Arbitrum:可扩展的私有智能合约

Harry Kalodner、Steven Goldfeder、Xiaoqi Chen、S. Matthew Weinberg 和
Edward W. Felten,普林斯顿大学

https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner

这篇论文包含在第 27 届 USENIX 安全研讨会论文集中。

2018 年 8 月 15-17 日 · 美国马里兰州巴尔的摩

978-1-939133-04-5

第 27 届 USENIX 安全研讨会论文集的公开访问由 USENIX 赞助。

Arbitrum:可扩展的私有智能合约

哈利·卡洛德纳
普林斯顿大学

史蒂文戈德费德
普林斯顿大学

Xiaoqi Chen
普林斯顿大学

S. Matthew Weinberg
普林斯顿大学

Edward W. Felten 普林斯顿大学

抽象的

我们介绍了 Arbitrum,这是一种支持智能合约的加密货币系统,不受之前系统(如以太坊)的可扩展性和隐私性的限制。Arbitrum 与 Ethereum 一样,允许各方通过使用代码指定实现合约功能的虚拟机 (VM) 的行为来创建智能合约。 Arbitrum 使用机制设计来激励各方在链下就 VM 的行为达成一致,因此 Arbitrum 矿工只需验证数字签名即可确认各方已就 VM 的行为达成一致。在各方无法在链下达成一致的情况下,Arbitrum 仍然允许诚实的各方在链上推进 VM 状态。

如果一方试图对 VM 的行为撒谎,验证者(或矿工)将通过使用利用 Arbitrum 虚拟机架构特性的高效基于挑战的协议来识别和惩罚不诚实的一方。以这种方式将 VM 行为的验证移至链外可显着改善可扩展性和隐私性。我们描述了 Arbitrum 的协议和虚拟机架构,并展示了一个工作原型实现。

1 简介

数字货币与智能合约的结合是天作之合。加密货币允许各方直接转移数字货币,依靠分布式协议、密码学和激励措施来执行基本规则。智能合约允许各方创建虚拟的可信第三方,这些第三方将根据任意商定的规则行事,从而允许创建复杂的多方协议且交易对手风险极低。通过在加密货币之上运行智能合约,人们可以在合约中编码货币条件和惩罚,这些将由底层共识机制强制执行。

以太坊 [31] 是第一个支持图灵完备状态智能合约的加密货币,但它在可扩展性和隐私方面受到限制。以太坊要求每个矿工模拟每个合约执行的每个步骤,这是昂贵的并且严重限制了可扩展性。它还要求每个合同的代码和数据都是公开的,没有某种类型的隐私覆盖功能,这会增加其自身的成本。

1.1 仲裁

我们介绍了 Arbitrum 的设计和实施,这是一种解决这些缺点的智能合约新方法。仲裁合约对于验证者来说管理起来非常便宜。 (如下所述,我们通常使用术语验证者来指代底层共识机制。例如,在比特币协议中,比特币矿工是验证者。)如果各方根据激励行事,Arbitrum 验证者只需要验证一些数字每份合同的签名。即使当事方违背他们的动机,Arbitrum 验证者也可以有效地裁定关于合约行为的争议,而无需检查合约对多个指令的执行情况。 Arbitrum 还允许合约私下执行,只发布合约状态的(可销售的)哈希值。

在 Arbitrum 中,各方可以将智能合约实现为对合约规则进行编码的虚拟机 (VM)。 VM 的创建者为 VM 指定一组管理器。Arbitrum 协议提供了一种任意信任的保证:任何一个诚实的管理者都可以强制虚拟机按照虚拟机的代码行事。对 VM 结果感兴趣的各方可以自己作为

经理或指定他们信任的人代表他们管理 VM。对于许多合同,经理人的自然集合在实践中会非常小。

依赖管理器,而不是要求每个验证者模拟每个 VM 的执行,允许 VM 的管理器以更低的成本提升 VM 的状态

给验证者。验证者只跟踪 VM 状态的散列,而不是完整状态。 Arbitrum 为管理人员创造了激励机制,让他们在带外就 VM 将做什么达成一致。任何由所有管理者认可的状态改变（并且不超支 VM 的资金)将被验证者接受。如果与激励相反,两个管理者对 VM 将做什么有不同意见,则验证者使用二分法将分歧缩小到执行单个指令,然后一个管理者提交该指令的简单证明验证者可以非常有效地检查执行情况。错误的经理向验证者支付巨额罚款,以阻止分歧。

各方可以向 VM 发送消息和货币,而 VM 本身可以向其他 VM 或其他方发送消息和货币。 VM 可能会根据收到的消息采取行动。 Verifier 跟踪 VM 收件箱的哈希值。

Arbitrum VM 和协议的架构旨在使验证者尽可能快速和简单地解决争议。设计细节将在本文后面出现。

Arbitrum 大大降低了智能合约的成本。如果参与者根据他们的激励行事,那么验证者将永远不必模仿或验证任何 VM 的行为。在这种情况下,验证者的唯一职责是进行简单的簿记以跟踪货币持有量、消息收件箱的哈希值以及每个 VM 的单个哈希状态值。如果参与者的行为不合理,可能需要验证者做少量的额外工作,但验证者将以牺牲非理性方为代价来（过度)补偿这项工作。

作为先前原则的必然结果,Arbitrum VM 可以是私有的,从这个意义上说,VM 可以创建并执行到完成而不会泄露 VM 的代码或其执行,除了它发送的消息和付款的内容和时间,并且(saltable)其状态的哈希值。 VM 的任何管理员都必须能够透露有关该 VM 的信息,但是如果管理员想要维护 VM 的隐私,他们可以这样做。

Arbitrum 与共识无关,这意味着它假定存在发布交易的共识机制,但 Arbitrum 的设计同样适用于任何共识机制,包括单个中心化发布者、基于仲裁的共识系统或 Nakamoto 共识用于比特币 [26]。

此外,现有的智能合约系统可以作为这种共识机制,假设它可以将 Ar bitrum 的规则编码为智能合约。在本文中,我们将共识实体或系统称为验证者（并将该共识系统中的参与者称为验证者）。

1.2 论文结构

在本文的其余结构如下。在第 2 节中,我们讨论了有效实施智能合约的困难,并提出了参与困境,这是一种关于参与博弈的新理论结果,表明一种激励智能合约验证的方法可能行不通。在第 3 节中,我们描述了 Arbitrum 的方法,在第 4 节中,我们提供了 Arbitrum 的协议和虚拟机架构的更多细节,它们共同允许对实施智能合约的虚拟机的操作进行更有效和隐私友好的验证。第 5 节描述了我们对 Arbitrum 的实施,并提供了一些性能基准以及证明和区块链交易的大小。第 6 节调查相关工作,第 7 节总结本文。

2 为什么扩展智能合约很困难

以通用和有效的方式支持智能合约是一个难题。在本节中,我们调查了一些现有方法的缺点。

2.1 验证者的困境

实现智能合约 VM 最明显的方法是让加密货币系统中的每个矿工模拟每个 VM 执行的每个步骤。这具有简单的优点,但它对可扩展性施加了严格的限制。

验证 VM 执行的高成本可能表现为验证者的困境 [22]。由于涉及由 VM 执行代码的交易的验证成本很高,因此本应验证这些交易的一方有动机通过接受交易而不验证它们来搭便车,希望阻止 (1) 不当行为由其他方进行验证,或 (2) 任何差异将不会被其他潜在验证者检测到,因为他们也不执行验证。这可能会导致一种平衡,在这种平衡中,一些交易在很少或没有验证的情况下被接受。

相反,在所有矿工都诚实地进行验证的情况下,一个矿工可以通过包括一个耗时的计算来利用这一点,这将花费其他矿工大量的时间来验证。

当所有其他矿工都在进行验证时,包含这种计算量大的交易的矿工可以在挖掘下一个区块时抢占先机,从而获得不成比例的收集下一个区块奖励的机会。存在这种困境是因为验证 VM 执行的成本很高。

2.2 参与困境

一种扩展验证的方法（例如在 TrueBit [30] 中使用)依赖于参与游戏,这是一种机制设计方法,旨在诱导有限但足够数量的参与方来验证每个 VM 的执行。这些系统面临我们所说的参与困境,即如何防止女巫攻击,在这种攻击中,可能诚实或不诚实的单个验证者声称自己是多个验证者,这样做可以将其他验证者赶出系统.

