\mathbf{E}_A), \mathcal{E}_D(\mathbf{E}_D)\right) \tag{C.13}$$

$$\mathcal{E}_T(\mathbf{E}_T) = \mathcal{E}(\updownarrow \mathbf{E}_T) \tag{C.14}$$

$$\mathcal{E}_P(\mathbf{E}_P) = \mathcal{E}(\updownarrow [(s, \updownarrow p) | (s, p) \lessdot \mathbf{E}_P]) \tag{C.15}$$

$$\mathcal{E}_G(\mathbf{E}_G) = \mathcal{E}(\updownarrow [(w, \mathcal{E}_4(t), \updownarrow a) | (w, t, a) \lessdot \mathbf{E}_G]) \tag{C.16}$$

$$\mathcal{E}_A(\mathbf{E}_A) = \mathcal{E}(\updownarrow [(a, f, \mathcal{E}_2(v), s) | (a, f, v, s) \lessdot \mathbf{E}_A]) \tag{C.17}$$

$$\mathcal{E}_D((\mathbf{v}, \mathbf{c}, \mathbf{f})) = \mathcal{E}(\updownarrow [(r, \mathcal{E}_4(a), [(v, \mathcal{E}_2(i), s) | (v, i, s) \lessdot \mathbf{j}]) | (r, a, \mathbf{j}) \lessdot \mathbf{v}], \updownarrow \mathbf{c}, \updownarrow \mathbf{f} \tag{C.18}$$

$$\mathcal{E}(\mathbf{H}) = \mathcal{E}_U(\mathbf{H}) \frown \mathcal{E}(\mathbf{H}_s) \tag{C.19}$$

$$\mathcal{E}_U(\mathbf{H}) = \mathcal{E}(\mathbf{H}_p, \mathbf{H}_r, \mathbf{H}_x) \frown \mathcal{E}_4(\mathbf{H}_t) \frown \mathcal{E}(¿\mathbf{H}_e, ¿\mathbf{H}_w, \updownarrow \mathbf{H}_o, \mathcal{E}_2(\mathbf{H}_i), \mathbf{H}_v) \tag{C.20}$$

$$\mathcal{E}(x \in \mathbb{X}) \equiv \mathcal{E}(x_a, x_s, x_b, x_l) \frown \mathcal{E}_4(x_t) \frown \mathcal{E}(\updownarrow x_{\mathbf{p}}) \tag{C.21}$$

$$\mathcal{E}(x \in \mathbb{S}) \equiv \mathcal{E}(x_h) \frown \mathcal{E}_4(x_l) \frown \mathcal{E}(x_u, x_e) \frown \mathcal{E}_2(x_n) \tag{C.22}$$

$$\mathcal{E}(x \in \mathbb{L}) \equiv \mathcal{E}(\mathcal{E}_4(x_s), x_c, x_y, \mathcal{E}_8(x_g), O(x_{\mathbf{d}}), x_u, x_i, x_x, x_z, x_e) \tag{C.23}$$

$$\mathcal{E}(x \in \mathbb{W}) \equiv \mathcal{E}(x_s, x_x, x_c, x_a, \updownarrow x_{\mathbf{o}}, \updownarrow x_1, \updownarrow x_{\mathbf{r}}, x_g) \tag{C.24}$$

$$\mathcal{E}(x \in \mathbb{P}) \equiv \mathcal{E}(\updownarrow x_{\mathbf{j}}, \mathcal{E}_4(x_h), x_u, \updownarrow x_{\mathbf{p}}, x_{\mathbf{x}}, \updownarrow x_{\mathbf{w}}) \tag{C.25}$$

$$\mathcal{E}(x \in \mathbb{I}) \equiv \mathcal{E}(\mathcal{E}_4(x_s), x_c, \updownarrow x_{\mathbf{y}}, \mathcal{E}_8(x_g), \mathcal{E}_8(x_a), \updownarrow \mathcal{E}_I^{\#}(x_i), \updownarrow [(h, \mathcal{E}_4(i)) | (h, i) \lessdot x_{\mathbf{x}}], \mathcal{E}_2(x_e)) \tag{C.26}$$

$$\mathcal{E}(x \in \mathbb{C}) \equiv \mathcal{E}(x_{\mathbf{y}}, x_r) \tag{C.27}$$

$$\mathcal{E}(x \in \mathbb{T}) \equiv \mathcal{E}(\mathcal{E}_4(x_s), \mathcal{E}_4(x_d), \mathcal{E}_8(x_a), \mathcal{E}(x_m), \mathcal{E}_8(x_g)) \tag{C.28}$$

$$\mathcal{E}(x \in \mathbb{O}) \equiv \mathcal{E}(x_h, x_e, x_a, \updownarrow x_{\mathbf{o}}, x_y, O(x_{\mathbf{d}})) \tag{C.29}$$

$$O(o \in \mathbb{J} \cup \mathbb{Y}) \equiv \begin{cases} (0, \updownarrow o) & \text{if } o \in \mathbb{Y} \\ 1 & \text{if } o = \infty \\ 2 & \text{if } o = \lightning \\ 3 & \text{if } o = \circledcirc \\ 4 & \text{if } o = \text{BAD} \\ 5 & \text{if } o = \text{BIG} \end{cases} \tag{C.30}$$

$$\mathcal{E}_I(h \in \mathbb{H} \cup \mathbb{H}^{\boxplus}, i \in \mathbb{N}_2 15) \equiv \begin{cases} (h, \mathcal{E}_2(i)) & \text{if } h \in \mathbb{H} \\ (r, \mathcal{E}_2(i + 2^{15})) & \text{if } \exists r \in \mathbb{H}, h = r^{\boxplus} \end{cases} \tag{C.31}$$

注意，上述内容中使用了 O 来简洁编码工作项的结果，同时对 $\mathbf{E}_G$ 和 $\mathbf{E}_P$ 进行了微调，以适应它们内部元组所含的可变长度序列项 a 和 p，这些序列项需要添加长度判别符。

附 录 D. 状态默克尔化

Merkle 化过程提供了一种加密承诺，能够高效快速地验证状态中任意信息的真实性。这一过程分为两个阶段来阐述：第一阶段是将 32 字节的序列转化为（无限长的）字节序列，这一过程称为状态序列化；第二阶段则基于这一序列化结果，生成一个 32 字节的承诺，即 Merkle 化结果。

D.1. 序列化

状态序列化主要是把 σ 的所有组件整合到一个映射里，这个映射以 32 字节的状态键为索引，对应着不定长的字节序列。状态键由哈希和章节组件组成，相当于状态组件或 δ 内部字典的服务索引。

我们定义了一个状态键构造函数，记为 C:

$$C: \begin{cases} \mathbb{N}_{2^8} \cup (\mathbb{N}_{2^8}, \mathbb{N}_S) \cup (\mathbb{N}_S, \mathbb{Y}) \to \mathbb{H} \\ i \in \mathbb{N}_{2^8} \mapsto [i, 0, 0, \ldots] \\ (i, s \in \mathbb{N}_S) \mapsto [i, n_0, 0, n_1, 0, n_2, 0, n_3, 0, 0, \ldots] \text{ where } n = \mathcal{E}_4(s) \\ (s, h) \mapsto [n_0, h_0, n_1, h_1, n_2, h_2, n_3, h_3, h_4, h_5, \ldots, h_{27}] \text{ where } n = \mathcal{E}_4(s) \end{cases} \tag{D.1}$$

