**反事实：广义状态通道**

Jeff Coleman, Liam Horne, and Li Xuanji

*L4*[[1]](#footnote-1)

(Dated: June 12, 2018)

摘要

状态通道是降低区块链用户交易费的重要技术。在其适用范围内，它们允许用户彼此进行交易，而无需支付区块链交易费用和即时终局性，并且是安全实现后一种属性的唯一技术。我们描述了通用状态通道，一种允许用户在现有通道中安装新功能而不接触区块链的结构，使用通道内合同的反事实实例化。我们提出了一种基于以太坊之上的面向对象方法，它将功能和状态封装在反事实的实例化契约中，与整体方法相比，提供了许多隐私、效率和安全方面的好处，并描述了一种新的面向对象元通道方法来构建状态通道网络。我们分析了通道的独特安全假设，并描述了通道用户可以从中受益的第三方服务。

介绍

状态通道通过将状态修改操作从一个区块链中去掉并直接在定义的参与者之间执行来提高区块链的性能。支付通道[23]是描述的第一种状态通道，使用链外交互修改锁定比特币的所有权，从而允许用户彼此进行“链外支付”。术语“状态通道”概括了这种支付之外的方法，包括所有类型的区块链状态修改，这些修改在与支付通道类似的安全范式中运行。

状态通道：基本概述

状态通道允许各方安全地修改区块链状态的锁定部分，称为状态抵押。这些抵押通常存放在多签名钱包中，状态通道的参与者是多签名的签署者。

参与者通过交换链外交互来更新状态通道。这些消息描述了对状态抵押的更新，例如，改变支付通道中双方余额的状态，或棋手在国际象棋状态通道中的下一步行动。参与者可以继续交换这些链外更新，而不产生任何链内费用，直到他们选择关闭通道，此时最新的状态更新作为一个单一交易发送到链内多签，并且抵押状态根据最终状态提取给当事方。

参与者存储这些链外状态更新的副本。因为每条消息都是加密签名的，多签地址包含验证这些签名和解释这些消息的代码，所以参与者可以随时在链上实现最新的状态更新。当事方被对方阻止提交旧消息：如果Alice提交旧状态，Bob将有机会通过广播一个较新的状态来“反驳”它。这种设计允许参与者将状态通道内的更新视为“最终的”，尽管完全是在链外进行的。

与其他技术的比较

状态通道是将状态修改移出区块链的几种技术之一，将这些操作重新定位到layer2环境中，在那里可以以较低的成本执行。在这些技术中，一个重要的区别在于，重新部署的业务是否引入了额外的共识假设，如果这些假设失败，将永久损害长期担保（如资产所有权的持续性）。侧链[10]是属于这类技术的一个例子；如果状态侧链到另一个区块链，而区块链的共识机制失败，则该状态的所有者或受益人可能会失去委托给该区块链的所有东西，即使原始区块链仍然是安全的。

相比之下，状态通道（以及Plasma[22]）等技术允许用户将其状态恢复到原始区块链（假设其保持安全），即使链外状态修改机制失败，也允许用户在其自身保持安全的情况下将其状态恢复到原始区块链。

为了制定这些“安全网”条款，Plasma和状态通道都依赖于这样一个假设：新交易可以发布到（并最终确定）基础区块链。这种交易不仅需要存入或退出该链上的状态抵押，而且还需要区块链作为可信任的第三方，包含解决参与者之间的某些类型的争议的机制。这一假设还特别包括审查阻力，因为对参与者提交给链的内容进行选择性审查可以防止参与者安全退出通道或等离子链。因此，状态通道的安全属性直接取决于其状态抵押被锁定的区块链的安全性和活跃性属性，以及通道参与者及其代表的特定能力。

在“安全网保留”方法的范畴内，状态通道与Plasma完全不同，因为它们是完全同意进行的。一个状态通道内的任何状态变化都需要被指定为“有兴趣”的所有当事方明确的加密同意。在将状态存入状态抵押之前，假设一方的“利息”由基础区块链本身的规则定义。当他们进入状态通道时，缔约方选择他们希望与之互动的其他缔约方，并将这些缔约方添加到必须获得其同意才能在交存国开展业务的集合中。由于持续需要获得和证明直接同意，状态通道在很大程度上依赖于状态通道参与者（或其代表）的记录和响应能力。

总之，所有这些因素使状态通道成为提高基于区块链的应用程序性能的最严格限制技术之一。然而，尽管有这些限制，与常规的连锁经营相比，状态通道仍具有许多优势。

状态通道的好处

首先，通过减少区块链本身的使用（区块链必须由整个网络处理），与基于通道的应用程序相关的总开销将大幅减少。这尤其适用于执行大量操作的应用程序，这些操作只直接影响少数参与方。单个通道通常具有比等离子更小的参与者集，这意味着通道内计算操作的总复制更少。因此，状态通道内的状态修改操作可以非常便宜地执行，比几乎任何其他区块链性能增强技术下的操作都要便宜。如小额支付和流式支付等技术，如果以常规的链上交易的形式实施，成本将高得令人望而却步，当通道化时，这些技术将变得可行。

第二，由于受状态通道控制的状态可以很快完成，因此状态通道可以在认为操作不受意外逆转影响之前，从区块链中消除等待“确认”的需要，从而显著改善用户体验。显然，完全同意的极端要求是这种快速终结性的根源，它还允许合作的和可利用的参与者立即从状态抵押中提取状态抵押，并控制其通道。相比之下，当将状态撤回到父链时，等离子链要求延迟。

第三，当状态通道连接到状态通道网络（如Lightning[23]或Raiden[3]网络）时，不直接共享状态抵押的缔约方可以通过链接连续的通道化协议进入通道化协议。这允许参与者在大型网络之间进行非常广泛和快速的交互，而无需任何链上事务，尽管对交互类型有一定的限制，并且在支持大型网络所需的锁定状态数量方面存在一定程度的额外开销。

如果能够开发出能够最大化状态通道优势的体系结构，同时最大限度地降低其成本和不便，那么基于区块链的应用程序的功能就有可能得到显著的改进。然而，到目前为止，状态通道实现的数量和质量仍然非常有限。这有几个原因。

一个是，单个应用程序的有效通道化需要大量的工程和博弈论设计工作。状态通道客户机软件必须考虑并正确处理各种可能性和边缘情况。没有关于如何实现通道化应用程序的标准库、参考实现或教程。目前，每个团队都必须从零开始，投入大量的时间和专业知识来产生可行的结果。

但从我们的角度来看，最关键的障碍是，大多数区块链用户不会在长时间内专门运行单个应用程序。由于进入和退出应用程序状态通道仍然需要提交正常的区块链交易，而该经验所需的所有费用和等待，因此通道化的好处仅限于应用程序本身的重复操作。对于为最终用户提供净经济效益的特定应用通道，必须以同一应用中的大量连续操作为规范。很少有应用程序适合这个特定的配置文件。更糟的是，强迫用户每次开始使用应用程序时都要等待，这会降低状态通道所提供的可用性优势。

状态通道的广义框架

在本文中，我们提出了一种基于区块链的应用程序通道化的替代方法。我们不要求每个应用程序开发人员从头构建一个完整的状态通道体系结构，而是建议构建一个通用框架，其中状态只存储一次，然后任何符合框架的应用程序都可以使用该框架。安装或打开和关闭新应用程序都不需要任何链上事务。用户默认经历通道化，也能够以可比的成本和方便进行正常交易，任何存储和保护其区块链状态的非通道方法。

从开发人员的角度来看，通用框架管理通道体系结构的“重量级提升”，同时向应用程序提供简单易懂的基本功能API。这样，当应用程序设计人员只需重新使用标准功能（如付款、有条件付款、原子交换等）时，他们就不必成为状态通道专家。对于高级开发人员，可以设计新功能并将其添加到框架中，这得益于对新的链上事务的需求同样不足。在一般情况下，用户还允许许多应用程序在为通用框架开发新功能后重新使用新功能。我们的期望是，基础共享功能将被仔细审查、正式验证等，以便提供可靠安全保证的高成本在应用程序开发人员之间共享，而不是在每次开发和部署新应用程序时重复。

我们的方法是开源、模块化和基于契约的。我们首先详细描述设计的需求和目标，然后继续描述如何在我们的框架内实现基本组件的细节。除了框架本身的基本功能外，还存在一些重要的周边服务，这些服务将保持通道化应用程序的可用性、廉价性和性能。其中一些服务，以及框架内更先进的重型优化技术，将仅作简要介绍，细节有待进一步研究。

要求和目标

为了提供一个能够支持尽可能多用例的泛化，我们首先需要了解状态通道将要满足的标准。只有这样，我们才能描述在我们提议的方法中满足这些标准的一组用例。我们首先确定四个基本标准。第一个是我们认为最基本的：

安全标准：将状态委托给正确初始化的状态通道的各方不应显著增加以未经授权的方式操纵该状态的风险，即使其通道的交易对手/中间人既恶意又不可靠（在明确指定的威胁模型的范围内）。此处明确包括无法从状态通道中撤消状态的风险。

这种不可分割的要求，有时也被称为不信任，“不信任”，或简单地“安全”，是任何状态通道建设的核心。状态通道的全部目的是在不让参与者暴露于不可接受风险的情况下，将操作从区块链中移出。意义的定义取决于具体的威胁模型，具体的结构是针对该模型的。我们将在第3节中讨论一些适合于现实状态通道环境的威胁模型。

我们的第二个要求是，可以在通道内执行新的操作，并且这些操作具有与直接在链上执行的类似操作相同的最终性和不可逆性。这一要求确保了通道支持实际的特性，即用户可以用有趣的方式修改状态，而不仅仅是简单地存储和提取状态。

我们认为，一个状态的修改如果最终确定，即当一方受到激励并能够将状态的修改带到链上时，就已经在一个通道中进行了：

通道中的最终性：当某个通道参与者a在链上无法阻止其选择实现x时，区块链属性x在状态通道中最终确定，并且他更喜欢实现x，而不是任何冲突的属性，他们同样可以保证能够实现。

请注意，在这个定义下，通道化状态不需要存在“存储”的实际位置（例如磁盘/内存位置或共享的“迷你链”）。相反，当一个或多个通道参与者确信该状态已最终确定时，就存在一个通道化状态。在实践中，参与者的软件通过将收到的消息（以及相关的区块链信息）与威胁模型进行比较，以确定其向用户报告的通道状态；并通过以维护状态最终性的方式响应来自用户/交易对手/区块链的新信息（因此在适当激励的代表的帮助下进行。

除了两个面向安全的基本标准之外，还有两个面向性能的基本标准，我们想确定。

响应性设计：如果一个通道的所有当事方都在线并就某个当前已定稿的状态达成一致，并且一些参与者满足该状态请求撤销的最终性标准，则其他当事方应优先选择立即批准撤销请求，而不是延迟批准或完全拒绝回复。

链外设计：除非正在进行状态撤销，或者需要一个链上操作来维持先前最终状态的最终性标准，否则所有当事方都应该更喜欢在现有通道内完成新的状态转换，而不是在区块链上完成相同的状态转换。

虽然很容易设计出一个可以即时提取的状态通道，但它是确保即时提取状态也是正常情况的响应标准，这对于确保将状态通道用作默认用户体验仍然是可行的至关重要。同时，链外准则尽可能地在通道内保持新的状态变化，从而最大限度地提高该状态对未来新的通道化应用程序的可用性。这些设计数据的组合实现使得在近即时的链外交互的默认首选情况和随时从链外环境中移除状态的简单、可访问的能力之间有了明确的分离。这些是我们要遵循的准则，以便使通道化应用程序严格优于其实际用户的链上备选方案。

除了这四个基本标准之外，还有许多其他的目标，它们都为我们的设计提供了信息，可以用来比较和对比迄今为止提出的不同的状态通道方法。在我们看来，一个有效的状态通道设计应该（没有特别的顺序）：

•最小化对链上操作和存储的使用

•允许参与者安装和使用新的基于通道的应用程序和通道操作类型，而无需链上操作

•支持高吞吐量

•支持隐私保护操作和技术，包括是否在使用通道

•允许在不同类型的硬件上部署（从嵌入式系统到高性能集群）

•防止跟踪状态和/或所需事务数量随时间不必要地膨胀（出于性能和DoS原因）

•最大限度地减少单一通道和跨通道网络的资本需求

•允许链上组件在一年内保持不变，如时间尺度。

•允许通道的可组合性，这样应用程序就不需要区分特定当事方组可用的网络状态和有助于使该状态的不同部分可用的单个通道关系。

•支持针对单个状态抵押同时进行多个并行操作（即关闭一个应用程序的通道操作，同时继续另一个应用程序的通道操作而不中断）

•支持在不关闭和重新启动通道运营的情况下对基础状态抵押进行链修改（包括存款或部分提取）

•使用模块化方法来封装不同的特性和通道操作类型（出于安全考虑，并且在部署新的基于通道的应用程序时尽量减少重复工作）

毫无疑问，状态通道设计还有更多的目标可以确定，我们希望在以后的研究中探索这些目标。然而，这个列表可以作为一个起始指标，用来评估我们提出的设计和目前存在的更广泛的状态通道生态系统。

威胁模型

最终用户将愿意信任和使用状态通道，只要这样做不会不必要地增加其状态被未经授权的方式操纵的风险。这一决定取决于他们所选择的威胁模型，即：

1.他认为对方可能采取的行动、环境，以及一般情况下他无法控制的任何行动，

2.他认为自己有能力采取什么行动，以及

3.他认为未经授权操纵状态的行为。

威胁模型决定了状态通道交互的不信任程度，因为不信任威胁模型之外的操作。因此，重要的是要记录现实的威胁模型，并分析我们在这些模型下构建的安全性和性能。

在本节中，除了威胁模型外，我们还指定了各种协议和行为。行为指定兼容客户端遵守的规则，而协议是由状态通道智能合约强制执行的一组规则。

一般假设

状态通道的安全性依赖于两个假设，而这两个假设不是在区块链上安全交易所需的假设。

3.1.1对区块链活力的依赖。状态通道威胁模型假定在任何攻击期间都可以成功地保留底层链的各种属性。这些假设中的一些，如假设在给定数量的确认（或某些类似的最终性指标）后成功锁定了对状态抵押的链上贡献，反映了通常由用户放在区块链上的安全期望。其他的，比如某个特定的通道参与者在一定大小的提交窗口上的审查阻力属性，构成可调参数，当状态抵押存储在更有效地满足所需属性的区块链上时，这些参数可以提高状态通道的性能和能力。