2.3 参与游戏

在本节中,我们证明了基于参与博弈的方法的新形式障碍。这个想法是玩家将“参与”一个代价高昂的过程。考虑以下游戏：

- 有n 个玩家,他们可以支付1 参与。
- 参与玩家 i 选择的 Sybils 数量si ⩾ 1。非参与玩家设置si = 0。
- 玩家i 收到奖励si · f(Σj sj),其中f : N → R+是奖励函数。

在本文的上下文中,将参与视为“验证计算”。验证计算需要花费一些费用,但是一旦你验证了它,你就可以声称已经从任何数量的额外女巫身上免费验证了它,而这些女巫与“真正的”验证者没有区别。然后目标将是设计一个参与游戏（即奖励函数 f(·)），使得在均衡状态下,没有玩家对 Sybil 有任何激励,并且期望数量的玩家参与,因此验证者的表观数量等于作为验证者的独立玩家的实际数量。

TrueBit 的作者正确地观察到函数族fc(m) = c · 2 非常适合参与游戏。具体来说,对于参与玩家的任何目标[illegible]的（达到对称性)纯纳什均衡,其中每个玩家都有si ∈ {0,1},并且恰好k玩家参与。事实上,一个更强大的属性成立：它始终是任何玩家设置si ⩽ 1 的最佳响应!1我们将此类奖励函数称为一次性Sybil-proof定义）。最初使参与游戏看起来像是可验证智能合约的有前途的途径,因为存在一次性 Sybil 奖励功能。

然而,之前的工作未能解决的一个问题是智能合约验证是一个重复的游戏。在重复博弈中,还有许多其他均衡

1也就是说,无论其他玩家做什么,玩家i都是严格的设置si = 1 比si > 1 更快乐。

不要投射到他们的一次性变体的纳什均衡上。对于直觉,回想一下经典的囚徒困境：2如果游戏只进行一次,那么唯一的纳什均衡就是两个玩家都背叛（而背叛甚至是一种严格的占优策略）。然而,在重复的囚徒困境中,还有许多其他均衡,包括著名的以牙还牙和冷酷触发策略 [29]。

我们在附录 A 中讨论了重复博弈的正式模型（这是标准的,但不是本文的重点）。但重点是重复博弈允许玩家牺牲现在以节省未来。例如,以下是 f(m) = (4.5)· 2 −m 的重复参与博弈的均衡。

玩家一使用策略：在所有回合中设置s1 = 2。
玩家 i > 2 在所有回合中设置si = 0。玩家 2 使用策略：如果在前两轮中的任何一轮中， Σj=2 sj ⩽ 1,则设置s2 = 1。否则,设置s2 = 0。

请注意,除了玩家 1 之外的所有玩家肯定是最佳响应者。他们目前的效用为零（因为玩家 1每轮都设置s1 = 2,因此他们都设置si = 0）。如果他们反而参加任何一轮,他们将获得负效用。另一方面,玩家 1 也是最好的回应!这是因为如果他们在任何一轮中减少了 Sybils 的数量,这将导致玩家 2 参与接下来的两轮（附录中的错误证明）。

请注意,这种均衡一点也不不自然：参与者 > 1 只是对前几轮市场的情况做出反应。玩家 1 领先游戏一步,并意识到无论如何都会有两个参与者处于均衡状态,因此玩家 1 与其分享奖励不如成为所有人。其实这不是第i个奖励函数c · 2特有的性质

, 但任何奖励功能。

定理 1.每个 One-Shot Sybil-Proof 参与游戏都承认只有一个玩家参与的纳什均衡。

在附录 A 中,我们提供了定理 1 的证明,以及对可能的开箱即用防御的讨论。
这些防御措施在技术上似乎具有挑战性（也许不可能)实施,但我们并没有证明这一点。然而,模拟确实表明实施这些防御的成本与单个玩家的计算能力呈线性关系,这可能会使它们变得不切实际（如果它们确实可行的话）。

因此,基于此类参与游戏的方法,包括先前工作 [30、32] 中提出的方法,似乎无法防止破坏智能合约验证信心的女巫攻击。

2有两个玩家。如果他们都背叛,则双方都获得收益 1,如果他们都合作,则获得收益 2。如果一个合作而另一个背叛,则背叛者得 4,合作者得 0。

3 决策系统概述

在本节中,我们将概述 Ar bitrum 的设计。

3.1 角色

Arbitrum 协议和系统中有四种类型的角色。

Verifier是验证交易有效性并发布已接受交易的全局实体或分布式协议。验证者可能是中央实体或分布式多方共识系统,例如分布式仲裁系统、中本聪共识协议 [26] 中的全球矿工集合,或者本身是现有加密货币的智能合约。由于 Arbitrum 的设计对于使用哪种类型的共识系统是不可知的,为了简洁起见,我们对正在运行的任何共识系统都使用单数术语 Verifier。

密钥是协议中可以拥有货币并提议交易的参与者。密钥由公钥（的散列)标识。它可以通过使用相应的私钥对交易进行签名来提议交易。

VM（虚拟机)是协议中的虚拟参与者。根据 Arbitrum 虚拟机 (AVM) 规范,每个 VM 都有定义其行为的代码和数据,该规范包含在本文的扩展版本中。与密钥一样,VM 可以拥有货币并发送和接收货币和消息。虚拟机是由一种特殊的交易类型创建的。

VM的管理者是监视特定 VM 的进度并确保 VM 的正确行为的一方。创建 VM 时,创建 VM 的事务会为 VM 指定一组管理器。

管理器由其公钥（的哈希值)标识。

3.2 虚拟机的生命周期

Arbitrum VM 是使用特殊事务创建的,它指定了 VM 的初始状态哈希、VM 的管理器列表和一些参数。如下所述,状态散列表示对 VM 状态（即其代码和初始数据)的加密承诺。可以同时存在任意数量的 VM,通常具有不同的管理器。

创建 VM 后,管理人员可以采取措施使该 VM 的状态发生变化。 Arbitrum 协议提供了一种任意信任的保证:任何一个诚实的管理者都可以强制 VM 的状态变化与 VM 的代码和状态一致,即根据 AVM 规范是有效的执行。

断言指出,如果某些先决条件成立,VM 的状态将以某种方式改变。如果 (1) 断言的先决条件成立,(2) VM 未处于停止状态,并且 (3) 断言不会花费比 VM 拥有的更多的资金。断言包含 VM 新状态的哈希值和 VM 采取的一组操作,例如发送消息或货币。

一致断言由该 VM 的所有管理器签署。如果一致断言是合格的,它会立即被验证者接受为 VM 的新状态。

有争议的断言仅由一名经理签署,并且该经理会在断言上附加货币保证金。如果有争议的断言是合格的,则该断言由验证者发布为未决。如果超时期限过去后没有任何其他管理器对未决断言提出质疑,则验证者接受该断言并且断言者取回其保证金。如果另一位经理质疑悬而未决的断言,挑战者会存入一笔货币保证金,两位经理将参与二分协议,以确定他们中的哪一位在说谎。骗子会损失押金。

如上所述,VM 继续推进其状态,直到 VM 达到停止状态。在这一点上,不可能有进一步的状态变化,验证者和管理者可以忘记虚拟机。

3.3 二分协议

当一位经理提出了一个有争议的断言,而另一位经理对该断言提出质疑时,二分法就开始了。两位经理都将存入货币保证金。

在二分协议的每一步,断言者将断言二分为两个断言,每个断言涉及 VM 计算步骤的一半,挑战者选择它想挑战哪一半。

他们继续这个二分协议,直到关于单个步骤的断言（即 VM 执行一条指令)受到挑战,此时断言者必须提供验证者可以检查的一步证明。

如果断言者提供了正确的证明,则断言者获胜;否则挑战者获胜。赢家取回押金,并拿走输家一半的押金。失败者押金的另一半交给验证者。

对分协议是通过断言者和挑战者进行的一系列区块链交易来执行的。在协议中的每一点,一方都有一个有限的时间间隔来进行下一步行动,如果他们未能在截止日期前采取有效行动,则该方将失败。

验证者只需要检查移动的表面有效性,例如,检查将一个断言分成两个一半大小的断言是否有效,因为这两个结果断言确实组成了原始断言。

3.4 验证者的角色

回想一下,验证者是一种机制,它可以是一个具有多个参与者的分布式协议,用于验证交易并发布经过验证的交易。

除了存储关于每个 VM 的一些参数（例如其管理器列表)之外，Verifier 还会跟踪有关每个 VM 随时间变化的三项信息:VM 状态的哈希值、VM 持有的货币数量以及VM 收件箱的哈希,其中包含发送到 VM 的消息。 VM 的状态是先进的,对应于 VM 程序的执行,通过 Verifier 接受 VM 的管理器所做的断言。

被挑战的断言不能被验证者接受,即使断言者赢得了挑战游戏。相反,断言在受到挑战时是“孤立的” 。3 挑战游戏结束后,服务器可以选择重新提交相同的断言,但如果断言不正确,这显然是愚蠢的。

协议设计确保单个诚实的管理者始终可以通过挑战来防止不正确的断言被接受。 （如果其他人在诚实的管理者可以这样做之前挑战断言,断言仍然不会被接受,即使挑战者是恶意的。)诚实的管理者还可以通过提出有争议的断言来确保 VM 取得进展,除了恶意的管理者可以通过强制执行一个它知道会丢失的二分协议,以一半的押金为代价,在一个二分协议的持续时间内延迟进度。

3.5 关键假设和权衡

Arbitrum 允许创建 VM 的一方指定该 VM 的代码、初始数据和一组管理器。验证者确保 VM 不能创建货币,而只能花费发送给它的货币。因此,不知道 VM 状态或不喜欢 VM 代码、初始数据或一组管理器的一方可以安全地忽略该 VM。假设当事方同意 VM 已正确初始化并且他们在其正确执行中有一定的利益时,他们只会关注 VM。任何一方都可以自由创建模糊或不公平的 VM;其他方可以随意忽略它。

根据 Arbitrum 的任意信任假设,各方应

如果他们信任至少一个 VM 的管理器,则只依赖 VM 的正确行为。获得信任的经理的一种方法是自己担任经理。我们还预计,一个成熟的 Arbitrum 生态系统将包括经理即服务业务,这些业务有动机维护诚实的声誉,并且可能额外承担未能履行诚实经理职责的法律责任。

Arbitrum 做出的一个关键假设是,管理者将能够在指定的时间窗口内向验证者发送质询或响应。在区块链设置中,这意味着能够在该时间内将交易包含在区块链中。尽管

至关重要的是,这种假设在加密货币中是标准的,并且可以通过在terval（这是每个 VM 的可配置参数)中扩展挑战来减轻风险。

有两个因素有助于降低针对诚实管理者的拒绝服务攻击的吸引力。首先,如果 DoS 攻击者不能确定阻止诚实的管理者提交挑战,而只能将挑战的概率降低到 p,招致惩罚的风险可能仍然足以阻止错误断言,特别是如果存款金额增加。其次,因为每个管理器仅由公钥标识,管理器可以使用复制来提高其可用性,包括使用攻击者不知道其存在或位置的“秘密”副本