状态序列化是指将各组件合并后生成的字典。通过密码散列确保，只要输入 C 不重复，状态键就不会重复。形式上，我们定义 T 来将状态 σ 转换为其序列化表示。

$$T(\sigma) = \begin{cases} C(1) \mapsto \mathcal{E}\left(\left[\updownarrow x | x \leqslant \alpha\right]\right), \\ C(2) \mapsto \mathcal{E}(\varphi), \\ C(3) \mapsto \mathcal{E}(\updownarrow [(h, \mathcal{E}_M(\mathbf{b}), s, \updownarrow \mathbf{p}) | (h, \mathbf{b}, s, \mathbf{p}) \leqslant \beta]), \\ C(4) \mapsto \mathcal{E}\left(\left\{\gamma_{\mathbf{k}}, \gamma_{z'}, \begin{cases} 0 \text{if} \gamma_{\mathbf{s}} \in [\mathbb{C}]_{\mathbf{E}} \\ 1 \text{if} \gamma_{\mathbf{s}} \in \left[\mathbb{H}_B\right]_{\mathbf{E}} \end{cases}, \gamma_{\mathbf{s}}, \updownarrow \gamma_{\mathbf{a}}\right\}\right), \\ C(5) \mapsto \mathcal{E}(\updownarrow [x \wr x \in \psi_{\mathbf{g}}], \updownarrow [x \wr x \in \psi_{\mathbf{b}}], \updownarrow [x \wr x \in \psi_{\mathbf{w}}], \updownarrow [x \wr x \in \psi_{\mathbf{o}}]), \\ C(6) \mapsto \mathcal{E}(\eta), \\ C(7) \mapsto \mathcal{E}(\iota), \\ C(8) \mapsto \mathcal{E}(\kappa), \\ C(9) \mapsto \mathcal{E}(\lambda), \\ C(10) \mapsto \mathcal{E}([¿(w, \mathcal{E}_4(t)) | (w, t) < \rho]), \\ C(11) \mapsto \mathcal{E}_4(\tau), \\ C(12) \mapsto \mathcal{E}_4\left(\chi_m, \chi_a, \chi_v\right) \sim \mathcal{E}\left(\chi_{\mathbf{g}}\right), \\ C(13) \mapsto \mathcal{E}(\mathcal{E}_4^{\#}(\pi_V), \mathcal{E}_4^{\#}(\pi_L), \pi_C, \pi_S), \\ C(14) \mapsto \mathcal{E}([\updownarrow (\mathbf{w}, \updownarrow \mathbf{d}) | (\mathbf{w}, \mathbf{d}) \leqslant \mathbf{i} | \mathbf{i} \leqslant \vartheta]), \\ C(15) \mapsto \mathcal{E}(\left[\uparrow \mathbf{i} | \mathbf{i} \leqslant \xi\right]), \\ \forall (s \mapsto \mathbf{a}) \in \delta : C(255, s) \mapsto \mathbf{a}_c \frown \mathcal{E}_8(\mathbf{a}_b, \mathbf{a}_g, \mathbf{a}_m, \mathbf{a}_o) \frown \mathcal{E}_4(\mathbf{a}_i), \\ \forall (s \mapsto \mathbf{a}) \in \delta, (k \mapsto \mathbf{v}) \in \mathbf{a}_{\mathbf{s}} : C(s, \mathcal{E}_4(2^{32} - 1) \frown k_{0\ldots28}) \mapsto \mathbf{v}, \\ \forall (s \mapsto \mathbf{a}) \in \delta, (h \mapsto \mathbf{p}) \in \mathbf{a}_{\mathbf{p}} : C(s, \mathcal{E}_4(2^{32} - 2) \frown h_{1\ldots29}) \mapsto \mathbf{p}, \\ \forall (s \mapsto \mathbf{a}) \in \delta, ((h, l) \mapsto \mathbf{t}) \in \mathbf{a}_{\mathbf{l}} : C(s, \mathcal{E}_4(l) \frown \mathcal{H}(h)_{2\ldots30}) \mapsto \mathcal{E}(\updownarrow [\mathcal{E}_4(x) | x \leqslant \mathbf{t}]) \end{cases} \tag{D.2}$$

大多数行展示的是自然键派生键与状态组件序列化之间的单一映射。而最后四行，每行都定义了一组映射关系，因为这些项适用于所有服务账户，特别是在最后三行中，它们还适用于嵌套字典里的服务键。

另外，状态中所有非判别式的数字都是按照固定长度进行序列化的，长度依据项的大小来确定。

D.2. 默克尔化

定义了 T 之后，我们接下来定义的 $\mathcal{M}_\sigma$ 其余部分，主要是将序列化映射转换为一个加密承诺。这个承诺采用二进制帕特丽夏·默克尔树（Patricia Merkle Tree）的根来表示，其格式经过特别优化，以适应现代计算硬件。通过调整大小，使其更贴合典型的内存布局，并减少了对不可预测分支的依赖。

D.2.1. 节点编码和字典树标识

我们通过根节点的哈希值来标识（子）树，有一个例外：空（子）树被标识为零哈希，即 $\mathbb{H}^0$。

每个节点大小为 512 位（64字节），分为分支节点和叶子节点两种，首位用于区分类型。

在分支节点中，511 位被分为两部分来存储子节点的哈希值：后 255 位用于标识 0 位（左）子树，完整的 256 位则用于标识 1 位（右）子树。

叶子节点分为嵌入值叶子和普通叶子两种，节点的第二个位用来区分这两种类型。

在嵌入值叶子节点中，第一个字节的剩下 6 位用来表示嵌入值的大小。紧接着的 31 个字节存储键的前 31 个字符。最后的 32 个字节用来存放值，如果值不足 32 个字节，就用零来补齐。

在普通叶子节点中，第一个字节的剩余 6 位全部为零。接下来的 31 个字节用来存储键的前 31 个字符。而最后的 32 个字节，则用来存储值的哈希值。

形式上，我们定义两个编码函数：B 和 L。

$$B: \begin{cases} (\mathbb{H}, \mathbb{H}) \to \mathbb{B}_{512} \\ (l, r) \mapsto [0] \frown \text{bits}(l)_{1\ldots} \frown \text{bits}(r) \end{cases} \tag{D.3}$$

$$L: \begin{cases} (\mathbb{H}, \mathbb{Y}) \to \mathbb{B}_{512} \\ (k, v) \mapsto \begin{cases} [1,0] \frown \text{bits}(\mathcal{E}_1(|v|))_{2\ldots} \frown \text{bits}(k)_{\ldots 248} \frown \text{bits}(v) \sim [0,0,\ldots] & \text{if } |v| \le 32 \\ [1,1,0,0,0,0,0,0] \frown \text{bits}(k)_{\ldots 248} \frown \text{bits}(\mathcal{H}(v)) & \text{otherwise} \end{cases} \end{cases} \tag{D.4}$$

然后，我们可以将基本的默克尔化函数定义为 $\mathcal{M}_\sigma$：

$$\mathcal{M}_\sigma(\sigma) \equiv M(\{(\text{bits}(k) \mapsto (k, v)) | (k \mapsto v) \in T(\sigma)\}) \tag{D.5}$$

$$M(d: \mathbb{D}(\mathbb{E} \to (\mathbb{H}, \mathbb{Y}))) \equiv \begin{cases} \mathbb{H}^0 & \text{if } |d| = 0 \\ \mathcal{H}(\text{bits}^{-1}(L(k, v))) & \text{if } \mathcal{V}(d) = \{(k, v)\} \\ \mathcal{H}(\text{bits}^{-1}(B(M(l), M(r)))) & \text{otherwise} \end{cases} \tag{D.6}$$

$$\text{where } \forall b, p: (b \mapsto p) \in d \Leftrightarrow (b_{1\ldots} \mapsto p) \in \begin{cases} l & \text{if } b_0 = 0 \\ r & \text{if } b_0 = 1 \end{cases}$$

附 录 E. 通用默克尔化

E.1. 二进制默克尔树

默克尔树是一种加密数据结构，能为特定的值序列提供哈希保证。其计算复杂度为 $O(N)$，包含证明大小则为 $O(\log(N))$。这种结构设计均衡，确保了所有叶子的最大深度最小，且在该深度下的叶子数量也达到最少。

我们默克尔树的核心是节点函数 $O(N)$，它接收一系列长度为 n 数据块作为输入，并输出这些数据块或相应的哈希值：

$$N: \begin{cases} (\lVert \mathbb{Y}_n \rVert, \mathbb{Y} \to \mathbb{H}) \to \mathbb{Y}_n \cup \mathbb{H} \\ (\mathbf{v}, H) \mapsto \begin{cases} \mathbb{H}^0 & \text{if } |\mathbf{v}| = 0 \\ \mathbf{v}_0 & \text{if } |\mathbf{v}| = 1 \\ H(\$\text{node} \frown N(\mathbf{v}_{\dots\lceil|\mathbf{v}|/2\rceil}, H) \frown N(\mathbf{v}_{\lceil|\mathbf{v}|/2\rceil\dots}, H)) & \text{otherwise} \end{cases} \end{cases} \tag{E.1}$$

敏锐的读者会发现，如果我们的 $\mathbb{Y}_n$ 正好等于 $\mathbb{H}$，那么这个函数总是会得出 $\mathbb{H}$。不过，为了保障其安全性，我们必须谨慎行事，防止存在前像碰撞的风险。为此，我们在哈希中加入了前缀 $node，以最大程度地降低这种风险。只要确保每个项目都用不同的前缀进行哈希，系统就可以被认为是安全的。

我们还定义了追踪函数 T，它能从树顶开始，一路导航到底部，找到与给定序列索引中项目相对应的叶子节点，并返回沿途经过的每个相对节点。这个函数在创建数据包含证明时非常有用。

$$T: \begin{cases} (\lVert \mathbb{Y}_n \rVert, \mathbb{N}_{|\mathbf{v}|}, \mathbb{Y} \to \mathbb{H}) \to \lVert \mathbb{Y}_n \cup \mathbb{H} \rVert \\ (\mathbf{v}, i, H) \mapsto \begin{cases} [N(P^\perp(\mathbf{v}, i), H)] \frown T(P^\intercal(\mathbf{v}, i), i - P_J(\mathbf{v}, i), H) & \text{if } |\mathbf{v}| > 1 \\ [\,] & \text{otherwise} \end{cases} \\ \text{where } P^s(\mathbf{v}, i) \equiv \begin{cases} \mathbf{v}_{\dots|\mathbf{v}|/2|} & \text{if } (i < \lceil|\mathbf{v}|/2\rceil) = s \\ \mathbf{v}_{\lceil|\mathbf{v}|/2\rceil\dots} & \text{otherwise} \end{cases} \\ \text{and } P_J(\mathbf{v}, i) \equiv \begin{cases} 0 & \text{if } i < \lceil|\mathbf{v}|/2\rceil \\ \lceil|\mathbf{v}|/2\rceil & \text{otherwise} \end{cases} \end{cases} \tag{E.2}$$