3.1.2对参与者可用性的依赖。与公共区块链的标准用户相比，状态通道威胁模型假定一组更为活跃和有能力的参与者。例如，虽然基于加密的区块链的所有用户都必须能够安全地存储和使用某种类型的私钥，但大多数区块链应用程序目前都假定拥有这些密钥是检索和修改用户基于区块链的权利和资产的当前状态的充分条件。相比之下，为了对开放通道的当前状态有信心，可能需要状态通道用户存储其密钥和通道内发生的各种记录，包括他们自己的操作和从通道对手收到的某些已签名消息。状态通道用户还必须有信心能够以一定的频率连接到区块链（以及互联网），以便在交易对手将其发布到状态抵押所在的基础区块链时，他们能够检测并响应过时的更新。尽管这些能力中的许多既可以调整到可接受的水平，也可以安全地外包，但它们对整体威胁模型的影响仍然必须考虑在内，我们的目标是尽可能明确这些假设。

3.1.3缺乏外部激励。与大多数密码经济结构中使用的威胁模型一样，我们也包括一些强调高度地方性经济激励的模型。它们的设计并不是为了在外部激励超过议定书本身所载的经济激励（例如贿赂、死亡威胁）的情况下建立模型或保持安全；在这种情况下，根据申请的细节和外部激励的类型，申请可能会维持或可能不会维持其担保。如果没有对交易对手的整个回报表的直接控制，任何地方的经济机制都不能比对激励行为的偏离施加一些成本更好。然而，我们确实注意到，有些担保更直接地依赖于密码假设，而不是经济假设，而且这些担保可能会持续存在，即使交易对手被任意地支付大笔款项以偏离当地的激励措施。

3.1.4无软件错误。在这些模型中没有显式地处理软件错误。软件中的错误，特别是涉及密钥处理或智能合同代码时，可能会导致部分或所有方的状态完全丢失或破坏。尽管如此，我们为所有状态通道设计包括以下设计：

可恢复性设计：在可能的情况下，应设计状态通道软件，以便通道内的故障可恢复，前提是通道各方应合作制定任何必要的步骤。

这种设计明确地将“人人都是诚实的”假设作为从软件错误中恢复的条件。在最坏的情况下，智能合约漏洞可能会让一方窃取通道中的所有资金。显然，在这种情况下，“窃取所有资金”的策略占据了所有其他策略的主导地位。我们认为支持这种类型的复苏是值得的，因为在许多情况下，外部激励（如声誉）会超过地方财政激励。

griefing

在分析密码经济系统时，griefing是指参与者为了伤害另一个参与者而偏离协议规定的行为的能力，这可能对他们自己造成一定的损失，而不会直接惠及griefer。

在下面的两个部分中，我们定义了两个我们的威胁模型必须考虑的主要griefing策略，即不可用griefing和发布过时状态。

3.3不可用griefing

我们将“不可用griefing”定义为“不可用”的具体策略，即偏离以下行为：

可用行为：当用户签署所有有效的状态更新并在可能的情况下建议离线而不是在线更新时，用户就实现了可用行为。

现有的行为模式为合作各方尽可能多地实现响应和链外设计提供了模型。如果各方都这样做，交易费用将降至绝对最低。一个偏离它的攻击者据说会使通道的其他成员感到悲伤，并因此招致交易费用。例如，考虑爱丽丝和鲍勃之间的状态通道。Alice提出了一个有效的状态更新，但是Bob只是选择不签名。Alice被迫与最近被接受的状态建立连锁关系，这要求Alice支付连锁交易费。

在其他密码经济系统中也存在同样的网格化策略；我们给出了两个具有代表性的具体攻击示例。在casper中，任何验证器都可以停止投票，导致所有验证器都亏损。在Plasma的“权威证明”链中，权威可以停止提交任何标题，强制所有参与者将状态撤回到主以太坊链，在这个过程中支付费用。

3.3.1为什么存在不可用griefing策略。在上述所有例子中，由于说话人/听众的错误等价性，griefing策略不受惩罚[26]：区块链无法区分Bob不可用的情况和Alice仅仅声称Bob不可用的情况。另一种说法是，不可用不是唯一可归因的故障[2]。如果这样的争议发生在一个通道上，那么Alice的连锁行为就相当于声称Bob不可用，但是区块链无法分辨出他们中的哪一个是错的。

3.3.2缓解不可用性。缓解不可用性灰化的技术是一个活跃的研究领域。有几种潜在的解决方案可用于状态通道：

1.使用可信的第三方作为“见证人”，将其作为数据可用性的真实来源。这是一个值得信任的一方，因为恶意证人与恶意通道参与者合作，将能够对其他通道参与者施加不公平的处罚。

2.使用“半可信”的第三方，该第三方可以获得有关通道中数据可用性的私人知识。相信第三方可以使用此信息，例如，设置保险费，以获得保险。见第7.2节。

3.承诺以某种方式作出回应，类似于在一场“鸡游戏”中“扔掉方向盘”，或在相互保证的破坏情况下进行第二次打击核报复。

4.以这样一种方式汇集数据可用性，即攻击者必须破坏大量通道或没有通道。

3.4记录过时状态

与不可用性相比，发布过时状态在大多数情况下是可归因的错误。如果攻击者成功提交旧状态，并且该状态不会受到质疑，那么他将恢复在通道中完成的状态。支付通道中此类攻击的一个具体示例是，攻击者通过通道完成支付，接收商品作为回报，然后在一段时间后提交链上的旧状态。我们可以将发布过时状态建模为与以下行为的偏差：

最新行为：在需要将状态发布到链的情况下，例如在响应其他用户或处理不可用的交易对手时，用户在发布他能够发布的最新状态时实现最新行为。

攻击成功的机会，因此偏离的动机被无法完全消除的环境风险所排除：

•链拥塞：其他用户向网络提交大量高收费交易，临时增加了及时执行交易所需的费用规模，以及任何给定交易将进入随机选择的矿工节点的内存池的不确定性。

•网络断开：由于电源故障、电信问题或其他原因，用户的计算机、手机或其他使用设备的通道与互联网断开。

外部的连锁拥挤源，即通道中任何一方都无法控制的资源，总是会出现。我们观察到，公共区块链的交易负载表现出非常“尖峰”的行为（例如，状态ICO、Cryptokitties发布），这与交易能力的非弹性供应相结合，导致极高的费用周期。这也导致了链上的问题，例如，比特币“dust utxos”[7]面额太小，无法消费（相对于目前的高交易成本）。链拥塞甚至不需要完全超出攻击者的控制范围：在通道中拥有重要资产并从关闭过时状态中集体受益的人群可能会串通，以交易或其他方式处理网络，从而淹没链。如果有必要考虑这种协调，可能会有更复杂的攻击，例如矿工们串通起来审查收尾交易。

3.4.1检测攻击。总的来说，我们必须假定任何攻击都会成功的一些基本的非零可能性，这意味着在没有提交过时状态的惩罚的情况下，尝试“永远不会有伤害”。这表明应采用以下规则：

惩罚过时状态协议：如果提交了可证明的过时状态，提交者将受到惩罚。

请注意，并非所有被取消的出口都是“可证实的过时状态”。在支付通道的情况下，与nonce n的尝试收尾可以被nonce n+1的另一个取消，但这并不意味着第一个提交者（称为爱丽丝）的行为是恶意的。如果Alice提议更新nonce n+1并发送其签名确认该更新，但交易对手没有响应，则可能发生这种取消。在这种情况下，Alice必须与nonce n链接，因为知道对方有nonce n+1。因此，交易对手的行为构成不可用性欺诈。一般来说，确保这一点不发生的问题，即，任何一方都收到了nonce n+1的签名，或者没有一方收到，就相当于没有第三方的签名是不可能公平交换的[20]。Alice与nonce n链接的情况相当于她将链用作第三方，并为此支付费用。

最大限度地增加可证明的过时状态的方法是让客户采用以下规则：

不跳过客户机规则：对于顺序更新，如果第一次更新没有收到副署，则不要在一行中签署两个更新。

其中顺序更新指的是付款通道中的nonce等情况。

然后，协议可以识别提交的可证明是过时的状态并对其进行惩罚。

支付通道的不跳过协议：如果参与者p开始争议并发布一个nonce n更新，随后被nonce严格大于n+1的更新覆盖，则p将丢失一笔保证金。

这也可以放宽，这样P只需支付第二次更新的费用，而不是丢失一笔保证金。当然，采用不跳行为会增加通信成本，如果风险价值不是很高，我们可能不想承担这种成本。在后面的部分中，我们将讨论一个使用根nonce的构造，它允许我们通过选择何时“折叠”一个通道来更精细地控制这种权衡，这样就不会更改任何应用程序状态，但是折叠更新之前的所有已签名消息都可以证明是过时的。

3.4.2处罚。提交可证实的失效状态的适当处罚将取决于存在风险的资金数量和攻击成功的可能性。任何攻击成功的概率将取决于关闭窗口的长度。

3.5发起人支付费用

我们分析了发起方支付争议费用的协议。这对于分析很有用，因为它易于实现，而且许多现有的通道实现[6，24]都使用它。

发起人支付费用协议：当纠纷发生连锁时，发起人支付交易费用。

该协议具有激励相容性，因为可用行为是纳什均衡，前提是由格栅造成的公用事业的总可能减少少于任何一方对状态抵押的所有权。大多数情况下，这也是唯一的纳什均衡，例外情况是当一方在下棋的最后一步会让另一方悲伤，在这种情况下，可用的行为和不签署最后更新的不可用策略都是纳什均衡。愿意进入本协议下所有状态通道的参与者隐含地假设一个威胁模型，“攻击者不能偏离协议指定的行为，除非它对他们有严格的好处”。我们认为这是一个不切实际的威胁模型。

3.6经济风险

在讨论了如何调整协议参数以处理对交易对手因犯可归因错误而攻击能力的期望之后，我们转向设计协议和选择威胁模型的问题，以处理对交易对手执行不可用griefing的能力或可能性的期望，这不是一个ATT。可分布故障。为此，我们引入了经济风险的概念，即一个参与者可能通过欺诈而失去的价值（效用）。我们介绍了威胁模型在经济风险量方面可能具有的三个性质。

3.1.1无风险。在这些威胁模型中，用户不承担任何经济风险，即，他要求交易对手不能降低其回报（与交易对手没有对他进行惩罚的情况相比）。这些是最具约束性的威胁模型；在这些模型下可以进行的交互很少。如果一个协议允许griefing，那么很明显这个风险模型不能被双方采用。事实上，即使在有争议的“没有应用程序逻辑”（即支付通道）的情况下，也可能发生欺诈，因为必须有人支付设立初始状态抵押持有人的费用。在这之后，交易对手可以退出；这是没有第三方的公平交换不可能的另一个后果。

但是，如果另一方不这样做，状态通道中的一方可能在这种威胁模式下运作。例如，单向支付通道，在这种情况下，付款人需要支付建立通道的所有费用。在这种情况下，支付方将依赖外部激励，如声誉。另一个角色颠倒的例子是，一家资金充足的初创公司提供一个DAPP来支付人们建立必要通道的费用，并依靠一些外部的反Sybil指标（例如，给我们发一张带护照的照片）作为一种外部机制来限制他们的财务风险。

3.6.2有限经济风险。在这些威胁模型中，用户愿意丢失有限数量的实用程序进行攻击。这是一个有用的威胁模型，因为在一般情况下，许多用户会很高兴地将自己暴露在少量的风险中，以换取性能或功能的大幅改进。当然，任何状态通道的互动都有一个隐含的界限，因为用户总是可以选择放弃，让交易对手索取所有的状态抵押。我们所说的有界经济风险是指，有界经济风险低于状态抵押的价值——通常要低得多。这意味着用户将只参与有限长度的交互，因此在链外执行整个交互的成本低于他的阈值。例如，一个井字游戏最多可以持续9个动作，如果一个用户做9个链上动作的预期成本低于他的门槛，他会愿意参与其中。

对于没有明确长度限制的游戏，我们可以修改游戏规则以强制执行规则，例如：

国际象棋的有界经济风险协议：当在链子上玩了k个以上的动作而游戏没有结束时，锅会返回给玩家（即，平均分割）。

以这种方式修改协议可能会改变游戏的性质。例如，在上述k=2的协议中，任何一个不离开将死配偶的玩家都可以玩“继续抽签”的移动。显然，这是一个不同于国际象棋的游戏，并不是一个非常有趣的游戏，这仍然适用于小K。

“经济风险”必须捕获的不仅仅是“具有经济价值的链上状态”；它必须包括任何人不想失去的东西，例如，锁定资金的时间价值，或在下棋时花费的精神努力。

3.6.3有界格栅系数。griefing factor是攻击者造成的损失与他造成的损失之比。例如，1:2意味着攻击者必须花费1美元来销毁2美元的价值。当我们说有界griefing因子时，我们的意思是每个参与者的griefing因子都有界于零之外。这是一个有用的威胁模型，因为如果另一个参与者必须承担重大成本才能强制实现这些风险，那么人们将接受更大的风险。说话人-听者故障等效表明，griefing因子边界不能全部优于1:1；在2方通道中实现1:1渐进性的简单协议是：

1:1 griefing factor协议：对于链上移动，交易成本必须在参与者之间平均分摊。

每个通道边界是不够的，因为它们可能仍然允许一种形式的大规模打磨。然而，即使是一个全球范围的griefing因子是否足够，取决于用户的模型，即他们自己的伤害对他们期望通过通道与之交互的代理的效用排名很高。此外，如果我们有一个没有限制经济风险的限制性griefing因子，可用的行为又会变成一个平衡，但这很大程度上取决于假设政党有非常大的预算。现实地说，一个有限的griefing因子威胁模型也应该实施有限的经济风险。

3.7其他有用的威胁模型

我们还没有探索其他有用的威胁模型，包括与交易对手有关的隐私和与中间节点有关的隐私。例如，对于某些应用程序，当事方可能决定将其交互编码为状态机，并且仅反事实地为状态转换函数实例化一个零知识验证器，这意味着当事方可以在链上发生争议，而不必透露有关其交互作用的敏感信息。关于支付通道的现有文献还考虑了支付金额和路由路径方面的隐私（例如，路由支付有时会向所有中间节点显示金额、发送方和接收方）。

此外，这里还没有包括响应委托模型（见第7.2节）。我们将简单地说，授权可用性假设意味着一个“1/n”安全标准，其中所有n个外包服务必须串通，必须放弃收入或损失一些存款（取决于设计）。