万斯。

最后,一个有动机的恶意管理器可以通过不断挑战关于其行为的所有断言来无限期地停止 VM。攻击者将至少损失每笔押金的一半,而每次此类损失只会延迟 VM 的进度,延迟时间仅为运行二分协议一次所需的时间。我们假设 VM 的创建者会将 VM 的存款金额设置得足够大以阻止这种攻击。

3.6 好处

可扩展性。也许 Arbitrum 的关键特性是它的可扩展性。管理人员可以无限期地执行一台机器,只需支付微不足道的交易费用,这些费用很小,而且与他们运行的代码的复杂性无关。如果参与者遵循激励,所有的主张应该是一致的并且永远不会发生争议,但即使确实发生争议,验证者也可以有效地解决它,而诚实方的成本很小（但不诚实方的成本很高）。

隐私。 Arbitrum 的模型非常适合私人智能合约。如果没有争议,则不会向验证者透露 VM 的内部状态。此外,如果各方根据他们的动机执行协议,则不应发生争议。即使在有争议的情况下,验证者也只会得到有关 ma 的单个步骤的信息

3我们拒绝了如果断言者赢得挑战游戏则允许断言被接受和执行的替代方案,以防止恶意挑战者故意输掉挑战游戏以接受虚假断言的攻击。我们选择的设计确保故意输的挑战者会将一半的押金损失给矿工（另一半给挑战者可能与之串通的断言者）,但恶意挑战者将无法强制接受一个无效的断言。

chine 的执行,但机器的绝大部分状态对验证者来说仍然是不透明的。在 4.4 节中,我们展示了我们甚至可以通过以隐私保护的方式进行一步验证来消除这种泄漏。

Arbitrum 的隐私并非巧合,而是其模型的直接结果。由于 Arbitrum 验证者（例如,Nakamoto 共识模型中的矿工)不运行 VM 的代码,因此他们不需要看到它。相比之下,在以太坊或任何试图实现“全局正确性”的系统中,所有代码和状态都必须公开,以便任何人都可以验证它,而这种模式从根本上与私人执行不一致。

灵活性。一致断言提供了很大的灵活性,因为管理人员可以选择将机器重置为他们希望的任何状态,并采取他们想要的任何行动（前提是机器有资金） 即使机器代码认为它们是无效的。这需要经理们的一致同意,所以如果任何一个经理是诚实的,只有当结果是一个诚实的经理会接受的结果时才会这样做 比如关闭一个由于故障而进入不良状态的虚拟机。软件错误。

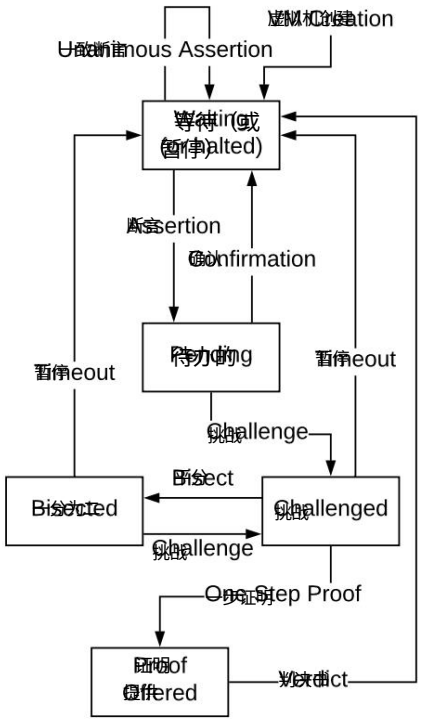

图 1:管理 Arbitrum 协议中每个 VM 状态的状态机概览。

4 决策设计细节

本节更详细地描述了 Arbitrum 协议和虚拟机设计。该协议管理公共进程,该进程管理和推进整个系统和每个 VM 的公共状态。VM 架构管理在 VM 中运行的 Arbitrum 程序的语法和语义。

4.1 仲裁协议

Arbitrum 使用简单的加密货币设计,增加了允许创建和使用虚拟机 (VM) 的功能,它可以体现任意功能。 VM 是在 Arbi trum 虚拟机架构上运行的程序,如下所述。

Arbitrum 协议识别两种参与者:密钥和虚拟机。密钥由公钥（的加密散列)标识,如果该操作由相应的私钥签名,则认为参与者已采取该操作。另一种参与者是虚拟机,它通过执行代码来采取行动。任何演员都可以拥有货币。 Arbitrum 跟踪每个参与者拥有多少货币。

VM 是使用特殊事务类型创建的。 VM 创建交易指定了 VM 初始状态的加密哈希,以及 VM 的一些参数,例如挑战期的长度、各方的各种付款和存款的金额

将随着协议的进一步执行而生成,以及 VM 管理器的列表。

对于每个 VM,验证者跟踪该 VM 的散列状态、VM 持有的货币数量及其收件箱的散列。可以通过有关 VM 执行的断言来更改 VM 的状态,断言指定 (1) VM 执行的指令数,(2) 执行后 VM 状态的散列,以及 (3) VM 采取的任何操作例如付款。此外,断言声明了一组前提条件,这些前提条件必须在断言之前为真,这些前提条件指定（1）执行前 VM 状态的散列，(2)断言包含在块中的时间的上限和下限,(3) VM 持有的余额的下限,以及 (4) VM 收件箱的哈希值。Arbitrum 的规则规定了在什么条件下可以接受 as sertion。如果断言被接受,则 VM 被视为已更改其状态,并按照断言的规定采取了公开可见的操作。

在最简单的情况下,断言由所有 VM 的管理器签署。在这种情况下,如果断言符合条件,即如果 (1) 断言的前提条件与 VM 的当前状态匹配,(2) VM 未处于暂停状态,并且 (3) VM 有足够的资金来支付断言指定的任何款项。一致断言相对便宜

供验证者验证,只需要检查资格和验证管理人员的签名,因此他们需要少量的交易费用。

在更复杂的情况下,只有其中一位经理签署了一项断言“有争议的断言”。与断言一起,断言经理必须托管一笔存款。这样一个有争议的断言不会立即被接受,而是,如果它是合格的,它会被发布为未决,并且其他经理有一个预先指定的时间间隔,他们可以在这个时间间隔内质疑该断言。

（有争议的断言中允许的步数限制为创建 VM 时设置为参数的最大值,以确保其他管理器有足够的时间在挑战前模拟声明的执行步数间隔到期。）如果在间隔期间没有挑战发生,则断言被接受,VM 被认为已经做出断言的状态更改并采取了断言的操作,并且断言管理器取回其押金。

4.2 二分协议

如果经理对断言提出质疑,则质疑者必须托管押金。现在,断言者和挑战者通过公共协议参与游戏,以确定谁是错误的。获胜方收回自己的押金,输方抽取一半押金。输家的另一半押金将交给验证者,作为对裁判比赛所需工作的补偿。

游戏以交替步骤进行。提出挑战后,断言者将获得一个预先指定的时间间隔来平分其先前的断言。如果之前的断言涉及VM中的N步执行,那么两个新的断言必须分别涉及N/2和N/2步,并且这两个断言必须结合起来等同于之前的断言。如果在时限内没有提供有效的二分法,则挑战者赢得比赛。在提供二分法后,挑战者必须在预先指定的时间间隔内挑战两个新断言之一。

两位玩家交替移动。在每一步,玩家必须在指定的时间间隔内移动,否则输掉游戏。每一步都需要玩家进行移动以进行小额额外存款,该存款将添加到游戏的赌注中。

经过对数次的二分后,挑战者将挑战涵盖单个执行步骤的断言。此时,断言者必须提供一步证明,它确定在断言的初始状态下,假设前提条件,在 VM 中执行单个指令将达到断言的最终状态并采取断言的公开可见的操作（如果有的话）。

这个一步证明由验证者验证。见图 1 概述了实现该协议的状态机。

4.3 VM架构的选择

Arbitrum VM 旨在使验证者检查一步证明的任务尽可能快速和简单。特别是,VM 设计保证了表示一步证明的空间以及生成和验证此类证明的时间受小常数的限制,与程序代码和数据的大小和内容无关。

作为支持常量有界证明的架构选择的一个例子,AVM 不提供大而平坦的内存空间。提供一个大的平面内存空间的高效向上数据哈希将需要空间以 Merkle Tree 样式进行哈希,证明者需要提供内存状态的 Merkle 证明,这需要对数证明空间和对数时间来证明和验证.相反,Arbitrum VM 提供了一种元组数据类型,最多可以存储八个值,它可以递归地包含其他元组。这允许构建相同类型的树表示,但它是由在 VM 内的应用程序中运行的 Arbitrum 代码构建和管理的。使用这种设计,读取或写入内存位置需要对数数量的恒定时间可证明的 Arbitrum 指令（而不是单个对数时间可证明的指令）。Arbitrum 标准库为程序员提供了一个大的平面内存抽象。

我们在这里提供了 VM 架构的概述。
有关更详细的规范,请参阅本文的扩展版本。

类型Arbitrum VM 的优化操作在很大程度上取决于其类型系统。在我们的原型类型中,类型包括:特殊的空值 None、布尔值、字符（即 UTF-8 代码点）、64 位有符号整数、64 位 IEEE 浮点数、长度最大为 32 的字节数组、和元组。元组是最多 8 个 Arbitrum 值的数组。元组的槽可以递归地保存任何值,包括其他元组,因此单个元组可能包含任意复杂的树数据结构。

所有值都是不可变的,实现会在创建每个元组时计算其散列值,因此任何值的散列值都可以在常数时间内（重新）计算。[4]