基于此，我们定义了其他默克尔化函数。

E.1.1. 平衡树

我们将平衡二进制默克尔函数定义为 $\mathcal{M}_B$：

$$\mathcal{M}_B: \begin{cases} (\lVert \mathbb{Y} \rVert, \mathbb{Y} \to \mathbb{H}) \to \mathbb{H} \\ (\mathbf{v}, H) \mapsto \begin{cases} H(\mathbf{v}_0) & \text{if } |\mathbf{v}| = 1 \\ N(\mathbf{v}, H) & \text{otherwise} \end{cases} \end{cases} \tag{E.3}$$

这种方法适合为长度不超过 32 字节的数据创建证明，因为它省去了对序列中每个项目的哈希处理步骤。对于包含更大数据项的序列，建议先对它们进行哈希处理，这样可以确保生成的证明大小最小，因为每个证明通常都会包含一个数据项。

注：在未提供哈希函数参数 H 的情况下，我们可假定使用 Blake 2b 哈希函数，$\mathcal{H}$。

E.1.2. 固定深度树

我们将固定深度的二进制默克尔函数命名为 $\mathcal{M}$。同时，我们定义了两个配套函数：$\mathcal{J}_x$ 和 $\mathcal{L}_x$，它们分别用于处理子树的页面。$\mathcal{L}_x$ 函数提供一个包含哈希处理后的前缀数据叶子的单独页面；而 $\mathcal{J}_x$ 函数则提供到达单个页面的默克尔路径。这两个函数都假设页面大小为 2^x 且对齐，并接收页面索引作为输入参数。

(E.4) $$\mathcal{M}: \begin{cases} (\llbracket \mathbb{Y} \rrbracket, \mathbb{Y} \to \mathbb{H}) \to \mathbb{H} \\ (\mathbf{v}, H) \mapsto N(C(\mathbf{v}, H), H) \end{cases}$$

(E.5) $$\mathcal{J}_x: \begin{cases} (\llbracket \mathbb{Y} \rrbracket, \mathbb{N}_{|\mathbf{v}|}, \mathbb{Y} \to \mathbb{H}) \to \llbracket \mathbb{H} \rrbracket \\ (\mathbf{v}, i, H) \mapsto T(C(\mathbf{v}, H), 2^x i, H) \ldots max\left(0, \left\lceil \log_2\left(max(1, |\mathbf{v}|)\right) - x \right\rceil\right) \end{cases}$$

(E.6) $$\mathcal{L}_x: \begin{cases} (\llbracket \mathbb{Y} \rrbracket, \mathbb{N}_{|\mathbf{v}|}, \mathbb{Y} \to \mathbb{H}) \to \llbracket \mathbb{H} \rrbracket \\ (\mathbf{v}, i, H) \mapsto [H(\$\text{leaf} \frown l) | l \in \mathbf{v}_{2^x i \ldots min(2^x i + 2^x, |\mathbf{v}|)}] \end{cases}$$

为了使 $\mathcal{J}_x$ 证明被接受，我们必须假设目标观察者不仅了解给定索引处的项目值，还掌握由 $\mathcal{L}_x$ 给出的 2^x 大小子树中所有其他叶子的信息。

如上所述，我们可以默认将 H 设定为 Blake 2b 哈希函数。

在证明和默克尔根的计算过程中，我们采用了一个恒定性预处理函数C。这个函数会对所有数据项先加上固定前缀 “leaf” 后进行哈希处理，然后将整体数据大小填充至最近的 2 的幂次方，填充时使用的是零哈希 $\mathbb{H}_0$ 。

(E.7) $$C: \begin{cases} (\llbracket \mathbb{Y} \rrbracket, \mathbb{Y} \to \mathbb{H}) \to \llbracket \mathbb{H} \rrbracket \\ (\mathbf{v}, H) \mapsto \mathbf{v}' \text{where} \begin{cases} |\mathbf{v}'| = 2^{\lceil \log_2(max(1, |\mathbf{v}|)) \rceil} \\ \mathbf{v}'_i = \begin{cases} H(\$1\text{eaf} \frown \mathbf{v}_i) & \text{if } i < |\mathbf{v}| \\ \mathbb{H}_0 & \text{otherwise} \end{cases} \end{cases} \end{cases}$$

E.2. 默克尔式山峦序列

Merkle Mountain Range（MMR）是一种只能追加的加密数据结构，它能为一系列值提供承诺。向 MMR 中添加元素或证明某个元素的存在，其时间和空间复杂度都是 $O(\log(N))$，其中 N 代表集合的大小。

我们定义 Merkle Mountain Range（默克尔山峦范围）为集合 $\llbracket \mathbb{H}? \rrbracket$ 内的一系列峰值，每个峰值对应一棵 Merkle 树的根，树中包含 2^i 个项目，i 为序列索引。由于集合大小不一定总是 2 的幂减 1，因此某些峰值可能为空（$\emptyset$），而非具体的 Merkle 根。

由于直接使用哈希序列作为承诺显得不够便捷，MMR 在发布前通常会经过再次哈希处理。这样的哈希处理能防止后续再追加元素，因此该范围会被保存在需要生成未来证明的系统上。

我们将追加函数 $\mathcal{A}$ 定义为：

(E.8) $$\mathcal{A}: \begin{cases} (\llbracket \mathbb{H}? \rrbracket, \mathbb{H}, \mathbb{Y} \to \mathbb{H}) \to \llbracket \mathbb{H}? \rrbracket \\ (\mathbf{r}, l, H) \mapsto P(\mathbf{r}, l, 0, H) \end{cases}$$

$$\text{where} P: \begin{cases} (\llbracket \mathbb{H}? \rrbracket, \mathbb{H}, \mathbb{N}, \mathbb{Y} \to \mathbb{H}) \to \llbracket \mathbb{H}? \rrbracket \\ (\mathbf{r}, l, n, H) \mapsto \begin{cases} \mathbf{r} \doteqdot l & \text{if } n \geq |\mathbf{r}| \\ R(\mathbf{r}, n, l) & \text{if } n < |\mathbf{r}| \wedge \mathbf{r}_n = \emptyset \\ P(R(\mathbf{r}, n, \emptyset), H(\mathbf{r}_n \frown l), n+1, H) & \text{otherwise} \end{cases} \end{cases}$$

$$\text{and } R: \begin{cases} (\llbracket T \rrbracket, \mathbb{N}, T) \to \llbracket T \rrbracket \\ (\mathbf{s}, i, v) \mapsto \mathbf{s}' \text{where } \mathbf{s}' = \mathbf{s} \text{ except } \mathbf{s}'_i = v \end{cases}$$

我们将 MMR 编码函数定义为 $\mathcal{E}_M$:

(E.9)
$$\mathcal{E}_M: \begin{cases} \llbracket \mathbb{H}? \rrbracket \to \mathbb{Y} \\ \mathbf{b} \mapsto \mathcal{E}(\updownarrow [\text{¿}x | x < \mathbf{b}]) \end{cases}$$

我们将 MMR 超级峰值函数定义为 $\mathcal{M}_R$:

(E.10)
$$\mathcal{M}_R: \begin{cases} \llbracket \mathbb{H}? \rrbracket \to \mathbb{H} \\ \mathbf{b} \mapsto \begin{cases} \mathbb{H}^0 & \text{if} |\mathbf{h}| = 0 \\ \mathbf{h}_0 & \text{if} |\mathbf{h}| = 1 \\ \mathcal{H}_K(\$\text{peak} \frown \mathcal{M}_R(\mathbf{h}_{\dots|\mathbf{h}|-1}) \sim \mathbf{h}_{|\mathbf{h}|-1}) & \text{otherwise} \\ \text{where } \mathbf{h} = [h | h \lessdot \mathbf{b}, h \neq \emptyset] \end{cases} \end{cases}$$

附 录 F. 洗牌

Fisher-Yates 洗牌算法的形式是:

$$\forall T, l \in \mathbb{N}: \mathcal{F}: \begin{cases} (\llbracket T \rrbracket_l, \llbracket \mathbb{N} \rrbracket_{l:}) \to \llbracket T \rrbracket_l \\ (\mathbf{s}, \mathbf{r}) \mapsto \begin{cases} \left[\mathbf{s}_{\mathbf{r}_0 \bmod l}\right] \frown \mathcal{F}\left(\mathbf{s}'_{\ldots l-1}, \mathbf{r}_{1\ldots}\right) \text{ where } \mathbf{s}' = \mathbf{s} \text{ except } \mathbf{s}'_{\mathbf{r}_0 \bmod l} = \mathbf{s}_{l-1} & \text{if } \mathbf{s} \neq [] \\ [] & \text{otherwise} \end{cases} \end{cases} \tag{F.1}$$