4前期工作

状态通道技术-通过完全同意锁定状态抵押和修改链外状态，建立在公共区块链之上，以提供安全性，但在乐观情况下，零边际交易-已经被我们之前的许多项目使用。研究探索了设计空间的两个重要和大致正交的维度：通道内可以做什么，以及通道如何组成网络。我们概述了之前/并行研究中在这两个维度上探索的想法，然后使用我们的需求和目标分析特定项目，并指出它们与我们的工作有何不同。

4.1通道内的一般性

提高一个通道的“通用性”是指扩大其内部可以做的事情的集合；例如，状态通道被认为是支付通道的通用性，即支付通道可以被实现为一种特定类型的状态通道。

4.1.1支付通道。支付通道可以看作是状态通道技术在支付的特定用例（链外令牌传输）中的应用。通过Mike Hearn、Alex Aakselrod和Bitcointalk论坛用户Hashcoin的工作，在比特币之上建立支付通道的提案进展缓慢（有关更全面的历史概况，请参见[5]）。即使在今天，仍然有大量的研究和项目关注或局限于支付通道的用例。由于支付通道只支持少数应用程序，因此我们的某些标准（例如安装新应用程序）不适用于它们。

4.1.2状态通道。除了支付通道之外，使用状态通道技术进行任意状态转换（不仅仅是支付）的想法已经不那么深入地探索，但仍然相对知名，有许多项目，如gnosis[16]和funfair[14]使用这种预测市场的非链外交易技术构建生产应用程序。对于无信任的赌场游戏。在社区中，这些通道被称为“状态通道”，通常与支付通道形成对比。对这一点的早期描述是在第一作者的博客文章[12]中，以及由[21]独立生成的早期概念验证代码，该代码将棋盘存储为反事实状态，并使用交互式验证进行棋盘计算。

4.1.3广义状态通道。我们的工作通过引入反事实实例化等新技术以及基于契约的设计框架来概括现有的状态通道，这两种方法都是我们在“广义状态通道”一词下所指的。与现有的“特定于应用程序”的状态通道相比，与最终用户最明显的区别是通道化安装新功能，这使得它更便宜（无事务处理）、即时且优化地保护隐私。在Perun[24]中独立发现并实现了类似的“链外实例化”（在第4.3.4节中讨论）。

4.2中介机构之间的互动

目前对支付通道的研究大多集中在探索如何通过中介机构进行支付结构的设计空间。假设爱丽丝和鲍勃有付款通道，鲍勃和卡罗尔有付款通道。如果Alice在链下支付Bob，然后Bob支付Carol相同的金额，这相当于Alice在链下支付Carol，而无需设置新的Alice Carol支付通道。

有多种方法可以做到这一点，即，确保爱丽丝支付鲍勃如果鲍勃支付卡罗尔。在闪电式散列时间锁合同（HTLC）中，来自两个支付通道的等量资金都被锁定，只有在某个散列在某个截止日期之前被显示出来（因此，被“散列”和“按时间”锁定），才能使用它们。从Bob的角度来看，如果锁是用相同的哈希设置的，但是出站锁的截止日期早于入站锁，那么他可以保证，如果出站付款，他也可以强制入站付款。为了保持术语的一致性，我们将这些“直接”支付通道称为一跳支付通道。

Sprites项目[6]观察到，这种机制意味着最后期限必须错开，这意味着最坏情况下的资本锁定时间在跃点数量上呈线性增长，因此总资本锁定成本在跃点数量上呈四次增长。他们设计了另一种解决方案，其中散列揭示截止时间设置为早于争议解决截止时间，允许所有锁具有相同的截止时间。因此，资本锁定时间与跃点数无关，总资本锁定成本与跃点数呈线性关系。

与这两个方案相比，我们的元通道构建使用“租用通道”方案。我们使用智能合约链将中介机构的资金锁定一定时间，而不是在每次支付中建立智能合约链，在此期间，可以进行任意数量的非链支付。在支付通道网络的具体情况下，其区别类似于电路交换网络和包交换网络之间的区别。我们的渐进资本成本和锁定时间与Sprites式HTLC相同。使用面向对象的框架，我们还可以构建哈希时间锁定条件支付，允许用户根据需要实现闪电式或精灵式支付路由。

4.3现有设计

4.3.1闪电网络-闪电网络可以说是使用支付通道技术的最知名的项目，短期内可能会看到按价值计算的最大交易量。它使用一些特殊用途的操作码，在比特币上建立连锁小额支付。雷电网络在[23]中提出，目前在[11]中规定。

Lightning Network通道使用资金交易中锁定的资金，这是由通道中的各方创建和拥有的多签名UTXO。余额更新是通过签署两个非对称承诺交易来完成的，每个交易都花费资金交易，立即向交易对手发放远程资金，并开始一个挑战期，之后广播公司将收到剩余的本地资金。承诺交易有一个撤销机制：旧的交易被撤销密钥的公开明确撤销，而过时的承诺交易txo允许任何拥有撤销密钥的人申请所有本地资金。在一个简单的设计中，所有旧的撤销密钥都必须存储在内存中，这使得客户端的内存需求与支付的数量成线性关系。挑战周期是使用op checksequenceverify相对时间锁操作码实现的，因此通道可以无限期打开。合作即时取款支持使用直接支出融资交易的结束交易；因此，合作关闭的通道与正常的多交易不可区分。通过合作将融资交易分为两个输出，一个部分支出和一个新的融资交易，可以支持部分结算。

4.3.2 RAIDen RAIDen项目[3]旨在使用以太坊智能合约建立支付通道网络，支持以太坊和ERC20代币。RAIDEN网络可以无限期开放。不支持合作即时取款，但有一个公开的问题需要添加。raiden使用闪电网络使用的相同哈希锁机制，即中介使用不同的到期时间，这意味着L-hop路径将抵押品锁定O（L）时间。除了支付，瑞登还计划支持原子交换。RAIDen项目还构建了微RAIDen，它的功能集比“全RAIDen”小，因为它只支持单向支付通道，不支持传递支付。

4.3.3 sprites sprites项目[6]利用我们一直称之为sprites-style htlc的技术，构建了支付通道网络，从而降低了闪电式htlc的最坏附带成本。此外，他们还构建了状态通道，用作OneHop支付通道和Sprites风格HTLCS的基本“构建块”。他们的状态通道在两个方面与我们的相似。首先，状态转换函数可以依赖于链状态（实际上，此功能用于引用公共的“事实来源”，以确定HTLC的散列是否及时显示）。第二，支持任意的状态抵押，因为用户承诺在某些情况下允许状态通道调用handleoutputs；实际上，对于他们的支付通道，使用单独的合同来保存存款。sprites风格的支付通道与我们的不同之处在于，它们不需要被反事实地实例化，并修复一些设计选择，例如选择使用标量nonce、签名算法和使用字节32存储状态。精灵式的支付通道也不支持合作取款。

4.3.4 Perun Perun通道与我们所采用的其他支付通道相比，有三个创新点：第一，其通过中间商（他们称之为虚拟通道）路由的技术与我们的技术非常相似，因为虚拟通道是“非交互的”，不需要中间商就电动汽车进行合作。ERY付款。其次，Perun状态通道具有与我们相同的模块化/并行结构。在他们的术语中，多状态通道的状态σ被分解为“子状态”（σ1，σ2…σn），可以同时更新和争议，这与我们将状态通道中的反事实状态分解为反事实实例化的契约相对应。这种平行性对于支持租金中介模式是必不可少的。第三，“子状态”集合可以在链外更新（例如，多状态通道的状态可以从（σ1，σ2）到（σ1，σ2，σ3），而不需要OnChain事务，在我们的范式中，它对应于用状态σ3反事实地实例化契约），从而允许在链外安装新的应用程序。

但是，Perun通道缺乏反事实寻址和反事实代码实例化（只有状态是反事实实例化的）。这意味着，虽然新的应用程序可以安装在链上，但该功能的代码必须已经部署在链上。这也意味着纳米合同不能相互参照，限制了它们的组成。此外，还没有普遍的状态抵押；纳米合约明确地“阻塞”了一定数量的乙醚（并且只有乙醚，作为其建设中的一种供奉货币），纳米合约的财务结果通过明确定义的API修改了阻塞的乙醚的所有权。

Perun论文本身只讨论了多跳支付通道，后续论文[25]描述了多跳状态通道和长度大于2的支付通道的构建。它们的构造被概括为递归地构建虚拟通道，与我们的构造不同，主要是由于缺乏反事实寻址。在我们的范例中，中介锁定了某个状态来构造一个元通道，其中一个支付对象保持锁定状态，元通道的各方随后反事实地实例化他们想要的任何功能，并从支付对象中为其提供资金。然而，在佩伦的范式中，多跳虚拟通道中的争议必须（无信任地）通过中介解决。更具体地说，如果通过Ingrid有一个Alice Bob Metachannel，并实例化了许多功能，Alice可以通过强迫Ingrid在Metachannel中播放所有的动作来哀悼Ingrid，而在我们的构建中，Alice只能对Bob这样做，Alice只能哀悼Ingrid，哀悼Ingrid所需的费用的价值，直到支付给主题。

5概念和定义

任何状态通道所使用的核心技术都是利用尚未实现的链上后果来执行和保障纯粹的链下活动。如果通道建设得当，许多后果将永远无法实现，即使是在预期中。然而，他们在游戏理论模型中的存在对于通道的设计和安全至关重要，激励参与者避免采取可能导致这些负面后果实际发生的行动。

因此，任何状态通道设置的必要步骤都必须是转移一些链上状态（以太坊、令牌所有权等），使其远离单个参与者的单边控制，以便执行行动能够修改该状态。这些资产包括状态抵押，只要它们在状态通道中，就被“锁定”。在锁定了状态存储之后，参与者可以在状态通道内通过向彼此传递特殊类型的消息进行“链外”交互。这些消息的传递会导致链上杠杆的变化（即参与者能够执行的链上操作的类型）；实际上，由于我们对“链下交互”的定义，这是他们唯一能做的事情。然而，这种看似有限的能力提供了足够的能力来取代一些链上操作。例如，在支付通道中，更新“仅”更改了杠杆率，但这足以让参与者将最近签署的余额视为“我在该支付通道中拥有的以太数量”6。还要注意，虽然我们有一些“链外状态”和“链外事务”的概念，但实际上没有任何EVM在链外运行；在通道各方之间传递的消息与计算状态的任务无关，而更多的是与说服另一方可以将状态视为最终状态的任务有关7。

5.1反事实术语

由于状态通道交互都是关于修改杠杆，我们花了很多时间讨论区块链上可能发生的事情，但没有。链上执行可能非常复杂，确实需要这样做，因为广义状态通道的全部功能都是通过修改执行杠杆来实现的。反事实术语是一种将杠杆分解成更小、更模块化的部分，并在不使用长的、严格限定的句子结构的情况下进行讨论的方法。粗略地说，对于任何连锁经营

可以通灵的x，我们用反事实的x来讨论

1.x可能发生在链条上，但不会

2.执行机制允许任何参与者单方实施X。

3.参与者可以表现得好像发生了X。

例如，在支付通道中，X可以是“4以太从智能合约转移到爱丽丝的账户，6以太从智能合约转移到鲍勃的账户”，如果双方都有最新签署的副本，则反事实X将是事务状态，其中记录爱丽丝的余额为4，鲍勃的余额为6。

对反事实术语的更完整的处理更为复杂，例如，需要对参与者、他们的能力和激励的定义、连锁经营的“类别”以及第3条对参与者的确切含义。请参阅附录A了解完整的处理方法；但是，我们建议首先阅读本节和以下内容，以便在理解一般定义之前对一些反事实术语有一个直观的理解。

5.2反事实状态

让我们更仔细地研究一下如何使用这个术语来描述现有的支付和状态通道建设。在以太坊中，帐户的状态由它的nonce、ether balance、合同代码和存储组成。当某些预先指定的用户（选择包括受反事实状态影响的用户）中的每个人都可以单方面将链上状态更新为给定的反事实状态时，正确构建的智能合约将保持反事实状态。

例如，让我们描述现有文献中使用的超时/挑战/终结机制。反事实状态分为nonce和特定于应用程序的状态，其中nonce是

一些计算成本。

6使用我们的术语，现有的支付通道解决方案使用相同的智能合约来持有状态抵押并执行；链外消息包含合同解释的签名消息。

7在实践中，应用程序编写器将定义一些标准的消息模板，用户客户端软件可以简单地检查消息，而不是要求用户“从头开始”解释消息。

单个uint256，合同定义了一个固定的参与者集。参与者签署包含新的应用程序特定反事实状态的更新，并且更新必须增加nonce。对于支付通道，应用程序特定的状态是从参与者到其余额的映射。对于一个通道化的象棋游戏，棋盘的状态作为应用程序特定的状态。在链上争议期间，反事实状态被视为临时状态，直到提交签名更新时对象进入最终状态，并且在未提交更新状态的情况下，挑战期结束

5.3反事实例示

以太坊通过支持智能合约为开发者提供通用计算平台。同样，状态通道中的参与者应该能够反事实地在状态通道中实例化智能合约，将新功能9安装到状态通道中，而不需要任何链上操作。反事实实例化后，用户将受到反事实实例化合同的约束。

一般来说，链上智能合约只有在控制链上有价值的状态时才有用。由于反事实实例化的契约实际上并没有部署在链上，因此它们不能直接取得链上资产的所有权。参与者必须向状态抵押持有人签署承诺书，以便执行过程以某种方式使用反事实的例示合同。因此，状态抵押的一部分被分配给反事实例示的合同。作为执行杠杆的一部分，参与者必须能够单方面地在链上实例化反事实实例化的合同，然后让合同控制如何支付状态抵押。

5.3.1反事实处理。一个潜在的问题出现了，因为在部署以太坊合同之前，我们通常不知道它将部署在哪个地址，因为当前合同的地址取决于合同创建者的地址和nonce[19]。状态抵押持有人事先不知道反事实例示的合同将有哪些地址，但需要作出涉及此地址的承诺。出于同样的原因，反事实例示的合同不能直接引用彼此。

为了解决这个问题，我们提供了一个全局注册表契约，它部署在链上，允许引用反事实的实例化契约。这里的“全局”意味着它的一个实例在所有的状态通道中共享。注册表需要将合同的反事实地址定义为合同初始化代码的确定函数。注册表提供了一个解析函数，将反事实地址映射到以太坊地址（类似于DNS如何将域名映射到IP地址），以及一个部署函数，用于部署新合同并注册它们。