VM状态 VM 的状态是分层组织的。这允许计算 VM 状态的哈希值

[4] 元组和扩展类型是我们 VM 设计的一个基本方面。其他非关键元素可能会发生变化。例如,可能支持更少的类型,例如仅支持元组和整数类型。

以 Merkle Tree 的方式,并逐步更新。
状态哈希可以随着机器状态的变化而有效地更新,因为 VM 架构确保指令只能修改状态树根附近的项目,并且状态树的每个节点的度数不超过八。

VM 的状态包含以下元素:

· 一个指令堆栈,它对当前程序计数器和指令进行编码(如下所述);

·值的数据堆栈5;

· 调用堆栈,用于存储过程调用的返回信息。

· 一个静态常量,它是不可变的;和

· 保存一个值的单个可变寄存器。

初始化 VM 时,指令堆栈和静态常量从 Arbitrum 可执行文件中初始化;数据和调用栈都是空的;寄存器为无。请注意,由于单个值可以通过元组的递归包含来保存任意数量的数据,因此静态常量可以保存任意数量的常量数据以供程序使用,并且单个寄存器可用于管理包含 ing 的可变结构任意数量的数据。许多程序员会选择使用平面内存抽象,构建在这种可变结构之上,例如 Arbitrum 标准库中提供的结构。

说明VM 使用基于堆栈的架构。
存在 VM 指令以操作堆栈顶部、将小整数压入堆栈、在堆栈顶部执行算术和逻辑运算、类型之间的转换、计算值的哈希值、计算字节数组的子序列, 并连接字节数组。

控制流指令包括条件跳转、过程调用和返回。对元组进行操作的指令包括创建一个填充 None 的新元组的指令,从一个元组中读取一个槽,以及在修改一个槽的值的同时复制一个元组。最后,还有与其他方交互的说明,如下所述。

指令栈Arbitrum 没有使用传统的程序计数器,而是维护一个“指令栈”,它保存程序剩余部分的指令。 Arbitrum VM 不是通过指令列表来推进程序计数器,而是弹出指令堆栈以获得下一条指令

5 堆栈表示为 None(表示空堆栈)或 2 元组 (top, rest),其中 top 是堆栈顶部的值,rest 是堆栈的其余部分,格式相同。

执行。 (如果指令堆栈为空,则 VM 停止。)
跳转和过程调用指令改变了指令栈,过程调用存储了旧的指令栈(将指令栈的副本压入调用栈),以便在过程返回时恢复。

这种方法允许一步证明使用常量空间,并允许在常量时间内验证当前指令6和下一条指令堆栈值。

因为堆栈可以表示为链表,所以 AVM 实现可能会遵循我们的原型实现,将程序中的所有指令安排到单个链表中,并将指令堆栈值维护为指向该链表的指针。

汇编程序和加载程序Arbitrum 汇编程序采用 Arbitrum 汇编语言编写的程序并将其翻译成 Arbitrum 可执行文件。汇编器提供了各种形式的语法糖,使编程更容易一些,包括控制结构,如 if/else 语句、while 循环和闭包。汇编器还支持包含库文件,例如标准库中的库文件。

标准库标准库是一组用 Arbitrum 汇编代码编写的有用工具。它包含大约 3000 行 Arbitrum 汇编代码,并支持有用的数据结构,例如任意大小的向量、键值存储、寄存器顶部平面内存空间的抽象,以及时间和传入消息的处理。

与其他 VM 或密钥交互 VM 通过发送和接收消息与其他方交互。一条消息由一个值、一定数量的货币以及发送者和接收者的身份组成。发送指令从栈顶获取值并将它们作为消息发送。如果消息无效,例如因为它试图发送比 VM 拥有的更多的货币,则无效消息将被丢弃而不是发送。

程序使用收件箱指令将机器的消息收件箱复制到堆栈中。标准库包含有助于管理传入消息的代码,包括在新消息到达时进行跟踪并将它们一条一条地提供给应用程序。

平衡指令允许 VM 确定
它拥有多少货币,以及时间指令

6一种更传统的方法是在指令数组上保留一个整数程序计数器、一个线性指令数组和一个预先计算的 Merkle 树哈希。然后一步证明将使用默克尔树证明来证明当前程序计数器下是哪条指令。这将需要对数(指令数)空间和对数检查时间来进行一步证明。

相比之下,我们的方法需要恒定的时间和空间。

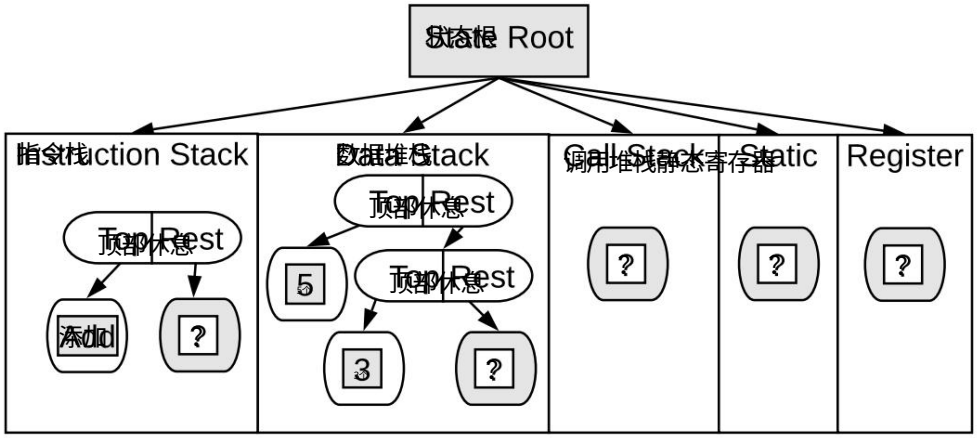

图 2:添加指令的一步证明中显示的信息。四舍五入的外框代表值哈希,内方框代表值本身。灰色框是在一步证明中由断言者发送给验证者的值。

降低 VM 以获得当前时间的上限和下限。

前提条件、断言和一步证明如上所述,断言是关于 VM 执行间隔的声明。每个断言都伴随着一组前提条件,包括:断言执行前 VM 状态的散列、VM 盒内内容的散列、VM 货币余额的可选下限,以及可选的下限和上限时间(以区块高度衡量)。除非所有先决条件都成立,否则断言将被视为不合格而被忽略。

(当事方可以选择存储不合格的断言,希望它以后变得合格。)

除了前提条件之外,断言还包含以下组件:执行后机器状态的哈希值、执行的指令数以及 VM 发出的消息序列。

Arbitrum 协议可能要求一方提供一步证明,这是正确性证明,假设一组前提条件,用于涵盖单个指令执行的断言。一步证明必须提供足够的信息,超出前提条件,使验证者能够模拟将要执行的单个指令。因为 VM 的状态被组织为 Merkle Tree,并且 VM 的起始状态哈希,也就是 Merkle Tree 的根哈希,作为前提条件给出,证明只需要扩展足够的初始状态状态 Merkle 树使验证者能够模拟单个指令的执行,计算在给定先决条件的情况下执行该指令所产生的唯一断言,并验证它是否与声明的断言匹配。

一步证明扩展了验证者所需的状态树的任何部分。例如,苏

假设要执行的指令从堆栈中弹出一个项目。回想一下,对于空堆栈,堆栈表示为 None,否则表示为 2 元组 (top, rest),其中 top 是堆栈的顶部项目,rest 是堆栈的其余部分。在这个例子中,如果栈哈希等于 None 的哈希,那么 Verifier 就会知道栈是空的。否则,证明者将需要提供 top 和 rest 的哈希值,以允许验证者检查这两个哈希值的组合是否产生预期的堆栈哈希值。类似地,如果指令应该将两个值相加,而验证者只有值的哈希值,则证明必须包括这两个值。在所有情况下,证明者都会提供验证者模拟指定指令所需的值,并且验证者会检查提供的值是否与验证者已经收到的哈希值一致。证明者使用的 Arbitrum VM 模拟器自动确定证明中必须提供哪些元素。请参见图 2,了解在添加指令的一步证明过程中向验证者透露的信息。

消息和收件箱消息可以通过两种方式发送到VM:密钥可以通过在区块链上放置特殊的消息传递交易来发送消息;另一个虚拟机可以使用发送指令发送消息。消息在逻辑上有四个字段:数据,它是一个 AVM 值(在区块链上编组为字节数组);从发送方转移到接收方的非负数额的货币;以及 mes 的发送者和接收者的身份

智者。

每个 VM 都有一个收件箱,其哈希值由验证者跟踪。空收件箱表示为 AVM 值 None。通过将收件箱设置为二元组 (prev, M),可以将新消息 M 附加到 VM 的收件箱,

其中 prev 是收件箱的先前状态。 VM 可以执行收件箱指令,该指令将 VM 收件箱的当前值推送到 VM 的堆栈上。

VM 的管理器跟踪其收件箱的状态,但验证者只需要跟踪收件箱的哈希值,因为这是验证 VM 接收收件箱内容的一步证明所需的全部。如果 VM 稍后处理收件箱内容,并且需要该处理的某个步骤的一步证明,则管理器将能够提供所需的任何值。

由于收件箱指令为 VM 提供了一个收件箱状态,该状态可能是多条消息的链接列表,因此程序员可能希望在 VM 内缓冲这些消息以提供一次接收一条消息的抽象。 Arbitrum 标准库提供代码来执行此操作以及跟踪新消息何时到达收件箱。

4.4 扩展

在本节中,我们描述了可能证明有用的 Arbitrum 设计的扩展,特别是当 Arbitrum Verifier 被实现为公共区块链时。

链下进度Arbitrum 允许虚拟机以相同的链上成本执行比现有系统多几个数量级的计算。然而,VM 的使用通常取决于 VM 的管理器和 VM 本身之间的通信。在我们之前对 Arbitrum 协议的描述中,这种通信必须在链上进行,因此受到共识机制速度的限制。 Arbitrum 与状态通道和侧链技术兼容,并且有几种结构允许管理者与 VM 通信并一致推进 VM 的状态链下。我们在本文的扩展版本中介绍了一种此类结构的详细信息。