为了满足根据哈希形式的随机种子来洗牌序列的需求，我们提供了洗牌函数 $\mathcal{F}$ 的另一种版本。这个版本接受一个 32 字节的哈希值作为输入，而不是数字序列。我们定义了一个函数 $\mathcal{Q}$，用于“从哈希生成数字序列”，具体细节如下:

$$\forall l \in \mathbb{N}: \mathcal{Q}_i: \begin{cases} \mathbb{H} \to \llbracket \mathbb{N}_{2^{32}} \rrbracket_l \\ h \mapsto [\mathcal{E}_4^{-1}(\mathcal{H}(h \frown \mathcal{E}_4(\lfloor i/8 \rfloor))_{4i \bmod 32 \cdots + 4}) | i \in \mathbb{N}_l] \end{cases} \tag{F.2}$$

$$\forall T, l \in \mathbb{N}: \mathcal{F}: \begin{cases} (\llbracket T \rrbracket_l, \mathbb{H}) \to \llbracket T \rrbracket_l \\ (\mathbf{s}, h) \mapsto \mathcal{F}(\mathbf{s}, \mathcal{Q}_l(h)) \end{cases} \tag{F.3}$$

附 录 G. Bandersnatch 可验证随机函数

Bandersnatch 曲线是 2021 年由 Masson、Sanso 和 Zhang 三人所定义的。

单上下文的 Bandersnatch 类 Schnorr 签名 $\mathbb{F}_k^m\langle c \rangle$，其定义基于 Hosseini 和 Galassi 在 2024 年作为 IETF VRF（验证随机函数）所指定的模板，并由 Goldberg 等人在 2023年 进行了进一步的详细阐述。

$$\mathbb{F}_{k \in \mathbb{H}_B}^{m \in \mathbb{Y}} \langle c \in \mathbb{H} \rangle \subset \mathbb{Y}_{96} \equiv \{x | x \in \mathbb{Y}_{96}, \text{verify } (k, c, m, x) = \text{T}\} \tag{G.1}$$

$$\mathcal{Y}(s \in \mathbb{F}_k^m\langle c \rangle) \in \mathbb{H} \equiv \text{output } (x | x \in \mathbb{F}_k^m\langle c \rangle)_{\ldots 32} \tag{G.2}$$

单上下文的 Bandersnatch RingVRF 证明 $\bar{\mathbb{F}}_k^m\langle c \rangle$，是一种基于 Pedersen VRF（验证随机函数）的零知识简洁非交互式论证（zk-SNARK）的类似实现。这一概念由 Hosseini 和 Galassi 在 2024 年提出，并由 Jeffrey Burdges 等人在 2023 年进行了进一步的详细阐述。

$$\mathcal{O}([\mathbb{H}_B]) \in \mathbb{Y}_R \equiv \text{commit}(\llbracket \mathbb{H}_B \rrbracket) \tag{G.3}$$

$$\bar{\mathbb{F}}_{r \in \mathbb{Y}_R}^{m \epsilon Y} \langle c \in \mathbb{H} \rangle \subset \mathbb{Y}_{784} \equiv \{x | x \in \mathbb{Y}_{784}, \text{verify } (r, c, m, x) = \text{T}\} \tag{G.4}$$

$$\mathcal{Y}(p \in \bar{\mathbb{F}}_r^m\langle c \rangle) \in \mathbb{H} \equiv \text{output}(x | x \in \bar{\mathbb{F}}_r^m\langle c \rangle)_{\ldots 32} \tag{G.5}$$

在构建环时，若某个密钥 $\mathbb{H}_B$ 缺少对应的 Bandersnatch 点，则应采用 Hosseini 和 Galassi 于 2024 年提出的 Bandersnatch 填充点来进行替代。

附 录 H. 擦除编码

JAM 数据可用性和分发系统的核心是基于 GF(2^{16}) 域的系统性里 Reed-Solomon 擦除编码函数，其码率为 342:1023，与 Lin、Chung 和 Han 在 2014 年提出的算法采用了相同的变换方法。我们采用小端 $\mathbb{Y}_2$ 形式表示 16 位的 GF 点，其功能等价性由 $\mathcal{E}_2$ 来保证。基于此，我们定义了编码函数 C，它将 342 个 $[\![\mathbb{Y}_2]\!]$ 形式的字节对 映射到 1023 个 $[\![\mathbb{Y}_2]\!]$ 形式的字节对，即 $C: [\![\mathbb{Y}_2]\!]_{342} \to [\![\mathbb{Y}_2]\!]_{1023}$，以及恢复函数 $\mathcal{R}$，它能够从任意 342 个不同的字节对及其索引中恢复出原始的 342 个字节对序列，即 $\mathcal{R}: \wp(\langle(\mathbb{Y}_2, \mathbb{N}_{1023})\rangle)_{342} \to [\![\mathbb{Y}_2]\!]_{342}$。编码过程是将一个 684 字节（在编码函数中作为 342 个字节对处理）的数据块扩展为 1023 个字节对，而恢复过程则是通过收集这些字节对及其索引，将其还原为最初的 342 个字节对序列。

实际上，这种擦除编码方法能够高效地对大小为 684 字节倍数的数据进行编码和恢复。对于长度不是 684 字节倍数的数据，我们会采用零填充的方式进行处理。在 JAM 协议中，我们主要在两种场景下应用这种编码技术：一是针对可变大小（但通常规模较大）的数据块进行编码，用于 Auditda 和区块分发系统；二是针对较小且固定大小的数据段进行编码，用于 Importda 系统。

对于 Importda 系统，我们处理 4,104 字节的输入，实现六级数据并行。若一次编码或恢复多个数据段，可提高并行度。但恢复时，各数据段需由同一组索引构成（具体依算法而定）。

H.1. Blob 编码与恢复

我们有一个数据块 $\mathbf{d} \in \mathbb{Y}_{684k}$，其中 $k \in \mathbb{N}$ 。这个数据块可以分成个 k 片段，每个片段都是 684 个八位字节的序列。我们把这些片段（数据并行的）记作 $\mathbf{p} \in [\![\mathbb{Y}_{684}]\!] = \text{unzip}_{684}(\mathbf{d})$。然后，我们把每个片段重新组合成 342 对八位字节，并用 C 进行纠删编码，这样每个片段就变成了 1,023 对八位字节。

得到的矩阵按索引分组后，被串联成 1,023 个块，每块含k对八位字节。任意选取其中 342 个块，即可重构原始数据 $\mathbf{d}$。

形式上，我们首先定义四个实用函数：一是将大序列分割成若干等大的子序列，二是将这些子序列重新组合回一个大序列：

$$\forall n \in \mathbb{N}, k \in \mathbb{N}: \text{split}_n(\mathbf{d} \in \mathbb{Y}_{k \cdot n}) \in \left[\!\left[\mathbb{Y}_n\right]\!\right]_k \equiv \left[\mathbf{d}_{0 \cdots + n}, \mathbf{d}_{n \cdots + n}, \cdots, \mathbf{d}_{(k-1)n \cdots + n}\right] \tag{H.1}$$

$$\forall n \in \mathbb{N}, k \in \mathbb{N}: \text{join}_n(\mathbf{c} \in \left[\!\left[\mathbb{Y}_n\right]\!\right]_k) \in \mathbb{Y}_{k \cdot n} \equiv \mathbf{c}_0 \frown \mathbf{c}_1 \frown \ldots \tag{H.2}$$

$$\forall n \in \mathbb{N}, k \in \mathbb{N}: \text{unzip}_n(\mathbf{d} \in \mathbb{Y}_{k \cdot n}) \in \left[\!\left[\mathbb{Y}_n\right]\!\right]_k \equiv \left[\left[\mathbf{d}_{j \cdot k + i} | j \lessdot \mathbb{N}_n\right] | i \lessdot \mathbb{N}_k\right] \tag{H.3}$$

$$\forall n \in \mathbb{N}, k \in \mathbb{N}: \text{lace}_n(\mathbf{c} \in \left[\!\left[\mathbb{Y}_n\right]\!\right]_k) \in \mathbb{Y}_{k \cdot n} \equiv \mathbf{d} \text{ where } \forall i \in \mathbb{N}_k, j \in \mathbb{N}_n: \mathbf{d}_{j \cdot k + i} = (\mathbf{c}_i)_j \tag{H.4}$$

我们定义转置运算符为：

$$^{\mathsf{T}}[[\mathbf{x}_{0,0}, \mathbf{x}_{0,1}, \mathbf{x}_{0,2}, \ldots], [\mathbf{x}_{1,0}, \mathbf{x}_{1,1}, \ldots], \ldots] \equiv [[\mathbf{x}_{0,0}, \mathbf{x}_{1,0}, \mathbf{x}_{2,0}, \ldots], [\mathbf{x}_{0,1}, \mathbf{x}_{1,1}, \ldots], \ldots] \tag{H.5}$$