通过计算一个合同的反事实地址，然后向状态抵押持有人签署一些承诺，向其分配一些状态抵押，从而反事实地例示了一个合同。注释

反事实实例化是签署这些承诺的行为，而不是实际调用链上部署的行为。一旦合同被反事实地实例化，用户就可以使用它的反事实地址来引用它。

5.4最小化链上功能和状态

反事实实例化允许我们在链外实例化特定于应用程序的智能契约，但我们也可以更抽象地将其视为一种将功能和状态移出链的技术。由于我们希望尽可能多地这样做，因此我们还反事实地实例化了争议解决功能（例如挑战/超时）。然后，状态抵押持有人可以是N对N多签名钱包，并且是必须为每个附加状态通道部署的唯一链上组件。我们认为这是必须放置在链上的功能和状态的最小数量；状态抵押必须由一个在其上执行基于一致同意的操作的实体持有，而multisig则完全实现了这一点，而不是更多。我们认为这是本文的一个重要观点：

一个足够强大的多签名钱包足以作为状态抵押持有人。

这不仅是可能的，而且实际上是一个有用的设计选择，提供两个主要好处：

1、隐私权。状态通道参与者可以选择不共享其已签署的承诺，在这种情况下，外部观察员只能看到链上的多西格钱包；希望隐藏他们之间存在状态通道的参与者可以使用“所有链上多西格集”作为匿名集。

2.可升级性。如果在争议解决代码中发现错误，或者更普遍地说，在multisig之外的任何状态通道代码中发现错误，则可以在双方合作的情况下进行链下更换。

将标准multisig钱包用作状态抵押持有人的另一个好处是，任何支持multisig钱包所有权的资产都可以由状态通道自动控制。例如，一个抽象的资产，如“设置ENS名称的权利”，可以存入状态抵押。

5.5组织链下状态和交互

5.5.1反事实对象。作为状态通道的模块化构建块，我们将反事实对象定义为包含反事实状态的反事实实例化契约。状态通道由反事实对象的集合组成，这些对象使用反事实寻址互相引用。这允许我们尽可能地划分状态通道中的功能、状态和交互。两个参与者可以通过使用一个反事实实例化的支付通道对象来打开一个包含他们之间的小额支付功能的状态通道，然后在一段时间后决定通过反事实实例化一个国际象棋反事实对象来玩一个链外的国际象棋游戏。逻辑和状态保持分离，并且它们之间的任何交互都被最小化。

5.5.2指定沉积状态。对于一个反事实对象o，为了控制由multisig m持有的某些状态抵押，必须承诺m使用最终确定的反事实状态o支付该状态抵押的部分。例如，如果o是一个支付反事实对象，具有反事实地址c和反事实状态（a，b），代表alice和bOB的平衡，应该有M的承诺形式“如果反事实对象在反事实地址C存在于反事实状态（A，B）的最终状态，然后发送一个以太给Alice和B给Bob”。可以看出，这些承诺相当复杂，因此我们的multisig必须能够执行复杂的多操作码操作。此外，重要的是，我们使用多重利益集团，一旦作出承诺，就不能单方面撤销。

图1：multisig执行对付款对象的调用，并使用结果确定转移到Alice和Bob的金额。

5.5.3反事实对象之间的关系。反事实对象可以在许多方面相互依赖。除非存在另一个反事实实例化的约定，否则可以编写对象，使其构造函数返回。或者，在完成另一个对象之前，对象可以拒绝转换到某个已完成的状态，我们称之为条件完成。在这两种情况下，反事实处理是必要的参考其他合同。理论上，这些关系可以变得任意复杂；一个对象可以拒绝完成，直到其他对象集存在，其中一个对象的nonce在给定范围内，而其他一些对象集不存在，等等。

状态通道的反事实状态是指各客体的局部反事实状态以及它们之间的关系网络。当我们说状态通道是以一致同意的方式运作时，我们的意思是任何一方都不能单方面改变这个状态，除非有关各方事先给予该方许可。

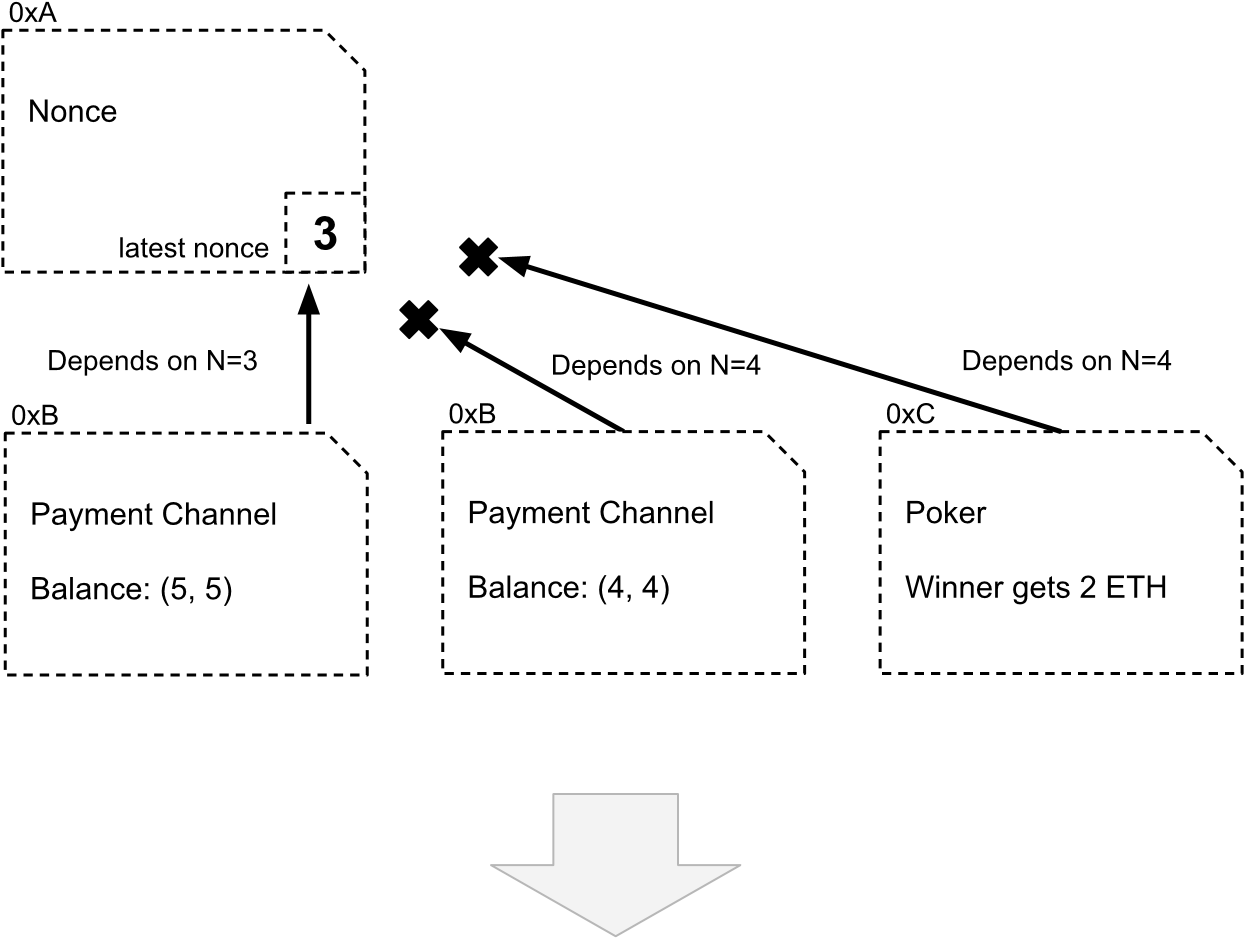
5.5.4非依赖条件终结。一种特殊的条件终结称为非依赖条件终结，它是我们用来组织反事实状态的关键工具。在它中，依赖对象v直到依赖对象u在给定的nonce n上完成时才结束。然后我们说v依赖于u在nonce n上。

图2：非依赖条件终结。通过将nonce对象的nonce增加到4，付款通道对象将无法再完成。

由于nonce也用于对消息排序，因此如果我们希望使用相同的nonce更新本地状态和依赖项，则会出现一个潜在的问题。例如，如果我们有一个基于支付通道对象的象棋游戏，发送一个小额支付会增加nonce；因此每次发送小额支付时，我们都必须确保象棋对象不会被删除。避免这种情况的一种方法是不要将同一个nonce同时用于两个目的，例如，通过重构来使用3个nonce；一个用于依赖项，另两个用于本地状态的更新。

图3：多个从属项。0xa的nonce纯粹用于管理依赖项，而0xb和0xc的nonce用于更新本地反事实状态。

5.5.5原子态转变。使用多个从属项，我们可以原子地进行复杂的状态转换。设C为一个具有反事实状态的反事实对象，它可以取A和B的值，目前是A。建立一个反事实对象网络，其存在取决于C的状态为A，另一个反事实对象网络，其存在取决于C的状态为B，并将C的状态从A更新为这也允许我们在对象之间创建更多的“概念”关系。例如，我们可以反事实地实例化一个扑克反事实对象，并通过原子性地减少支付通道余额和增加分配给扑克反事实对象的状态抵押量，从现有的支付通道为其提供资金，从而实现它们之间平衡关系的概念性保护。



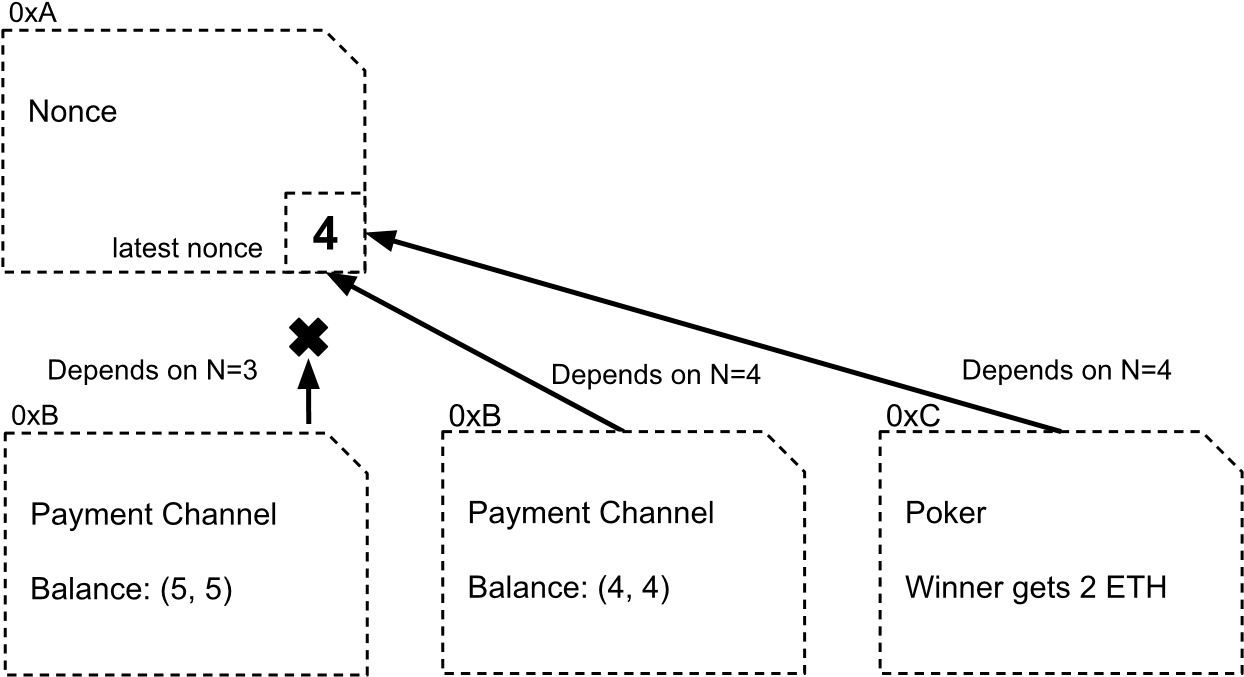


图4：原子操作。

使用一个专用的nonce对象的另一种选择是使用向量nonce，其中nonce被定义为自然数的向量，nonce可以在词典中进行比较。然后，依赖对象可以依赖于nonce的低阶字段。

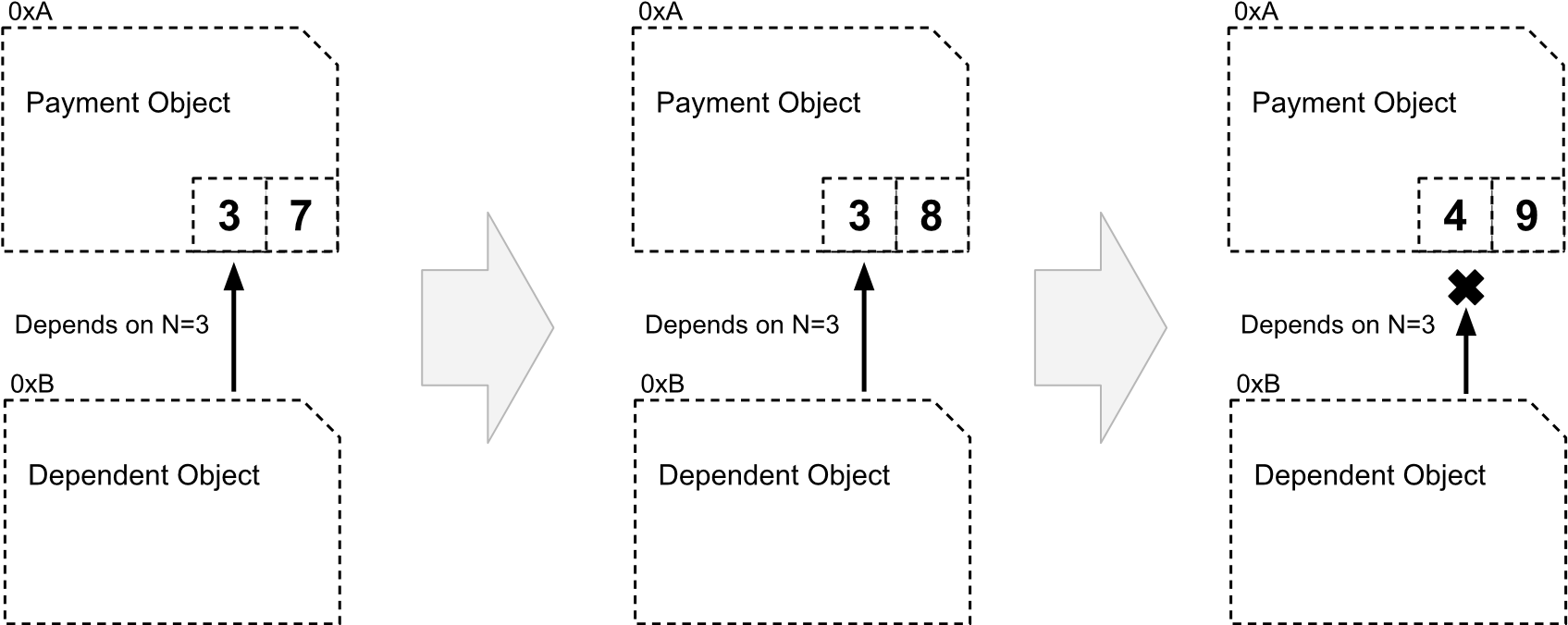


图5：向量nonce。当依赖对象仍然存在时，通过增加最低有效位来更新平衡。然后，更新平衡并原子地删除从属对象。

非依赖条件终结可传递地应用；如果a依赖b依赖c，则a无法终结，除非c终结。这就创建了一个层次关系，对象可以排列在一个有向非循环依赖关系图中。为了防止在链上发生冲突时，在尝试完成根到叶路径时超时“堆积”，我们使用一个称为递归终结的方案；所有的挑战超时都可以同时进行，当对象接收到ISfinal查询时，它会递归地对其父级执行ISfinal查询。