零知识一步证明虽然 Arbitrum 具有良好的隐私属性,但在一种情况下可能会出现小的隐私泄露。提交一步证明的经理将被迫透露一些状态作为证明的一部分。虽然每次挑战只会显示一小部分状态,并且只有在管理人员未能就一致断言达成一致时,这可能是敏感数据。

我们可以使用 Bulletproofs [7] 将一步证明实现为零知识协议。为此,需要将一步 VM 转换编码为算术电路,并证明该转换是有效的。

虽然我们可以使用 SNARKs [4、16、27],但 Bulletproofs 的好处是它们不需要可信设置。
虽然 Bulletproofs 的验证时间在电路中是线性的,但考虑到单步转换电路会很小,并且一步证明会是偶发事件,这在实践中应该不是问题。

虽然理论上可以使用零知识证明来证明整个状态转换(而不仅仅是单个步骤)的正确性,但是对于复杂的计算来说,使用当前的工具这样做是不可行的。仅在最后一步将挑战和二分协议与零知识证明相结合,使我们能够同时实现可扩展性和完全隐私。这利用了 Arbitrum VM 旨在简化一步证明的事实。

读取区块链在我们目前的设计中,Arbi trum VM 没有直接读取区块链的能力。

如果作为公共区块链启动,我们可以轻松扩展 VM 指令集以允许 VM 直接读取区块链。为此,我们将创建一个块的规范编码作为 Arbitrum 元组,该元组的一个字段包含代表区块链中前一个块的元组。这将允许具有当前块的元组的 VM 读取更早的块。断言的前提条件将指定最近的块高度,并且 VM 将有一条特殊指令将关联的块元组推送到堆栈。为了能够验证该指令的一步证明,验证者只需要跟踪每个块的 Arbitrum 元组哈希(每个块只有一个哈希)。

我们强调读取区块链不需要将大量数据放在 VM 的数据堆栈上。区块链读取包括仅将指定块的顶级元组放入堆栈。为了更深入地读取区块链,这个元组可以延迟扩展,只为 VM 提供读取所需位置所需的数据。 7

7请注意,以这种方式读取区块链支持与零知识证明兼容的不经意读取,因为验证者不需要知道正在读取区块链中的哪个位置(如果有的话)。

验证者只需要验证顶级元组哈希,即最近一个块的哈希。如果元组被扩展以更深入地读取到区块链中,这一切都发生在 Arbitrum 应用程序代码中,读取的位置将不会在链上发布。通过这种方式,区块链读取完全兼容零知识一步证明。特别是,验证者将始终提供指定的块元组哈希作为零知识证明的输入。如果一步证明确实是在读取区块链指令上,则该证明将验证正确的哈希值是否已放入堆栈。零知识证明不会泄露关于区块链是否被实际读取的信息(因为即使没有读取发生,块哈希也始终是证明的输入)或区块链上发生读取的位置(因为当前块元组可以已在 Arbitrum 应用程序代码中进行扩展,以读取区块链中的任何位置)。

5 实施和基准

为了完善和评估 Arbitrum,我们制作了 Arbitrum 系统的完整实施。这包括代表所有相关方的代码:一个集中的验证者、一个虚拟机、一个诚实的管理者和一个基于密钥的参与者。这些各方完全有能力执行 Arbitrum 协议的所有部分。我们的实现包括大约 6800 行 Go 代码,其中大约 3400 行用于 VM 模拟器,1350 行用于 as sembler 和 loader,650 行用于 honest manager,550 行用于 Verifier,其余为各种共享代码。

为了简化更强大的智能合约 VM 的编码,我们实现了 Arbitrum 标准库,其中包含约 3000 行 Arbitrum 汇编代码,支持有用的数据结构,如大型元组、键值存储、队列和字符串;以及用于处理消息、货币和时间的实用程序。

我们展示了这个 im 的强大功能和多功能性
通过实施两个智能合约来补充。

5.1 托管合约

我们首先讨论一个简单的托管合约。托管代码首先等待一条消息,其中包含三方（Alice、Bob 和 Trent)的身份和整数截止日期,以及 VM 将持有的一定数量的货币。 VM 然后等待来自 Trent 的消息,忽略来自其他任何人的消息。

如果来自 Trent 的消息包含偶数整数,则 VM 将货币发送给 Alice 并停止。如果来自 Trent 的消息包含其他内容,则 VM 将货币发送给 Bob 并停止。如果当前时间超过截止时间,VM 将一半的货币发送给 Alice,将剩余的货币发送给 Bob,然后停止。这需要 59 行 Arbitrum 汇编代码,这充分利用了标准库。汇编程序生成的可执行文件包含 4016 条指令。

执行合同需要将 5 笔交易添加到区块链中。初始创建 VM 事务为 309 字节。之后,一条 310 字节的消息被发送到 VM,传达相关各方的身份和截止日期,并向 VM 提供货币。

接下来,Trent 通过向 VM 发送一条 178 字节的消息来表明他的判断。

接下来,必须执行 VM 才能真正导致支付。首先广播一个 350 字节的断言,断言 2897 条 AVM 指令的执行,使 VM 处于暂停状态。接下来,在挑战窗口结束后,将广播一个 113 字节的确认交易,确认并接受断言的执行。整个过程一共需要1260字节

写入区块链。

5.2 迭代哈希

Arbitrum 的亮点之一是它执行 VM 计算的效率。为了证明这一点,我们测量了执行迭代 SHA-256 哈希的 Arbitrum VM 的吞吐量。这个 VM 的代码是一个无限循环,其中 VM 散列 1000 次,然后跳回到开头。 VM 代码使用 AVM 的散列指令,该指令在本机代码中实现。

我们在 2013 年初的 Apple MacBook Pro、2.7GHz Intel Core i7 上评估了该 VM 的运行性能。作为基准,在同一台机器上使用本机代码,我们每秒能够执行 1,700,000 次哈希。连续运行虚拟机,我们能够以每秒 970,000 哈希值的速度推进虚拟机。我们的实施能够实现本机代码原始性能的一半以上。这与以太坊相比,以太坊每秒能够处理总共大约 1600 个哈希（受限于以太坊的全球气体限制,这是由于验证者困境所必需的）。

Arbitrum 的性能优势进一步扩大。
虽然我们演示了单个 VM 的当前执行限制,但 Verifier 能够同时处理大量 VM。通过实例化迭代哈希 VM 的多个副本,我们测得在我们的机器上运行的 Verifier 节点每秒能够处理超过 5000 个有争议的断言。

与以太坊的 1600 相比,这使总可能的网络吞吐量达到每秒超过 40 亿哈希。

6 背景及相关工作

6.1 裁判委托

委托计算的问题涉及资源受限的客户端将计算外包给更强大的服务器。服务器应该提供它正确执行计算的证明,并且检查证明对于验证者来说应该比执行计算本身更有效 [17]。

Refereed-delegation (RDoC) 是一种用于解决委托计算问题的双服务器协议 [10, 11]。计算被委托给多个服务器,这些服务器独立地将结果报告给客户端。如果他们同意,则客户接受结果。然而,如果服务器不同意,它们将进行二分协议以识别一步不同意。然后客户端可以有效地评估单个步骤以确定哪个服务器在说谎。 Arbitrum 的二分协议的各个方面非常

类似于 RDoC。在 Arbitrum 中,就好像验证者将 VM 的计算外包给 VM 的管理者,在许多情况下,他们是对 VM 的计算感兴趣的各方。Arbitrum 的 VM 架构使得争议解决非常高效。

6.2 比特币

比特币是一种去中心化的数字货币 [26]。

比特币本身只支持一种非图灵完备的简单脚本语言,主要用于签名验证。已经开发了许多技术来允许在比特币的脚本语言之上编写更复杂的脚本。这些通常分为两类:(1)使用加密工具实现更复杂功能的协议,同时将它们自身限制在比特币的脚本语言中,以及(2)使用比特币作为共识层的协议,包括区块链上的原始数据具有运行协议的节点已知的附加验证规则,但未经比特币矿工验证。

第一种脚本增强功能包括零知识或有支付 [3, 9, 23],能够实现数字商品的公平交易。虽然强大高效,但零知识或然支付的局限性无法实现一般的智能合约。后者包括交易方 [1] 和开放资产 [12],将验证的全部工作推到每个钱包上。在这些覆盖协议中,每个节点都必须验证每笔交易(即使是它们不属于的交易),以便对正确性有信心。将此与 Arbitrum 进行对比,在 Arbitrum 中,矿工保证所有货币交易的正确性,而节点必须只监控他们关心的 VM 的内部状态。

以太坊的目标是“全局正确性”,或者系统中每个参与者都相信每个合同都已正确执行的能力,这取决于挖掘共识过程是否按预期工作。相比之下,Arbitrum 不会尝试向对该 VM 不感兴趣的各方提供 VM 的正确性保证,这使 Arbitrum 获得了巨大的优势在可扩展性和隐私方面。在 Arbitrum 中,各方可以安全地忽略他们不感兴趣的 VM。

以太坊式智能合约的局限性

以太坊的智能合约方法有几个缺点。

可扩展性。众所周知,以太坊的模型无法扩展。要求矿工模拟每个智能合约是昂贵的,而且每个矿工都必须重复这项工作。虽然以太坊确实要求对计算感兴趣的各方补偿矿工(用“gas”)的执行成本,但这并没有降低成本 它只是转移了成本。