然后，我们可以定义一个纠删编码分块函数，它接收一个长度能被 684 个八位字节整除的数据块，并输出 1,023 个包含较小数据块的序列：

$$C_{k \in \mathbb{N}}: \begin{cases} \mathbb{Y}_{684k} \to \left[\!\left[\mathbb{Y}_{2k}\right]\!\right]_{1023} \\ \mathbf{d} \mapsto \left[\text{join}(\mathbf{c}) | \mathbf{c} \lessdot {}^{\mathsf{T}}[\mathcal{C}(\mathbf{p}) | \mathbf{p} \lessdot \text{unzip}_{684}(\mathbf{d})]\right] \end{cases} \tag{H.6}$$

原始数据可由 1023 项中的任意 342 项及其索引重建，已知这 342 项即可直接拼接重建。

$$\mathcal{R}_{k\in\mathbb{N}}\begin{cases}\{(\mathrm{Y}_{2k},\mathbb{N}_{1023})\}_{342}\to\mathrm{Y}_{684k}\\ \mathbf{c}\mapsto\begin{cases}\mathcal{E}([\mathbf{x}|(\mathbf{x},i)\lessdot\mathbf{c}]) & \text{if } [i|(\mathbf{x},i)\lessdot\mathbf{c}]=[0,1,\ldots 341]\\ \mathrm{lace}_k([\mathcal{R}([(\mathrm{split}_2(\mathbf{x})_p,i)|(\mathbf{x},i)\lessdot\mathbf{c}])|p\in\mathbb{N}_k & \text{always}\end{cases}\end{cases}\tag{H.7}$$

段编码和解码可用同一函数实现，只需设定常数 $k=6$。

H.2. 码字表示

为简化表述，我们将每八个比特（八位组）的组合称为一个码字，所有码字（含消息码字）均视为有限域 $\mathbb{F}_2 16$ 中的元素，该域是通过扩展 $\mathbb{F}_2$ 并应用不可约多项式而得到的：

$$x^{16}+x^5+x^3+x^2+1\tag{H.8}$$

因此：

$$\mathbb{F}_{16}\equiv\frac{\mathbb{F}_2[x]}{x^{16}}+x^5+x^3+x^2+1\tag{H.9}$$

我们将有限域 $\frac{\mathbb{F}_{16}}{\mathbb{F}_2}$ 的生成元（即上述多项式的根）命名为 α，从而有 $\mathbb{F}_{16}=\mathbb{F}_2(\alpha)$。

我们不采用标准基$\{1,\alpha,\alpha^2,\ldots,\alpha^{15}\}$，而是选用一种在编码解码时更高效的 $\mathbb{F}_{16}$ 表示法。为此，我们特将 $\mathbb{F}_{16}$ 的这种特殊表示命名为 $\widetilde{\mathbb{F}}_{16}$，并定义其为由 Cantor 基生成的向量空间。

v_0	1
v_1	$\alpha^{15}+\alpha^{13}+\alpha^{11}+\alpha^{10}+\alpha^7+\alpha^6+\alpha^3+\alpha$
v_2	$\alpha^{13}+\alpha^{12}+\alpha^{11}+\alpha^{10}+\alpha^3+\alpha^2+\alpha$
v_3	$\alpha^{12}+\alpha^{10}+\alpha^9+\alpha^5+\alpha^4+\alpha^3+\alpha^2+\alpha$
v_4	$\alpha^{15}+\alpha^{14}+\alpha^{10}+\alpha^8+\alpha^7+\alpha$
v_5	$\alpha^{15}+\alpha^{14}+\alpha^{13}+\alpha^{11}+\alpha^{10}+\alpha^8+\alpha^5+\alpha^3+\alpha^2+\alpha$
v_6	$\alpha^{15}+\alpha^{12}+\alpha^8+\alpha^6+\alpha^3+\alpha^2$
v_7	$\alpha^{14}+\alpha^4+\alpha$
v_8	$\alpha^{14}+\alpha^{13}+\alpha^{11}+\alpha^{10}+\alpha^7+\alpha^4+\alpha^3$
v_9	$\alpha^{12}+\alpha^7+\alpha^6+\alpha^4+\alpha^3$
v_{10}	$\alpha^{14}+\alpha^{13}+\alpha^{11}+\alpha^9+\alpha^6+\alpha^5+\alpha^4+\alpha$
v_{11}	$\alpha^{15}+\alpha^{13}+\alpha^{12}+\alpha^{11}+\alpha^8$
v_{12}	$\alpha^{15}+\alpha^{14}+\alpha^{13}+\alpha^{12}+\alpha^{11}+\alpha^{10}+\alpha^8+\alpha^7+\alpha^5+\alpha^4+\alpha^3$
v_{13}	$\alpha^{15}+\alpha^{14}+\alpha^{13}+\alpha^{12}+\alpha^{11}+\alpha^9+\alpha^8+\alpha^5+\alpha^4+\alpha^2$
v_{14}	$\alpha^{15}+\alpha^{14}+\alpha^{13}+\alpha^{12}+\alpha^{11}+\alpha^{10}+\alpha^9+\alpha^8+\alpha^5+\alpha^4+\alpha^3$

v_{15}	$\alpha^{15} + \alpha^{12} + \alpha^{11} + \alpha^{8} + \alpha^{4} + \alpha^{3} + \alpha^{2} + \alpha$

每个消息码字 m_i，由 16 位组成，即 $m_i = m_{i,15}...m_{i,0}$，可视为一个长度为 16 的二进制向量：

(H.10) $$m_i = (m_{i,0}...m_{i,15})$$

其中，$m_{i,0}$ 是消息码字 m_i 的最低有效位。因此，我们用域元素 $\tilde{m}_i$ 来表示该消息码字，其定义为 $\tilde{m}_i = \sum_{j=0}^{15} m_{i,j} v_j$（其中 j 从 0 到 15）。

同样地，我们给每位验证者分配一个 0 到 1022 之间的唯一编号，并用域元素来表示验证者 i 。

(H.11) $$\tilde{i} = \sum_{j=0}^{15} i_j v_j$$

其中，i 的二进制表示为 $i = i_{15}...i_0$。

H.3. 生成多项式

为了将一条由 342 个词组成的消息编码为 1023 个码字，我们将每个消息表示为前面章节中所述的域元素，并构造一个最高次项为 341 的多项式 $p(y)$，该多项式需满足以下等式：

(H.12) $$\begin{aligned} p(\tilde{0}) &= \widetilde{m_0} \\ p(\tilde{1}) &= \widetilde{m_1} \\ &... \\ p(\widetilde{341}) &= \widetilde{m_{341}} \end{aligned}$$

在确定满足这些性质的 p(y) 后，我们对 p 在以下点进行评估：

(H.13) $$\begin{aligned} \widetilde{r_{342}} &:= p(\widetilde{342}) \\ \widetilde{r_{343}} &:= p(\widetilde{343}) \\ &... \\ \widetilde{r_{1022}} &:= p(\widetilde{1022}) \end{aligned}$$

然后，我们依据验证者的索引，将消息词及额外的码字分配给相应的验证者。

附 录 I. 符号索引

I.1. 集合

I.1.1. 常规符号

$\mathbb{N}$：非负整数集。下标表示最大值加一。详见第 3.4 节。

$\mathbb{N}^+$： 正整数集（不包括零）。

$\mathbb{N}_B$： 余额值的集合。等价于$\mathbb{N}_{2^{64}}$。详见方程 4.21。

$\mathbb{N}_G$：无符号 Gas 值的集合。等价于$\mathbb{N}_{2^{64}}$。详见方程 4.23。

$\mathbb{N}_L$：Blob 长度值的集合。等价于$\mathbb{N}_{2^{64}}$。详见第 3.4 节。

$\mathbb{N}_R$：寄存器值的集合。等价于$\mathbb{N}_{2^{64}}$。详见方程 4.23。

$\mathbb{N}_S$：服务索引所取的集合。等价于 $\mathbb{N}_{2^{32}}$。详见第 9.1 节。

$\mathbb{N}_T$：时隙（Timeslot）值的集合。等价于$\mathbb{N}_{2^{32}}$。详见方程 4.28。

$\mathbb{Q}$：有理数集（未使用）。

$\mathbb{R}$：实数集（未使用）。

$\mathbb{Z}$： 整数集。下标表示范围。详见第 3.4 节。

$\mathbb{Z}_G$：有符号 Gas 值的集合。等价于 $\mathbb{Z}_{-2^{63}\ldots 2^{63}}$。详见方程 4.23。

I.1.2. 自定义符号

$\mathbb{A}$：服务账户的集合。详见方程 9.3。

$\mathbb{B}$：布尔序列/比特串的集合。下标表示长度。详见第 3.7 节。

$\mathbb{C}$：Seal-key 票据的集合。详见方程 6.6。（注意：$\mathbb{C}$ 并不表示复数集合。）

$\mathbb{D}$：字典的集合。详见第 3.5 节。

$\mathbb{D}\langle K \to V\rangle$： 从 K 到 V 的部分双射字典集合。详见第 3.5 节。

$\mathbb{E}$：有效的 Ed25519 签名集合，为 $\mathbb{Y}_{64}$ 的子集。详见第 3.8 节。

$\mathbb{E}_K\langle M\rangle$：钥匙 K 和消息 M 的有效 Ed25519 签名集合，为 $\mathbb{E}$ 的子集。详见第 3.8 节。