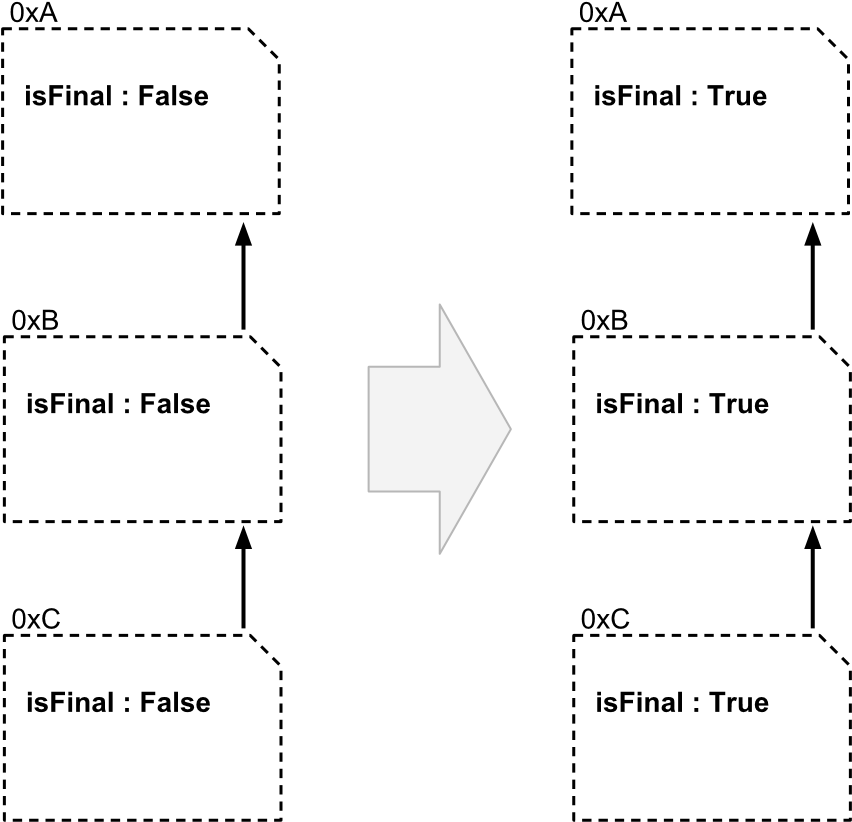


图6：分层定案。在一次争端中，三个反事实的对象在一次交易中最终确定。

5.5.6根nonce。应该有一个“根nonce”反事实对象，它有一个标量nonce，不包含任何状态，并且所有其他对象都依赖于它。根nonce的存在允许我们原子地进行影响多个对象的任意复杂更改，包括删除对象。此外，在争议中，我们可以通过发布根nonce的更新值来响应依赖根nonce的过时值的过时状态。如果所有过时状态都依赖于根nonce的过时值，那么我们对所有过时状态都有一个O（1）响应，并且我们可以在脱机之前以这种方式“折叠”通道的状态。

5.6与链上状态的交互

可以对许多引用链状态的应用程序进行通道化。例如，两名玩家可能希望在某个状态通道内就一些知名的未来活动（例如，2020年美国总统选举的获胜者）下注，并使用链上分散的Oracle，如Augur。如果甲骨文是适当设计的，这是非常直接的；例如，占卜市场有一个固定的地址，并解决成一个不变的状态，因此一个反事实的实例化合同可以简单地引用它。这表明，在反事实对象内确定的状态可能会发生变化，即使相关各方之间没有传递消息。

5.7面向对象方法

我们将组织状态通道的方法描述为反事实对象，每个对象都包含自己的功能和反事实状态，并将其描述为状态通道的“面向对象”方法。

5.7.1有条件付款。我们提倡的一种结构是通过使用有条件的支付反事实对象来分离支付逻辑和应用逻辑。条件付款反事实对象包含观察另一个反事实对象的功能（通过解析反事实地址，然后对其执行消息调用），然后使用结果支付指定的状态抵押。观察到的对象实现了特定于应用程序的功能，并且多个条件付款对象可以观察到相同的反事实对象（例如，在mkr和rep的原子交换中，mkr和rep的反事实付款对象应该观察到相同的对象）。这种设计有两个优点：

1.风险所在地。应用程序逻辑中的软件错误只会对分配给条件付款对象的值产生风险，前提是条件付款对象中没有其他错误。这一点很有用，因为虽然支付逻辑将受到严格审查，但将编写更大的应用程序逻辑，并且应尽可能多地包含其中的错误。

2.不受限制的状态抵押。任何可以由多袋钱包持有的东西都可以作为状态抵押持有，并分配给有条件的支付反事实对象。作为一个思想实验，假设将来某个时候提出了一个ERC2000规范，它表示可替换的令牌，但与ERC20不兼容；现有的用户将不存在使用ERC2000令牌作为状态抵押的问题。

5.7.2即时提取和补充。面向对象的方法很容易支持即时提取和即时补充，这意味着附加状态可以（在链上）存储到通道中或提取，而不需要挑战周期。让我们考虑立即撤军。首先，一些以太被分配给一个反事实对象，其状态取决于多重对称的平衡。然后，提款交易自动将资金移出状态通道，并减少分配给状态通道内提款接收者的余额，而追加交易自动将资金移入状态通道，并增加分配给出资者的余额。

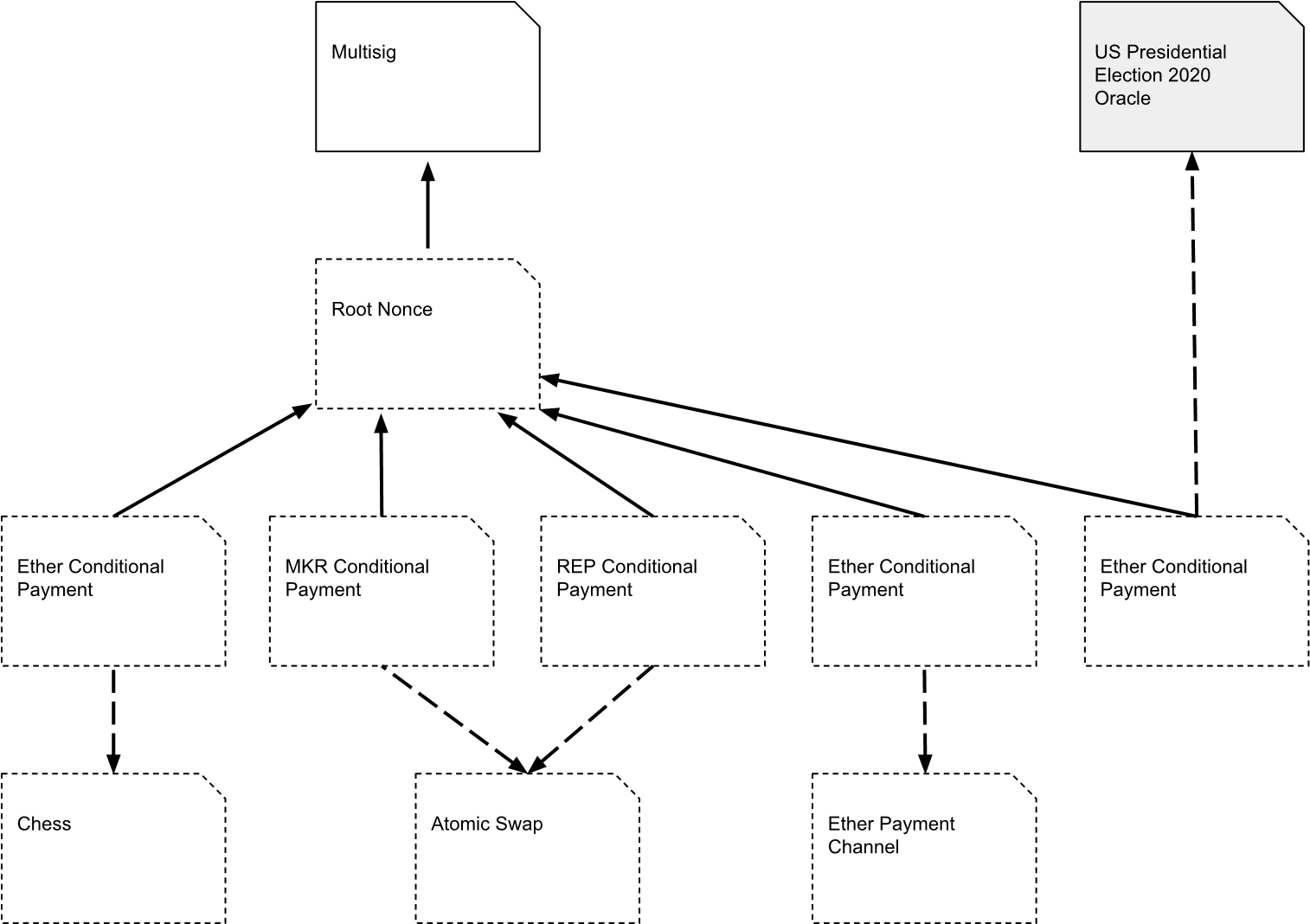


图7：典型状态通道的结构。实心箭头表示“依赖”关系，虚线表示“观察”关系。

5.7.3利益。我们相信这种面向对象的方法通过提供以下好处来最大限度地提高灵活性和模块性。

1.功能的可组合性。单个反事实对象可以在状态通道中以多种方式互相“插入”。例如，一旦编写了一个扑克反事实对象，它就可以在一个有任何类型的pot的状态通道中使用（只需让适当类型的条件支付通道查看它），以及任何随机性源（通过让扑克对象查看“随机性游戏”反事实对象）。应用程序编写者只需要指定扑克规则，不需要是ERC20令牌接口或随机性游戏的专家。

2.重新使用链上设计技术。由于以太坊状态也是以面向对象的方式组织的（将功能和状态结合在一起的智能合约），因此EVM针对支持该组织进行了优化，我们可以从设计链上DAPP时使用的相同设计技术中获益。例如，一种常见的技术是，广泛使用的契约通过代理契约[15]共享功能（但不是状态），我们可以在通道中使用完全相同的技术。

3.梅克尔化争端。如果Alice和Bob需要对链上的国际象棋游戏产生争议，那么就没有理由在链上拖动一个不相关的原子交换对象。它还为我们提供了一种方法来控制应用程序逻辑（bug等）中的风险，方法是将撤销的内容限制到有条件的支付通道。

5.8更改参与者集

到目前为止，我们的状态通道已经是N方状态通道，在N上没有任何上限，而任意大的支付通道可以分解为2方状态通道加上支付路由的集合，状态通道的情况就不一样了。例如，以不可撤销代币（NFT）作为赌注的一部分进行的4人扑克游戏不能分解为2人状态通道加路由，并且通灵的4人扑克游戏必须获得所有4人的同意才能更新状态。

反事实对象还可以指定允许谁修改其反事实状态（例如，所需的签名集），特别是不同于“所有状态通道参与者”，这是我们隐式使用的。假设Alice、Bob和Carol建立了一个三方状态通道；拥有所有者设置Alice和Bob的支付子通道拥有的财产不需要Carol同意即可更改余额，因此其行为与Alice和Bob之间的正常二方状态通道非常相似。

5.9元通道

元通道是一个面向对象的解决方案，解决了跨中介交互的问题。我们首先为付款用例描述它们。假设Alice与Ingrid有一个状态通道，Ingrid与Bob有一个状态通道，Alice希望与Bob建立一个支付通道。首先，我们观察到Alice和Bob可以创建一个反事实实例化的自己拥有的支付通道对象。把这个对象称为O。现在，我们必须使O的反事实状态（即爱丽丝平衡和鲍勃平衡）有意义。我们通过创建两个代理支付反事实对象来实现这一点，一个在Alice Ingrid和Ingrid Bob通道中，每个对象都分配有状态抵押，并且遵守O。