以太坊通过“全局气体限制”来应对验证者的困境,该限制严重限制了每个块中可以包含的计算量。 8 以太坊的全局气体限制是一个重要的限制,它会进行许多计算 这只需要几秒钟在现代 CPU 上执行 无法实现 [8, 24]。

即使对于低于气体限制的计算,以太坊的按指令付费模型也会变得非常昂贵。

隐私。所有以太坊合约代码都是公开的,这是该模型的必要条件,因为每个矿工都需要能够模拟所有代码。以太坊中的任何隐私都必须作为覆盖层出现。在以太坊中使用 zkSNARKs [4, 16, 27] 已经取得了进展,这样矿工就可以在对合约调用的输入保持隐藏的情况下验证证明。然而,由于验证 SNARK 的成本很高 9,因此在实践中这样做的能力受到严重限制,因此吞吐量将被严重限制为每个块只有几个这样的交易。此外,SNARK 给证明者带来了沉重的计算成本。

6.3 以太坊

以太坊 [31] 是一种数字货币,支持状态完整、图灵完备的智能合约。矿工模拟合约代码并相应地更新状态。为了使以太坊区块有效,矿工必须正确模拟他们包含在区块中的所有合约计算,并正确更新状态(包括货币余额)以反映这些变化。如果一个矿工没有正确更新状态,其他矿工将拒绝该区块。

缺乏灵活性。在法律合同中,合同当事人可以通过协商一致变更或者解除合同。
这被认为是法律合同的一个重要特征,因为它可以防止当事人被错误的合同或不可预见的情况所困。对于以太坊风格的智能合约,偏离代码

8虽然在某些情况下 Arbitrum 确实限制了断言中的计算步骤数,但 Arbitrum 的限制要少得多。
仲裁限制仅适用于有争议的断言,不适用于可以包括无限数量步骤的一致断言。此外,Arbitrum 的限制在适用时是针对每个 VM 的,并假设可以并行管理许多 VM,而以太坊的限制是对所有 VM 的总计算量的全局限制。事务以太坊测试网 (0x15e7f5ad316807ba16fe669a07137a5148973235738ac424d5b70fk8 9ae7625e3) 使用 1,933,895 gas 验证了 SNARK。在目前 7,976,645 的主网气体限制下,这将只允许每个 9a 区块进行 4 笔交易。 上 这

不可能。在 Arbitrum 中,可以修改合约 VM,只要 VM 的所有诚实管理者都同意即可。

6.4 其他建议的解决方案

我们现在讨论其他针对智能合约可扩展性和/或隐私提出的解决方案,并将它们与 Arbitrum 进行比较。

零知识证明。 Hawk [18] 是一个提议的系统,用于使用 zkSNARKs [16、27] 的私人智能合约。 Hawk 有很强的隐私目标,包括隐藏货币转移的金额和交易方,对非参与者隐藏合同状态,以及支持甚至对合同中的其他参与者隐藏的私人输入。然而,Hawk 有几个缺点,使其在实践中不可行。首先,SNARK 需要每个电路的可信设置,这意味着对于合约实现的每个不同程序,都需要一个新的可信设置。虽然可以使用多方计算来减少对设置的信任,但这无法按照 Hawk 的要求在每个电路的基础上执行。其次,Hawk 并没有提高可扩展性,因为每个合约都需要将千字节的数据放在链上。最后,Hawk 中的隐私依赖于信任可以查看所有私人数据的第三方管理器。

可信执行环境 (TEE)。多项提案 [6、13、20、33] 将区块链与可信执行环境（如英特尔 SGX)相结合。

Ekiden [13] 使用 TEE 来实现可扩展和私有的智能合约。 Arbitrum 向外部各方隐藏了智能合约的代码和状态,而 Ekiden 向外部各方隐藏了状态,并且还允许合约的各方隐藏彼此的私人输入。

Ekiden 和更普遍地依赖 TEE 的系统的缺点是隐私和合约执行的正确性需要额外的信任。这包括相信硬件正在正确和私密地执行,以及相信证明密钥的发行者（例如,英特尔）。

安全多方计算。安全的多方计算是一种密码技术,它允许各方在私有输入上计算函数,除了他们的输出 [21] 之外什么都不知道。一些工作已经提议将安全的多方计算纳入区块链 [2, 19, 34]。这使得能够将货币条件附加到计算结果并激励公平（通过惩罚中止方）。

与即使节点离线也能取得进展的 Arbitrum 不同,基于 MPC 的系统需要主动（和交互)所有计算节点的参与。
即使最近在安全多方计算的性能方面取得了进展,加密工具也会带来显着的效率负担。

通过激励验证者实现可扩展性。一些提案（例如,[30, 32])有独立的各方（矿工除外)执行计算验证,但根据验证者的奖励方式,这些结果可能成为参与困境的牺牲品。

这些系统中最流行的是 TrueBit [30]。
与 Arbitrum 不同,TrueBit 是无状态的,不是独立系统。 TrueBit 为以太坊合约提供了一种外包计算并以低于以太坊 gas 价格的成本向合约接收结果的机制。在 TrueBit 中,第三方求解器执行计算任务,他们的工作由第三方验证器（其扮演的角色与 Arbitrum 验证器不同)进行检查。 TrueBit 验证者可以对 Solver 给出的结果提出异议,并且争议通过类似于 Arbitrum 中使用的质询-响应协议来解决。

TrueBit 试图通过激励 TrueBit 验证者检查计算和挑战不正确的断言来实现全局正确性。要参与,TrueBit 验证者必须存入保证金,如果他们谎报错误,他们将损失这笔保证金。为了激励验证者参与,TrueBit 协议偶尔会故意引入错误,TrueBit 验证者会因发现这些错误而获得奖励。

如果 m 个 TrueBit 验证者发现相同的错误,他们将使用 $f_c(m) = c \cdot 2 - m$ 形式的函数分配奖励。
如第 2.3 节所示,这是一次性 Sybil 证明。
然而,由于这是一个参与博弈,他们容易受到参与困境的影响,并且根据定理 1,TrueBit 承认只有一个 TrueBit 验证者（使用多个 Sybils）的均衡,如果发生这种情况,该验证者可以随意作弊。

尽管他们没有对其进行正式分析,但 TrueBit ac 知道这种类型的攻击并提出了一些特别的防御措施。首先,他们假设单个验证者没有足够的钱来支付成功欺负所有其他验证者所需的保证金。虽然这个假设可能有帮助,但尚不清楚它是否成立,特别是多个对手可能会集中他们的资金来发起这次攻击。 （请注意,攻击者不会为了执行此攻击而伪造这些资金,而只需要手头有这些资金即可。）

即使假设确实成立,对手仍然有可能通过与多个 Sybils 验证合约来欺负特定合约中的所有其他验证者。为了抵御这种情况,TrueBit 提出了一种“默认策略”,其中验证者随机选择要验证的任务,而不考虑

之前验证合约的验证者数量。然而,这个提议是有问题的,因为默认策略是主导的:不是选择随机验证的位置,如果验证者选择具有较少额外验证者的任务,它会更好。遵循“默认策略”不仅不是均衡,而且无论其他人做什么,都会被更好的策略所支配。

TrueBit 也不提供隐私,因为它允许任何主体作为验证者加入系统,因此任何人都必须能够了解任何 VM 的完整状态。

TrueBit 和 Arbitrum 之间的另一个主要区别是,在 TrueBit 中,计算成本与执行的步骤数成线性关系。对于在 TrueBit 中执行的每项计算任务,该方必须缴纳税款以资助该任务的解决和验证。TrueBit 论文估计这种税收占实际计算成本的 500%-5000%。尽管 TrueBit 的计算成本低于以太坊的成本,但它仍然存在线性成本。

TrueBit 建议对 VM 架构使用 Web Assembly。然而,与 Arbitrum Virtual Machine 不同,它确保一步证明将具有较小的常数大小,Web Assembly 没有这样的保证。

等离子体。 Plasma [28] 试图通过引入子链的概念在以太坊之上实现扩展。子链使用自己的共识机制

nism 选择发布哪些交易。这种共识机制强制执行编码在以太坊智能合约中的规则。如果子链上的用户认为子链的行为是正确的或恶意的,他们可以向主链上的合约提交欺诈证明,以便用他们的资金退出子链。

这种方法存在许多问题。
首先,与分片类似,Plasma 子链各自存在于自己的孤立世界中,因此不同子链上的人之间的交互很麻烦。其次,缺乏关于如何在 Plasma 合约中实际构建复杂欺诈证明的细节。 Plasma 合约需要以某种方式指定所有共识规则和方法来证明新定义的区块链上的欺诈行为,这是以太坊合约中一个复杂且目前尚未解决的问题。最后,将数据移出主区块链会带来数据可用性挑战,因为为了生成欺诈证明,您必须能够访问 Plasma 区块中的数据,并且没有保证访问此数据的机制。由于这个问题,Plasma 包含许多缓解措施,如果出现任何问题,用户将退出 Plasma 区块链。

由于实施具有智能合约功能的 Plasma 子链（如以太坊)的复杂性,目前实施 Plasma 的所有努力都使用

简单的基于 UTxO 的系统,无需按顺序编写脚本,允许简单的证明。Plasma 提议使用 TrueBit 作为子组件,在具有智能合约的子链中进行有效的欺诈证明,但如前所述,TrueBit 使用现成的 VM,它不能保证证明的大小或效率。事实上,Plasma 可能会受益于使用 Arbitrum 虚拟机。

状态通道。状态通道是一类通用技术,可提高一小部分固定参与者之间智能合约的可扩展性。之前的状态通道研究 [5、14、15、25] 主要关注与 Arbitrum 实现的不同类型的扩展。