$\mathbb{F}$：Bandersnatch 签名集合，为 $\mathbb{Y}_{64}$ 的子集。详见第 3.8 节。（注意：$\mathbb{F}$ 不表示有限域。）

$\mathbb{F}_K^M\langle C\rangle$： 由公钥 K、上下文 C 和消息 M 组成的 Bandersnatch 签名集合，为 $\mathbb{F}$ 的子集。详见第 3.8 节。

$\bar{\mathbb{F}}$：Bandersnatch RingVRF 证明的集合。详见第 3.8 节。

$\bar{\mathbb{F}}_K^M\langle C\rangle$：由根 R、上下文 C 和消息 M 组成的 Bandersnatch RingVRF 证明集合，为 $\bar{\mathbb{F}}$ 的子集。详见第 3.8 节。

$\mathbb{G}$：数据段的集合，相当于长度 $\mathbb{W}_G$ 的八位字节序列。详见方程 14.1。

$\mathbb{H}$：32 字节的加密值集合，为 $\mathbb{Y}_{32}$ 的子集。$\mathbb{H}$ 无下标时通常表示哈希函数的结果。详见第 3.8 节。（注意：$\mathbb{H}$ 不表示四元数。）

$\mathbb{H}_B$：Bandersnatch 公钥的集合，为 $\mathbb{Y}_{32}$ 的子集。详见第 3.8 节和附录 G。

$\mathbb{H}_E$：Ed25519 公钥的集合，为 $\mathbb{Y}_{32}$ 的子集。详见第 3.8.2 节。

$\mathbb{I}$：工作项的集合。详见方程 14.3。

$\mathbb{J}$：工作执行错误的集合。

$\mathbb{K}$：验证者密钥集的集合。详见方程 6.7。

$\mathbb{L}$：工作结果的集合。

$\mathbb{M}$：PVM 的RAM 状态的集合，为 $\mathbb{Y}_{2^{32}}$ 的超集。详见附录 A。

$\mathbb{O}$：累积运算元元素，对应于单个工作结果。

$\mathbb{P}$：工作包的集合。详见方程 14.2。

$\mathbb{S}$：可用性规范的集合。

$\mathbb{T}$：延迟转账的集合。

$\mathbb{U}$：在累积过程中使用的部分状态集合。详见方程 12.13。

$\mathbb{V}_\mu$：PVM RAM μ 的可读索引集合。详见附录 A。

$\mathbb{V}^*_\mu$：PVM RAM μ 的可写索引集合。详见附录 A。

$\mathbb{W}$：工作报告的集合。

$\mathbb{X}$：细化上下文的集合。

$\mathbb{Y}$：八位字节字符串（"blobs"）的集合。下标表示长度。详见第 3.7 节。

$\mathbb{Y}_{BLS}$：BLS 公钥的集合，为 $\mathbb{Y}_{144}$ 的子集。详见第 3.8.2 节。

$\mathbb{Y}_R$：Bandersnatch 环根的集合，为 $\mathbb{Y}_{144}$ 的子集。详见第 3.8 节和附录 G。

I.2. 函数

Δ ：累积函数；特定下标表示辅助函数：

Δ_1：单步累积函数。

Δ_*：并行累积函数。

Δ_+：完整的顺序累积函数。

Λ：历史查找函数。详见方程 9.7。

Ξ：工作结果计算函数。详见方程 14.11。

Υ：通用状态转换函数。详见方程 4.1、4.5。

Φ：密钥无效化（key-nullifier）函数。详见方程 6.14。

Ψ：整个程序的 PVM 机器状态转换函数。详见方程 A。

Ψ_1：单步 PVM 机器状态转换函数。详见附录 A。

Ψ_A：累积（Accumulate）PVM 调用函数。详见附录 B。

Ψ_H：主机函数（host-function）调用（PVM），带有主机函数封送（marshalling）。详见附录 A。

Ψ_I：授权检查（Is-Authorized）PVM 调用函数。详见附录 B。

Ψ_M：封送整个程序的 PVM 机器状态转换函数。详见附录 A。

Ψ_R：细化（Refine）PVM 调用函数。详见附录 B。

Ψ_T：传输（On-Transfer）PVM 调用函数。详见附录 B。

Ω：虚拟机主机调用函数。详见附录 B。

Ω_A：分配核心（Assign-core）主机调用。

Ω_B：授权服务（Empower-service）主机调用。

Ω_C：设定检查点（Checkpoint）主机调用。
Ω_D：指定验证者（Designate-validators）主机调用。
Ω_E：导出数据段（Export segment）主机调用。
Ω_F：忘记预影像（Forget-preimage）主机调用。
Ω_G：剩余 Gas 查询（Gas-remaining）主机调用。
Ω_H：历史查找预影像（Historical-lookup-preimage）主机调用。
Ω_I：服务信息查询（Information-on-service）主机调用。
Ω_J：驱逐服务（Eject-service）主机调用。
Ω_K：启动 PVM（Kickoff-pvm）主机调用。
Ω_L：预影像查找（Lookup-preimage）主机调用。
Ω_M：创建 PVM（Make-pvm）主机调用。
Ω_N：新建服务（New-service）主机调用。
Ω_O：触发 PVM（Poke-pvm）主机调用。
Ω_P：预览 PVM（Peek-pvm）主机调用。
Ω_Q：预影像查询（Query-preimage）主机调用。
Ω_R：读取存储（Read-storage）主机调用。
Ω_S：请求预影像（Solicit-preimage）主机调用。
Ω_T：传输（Transfer）主机调用。
Ω_U：升级服务（Upgrade-service）主机调用。
Ω_V：清空 PVM 内存（Void inner-pvm memory）主机调用。
Ω_W：写入存储（Write-storage）主机调用。
Ω_X：清除 PVM（Expunge-pvm）主机调用。
Ω_Y：获取数据（Fetch data）主机调用。
Ω_Z：归零 PVM 内存（Zero inner-pvm memory）主机调用。
Ω_ϱ：产出累积 Trie 结果（Yield accumulation trie result）主机调用。

I.3. 工具函数、外部依赖与标准函数

$\mathcal{A}(\ldots)$：默克尔山脉（Merkle Mountain Range, MMR）追加函数。详见方程 E.8。
$\mathcal{B}_n(\ldots)$：n 个八位字节转换为比特串的函数。上标 -1 表示反向操作。详见方程 A.12。
$\mathcal{C}(\ldots)$：纠删编码函数组。
$\mathcal{C}_n(\ldots)$：适用于 n 个数据块的纠删编码函数。详见方程 H.6。
$\mathcal{E}(\ldots)$：八位字节序列编码函数。上标 -1 表示反向操作。详见附录 C。
$\mathcal{F}(\ldots)$：Fisher-Yates 洗牌函数。详见方程 F.1。
$\mathcal{H}(\ldots)$：Blake2b 256-bit 哈希函数。详见第 3.8 节。
$\mathcal{H}_K(\ldots)$：Keccak 256-bit 哈希函数。详见第 3.8 节。
$\mathcal{J}_x$：指向特定 2^x 大小页面的默克尔树路径。详见方程 E.5。
$\mathcal{K}(\ldots)$：字典的键集合。详见第 3.5 节。
$\mathcal{L}_x$：2^x 大小页面的函数，适用于固定深度的默克尔树。详见方程 E.6。
$\mathcal{M}(\ldots)$：固定深度的二进制默克尔化（Merklization）函数。详见附录 E。
$\mathcal{M}_B(\ldots)$：平衡良好的二进制默克尔化（Merklization）函数。详见附录 E。

$\mathcal{M}_\sigma(\ldots)$：状态默克尔化（State Merklization）函数。详见附录 D。

$\mathcal{N}(\ldots)$：纠删编码块（erasure-coding chunks）函数。详见附录 H。

$\mathcal{O}(\ldots)$：Bandersnatch 环根函数。详见第 3.8 节和附录 G。

$\mathcal{P}_n(\ldots)$：八位字节数组的零填充（zero-padding）函数。详见方程 14.17。

$\mathcal{Q}(\ldots)$：从哈希值生成数值序列的函数。详见方程 F.3。

$\mathcal{R}(\ldots)$：纠删编码数据恢复函数组。

$\mathcal{S}_k(\ldots)$：通用签名函数。详见第 3.8 节。

$\mathcal{T}(\ldots)$：以 Jam 共同纪元（Jam Common Era）起始时间为基准的当前秒数。详见第 4.4 节。

$\mathcal{U}(\ldots)$：替换空值（substitute-if-nothing）函数。详见方程 3.2。

$\mathcal{V}(\ldots)$：字典或序列的值集合。详见第 3.5 节。

$\mathcal{X}_n(\ldots)$：在 $\mathbb{N}_{2^{8n}}$ 范围内的有符号扩展（signed-extension）函数。详见方程 A.16。