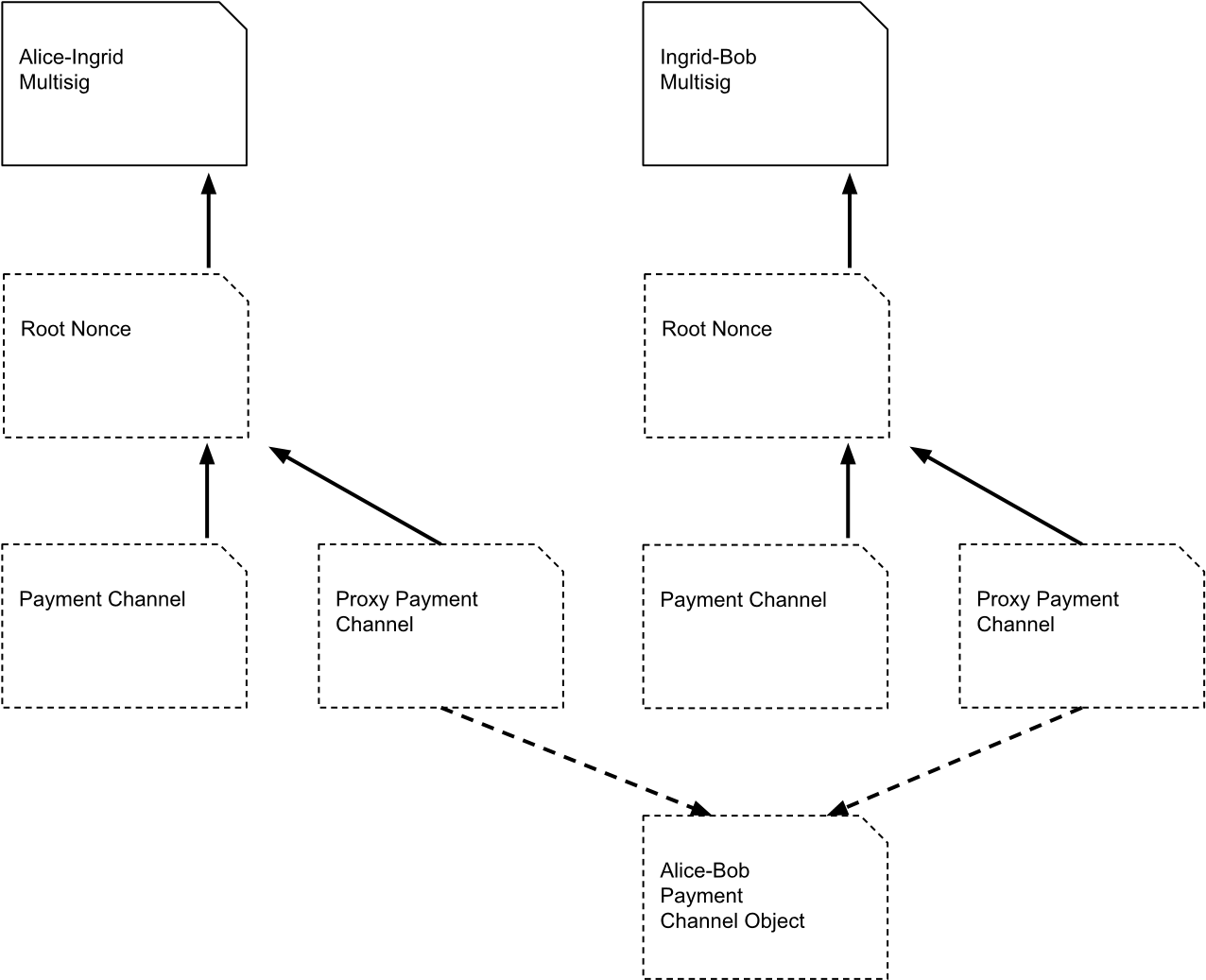


图8:Alice和Bob之间通过Ingrid的元通道结构，实现支付通道功能。

假设Alice Bob支付反事实对象具有分别代表Alice和Bob余额的反事实状态（A、B）。左边的代理支付对象将A分配给Alice，B分配给Ingrid，右边的代理支付对象将A分配给Ingrid，B分配给Bob。这样，英格丽德总是有A+B分配给她。

我们可以使用相同的“将Alice的余额分配给左侧，将Bob的余额分配给右侧”，将其归纳为任意长度的中间链。

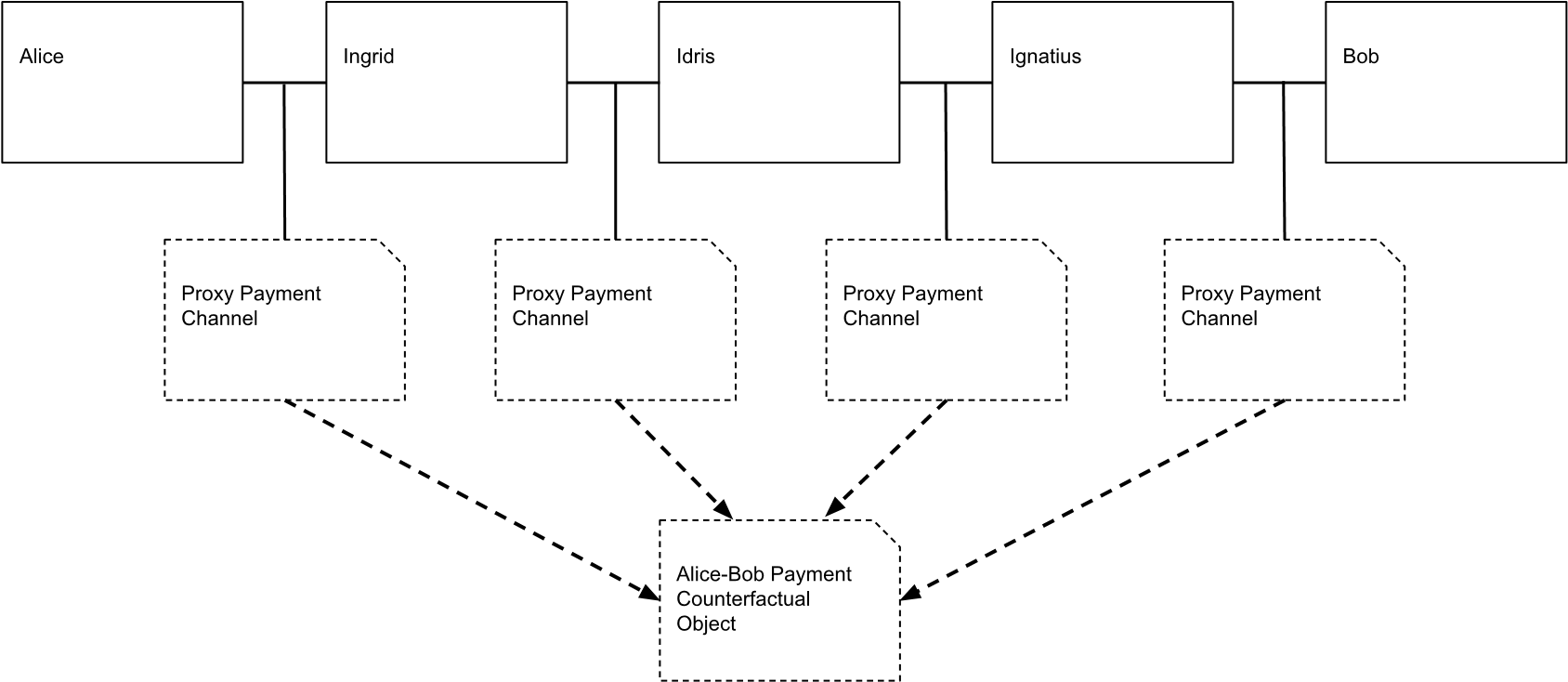


图9:Alice和Bob之间的元通道，通过由3个中介组成的中介链，实现支付通道功能。

最后，我们观察到一个足够强大的multisig对于一个状态通道是足够的，并且我们可以将multisig功能放入o中，从而允许o将其状态存储重新分配给Alice和Bob控制的其他对象。一般的多重投资通过打电话给其他合同（转让乙醚、在ERC20合同中更改余额）来执行对其的承诺，而o只通过向其自己的仓库写入来执行承诺。

6构造广义状态通道

在本节中，我们为通用状态通道提供一种构造，即一组用于创建状态通道、更新状态通道、安装新功能和撤消状态的协议。请注意，我们已经做出了详细指定协议的实现选择（例如，契约的函数签名和消息格式）。可能有不同的选择，并且有许多特定的协议使用与第5节中相同的概念。所提供的规范是设计空间中的一点，我们并不认为这是最理想或最简单的气体。

我们使用粗体无衬线字体（例如，multisig）引用链上合同。反事实实例化的合同类型以相同字体的斜体和非粗体表示（例如rootnonce）。协议使用粗体（例如，实例化）编写。

6.1实施技术

在实现通用状态通道时，必须做出各种不改变核心概念或定义的实现选择，但这些选择并不琐碎，足以进行描述。

6.1.1基于委托呼叫的承诺。如第5节所述，通过签署承诺书，将状态抵押分配给反事实对象，该承诺书承诺根据部署的反事实对象的状态支付状态抵押。因此，“承诺”是指承诺（原子地）执行具有条件逻辑的复杂多操作码操作，multisig wallet必须能够执行此类操作。由于以太坊不直接支持执行通过交易数据提供的操作码（例如，通过假设的eval操作码），我们使用delegatecall操作码在钱包中提供多操作支持。DelegateCall允许调用者契约A在A的上下文中运行时执行位于不同地址B的代码，保留A的msg.sender、msg.value、this、this.balance和storage值。

6.1.2反事实承诺。使用基于DelegateCall的承诺，可以通过承诺对包含“实际”承诺的链上地址执行DelegateCall来作出复杂的承诺。为了避免“实际”的承诺在链上，我们可以使承诺成为对代码的承诺，而代码是反事实对象的一部分，也就是说，承诺委托调用执行查找的链上代码块，然后委托调用它。在我们的设计中，这个链上的代码块被放置在注册表中。

6.1.3注册中心的认证。我们为注册表提供两种方法来验证部署对象的请求。第一种方法是检查msg.sender。第二种方法是显式地提供签名列表；这是用于元通道的技术。

6.1.4多任务所有人。在一个n党状态通道（没有元通道）中，每个反事实对象都属于n党。实现这一点的一种方法是让反事实对象具有更新功能，该更新功能必须提供N个签名。我们选择让multisig“代表”n方拥有反事实对象；承诺更新对象o的反事实状态是承诺multisig在o的反事实地址向合同发出具有某些固定（在承诺签署时）参数的呼叫。这有两个好处：第一，对于一个更新，只需要执行一次组签名验证（这可能非常昂贵，例如n ecrecovers），而不是两次。其次，更新可以在Merkle树结构中进行批处理（仅限Merkle根签名），将单个组签名验证的成本分摊到多个更新和部署上。

6.1.5根nonce。对于本节中的构造，我们对直接依赖根nonce的所有对象作出设计决策。如前所述，这并非所有状态通道都需要，在某些情况下可能是次优的。

6.2链组件

6.2.1多签名钱包。我们要求该多签名钱包需要所有状态通道参与者的同意（即一致同意），例如通过n对n阈值。这是用于控制和持有所有状态抵押的对象，也是我们设计中的唯一组件，必须在安全区块链层上为任何新的状态通道实例化。由于MuleSIG钱包在状态通道的上下文之外也是有用的（例如，在MulthIG钱包中持有基金的基础），所以我们描述了MulthIG钱包必须支持的一些特性，以便作为状态通道的状态保证金持有者可用。

•需要一致同意。

•能够自动执行复杂的操作。

•一旦同意，不得仅由通道中的一个或几个参与者撤销。

•承诺不得要求或强制执行特定命令以保持有效。

•承诺必须确保不受重播考虑的影响。

•承诺必须由同意方以外的实体提交。

为了满足这些要求，我们的multisig有一个公共的executeTransaction函数，它可以使用基于delegateCall的承诺在multisig上下文中执行任意事务（前提是这些事务由所有所有者签名），也就是说，executeTransaction可以执行任意的单个delegateCall。我们还允许ExecuteTransaction执行单个任意调用，以避免在DelegateCall中包装调用。这使得任何一方都可以提交所有同意方签署的交易，以“代表”multisig执行任意功能。

6.2.2登记处。注册表将反事实地址映射到部署在链上的合同的地址。它是一个状态契约，在存储中包含一个从反事实地址到部署地址的映射；我们使用字节32作为反事实对象的空间，因此映射将具有solidity类型映射（bytes32=>address）。注册表总共提供了五个函数；我们在这里指定了其中四个函数，并将最后一个函数的描述推迟到第6.7.1节。我们可以将这些函数分为两组；前两个函数提供支持反事实实例化所需的最小值，后两个函数是实用函数。

•部署：使用给定的代码和构造函数参数部署契约，计算反事实地址，并将反事实地址=>部署的地址映射写入存储。

•解决：在存储器中查找给定的反事实地址，并返回相应的地址。

在我们的注册表中，C.code和C.args通过连接在客户端进行组合；solidity abi指定当作为create opcode调用的参数广播时，合同的编译代码将执行构造函数，在编译代码之后查找构造函数的参数。因此，我们的deploy函数只接受一个字节码块来传递给create。因此，deploy具有签名（bytes32）=>（）。

其他两个函数不一定需要放在注册表中，而是在这里找到它们的逻辑原点。它们用于支持基于代表电话的承诺和反事实的承诺。它们是简单的代理函数；就像一个人可以调用以太坊地址一样，他使用proxy call来调用一个反事实地址（也就是说，调用一个与给定的已部署的反事实地址反事实实例化的契约）。

•proxycall：查找提供的反事实地址，并使用提供的calldata对结果地址进行调用。

•proxyDelegateCall：查找提供的反事实地址，并使用提供的CallData对结果地址进行委托调用。

关于注册合同的一个示例，请参见附录B。

6.3反事实对象

为了正式定义反事实对象，我们需要澄清继承自链上智能合约的术语中的歧义。使用solidity，单词“smart contract”可以指代1。合同的内容……}块，它将存储布局声明为一组变量、一组函数和一个构造函数，或者

2.前面的概念，但指定了构造函数参数，或

3.通过广播创建操作码的交易而创建的具有代码和存储（与外部拥有的帐户不同）的帐户；有时也称为“智能合约实例”。

在我们的术语中，“反事实对象”是指“智能合约”的第三个含义：具有代码和存储的帐户。在链上，共享代码的不同智能合约通过具有不同的地址来区分；对于反事实对象，我们要求它们具有ID属性。这可以防止不同反事实对象之间的重放攻击，并允许使用不同ID实例化多个反事实对象。因此，反事实对象抽象地定义为形式的属性元组：

C = (C*.*id*,*C*.*code*,*C*.*state*,*C*.*args)

其中c.code是合同的源代码，c.id是唯一标识符，c.state是反事实状态。我们使用C.code包含构造函数代码的约定；因此广播包含C.code和C.args的事务会在链上部署一个反事实的实例化契约。

6.3.1 API。因为反事实对象的代码c.code是一个任意的字节序列，所以有很多可能的方法可以实现它们。反事实实例化过程采用这个字节码和一些构造函数参数，并使用这些参数定义对象的初始状态c.state和两者的签名副本，以确定反事实地址。在实践中，我们对每一个反事实的对象都设定了一些期望：

1.它存储一个作为构造函数参数传递的唯一标识符。

2.由多签名钱包实例化。我们希望用C.code编写的构造函数包含将存储中的所有者变量分配给发送方的逻辑，除非另有规定。

3.在反事实对象的onchain部署情况下，它实现了一个包含multisig使用delegatecall操作码执行的精确代码的撤消功能，以发送锁定在状态存储中的状态。

4.它实现了在C.code中定义的超时长度。

5.通过将反事实地址映射到构成反事实对象依赖关系的nonce数字来参数化。我们需要以下条件来完成：对于映射中的每个键值对k:v，必须初始化具有反事实地址k的反事实对象，使nonce值正好等于v，并且（递归地）可以完成。