Arbitrum 允许具有非常大的计算量和状态的链上交易,成本低。状态通道允许一组各方相互同意链下的一系列消息,并且在处理完所有消息后仅发布一个聚合交易。

状态通道的构建侧重于各方都诚实可用的乐观情况,但在其他情况下无法顺利有效地工作。具体来说,如果通道的任何成员拒绝或无法继续参与,状态通道必须准备好在链上解决。这种链上解决机制需要在链上执行整个状态转换。因此,状态通道仅限于进行各方可以在链上进行的计算,否则争议解决将是不可行的。即使经理们并非一直都在行动,或者存在争议,仲裁仍然是有效的。

7 结论

我们推出了 Arbitrum,这是一个新的智能合约平台,与以前的解决方案相比具有更好的可扩展性和隐私性。我们的解决方案是共识无关的,并且可以插入任何现有的机制以在区块链上达成共识。Arbitrum 以其简单性而优雅,其直接和直观的激励结构避免了许多影响其他提议系统的陷阱。

Arbitrum 激励各方在链下就智能合约 VM 将做什么达成一致,即使各方违反激励措施,矿工或其他验证者的成本也很低。Arbitrum 还使用定制设计的虚拟机架构来降低链上争议解决的成本。将 VM 行为的执行主要转移到链下,并降低链上解析的成本,使 Arbitrum 在可扩展性和隐私方面具有优势。

8 致谢

Steven Goldfeder 得到 NSF 研究生研究奖学金的支持，资助 DGE 1148900。S。
Matthew Weinberg 得到 NSF 赠款 CCF 1717899 的支持。

参考

[1] 交易对手协议规范。 https://counterparty.io/docs/protocol_specification/ ，访问时间： 2018-01-01

[2] Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.：比特币的安全多方计算。在：安全和隐私 (SP)，2014 年 IEEE 研讨会

[3] Banasik, W., Dziembowski, S., Malinowski, D.：无脚本加密货币中的高效零知识或有支付。在：欧洲计算机安全研究研讨会。页数

261–280。施普林格 (2016)

[4] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.：C 的 SNARKs：在零知识的情况下简洁地验证程序执行。在：Ad vances in Cryptology – CRYPTO 2013，第 90-108 页。

施普林格 (2013)

[5] Bentov, I., Kumaresan, R., Miller, A.：Instanta neous decentralized poker。在：密码学和信息安全理论与应用国际会议。第 410-440 页。

施普林格 (2017)

[6] Brandenburger, M.、Cachin, C.、Kapitza, R.、Sorniotti, A.：区块链和可信计算：超级账本结构的问题、陷阱和解决方案。arXiv 预印本 arXiv:1805.08541 (2018)

..

[7] Bunz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.：Bulletproofs：机密交易的有效范围证明。技术。代表

[8] Bunz, B., Goldfeder, S., Bonneau, J.：以太坊中延迟和随机信标的证明。在：第一届IEEE 安全与隐私区块链研讨会论文集（2017 年 4 月）

[9] Campanelli, M.、Gennaro, R.、Goldfeder, S.、Niz zardo, L.：重新访问零知识或有支付：攻击和服务支付。在：2017 年 ACM SIGSAC 计算机和通信安全会议论文集。第 229–243 页。ACM (2017)

[10] Canetti, R., Riva, B., Rothblum, GN：使用多个服务器的实际计算委托。
在：第 18 届 ACM 计算机和通信安全会议论文集中。第 445–454 页。ACM (2011)

[11] Canetti, R., Riva, B., Rothblum, GN：计算的裁判委托。信息与计算 226, 16–36 (2013)

[12] Charlon, F.：开放资产协议 (oap/1.0)。在线，https://github.com/OpenAssets/open-assets protocol/blob/master/specification.mediawiki (2013)

[13] Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., Song, D.：驿传：一个保密、可信和高性能智能合约执行平台。 arXiv 预印本 arXiv:1804.05141 (2018)

[14] Coleman, J.：状态通道 (2015)

[15] Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.：Perun：基于加密货币的虚拟支付渠道。技术。代表，IACR Cryptology ePrint Archive，2017：635 (2017)

[16] Gennaro, R.、Gentry, C.、Parno, B.、Raykova, M.：没有 pcp 的二次跨度程序和简洁 nizks。在：密码技术理论和应用年度国际会议。施普林格 (2013)

[17] Goldwasser, S., Kalai, YT, Rothblum, GN：删除门控计算：麻瓜的交互式证明。
在：第四十届年度 ACM 计算理论研讨会论文集。第 113-122 页。美国计算机学会 (2008)

[18] Kosba, A., Miller, A., Shi, E., Wen, Z., Papaman thou, C.：Hawk：密码学和隐私保护智能合约的区块链模型。在：安全和隐私 (SP)，2016 年 IEEE 研讨会上。第 839-858 页。 IEEE (2016)

[19] Kumaresan, R., Moran, T., Bentov, I.：如何使用比特币玩去中心化扑克。在：CCS

[20] Lind, J., Eyal, I., Kelbert, F., Naor, O., Pietzuch, P., Sirer, EG：Teechain：使用可信执行环境的可扩展区块链支付。arXiv 预印本 arXiv:1707.05454 (2017)

[21] Lindell, Y., Pinkas, B.：隐私保护数据挖掘。在：年度国际密码学会议。第 36-54 页。施普林格 (2000)

[22] Luu, L., Teutsch, J., Kulkarni, R., Saxena, P.:共识计算机中神秘的激励机制。
在:第 22 届 ACM SIGSAC 计算机和通信安全会议论文集。第 706–719 页。美国计算机学会 (2015)

[23] Maxwell, G.:零知识或有支付。网址： https: //en.bitcoin.it/wiki/ Zero_Knowledge_Contingent_Payment (2011)

[24] McCorry, P., Shahandashti, SF, Hao, F.:具有最大选民隐私的董事会投票智能合约。 IACR Cryptology ePrint Archive 2017, 110 (2017)

[25] Miller, A.、Bentov, I.、Kumaresan, R.、Cordi, C.、McCorry, P.:Sprites 和状态通道:比闪电还快的支付网络

[26] Nakamoto, S.:比特币:一种点对点电子现金系统（2008 年）

[27] Parno, B.、Howell, J.、Gentry, C.、Raykova, M.:匹诺曹:几乎实用的可验证计算。
在:IEEE 安全和隐私研讨会,2013 年

[28] Poon, J., Buterin, V.:等离子:可扩展的自主智能合约。白皮书 (2017)

[29] Roughgarden, T.:第 5 讲:对等网络中的激励。 http://theory.stanford.edu/ ~tim/f16/l/l5.pdf （2016 年 10 月）

[30] Teutsch, J., Reitwiener, C.:区块链的可扩展验证解决方案 (2017)

[31] Wood, G.:以太坊:一种安全的分散式通用交易分类账。以太坊项目黄皮书 151, 1–32 (2014)

[32] Wood, G.:Polkadot:异构多链框架的愿景（2017 年）

[33] Zhang, F., Daian, P., Kaptchuk, G., Bentov, I., Miers, I., Juels, A.:Paralysis proofs: Secure dynamic access structures for cryptocurrencies and
更多的

[34] Zyskind, G.、Nathan, O.、Pentland, A.:Enigma:具有保证隐私的分散式计算平台。 arXiv 预印本 arXiv:1506.03471

参与游戏:充分证明和讨论

首先,我们提供定理 1 的证明。为此,我们需要比第 2.3 节中提供的更正式的设置。

每一轮都进行参与游戏。对于某些贴现参数 γ < 1,玩家具有时间贴现效用。也就是说,第 r 轮的效用贴现乘以第一轮的收益。笔记

以 γ 的比率,这
是必要的,以便收益是有限的,并且最佳响应的概念是有意义的。我们将取 γ → 1。也就是说,游戏是针对固定的 γ < 1 进行的,但我们会考虑 γ 非常接近 1 的情况。

定义 1 （一次性 Sybil 证明）。如果对于所有 k,· f(k +) ⩽ f(k +1),我们说参与游戏 f(·) 是一次性女巫证明。请注意,这相当于说策略si = 1 始终是最佳响应。

观察 1.每个 One-Shot Sybil-Proof 参与游戏都有 f(n+1) ⩽ f(n)/2。

证明。考虑一次 Sybil-Proof 定义中的 = 2。紧随其后的是索赔。 □

定义 2 （参与参数）。将 Sybil 证明参与游戏的参与参数定义为最大 k,使得 f(k) > 1。

定理证明 1. 令 k 为参与博弈的参与参数。如果 k = 1,那么在每轮s1 = 1 的情况下,玩家一参与是平凡的均衡,而所有其他玩家不参与,定理得到证明。

如果 k > 1,我们将考虑任何 1 > γ ⩾ 1− 3k f(1) 。考虑以下平衡:

- 玩家一参与并设置s1 = k 在每个圆形的。
- 玩家 i ∈ [2, k] 使用以下策略:如果在之前的任何 R = 12k f(1) 轮中， Σj=i sj < k−i−1,则设置si = 1。否则,设置si = 0。
- 玩家 i > k 设置si = 0。

首先,根据参与参数的定义,观察所有 i > 1 的玩家都做出最佳响应。玩家 1无论如何都会在每一轮设置s1 = k,因此所有其他玩家都会设置sj = 0。因此,在任何一轮中,玩家 i 面临的决定只是是否设置 sj = 并获得奖励 · f(k +) ,而不会影响任何人在任何未来回合中的策略。由于 f(·) 是一次性 Sybil 证明,我们有· f(k +) ⩽ f(k +1)。根据参与参数的定义,f(k + 1) ⩽ 1。所以玩家 i 最多只能通过参与获得 1 的奖励,并且必须支付 1 的成本,通过参与给他们带来非正效用。因此,所有 i > 1 的玩家都是最佳响应（效用为零,但没有提供更高效用的选项）。