$\mathcal{Y}(\ldots)$：Bandersnatch VRF 签名/证明的别名、输出或熵函数。详见第 3.8 节和附录 G。

$\mathcal{Z}_n(\ldots)$：在 $\mathbb{N}_{2^{8n}}$ 范围内的签名转换函数。上标 $^{-1}$ 表示反向操作。详见方程 A.10。

$\wp(\ldots)$：幂集（power set）函数。

I.4. 值（Values）

I.4.1. 区块上下文术语（Block-context Terms）

这些术语均与单个区块相关。它们可能带有上标，以更改上下文并引用其他区块。

A：该区块的祖先集合。详见方程 5.3。

B：该区块。详见方程 4.2。

C：服务累积承诺（Service Accumulation Commitment），用于构造 Beefy 根（Beefy Root）。详见方程 ??。

E：该区块的交易（Extrinsic）。详见方程 4.3。

$\mathbf{F}_v$：验证者 v 的 Beefy 签名承诺。详见方程 18.1。

G：核心（Cores）到担保密钥（Guarantor Keys）的映射。详见第 11.3 节。

$\mathbf{G}^*$：上一次轮换的核心到担保密钥的映射。详见第 11.3 节。

H：该区块的区块头（Block Header）。详见方程 5.1。

I：在该区块中被累积的工作报告（Work-Reports）序列。详见方程 ??。

Q：验证者确定必须审计的就绪工作报告选择集。详见方程 17.1。

R：生成工作报告的 Ed25519 担保密钥集合。详见方程 11.26。

S：该区块中已累积（“进展”）的服务索引集合。详见方程 ??。

T：票据条件（Ticketed Condition），如果区块是用票据签名（Ticket Signature）封装的，而不是后备方案（Fallback），则该条件为真。详见方程 6.15 和 6.16。

U：审计条件（Audit Condition），一旦区块被审计，该条件等于 τ 详见第 17 节。

V：当前区块中的判决（Verdicts）集合。详见方程 10.11。

W：已可用于累积的工作报告序列。详见方程 11.16。

X：由该区块累积所隐含的传输序列。详见方程 ??。

如果没有上标，默认情况下，区块指的是当前导入的区块。如果没有正在导入的区块，则指最佳链（Best Chain）的头区块（详见第 19 节）。明确的区块上下文上标包括：

$\mathbf{B}^{\natural}$ ：最新的已最终确定区块。详见方程 19。

$\mathbf{B}^{\flat}$: 最佳链的头部区块。详见方程 19。

I.4.2. 状态组件（State Components）

在此，撇号（'）表示后态（posterior state）。单个组件可通过字母下标进行标识。

α: 核心授权池（Core authorizations pool）。详见方程 8.1。

β: 最新区块的信息。

γ: 与 Safrole 相关的状态。详见方程 6.3。

- γ_a: 封印彩票票据累加器（Sealing lottery ticket accumulator）。
- γ: 下一 epoch 验证者的密钥，等价于 γ_z 组成的密钥。
- γ: 当前 epoch 的封印密钥序列（Sealing-key sequence）。
- γ_z: 当前 epoch 提交的票据的 Bandersnatch 根（Bandersnatch root）。

δ: 之前的服务账户状态（Prior state of the service accounts）。

- δ†: 后累积、传输前的中间状态（Post-accumulation, pre-transfer intermediate state）。
- δ‡: 后传输、预影像整合前的中间状态（Post-transfer, pre-preimage integration intermediate state）。

η: 熵累加器（Entropy accumulator）和 epoch 随机性（Epochal randomness）。

ι: 将要被抽取的验证者密钥及元数据。

κ: 当前活跃的验证者密钥及元数据。

λ: 在前一个 epoch 里活跃的验证者密钥及元数据。

ρ: 每个核心（Core）的待处理报告（Pending reports），这些报告在累积前被提供。

- $\rho^{\dagger}$: 后判决、保证交易前的中间状态（Post-judgment, pre-guarantees-extrinsic intermediate state）。
- $\rho^{\ddagger}$: 后保证交易、担保交易前的中间状态（Post-guarantees-extrinsic, pre-assurances-extrinsic intermediate state）。

σ: 系统的整体状态（Overall state of the system）。详见方程 4.1 和 4.4。

τ: 最新区块的时隙（Timeslot）。

φ: 授权队列（Authorization queue）。

ψ: 过往的工作报告判决（Past judgments on work-reports）和验证者（Validators）。

- ψ_b: 被判定为不正确的工作报告。
- ψ_g: 被判定为正确的工作报告。
- ψ_w: 其有效性被判定为未知的工作报告。
- ψ_o: 被判定为做出错误判断的验证者。

χ: 特权服务索引（Privileged service indices）。

- χ_m ：受祝福服务（Blessed Service）的索引。
- χ_v ：指定服务（Designate Service）的索引。
- χ_a ：分配服务（Assign Service）的索引。
- χ_g ：持续累积（Always-Accumulate）服务索引及其基本 Gas 配额。

π: 验证者的活动统计数据（Activity statistics for the validators）。

ϑ: 累积队列（Accumulation queue）。

ξ: 累积历史（Accumulation history）。

I.4.3. 虚拟机组件（Virtual Machine Components）

ε: 由所有机器状态转换引起的退出原因（The exit-reason resulting from all machine state transitions）。

ν: 指令的立即数值（The immediate values of an instruction）。

μ: 内存序列（The memory sequence）；属于集合 $\mathbb{M}$ 的成员。

ϱ: Gas 计数器（The gas counter）。

ω: 寄存器（The registers）。

ξ: 指令序列（The instruction sequence）。

ϖ: 程序的基本代码块序列（The sequence of basic blocks of the program）。

$\imath$: 指令计数器（The instruction counter）。

I.4.4. 常数（Constants）

$A = 8$: 审计批次之间的时间周期（秒）。

$B_I = 10$: 每项选举服务状态所需的额外最低余额。

$B_L = 1$: 每个选举服务状态所需的额外最低余额单位。

$B_S = 100$: 所有服务要求的基本最低余额。

$C = 341$: 总核心数。

$D = 192{,}00$: 未引用预影像可被删除的时隙周期。

$E = 600$: Epoch 的时隙长度。

$F = 2$: 审计偏差因子（Audit Bias Factor），即预期在上一个批次中缺席的附加验证者将审计工作报告的数量。

$G_A = 10{,}000{,}000$: 分配给调用工作包“累积逻辑（Accumulation Logic）”的 Gas 量。

$G_I = 50{,}000{,}000$: 分配给调用工作包“Is-Authorized 逻辑”的 Gas 量。

$G_R = 5{,}000{,}000{,}000$: 分配给调用工作包“Refine 逻辑”的 Gas 量。

$G_T = 3{,}500{,}000{,}000$: 所有累积过程的 Gas 总分配量，应不少于$G_A \cdot C \sum_{g \in v(x_g)_{8n}} (g)$。

$H = 8$: 最近历史的区块大小。

$I = 16$: 每个工作包中的最大工作项数量。

$J = 8$: 工作报告中依赖项的最大数量。

$K = 16$: 单个 Extrinsic（链上交易）中可提交的最大票据数。

$L = 14{,}000$: 查找锚点的最大时隙周期。

$N = 2$: 每个验证者的票据项数量。

$P = 8$: 授权池中的最大项目数。

$Q = 80$: 授权队列中的最大项目数。

$R = 10$: 验证者核心分配轮换的时隙周期。

$S = 1024$: 累积队列中的最大条目数。

$T = 128$: 工作包中的最大 Extrinsic 数量。

$U = 5$: 报告后但不可用的工作可能被替换的时隙周期。

$V = 1023$: 验证者总数。

$W_B = 12 \cdot 2^{20}$: 编码的工作包及其 Extrinsic 数据和导入内容的最大尺寸（单位：字节）。

$W_C = 4{,}000{,}000$：服务代码的最大大小（单位：字节）。

$W_E = 684$：基本纠删码块大小（单位：字节）。详见方程 H.6。

$W_G = W_P W_E = 4104$：片段的大小（单位：字节）。

$W_M = 3{,}072$：工作包中最大导入和导出数（单位：字节）。

$W_P = 6$：每个片段中的纠删码块数量。

$W_R = 48 \cdot 2^{10}$：工作报告中所有输出数据（Blobs）的最大总尺寸（单位：字节）。

$W_T = 128$：传输项的大小（单位：字节）。

X：上下文字符串，（Context Strings）详见下文

$Y = 500$：票据提交结束的 Epoch 时隙数量。

$Z_A = 2$：PVM 动态地址对齐因子（Dynamic Address Alignment Factor）。详见方程 A.18。

$Z_I = 2^{24}$：标准 PVM 程序初始化数据大小。详见方程 A.7。

$Z_P = 2^{12}$：标准 PVM 内存页面大小。详见方程 4.24。

$Z_Z = 2^{16}$：标准 PVM 程序初始化区大小。详见方程 A.7。

I.4.5. 签名上下文（Signing Contexts）

X_A = $jam_available: Ed25519 可用性保证（Availability assurances）。

X_B = $jam_beefy: BLS 累积结果根 MMR 承诺（Accumulate-result-root-MMR commitment）。

X_E = $jam_entropy: 链上熵生成（On-chain entropy generation）。

X_F = $jam_fallback_seal: Bandersnatch 备用区块封印（Fallback block seal）。

X_G = $jam_guarantee: Ed25519 担保声明（Guarantee statements）。

X_I = $jam_announce: Ed25519 审计公告声明（Audit announcement statements）。

X_T = $jam_ticket_seal: Bandersnatch RingVRF 票据生成与常规区块封印（Ticket generation and regular block seal）。