（六）是否定案，定案后，状态不再改变。它实现了一个isfinished函数，该函数可以检测合同是否已处于finished状态。

7.我们希望撤销功能在进行任何状态更新之前断言合同处于最终状态。

6.4基本协议

我们假设存在一个著名的地址注册表，一个实现keccak256抗冲突哈希函数的纯函数keccak256，一个接受ecdsa私钥和摘要并返回签名（由私钥签名的摘要）的纯函数ecsign（privkey，digest），以及一个接受函数的纯函数abiencode。n选择器和参数，并使用solidity契约abi将它们编码为正确的calldata。

6.4.1链上交易。我们假设参与者能够进行链上交易，例如部署合同和传输以太网络。为了简单起见，我们将这些连锁行动视为立即完成（即，我们不考虑它们恢复的风险）；在实践中，参与者必须在进行此类交易后等待充分的确认，然后再执行其他依赖于这些交易的最终性的行动。

6.4.2签署承诺。signcommitment协议提交一个multisig来执行某个操作。操作分为三种类型-发送以太网、呼叫和委托呼叫。在调用的情况下，将指定CallData。

4.4.4更新。此协议更新反事实对象的反事实状态。我们假设参与者在内存中存储每个反事实对象的源代码，这些对象的实例化是其一部分；也就是说，给定一个反事实地址c，存在一个反事实对象c，这样c=keccak256（c.dbytecode，multisig.address）。让这样C到C的映射称为查找，是keccak256的部分逆。

例如，CO提取功能将乙醚从多功能仪表盘中转移。为了让co.withdraw中的代码检查co的存储，必须调用co。



6.5反事实对象原语

在本节中，我们将指定一些常见的反事实对象，这些对象将在以后的示例中使用。

6.5.1根nonce。写为crootnonce，这个对象在它的nonce之外没有状态。它的目的是依赖于其他反事实对象，当它的nonce增加时，可以有效地“删除”任何依赖它的反事实对象。

c.状态=（c.nonce）

6.5.2付款合同。写为cpaymentchannel，这个对象只存储状态通道中的当事方到余额的映射，以及它的nonce。它只是作为一个原始的状态管理对象，大多数其他反事实对象将从中获取状态。

c.state=（c.nonce，c.balances）

6.5.3余额退款合同。这被写为cBalanceRefunder，并由阈值t和接收者p参数化。本合同的语义是：在t以上的multisig中的任何余额被分配给p（并可由p提取）。“BalanceRefunder”的名称来自这样一个事实：当一个参与者将ether放入一个状态通道时，这个对象被用来分配他对沉淀的乙醚。如果参与者存入乙醚，然后交易对手下线，在没有余额退款对象的情况下，存入的乙醚可能被扣押为人质。余额退款对象允许储户在这种情况下检索存储的乙醚。事实证明，完全相同的对象也可以用于即时取款。

所有对象还存储c.dependencies，它是反事实地址到非ce版本号需求的映射。

6.6协议

6.6.1创建新的状态通道。这是在一些参与方之间安全地创建新的状态通道的握手。目标是让每一方都有保证，在存入多个投资组合时，不会从中窃取他们的状态，同时在建立整个反事实结构之前，尽量减少对任何一个链交易的需要。

1。在安全通信层上交换公用地址。枚举p中每个参与方将使用的密钥集；这些密钥将作为所有者列在multisig中。所有各方都必须为要部署的multisig商定一个唯一标识符，以防止跨不同multisig的重播攻击。在这些示例中，我们将使用λp.id作为唯一标识符。

2。选择要部署multisig钱包的参与者。V=

## *randint*(0*,n*)

3.部署一个独特的多签名钱包。

***multisig*** := **Deploy**(*Pv.privkey,Multisig(P)*)

从这里开始，我们使用协议实例化、CommitWithDrawal和更新的约定，可以编写它们的第一个参数committed，在这种情况下，我们将该参数设置为multisig。

（四）建立反事实结构，允许第一笔状态抵押。（a）反事实地实例化根nonce契约。

*r* := **Instantiate**(*RootNonce*) (b)

反事实地例示一个余额退款合同。

*b*0 := **Instantiate** *BalanceRefund* recipient=*P*0 dependencies=*{r*:0*}*

（c）将状态分配给反事实余额退款合同。

**CommitWithdrawal**(*b*0)

请注意，在协议中的这一点上，除了空的multisig钱包之外，没有部署任何东西；对未部署的反事实实例化对象调用commitWithDrawal。

5.缴纳第一笔状态抵押。p0将S0存入多签名钱包。

6.建立反事实结构，允许下一个状态抵押。

对其余的参与方P1，…，PN−1重复步骤（b）中的过程。这个过程在结构上与（b）相同，但是这次由于根nonce已经被反事实地实例化，所以只需要更新它。对于k-th步骤（k=1，…，n−1），遵循以下过程以允许pk安全地存放sk：

1. 计算到目前为止存入的金额和各方的余额。

*k−*1

*Sk* := X *si*

*i*=0

*Bk* := *{Pi* : *si |* 0 *≤ i < k}*

(b）反事实地例示一个新的余额退款合同。

*bk* := **Instantiate***BalanceRefund* thresholdrecipient==*PSkk*

dependencies=*{r*:*k}*

1. 将状态抵押分配给该对象。

**CommitWithdrawal**(*bk*)

1. 反事实地例示一个新的支付合同，以存储已经存入的金额。

*pk* := **Instantiate***PaymentChannel*dependenciesbalances==*B{kr*:*k}*

e）将状态抵押分配给该对象。

**CommitWithdrawal**(*pk*) (f)

反事实地将根nonce增加到版本k。

**Update**(*r,k*)

7.进行第k个状态的存款。PK将sk存入多签名钱包。

8.重复第6步和第7步，直到所有参与者都进行了状态抵押。

9.实例化常规付款对象并为其分配状态。既然多签名钱包状态抵押中存在PNI=0−1si的余额，并且各方都已完成存款，现在我们可以删除最后一个余额退款对象bn−1并创建基本付款对象。

1. 反事实地实例化付款对象。

*B* := *{Pi* : *si |* 0 *≤ i < n}*

*p* := **Instantiate***PaymentChannel*dependenciesbalances==*B{r*:*n}* (b)

将状态抵押分配给此对象。

**CommitWithdrawal**(*p*) (c)

反事实地将根nonce增加到版本n。

**Update**(*r,n*)

我们将回滚协议的规范和安全证明的草图推迟到附录C中。直观地说，此协议是安全的，因为在任何时间点上，已存款的人拥有通过付款通道或平衡退款对象分配给自己的存款金额的所有权。我们通过更新根nonce或在链上存款，原子地在这个不变量持有的状态之间转移。

6.6.2在支付通道进行支付。在完成前一节中的所有步骤后，状态通道中的缔约方可以通过更新付款余额反事实对象开始付款，这只需要签署一条消息。因此，例如，如果余额从支付通道中的pi-having si更改为S0i，则双方将使用以下参数反事实地将支付合同更新为其余额的新值：

**Update** *i < n}*)

6.6.3在状态通道中安装新应用程序。假设状态通道的当事人决定玩扑克，而每个参与者决定把圆周率放进罐子里。我们将把CPoker定义为定义扑克规则的合同。让根nonce位于序列号k处。

1. 反事实地实例化扑克对象。

*o* := **Instantiate***Poker* dependenciespot=*{Pi*:*pi |*=0*{≤r*:*i<nk*+1*}},*

2.将部分状态抵押分配给对象。

**CommitWithdrawal**(*o*) 3.

使用减少的S创建新的付款对象。

*p* := **Instantiate***PaymentChannel*dependenciesbalances=*{P*=*i*:*{sri*:*−kp*+1*i}i}*

4。将部分状态抵押分配给对象。

**CommitWithdrawal**(*p*) 5.

反事实地将根nonce增加到k+1。

**Update**(*r,k* + 1)

从这里开始，玩游戏与在付款对象中付款完全相同，唯一不同的是两个对象用于更新的不同参数。

6.6.4增加状态抵押。通常情况下，状态通道中的一些参与者可能希望将更多的状态存储到状态通道中。例如，p0可能希望将w存入multisig。让根nonce位于序号q处，并且付款合同中的起始余额为

*{Pi* : *si |* 0 *≤ i < n}*. Let *S* = P*i si*.

1。建立反事实结构，允许状态抵押。

（a）反事实地例示余额退款合同。

阈值＝S

B：=实例化BalanceSunkRecipient=p0 Dependencies=r:q\_

（b）将部分状态抵押分配给该对象。

提交图纸（B）2.追加状态抵押。p0在多签名钱包中额外存入S00。

三。将反事实状态更新为“删除”余额退款合同。

（a）反事实地例示一个新的付款合同。

P：=实例化PaymentChannelBalancesDependencies=PI:S=I R:0Q≤+1I<N）

（b）将部分状态抵押分配给该对象。

committwithdrawal（p）（c）反事实地将根nonce增加到q+1。

更新（r，q＋1）

6.6.5立即退出。有些政党可能会将部分状态退出状态通道，这也是很常见的。例如，p0可能希望从状态抵押中提取w。让根nonce位于序号q处，付款合同中的起始余额为pi:si 0≤i<n。设S＝Pi-Si。

（一）建立反事实结构，允许即时撤诉。

（a）反事实地例示余额退款合同。

B：=实例化平衡退款阈值接收者==sp−0w

依赖项=r:q+1\_

（b）将部分状态抵押分配给该对象。

佣金支出（B）

（c）以减少的余额反事实地例示新的付款合同。

P：=实例化PaymentChannelBalances=依赖关系P0:S0−W=PR:IQ:S+1I 1≤I<N

（d）将部分状态抵押分配给对象。

committwithdrawal（p）（e）反事实地将根nonce增加到q+1。

更新（r，q＋1）

2.签署即时取款交易。P中的所有当事方均向multisig签署提款交易。此事务在链上执行。

签名承诺（多签名，（转让，p0，w））

6.7元通道

为了给出创建元通道的协议，我们必须引入两个新的基本协议和一个新的反事实对象原语。新的协议是对实例化和使用显式所有者集更新的修改，因为元通道参与者不共同拥有一个multisig。我们不追求的另一种设计选择是使用一个反事实的多签名钱包。

6.7.1用显式所有者实例化。该协议在注册表中使用基于ecrecover的显式消息身份验证来反事实地实例化反事实对象。

在争议期间，业主的任何成员都可以单方面广播包含承诺的适当格式的交易，导致注册处在cfaddress部署合同。

6.7.2与明确的所有者进行更新。我们假设具有显式所有者的反事实对象具有更新函数，该函数接受新状态和签名，验证签名是否签署新状态摘要，并执行更新。

6.7.4创建元通道的协议。我们假设Alice Ingrid State通道有一个multisig mav，一个根nonce带有反事实地址rav带有nonce k，一个支付通道位于地址pav带有余额alice:a，Ingrid:va。我们假设Ingrid Bob状态通道有多个sig mvb，根nonce有反事实地址rvb有nonce l，支付通道在地址pvb有余额Ingrid:vb，Bob:b。我们建立了余额为alice:a，bob:b的元通道支付通道，约束条件为a≥a，vb≥a，b≥b，va≥b（否则不可能实现无信任传递支付）。

1.反事实地例示Alice和Bob之间的付款合同。

pab：=用explicitTowners实例化a，b，v，paymentChannelOwnersBalances==alice，alid:a，bob，bob:b

2.反事实地例示Alice和Ingrid之间的代理合同。

金额=A+B

发送者=爱丽丝

xav：=实例化（mav，proxycontract intermediate=ingrid pcaddr=pab dependencies=rav:k+1）

3、承诺。

提交图纸（MAV、XAV）

4.反事实地举例说明余额减少的新支付合同。

p0av：=实例化mav，paymentchannelbalancesdependencies=alice:a=-ra，avingrid:k+1:v a−b

5、承诺。

提交图纸（mav，p0av）6.增加alice-ingrid根nonce。

更新（mav，k+1）

7.反事实地例示英格丽德和鲍勃之间的代理合同。

金额=A+B

发送器= BOB

xbv：=实例化（mbv，proxycontract intermediate=ingrid pcaddr=pab dependencies=rvb:l+1）

8、承诺。

提交图纸（MBV、XBV）

9.反事实地举例说明余额减少的新付款合同。

p0vb：=实例化mvb，paymentChannelBalancesDependences=Ingrid：=vb−rvba，：bobl+1:b−b

10、承诺。

提交图纸（MBV，P0VB）

11.增加Ingrid Bob根nonce。

更新（MVB，L+1）

12.反事实更新付款合同，以消除Ingrid的参与。

更新（pab，owners=alice，bob）之后，元通道支付通道可以通过简单更新pab的状态来使用，无需Ingrid的同意即可完成。

未来工作

7.1通用状态通道的能力

状态通道在其他可扩展性解决方案（如基础层切分或等离子）中没有限制。其中最重要的是，通道中的DAPP必须有一个已定义的参与者集。例如，不能向尚未在支付通道网络中的人发送通道化支付，而可以在Plasma中执行此操作。支付通道网络中的参与者集合虽然很大，但可以枚举，并且在所有现有的设计中，必须进行链上交易以扩大该集合。

这会阻止某些应用程序使用通道。例如，一个sha3赏金dapp（一个智能合约，它不信任地向能够提供sha3冲突的人支付赏金）的安全取决于“公众”能否参与。其逻辑是：如果存在对sha3的实际攻击，“任何知道它的人”都有通过收集赏金来揭示这一点的动机（等于赏金金额）；因此，赏金未被收集的事实提供了某种程度的保证，即实际上不存在对sha3的实际攻击。如果我们把它放到一个通道中，就会遇到两个问题。首先，可能希望领取奖金的人（例如，不满的国家安全局雇员）还没有进入某个通道，因此必须在领取奖金之前加入该通道，而且我们真的没有提供任何好处，而不是把奖金挂在链条上。第二，还没有进入通道的公众，如果不加入通道，就不能真正知道奖金是“可收藏的”。第三，即使有人设法加入通道并证实赏金是可收集的，他们也无法说服其他人相信这一事实；可能是，赏金只提供给一个人，试图证实它是公开的。

状态通道的另一个限制是时间粒度不能降低到根链的粒度以下。例如，如果以太坊数据块时间为5秒，并且我们非常乐观地假设数据块在创建后就以某种方式最终确定，那么就不能在通道内输入合同，从而使合同仅在1秒内有效（例如，有效时间为1秒的限制订单）。