现在,我们希望证明参与者 1 也是最佳响应者。请注意,玩家 1 当然有可能

为了提高他们在一轮中的收益：他们可以在他们设置$s_i = k$ 的一轮之后立即实现 · f()。立即从 One-Shot Sybil-Proof 的定义中，我们看到玩家 1 通过设置$s_i = 1$ 在这一轮中获得更多利润。但是，这将在以后的回合中让他们付出代价，并导致其他玩家参与。

具体来说，首先观察玩家 1 在任何回合中设置$s_1 = k$ 比$s_1 > k$ 严格更好。这是因为无论$s_1 = k$ 还是$s_1 > k$，所有其他玩家在未来每一轮的行为都相同，而$s_1 = k$ 在本轮中会产生更高的奖励。所以我们只需要考虑$s_1 < k$ 的偏差。

现在考虑如果玩家 1在每一轮都设置$s_1 = k$ 时的收益。每轮他们都会得到 $k \cdot f(k) - 1 := A$。所以玩家 1 得到奖励 $\geqslant A/(1-\gamma)$。

如果玩家 1在某轮中设置$s_1 = < k$，则考虑玩家 1 的最大收益。在这一轮中，玩家 1 将获得收益 · f() − 1 > ε。但现在考虑随后的 R 轮，并将这组轮称为 R。在这些轮中最多 k 轮中，$\sum_j s_j < k$ 是可能的。

这是因为$\sum_{j=1} s_j \geqslant k - X$，其中 X 是$s_1$在 R 的前几轮中玩过的最小值。这是因为如果在 R 的任何前一轮中我们有$s_1 = X$，那么玩家 2,.. ., $k - X + 1$ 将全部参与 R 中剩余的 ing 轮次。因此我们可能有$\sum_j s_j < k$ 的唯一方法是如果$s_i < X$。因为只有 k 个可能的值要报告，X 只能最多减少 k 次，这意味着最多有 k 轮$\sum_j s_j < k$。直觉上，每次玩家 1 将他们的 Sybil 计数从之前的最小值降低时，他们就会获得一个很棒的回合，其中参与者总数 < k。但是 R 中所有未来的轮次都增加了其他人的参与，因此总参与度将至少为 k，直到玩家 1 进一步降低他们的 Sybil 计数。

在这 k 轮的每一轮中，参与者 1 可能获得高达 $f(1) - 1 = C$ 的收益（这是一个非常宽松的上限）。
然而，在其他每一轮中，参与者 1 最多得到 $(k - 1)f(k) - 1 \leqslant A - 1$ 的回报。这是因为在所有其他轮中至少有 k 名参与者，在其中至少有一个不是玩家 1。因此，如果玩家 1 正在参与，对他们来说最好的情况是他们是 k-1 个参与者，只有一个其他参与者。因此，玩家 1 在这些 R 轮中的总收益上限为：

$$\sum_{r=0}^{R-1}(A-1)\gamma^r + kf(1) = (A-1)(1-\gamma^R)/(1-\gamma) + kf(1)$$
$$= A(1-c^R)/(1-\gamma) + kf(1) - (1-\gamma^R)/(1-c)。$$

最后，观察从 R + 1 到终止的整个剩余游戏的总收益为

大多数 $^R \cdot f(1)/(1-\gamma)$。这是因为最值
R 到 ∞γ r f(1)，[illegible] 他们的总收益最多为：

$$A/(1-\gamma) + kf(1) - (1-\gamma^R)/(1-c) + c^R f(1)/(1-c)。$$

观察第一项恰好是每轮设置$s_1 = k$所获得的奖励。通过适当设置γ，R，可以使附加项任意为负。特别地，设置 $\gamma = 1 - R = 3k\, f(1)$，$12k\, f(1)$

产量：

$$kf(1) - (1-\gamma^R)/(1-c) + c^R f(1)/(1-c)$$
$$= kf(1) - 3k\, f(1) \cdot (1-\gamma^R) + c^R \cdot 3k\, f(1)^2$$
$$= kf(1) - 2 + 3(f(1)+1)\gamma^R < 0。$$

最后的不等式如下，因为 R 足够大。

□

有必要对定理 1 进行快速评论。首先，为了使证明尽可能简单，观察我们的常数 γ 和 R 真的很浪费。当然我们可以优化常数，但这不是定理的重点。此外，我们当然不会声称预测这就是玩家在参与游戏中的行为方式。有许多均衡点。我们要指出的是，尽管单次推理的逻辑合理，但在重复博弈中存在可证明的不良均衡，并且这些均衡是相当（定性地）自然的：大多数玩家对市场做出反应，而一个玩家聪明地保持领先一步。鉴于此，以及其他不受欢迎的均衡的非常合理的存在，我们不会预测在重复博弈中会出现单发女巫证明均衡。

A.1 讨论可能的防御措施

在本节中，我们概述了一些针对参与者困境的“开箱即用”防御措施。这些防御措施似乎 a) 在技术上具有挑战性（也许是不可能的）并且 b) 成本高昂 - 与可能的对手的计算能力成线性关系。主要思想是我们对参与游戏的分析考虑了一个孤立的任务，每个玩家都可以参与每一轮。

相反，考虑一组并行玩的 T 参与游戏，限制是任何玩家最多可以同时进入其中的 A。边界 A 可能来自计算能力的限制，或者需要

货币存款。然而，“自然”的事态是 A > T，将我们还原到最初的参与博弈。也就是说，人们应该期望单个验证者（或验证者集团）拥有处理所有合约的计算能力。类似地，假设任何普通参与者都可以积累资金进行存款，那么一个富有的验证者（或企业集团）当然应该能够积累资金以在任何地方进行存款。所以这种做法一开始好像买不到什么东西。

一种潜在的防御途径是引入与其他合约无法区分的虚拟合约，人为地夸大 T > A。这样做的缺点是，如果虚拟合约要与其他合约无法区分，它们还必须奖励验证者，因此系统的成本将会爆炸。即使有人愿意付出代价，这个解决方案也有一些缺陷：

- 目前尚不清楚如何设计真正与其他交易无法区分的虚拟交易。
- 即使虚拟交易无法与其他交易区分开来，对手仍然可以尝试对他们投资的特定合约进行大量验证，鼓励其他人将有限的存款/计算能力花在其他地方进行验证。

如果有人能够以某种方式绕过上述问题，那么实施虚拟合约的成本会随着比率 A/T（其中 T 是自然期望的吞吐量）而增长。我们在下面提供了一些模拟结果来证实这一点。

有了足够多的虚拟交易，游戏就变成了：每个玩家同时选择一定数量的 Sybils si 。然后，随机统一选择A个参与游戏，玩家i在每个游戏中输入si个Sybils（注意，不失一般性，每个游戏者每局选择相同数量的Sybils是对称的）。如果 A/T（T 包括虚拟合约）很小，那么即使一个玩家引入了很多 Sybils，仍然有很大的机会在他们根本不参与的合约中结束，这仍然会产生合理的收益报酬。但是，我们当然需要 T > A 才能完成此操作，并且虚拟交易也需要付款。

下图描述了以下内容：假设 A/T 的初始比率（在图中称为“A”人们可以自然地认为 T 被归一化为 1）。然后，选择一个虚拟合约的比率来增加 T 到 T > A，以及形式为 f(m) = c · 2 −m 的奖励函数 f(·)。玩家 1 然后将选择s1进入每轮 A 参与游戏，因为他知道所有其他玩家都会对此作出最佳反应，以最大化他们自己的收益。最后，对于给定的 k（每个合同的不同参与者的期望数量），我们优化 T c 的所有选择以找到

,

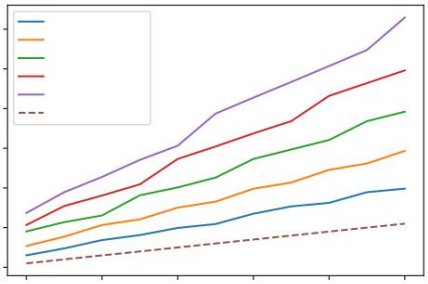

图 3：当一个用户针对各种初始A/T比率进行最佳女巫攻击时，保证x个不同参与者达到预期所需的总成本图。

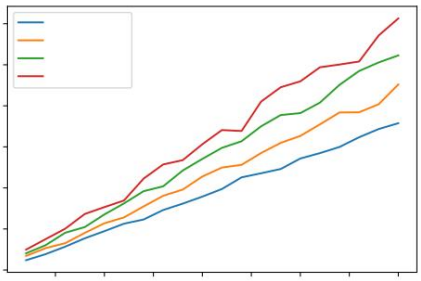

图 4：保证{2,3,4,5}个不同参与者期望的总成本图，当一个用户进行最佳女巫攻击时，作为初始比率A/T 的函数。

最小成本解决方案，保证每个合同的 k 个不同参与者的期望（以上述均衡形式）。我们在下面包括两个图。

这两个数字在 y 轴上都有总成本。图 3 在 x 轴上具有所需数量的不同参与者。虚线描绘了理想成本：我们必须为每份合约支付多少才能获得 x 个不同的验证者（这只是 x）。实线绘制了针对各种 A/T 初始值使用虚拟合约的最佳解决方案的成本。从第一个图中得出的结论是，如果 A > T，则理想成本和必要成本之间存在明显的分离。

图 4 的 y 轴为 A，实线绘制了使用虚拟合约作为 A 的函数的最佳解决方案的成本。在这里，很容易看出对于所有所需的不同数量的不同，成本在 A 中呈线性验证者。请注意，由于其他问题，此爆破将出现在基于参与游戏的作品中已经确定的任何爆破之上。