## BitVM中文社区 BitVM白皮书讲解 内容 解释 - BitVM中文社区 @Bitvm\_cn 凸 BitVM中文社区致力于推广和传播BitVM技术的概念和愿景。 Bitlayer@Oxkevinhe 位 Twitter 贡献者 Web3是一个开源、去中心化的生态系统,吸引着众多个体自发产出丰富多样的内容。这种自主参与的特性使得Web3成为一个充满创新和协作的数字化空间。在这个生态中,人们不仅仅是消费者,更是内容的创造者,推动着生 蓝港互动Web3研究小组 @XM\_LKWeb3Team 🖸 基础部分内容是引用蓝港互动的-比特币网络生态导航图 态的不断发展和演进。 特别感谢 Twitter 部分内容和图片引用 - 极简解读BitVM:如何在BTC链上验证欺诈证明(执行任意计算任计算 极客Web3 @eternal1997L 凸-BitVM技术通过允许在不改变比特币网络共识规则的前提下表达图灵完备的比特币合约,实现了执行任何可计算的函数、进行离线计算,并在链上不留下任何痕迹的特性。提供了一种全新的方式来探索复杂的区块链计算,同时保 持比特币网络的原则和安全性。这使得比特币能够在无需直接修改其核心协议的情况下,扩展了更广泛的使用案例和创新可能性。 BitVM中文社区将持续关注和总结有关BitVM的最新发展,期待更多志同道合的小伙伴加入BitVM中文社区,共同建设。 Crypto White @biquanlibai [2] -- 部分内容引用币圈李白老师的 - BitVM 在比特币上实现智能合约 20231019 比特币的核心是一种称为区块链的技术,它是一种去中心化的账本,记录着所有比特币交易的历史。每个区块包含一定数量的交易,并通过复杂的加密算法与前一个 比特币软分叉是指对比特币网络的一种更新,它允许旧版本节点和新版本节点继续共存于同一个网络中。这与硬分叉不同,后者创建了完全不兼容的新链。软分叉通常是为了引入新功能或 区块链接在一起,从而形成一个连续的链条。这种设计确保了比特币网络的安全性和透明度。 改进,同时保持向后兼容性。 (Bitcoin Core), 主要是用C++编程语言编写的。 一个简单的比特币交易(比如一个输入,两个输出)通常在250字节到600字节之间。更复杂的交易(比如多个输入和输出)可能会占用更多的字节。 提高网络性能和安全性:许多软分叉旨在提升比特币网络的性能、安全性和可扩展性。例如,SegWit软分叉增加了交易吞吐量并解决了交易可延展性问题。 - 2008年,化名为中本聪(Satoshi Nakamoto)的人或团体发表了一篇名为《比特币:一种点对点的电子现金系统》的论文: • 支持新功能和创新:软分叉使得在不割裂网络的情况下引入新功能成为可能,如更高效的签名算法、改进的智能合约功能等。 比特币诞生 2009年1月3日,中本聪挖出了创始区块。2009年1月12日进行了首次比特币交易。 - 升级压力:虽然软分叉保持向后兼容,但节点运营者和矿工可能会感受到升级到新版本以维持最佳性能和兼容性的压力。 比特币区块大约每10分钟产出一个,理论上每天大约会有144个区块被挖出。这取决于全网算力的变化,这个出块时间会有所波动。目前共计 一进度 82万余个区块。直到2140年。届时所有的比特币(2100万)全部发行完毕。换句话说在2140年之后,经过1,344万个区块,不会再有新的比 ·实施挑战:软分叉通常需要网络的广泛共识。获取足够多的网络参与者的支持来实施软分叉可能是一个复杂且耗时的过程。 特币产生。 比特币改进提案(Bitcoin Improvement Proposal,简称BIP)是一种设计文件,为比特币社区提供信息或提出新功能或改进。