7.2配套服务

我们补充了第三方服务的生态系统，这些服务将由状态通道用户使用。例如，在支付通道网络的背景下，中心是一个巨大的资本池，可以与最终用户打开许多小通道，允许用户通过中心路由支付。

Lightning Network为第三方引入了术语“望塔”，第三方持有终端用户最新状态（和签名消息）的副本，以代表终端用户作出响应。这使得终端用户可以长时间离线，代价是将安全性假设弱化为“发生争议时，至少有一个了望台会做出响应”。终端用户可以不信任地与望塔签订合同，向他们支付费用，或者在成功的纠纷发生时支付费用，或者在无法帮助解决纠纷的情况下惩罚他们。

在状态通道中，另一个第三方是保险提供者。这些保险公司的运作方式与传统保险公司很相似，他们收取一笔预付保险费，然后在发生欺诈（在本例中定义为发生在链外的交易）的情况下，分配一笔支付。由于griefing不可能消除，保险公司不是一个可信的第三方，有权声称一方在griefing，从而实施处罚；它所能做的只是获得一方正在griefing的私人知识。保险公司会收集Alice的签名信息，检查它们是否有效，然后将它们显示给Bob以尝试收集签名。如果Bob没有提供签名，那么保险人将提高保险费，或者将来拒绝与Bob签订合同。

7.3附加技术和子通道类型

还有其他有趣的技术和通道类型。

7.3.1链外时间见第一作者的谈话[13]。

7.3.2高级密钥管理类似于Coinbase等现有机构如何从使用高级密钥管理中获益（例如，冷/热钱包、支出限制），通道用户可以从使用高级密钥管理中获益。

7.3.3高吞吐量支付当我们将支付通道上的吞吐量要求提高到足够远时，即使是低成本的通道内支付也可能变得过高。例如，验证ECDSA签名（不是在链上，而是由通道参与者私下验证）的成本可能会变得太高。围绕散列揭示的技术可以缓解这种情况。

致谢

我们感谢Vitalik Buterin、Tom Close、Erik Bryn、Josh Stark、Nima Vaziri、Armani Ferrante、Lisa Eckey、Kristina主持人Akov'a、Yoichi Hirai、Sylvain Laurent和Alex Xiong的讨论和反馈。

1. Counterfactual conditional — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/wiki/ Counterfactual\_conditional,](https://en.wikipedia.org/wiki/Counterfactual_conditional) 2018.
2. A note on data availability and erasure coding. [https://github.com/ethereum/research/wiki/ A-note-on-data-availability-and-erasure-coding,](https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding) 2018.
3. Raiden specification. [http://raiden-network.readthedocs.io/en/stable/spec.html,](http://raiden-network.readthedocs.io/en/stable/spec.html) 2018.
4. Solidity documentation - units and globally available variables. [https://solidity.readthedocs.io/ en/develop/units-and-global-variables.html,](https://solidity.readthedocs.io/en/develop/units-and-global-variables.html) 2018.
5. Aaron van Wirdum. The history of lightning: from brainstorm to beta. [https://bitcoinmagazine. com/articles/history-lightning-brainstorm-beta/,](https://bitcoinmagazine.com/articles/history-lightning-brainstorm-beta/) 2018.
6. Andrew Miller et. al. Sprites and state channels: Payment networks that go faster than lightning. [https://arxiv.org/abs/1702.05812,](https://arxiv.org/abs/1702.05812) 2017.
7. Antoine Le Calvez. When the bitcoin dust settles. [https://medium.com/@alcio/ when-the-bitcoin-dust-settles-878f3431a71a,](https://medium.com/@alcio/when-the-bitcoin-dust-settles-878f3431a71a) 2018.
8. Augusto Hack. Add cooperative channel closing. [https://github.com/raiden-network/raiden/ issues/217,](https://github.com/raiden-network/raiden/issues/217) 2016.
9. Vitalik Buterin. A next generation smart contract and decentralized application platform. [https:](https://github.com/ethereum/wiki/wiki/White-Paper)

[//github.com/ethereum/wiki/wiki/White-Paper,](https://github.com/ethereum/wiki/wiki/White-Paper) 2013.

1. Adam Back et al. Enabling blockchain innovations with pegged sidechains. [https://blockstream. com/sidechains.pdf,](https://blockstream.com/sidechains.pdf) 2014.
2. Rusty Russell et. al. Lightning network in-progress specifications. [https://github.com/ lightningnetwork/lightning-rfc,](https://github.com/lightningnetwork/lightning-rfc) 2018.
3. Jeff Coleman. State channels. [http://www.jeffcoleman.ca/state-channels/,](http://www.jeffcoleman.ca/state-channels/) 2015.
4. Jeff Coleman. Universal hash time. [https://www.youtube.com/watch?v=phXohYF0xGo,](https://www.youtube.com/watch?v=phXohYF0xGo) 2015.
5. Jeremy Longley and Oliver Hopton. Funfair technology roadmap and discussion. [https://funfair. io/wp-content/uploads/FunFair-Technical-White-Paper.pdf,](https://funfair.io/wp-content/uploads/FunFair-Technical-White-Paper.pdf) 2017.
6. Jorge Izquierdo and Manuel Araoz. EIP 897: ERC delegateproxy. [https://eips.ethereum.org/ EIPS/eip-897,](https://eips.ethereum.org/EIPS/eip-897) 2018.
7. Martin K¨oppelmann. How offchain trading will work. [https://forum.gnosis.pm/t/ how-offchain-trading-will-work/63,](https://forum.gnosis.pm/t/how-offchain-trading-will-work/63) 2015.
8. Michael Lewis-Beck, Alan E Bryman, Tim Futing Liao. Counterfactual. the sage encyclopedia of social science research methods.
9. Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system. [https://bitcoin.org/bitcoin. pdf,](https://bitcoin.org/bitcoin.pdf) 2008.
10. Oleskii Matiiasevych. How is the address of an ethereum contract computed? [https://ethereum.](https://ethereum.stackexchange.com/questions/760/how-is-the-address-of-an-ethereum-contract-computed)

[stackexchange.com/questions/760/how-is-the-address-of-an-ethereum-contract-computed,](https://ethereum.stackexchange.com/questions/760/how-is-the-address-of-an-ethereum-contract-computed) 2016.

1. Henning Pagnia and Felix C. G¨artner. On the impossibility of fair exchange without a trusted third

party. [https://www.cs.utexas.edu/˜shmat/courses/cs395t\_fall04/pagnia.pdf,](https://www.cs.utexas.edu/~shmat/courses/cs395t_fall04/pagnia.pdf) 1999.

1. Paul Grau. Lessons learned from making a chess game for ethereum. [https://medium.com/ @graycoding/lessons-learned-from-making-a-chess-game-for-ethereum-6917c01178b6,](https://medium.com/@graycoding/lessons-learned-from-making-a-chess-game-for-ethereum-6917c01178b6) 2016.
2. Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. [https://plasma. io/plasma.pdf,](https://plasma.io/plasma.pdf) 2017.
3. Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. [https://lightning.network/lightning-network-paper.pdf,](https://lightning.network/lightning-network-paper.pdf) 2016.
4. Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Daniel Malinowski. Perun: Virtual payment hubs over cryptographic currencies. [https://eprint.iacr.org/2017/635,](https://eprint.iacr.org/2017/635) 2017.
5. Stefan Dziembowski, Sebastian Faust, Kristina Hostakova. Foundations of state channel networks. [https://eprint.iacr.org/2018/320,](https://eprint.iacr.org/2018/320) 2018.
6. Vitalik Buterin. The triangle of harm. [https://vitalik.ca/general/2017/07/16/triangle\_of\_ harm.html,](https://vitalik.ca/general/2017/07/16/triangle_of_harm.html) 2017.
7. Vitalik Buterin. Account abstraction for main chain. [https://github.com/ethereum/EIPs/issues/ 859,](https://github.com/ethereum/EIPs/issues/859) 2018.
8. Gavin Wood. Ethereum: A secure decentralized generalised transaction ledger. [https://ethereum.](https://ethereum.github.io/yellowpaper/paper.pdf)

[github.io/yellowpaper/paper.pdf,](https://ethereum.github.io/yellowpaper/paper.pdf) 2018.

反事实术语

在哲学中，反事实条件[1]是形式“如果x那么y”的陈述，但如果x不是实际情况，则使先行词“反事实”。换言之，这是一种形式的陈述，“如果是x，那么y会是，即使x不是”。例如，“如果奥斯瓦尔德没有射杀肯尼迪，那么其他人也会有”这样的说法是反事实的，因为前因是错误的。一个反事实条件不能被评价为一个真理功能条件，因为一个带有错误前因的真理功能条件是真的[17]；在奥斯瓦尔德·肯尼迪的例子中，它表达了两个事件之间的因果关系，而不是逻辑蕴涵。

在区块链中，我们以不同的意义使用这个术语，我们在5.1中简要描述了这个术语，我们将在本附录的其余部分中更全面地定义它。我们首先给出反事实术语的一般定义，然后将这个定义限制在我们在6个状态通道中使用的案例中。

1初步定义

首先，我们从白皮书[9]中概述了以太坊状态和以太坊状态转换函数的定义。以太坊状态是帐户到帐户状态（nonce、以太坊余额、合同代码和存储）的映射。事务是从外部拥有的帐户发送的签名消息。状态转换函数是一个函数apply（s，tx）→s0，它接受一个启动状态和一个事务，并输出一个新的状态，这是将事务应用到该状态的结果。演员和能力。我们将外部拥有的帐户所有者建模为具有某些有限功能的参与者：他们可以保守秘密，不能承诺彼此合作，不能反转哈希函数等。

B状态转换。通过考虑函数apply（·，tx）：t→t，可以将以太坊事务视为状态转换，其中t是一组有效的以太坊状态，是特殊的错误状态。但是，对于我们的目的来说，这太过约束；我们希望定义例如“事务”例如“帐户0x407…”将5个以太网传输到帐户0x00B…”，而不限制发送帐户必须位于的当前位置。因此，我们将状态转换定义为T上的关系，以捕获这个直观的概念。此定义还包括“块编号增加”作为状态转换，包括可能不适合于一个事务的状态转换，并包括可以选择事务来Fufil它们的状态转换（例如，一个帐户将以太网传输到两个帐户中的任意一个）。

2核心定义

让p成为一个参与者，x是一个状态转换，s是一个状态（被认为是当前的区块链状态）。反事实（p，x）成立的一个必要条件是p能够适用

x到s。

可达性：P能够执行导致状态为S0的区块链操作序列，从而X（S，S0）持有

除P:不可撤销：除P之外，没有一组参与者能够执行一系列区块链操作，而P没有执行某些操作，从而导致状态为S00，使得P不能将X应用于S00，这也是必要的。

如果这两个条件都满足，我们就说反事实（p，x）适用于s。

四十四

举个例子。考虑图A17中的智能合约，让我们假设Alice调用了Fund，其中常量bobaddr=0xb7…是由Bob控制的帐户，Bob不知道y的预映像。让x=5 ether从合同转移到Bobaddr。

四十五

B A非示例。考虑修改智能合约，我们将ens.resolve（“bob.eth”）替换为msg.sender，如图b所示。还假设除了将preimage传输给bob之外，Alice还将其广播到公共场所。

5威胁模型

每个参与者操作的威胁模型指定了每个参与者拥有的能力，因此通过“不执行某些操作”一句影响反事实术语的定义。虽然这还没有影响到我们迄今为止的任何例子，但在与状态通道实际相关的例子中是很重要的；在我们的有界经济风险和有界griefing因子的“违约”模型中，在支付通道中（假设pi的余额bi为i=1,2），我们可以说反事实（p1，x）反事实（p2，x），其中x是“b1，ether i”S转入P1账户，B2醚转入P2账户”。在这种情况下，第2条中的操作包括“p1无法响应过时的数据”（这是一种未能执行操作的行为）。但是，如果一个参与者是在完全公平的威胁模式下行事的，那么情况就不再是这样了，因为支付通道中可能存在欺诈。

6个状态通道

在一个含有p1，…pn的状态通道中，对于具有正效用的x，我们使用“反事实x”作为简写，表示“对于所有i，都有反事实（pi，x）的常识”。这与我们在一个通道中确定的x的定义相对应。

7个反事实名词

名词n的反事实n是一个简写，意思是大致反事实x，其中x是n放在区块链上。因此，严格的定义取决于n。

例如，如果n是“一个特定的智能合约”，则存在一个反事实合约（或我们一直称之为反事实实例化的合约，为了清晰起见），其意义是利益相关方（即状态通道中的各方）可以单方面实施与n（威胁模型的模件）相同的后果。

就“执行区块链操作的权利”通常用名词（例如，“ERC-721代币”或“Cryptokitty”）来描述而言，也可以构建支持这些反事实版本的合同。

上述评论的一个例外是“反事实地址”，它与“以太坊地址”不对应（类比除外，即它们具有相同的用途）。

B登记合同

四十八

C融资协议：回滚和安全证明

让我们从p0的角度画出一个简短的安全证明。我们希望证明，在他存款后的每一点（即步骤5），无论其他人做什么，他都能从通道中收回减去少量费用的S0醚。在步骤5之前（即步骤4c之后），已将阈值为0且接收者p0的余额退款提交给。如果在第5步之后到第6步之前，其他参与者停止响应，p0可以实例化r和b0，然后在b0上调用withdraw，将multisig中的所有以太发送回自己。由于b1和p1还不能最终确定，p0完成这项任务的能力在6e的第一次迭代之后仍然保持不变。6F后，p0可通过实例化p1来回收S0醚。在另一个迭代过程中，每次执行步骤6f时，都会有类似的参数。此外，由于p0从未授权从multisig中退出，因此在pi和b0上退出的必要调用不会失败。

我们还必须考虑signcommitment的公平交换失败，例如在步骤6f中（回顾6f执行一个更新协议，该协议作为子协议包含signcommitment的实例）。这意味着p0已经给出了他的承诺，允许6f成功完成，但还没有收到其他承诺，所以不能保证他可以使6f发生。在这种情况下，在等待一些主观超时之后，p0应该实例化r并尝试将其最终确定为版本0。这要么成功，要么失败，因为它最终被定为版本1。在这两种情况下，p0都可以分别利用b0或p1的抽出功能来回收S0醚。K>0的其他方pk的证明类似。但是，如果r在pk存款后j<k在nonce j发布并最终确定，那么pk将亏损；因此，pk必须挑战并将nonce更新为最新的可用值，即≥k。

4贡献

注意到报纸有问题吗？请在https://github.com/counterfacture/paper上向承载此文档的github存储库提交请求。

1. Contact L4 Research at research@l4v.io [↑](#footnote-ref-1)