X_U = $jam_audit: Bandersnatch 审计选择熵（Audit selection entropy）。

X_T = $jam_valid: Ed25519 有效工作报告的判决（Judgments for valid work-reports）。

X = $jam_invalid: Ed25519 无效工作报告的判决（Judgments for invalid work-reports）。

参考文献

Bertoni, Guido 等. (2013). "*Keccak*"。载于《*国际密码学理论与应用年会*》。Springer, pp. 313–314.

Bögli, Roman (2024). *"使用 ZKit 评估 RISC Zero：一个用于 ZKP 框架的可扩展测试和基准套件"*。OST 东瑞士应用科技大学博士论文。

Boneh, Dan, Ben Lynn 和 Hovav Shacham (2004). *"基于 Weil 配对的短签名"*。载于《*J. Cryptology*》17, pp. 297–319. doi: 10.1007/s00145-004-0314-9.

Burdges, Jeff, Alfonso Cevallos 等 (2024). *"拜占庭假设下的区块链高效执行审计"*。密码学电子印刷档案 (Cryptology ePrint Archive), Paper 2024/961。
https://eprint.iacr.org/2024/961.

Burdges, Jeff, Oana Ciobotaru 等 (2022). *"具有 Chaum-Pedersen 证明的高效可聚合 BLS 签名"*。密码学电子印刷档案 (Cryptology ePrint Archive), Paper 2022/1611。
https://eprint.iacr.org/2022/1611.

Burdges, Jeffrey 等 (2023). *"环可验证随机函数与零知识延续"*。密码学电子印刷档案 (Cryptology ePrint Archive), Paper 2023/002。
https://eprint.iacr.org/2023/002.

Buterin, Vitalik (2013). *"以太坊：下一代智能合约和去中心化应用平台"*。
https://github.com/ethereum/wiki/wiki/White-Paper.

Buterin, Vitalik 和 Virgil Griffith (2019). *"Casper：友好的终结性机制"*。arXiv: 1710.09437 [cs.CR].

Cosmos Project (2023). *"互链安全开启 Cosmos 新纪元"*。检索时间：2024年3月18日。
https://blog.cosmos.network/interchain-security-begins-a-new-era-for-cosmos-a2dc3c0be63.

Dune 和 hildobby (2024). *"以太坊质押"*。检索时间：2024年3月18日。
https://dune.com/hildobby/eth2-staking

Ethereum Foundation (2024a). *"全球规模的数字未来"*。检索时间：2024年4月4日。
https://ethereum.org/en/roadmap/vision/

Ethereum Foundation (2024b). *"Danksharding"*。检索时间：2024年3月18日。
https://ethereum.org/en/roadmap/danksharding/.

Fisher, Ronald Aylmer 和 Frank Yates (1938). 《*生物、农业与医学研究的统计表*》。Oliver and Boyd 出版社。

Gabizon, Ariel, Zachary J. Williamson 和 Oana Ciobotaru (2019). *"PLONK：基于拉格朗日基的置换，用于通用非交互式知识论证"*。密码学电子印刷档案 (Cryptology ePrint Archive), Paper 2019/953。
https://eprint.iacr.org/2019/953.

Goldberg, Sharon 等. (2023年8月). *"可验证随机函数 (VRFs)"*。RFC 9381. doi: 10.17487/RFC9381.
https://www.rfc-editor.org/info/rfc9381.

Hertig, Alyssa (2016). *"以太坊的区块链仍然在遭受攻击..."*。检索时间：2024年3月18日。
https://www.coindesk.com/markets/2016/10/06/so-ethereums-blockchain-is-still-under-attack/.

Hopwood, Daira 等 (2020). *"BLS12-381"*。
https://z.cash/technology/jubjub/.

Hosseini, Seyed 和 Davide Galassi (2024). *"Bandersnatch VRF-AD 规范"*。检索时间：2024年4月4日。
https://github.com/davxy/bandersnatch-vrfs-spec/blob/main/specification.pdf.

Jha, Prashant (2024). *"Solana 停机事件引发了关于客户端多样性和 Beta 状态的质疑"*。检索时间：2024年3月18日。
https://cointelegraph.com/news/solana-outage-client-diversity-beta.

Josefsson, Simon 和 Ilari Liusvaara (2017年1月). *"Edwards 曲线数字签名算法 (EdDSA)"*。RFC 8032. doi: 10.17487/RFC8032. https://www.rfc-editor.org/info/rfc8032.

Kokoris-Kogias, Eleftherios 等 (2017). *"全能账本（OmniLedger）：通过分片实现安全、可扩展的去中心化分类账"*。密码学电子印刷档案 (Cryptology ePrint Archive), Paper 2017/406。 https://eprint.iacr.org/2017/406.

Kwon, Jae 和 Ethan Buchman (2019). *《Cosmos 白皮书》*。*《分布式分类账网络》* 27, pp. 1–32。

Lin, Sian-Jheng, Wei-Ho Chung 和 Yunghsiang S. Han (2014). *"新型多项式基及其在 Reed-Solomon 纠删码中的应用"*。IEEE FOCS 2014，pp. 316–325. doi: 10.1109/FOCS.2014.41.

Masson, Simon, Antonio Sanso 和 Zhenfei Zhang (2021). *"Bandersnatch：基于 BLS12-381 标量域的快速椭圆曲线"*。密码学电子印刷档案 (Cryptology ePrint Archive), Paper 2021/1152。 https://eprint.iacr.org/2021/1152.

Ng, Felix (2024). *"在 2024 年测量区块链的每秒交易数（TPS）是愚蠢的吗？"*。检索时间：2024年3月18日。 https://cointelegraph.com/magazine/blockchain-transactions-per-second-tps-stupid-big-questions/.

Polkavm Project (2024). *"PolkaVM/RISC0 基准测试结果"*。检索时间：2024年4月3日。 https://github.com/koute/risc0-benchmark/blob/master/README.md.

Saarinen, Markku-Juhani O. 和 Jean-Philippe Aumasson (2015年11月). *"BLAKE2 加密哈希与消息验证码 (MAC)"*。RFC 7693. doi: 10.17487/RFC7693. https://www.rfc-editor.org/info/rfc7693.

Sadana, Apoorv (2024). *"将 波卡 技术引入以太坊"*。检索时间：2024年3月18日。 https://ethresear.ch/t/bringing-polkadot-tech-to-ethereum/17104 .

Sharma, Shivam (2023). *"以太坊的 Rollup 是中心化的"*。 https://public.bnbstatic.com/static/files/research/ethereums-rollups-are-centralized-a-look-into-decentralized-sequencers.pdf

Solana Foundation (2023). *"Solana 数据上线 Google Cloud BigQuery"*。检索时间：2024年3月18日。 https://solana.com/news/solana-data-live-on-google-cloud-bigquery.

Solana Labs (2024). *"Solana 验证者要求"*。检索时间：2024年3月18日。 https://docs.solanalabs.com/operations/requirements

Stewart, Alistair 和 Eleftherios Kokoris-Kogia (2020). *"Grandpa：拜占庭最终确定性机制"*。arXiv 预印本。 arXiv:2007.01560.

Tanana, Dmitry (2019). "用于分布式计算安全的 Avalanche 区块链协议"。2019 年 IEEE 黑海国际通信与网络会议 (BlackSeaCom), IEEE, pp. 1–3.

Thaler, Justin (2023). *"关于 Lasso、Jolt 及 SNARK 设计最新进展的技术 FAQ"*。检索时间：2024年4月3日。 https://a16zcrypto.com/posts/article/a-technical-faq-on-lasso-jolt-and-recent-advancements-in-snark-design/ .

Wikipedia (2024). *"Fisher-Yates 洗牌：现代算法"*。 https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle#The_modern_algorithm.

Wood, Gavin (2014). *《以太坊：安全的去中心化通用交易分类账》*。以太坊黄皮书 151, pp. 1–32。

Yakovenko, Anatoly (2018). *"Solana：高性能区块链的新架构 v0.8.13"*。