它是比特币开发流程的标准化方法, · 2024.1.11 SEC正式批准通过比特币ETF申请 比特币软分叉 ·BIP介绍 ———— 允许任何人提交新的想法或对现有协议的改进。BIP涵盖了广泛的技术内容,包括协议改进、新功能以及比特币社区中的各种问题。BIP的目的是确保所有的改进都经 过充分的讨论和审查, 以促进比特币的健康发展。 特币网络中协议的一个重大改变,这种改变是不向后兼容的。硬分叉会导致区块链分裂成两个不同的链,其中一个链遵循新的协议,另一个 链则继续遵循原有的协议。 - BIP 34 (2012): 这个软分叉添加了区块版本号,从而为未来的协议升级提供了更好的途径。 ·新币产生:例如Bitcoin Cash - BIP 66 (2015): 这次软分叉是为了增强网络的安全性,通过强制使用严格的DER签名来减少区块链的不确定性 - 安全风险: 硬分叉可能会暴露新的安全漏洞, 因为新协议可能没有经过充分的测试。 BIP 65 (2015): 引入了OP\_CHECKLOCKTIMEVERIFY(CLTV)操作码,使得比特币脚本可以执行基于时间的交易,这对于智 一 后果 基础知识 能合约的实现非常重要。 - 社区分裂:硬分叉可能导致社区分裂,不同的用户和矿工可能选择支持不同的链。 分离签名信息:在此之前,比特币交易包含了发送者的签名信息(称为见证数据),这部分数据被用来验证交易的合法性。 技术演进 ·价格波动:硬分叉的预期和实施往往会导致相关加密货币价格的大幅波动。 比特币硬分叉 Segregated Witness (SegWit, BIP 141, 2017): 这 SegWit的核心是将这些签名信息从交易的其余部分(例如输入和输出数据)中分离出来。 可能是最知名的比特币软分叉,又称"隔离见证", 2010 年 8 月 15 日,一名黑客利用比特币协议中的一个漏洞,生成了 1,844 亿枚比特币。比特币社区迅速反应,在数小时内发布了修正版 旨在解决可扩展性问题,减少交易大小,并提高区 区块结构的变化:传统比特币区块由一个区块头和许多交易组成,每个交易都包含签名信息。在SegWit实施后,签名信 本。2010年8月16日,比特币通过硬分叉化解了危机。 块链的效率。SegWit通过移除签名信息来实现这 息被移出交易本身,并被放置在区块的特殊部分,这个部分仅对支持SegWit的节点可见。 一点,并引入了对闪电网络这类二层解决方案的支 2015年澳本聪事件引发了广泛的争议和讨论,涉及一位名叫克雷格·赖特(Craig Wright)的澳大利亚计算机科学家,他自称是神秘的比 支持二层协议:SegWit的实施为比特币网络上的二层协议,如闪电网络(Lightning Network),提供了更好的基础。 持。 ·特币创始人中本聪(Satoshi Nakamoto)。赖特后来继续参与加密货币相关的项目和活动,他还声称参与了比特币SV(BSV)的开 🖒 这些协议旨在提高交易速度并降低成本,而不需要改变比特币的核心协议。 发,这是比特币的一个分叉版本。附推特链接。 Schnorr 签名(BIP 340):引入了Schnorr签名机制,允许多个输入被聚合在一个单独的签名中,这有助于提高隐私和 2017 年 8 月 1 日,比特币多年来的扩容争议导致比特币硬分叉,"Segwit"优化方案的支持者继续支持比特币,"大区块"方案(一部分社区成 效率,特别是在复杂的智能合约和多签名交易中。 员支持将区块大小增加到8MB)的支持者则创造出比特币现金Bitcoin Cash(BCH),并定向对原比特币持有账户进行 1:1 空投。 Taproot (BIP 341, BIP 342, 2021): 这次软分叉增 加了对Schnorr签名的支持,提高了隐私性和效 结合了MAST(Merkelized Abstract Syntax Trees),这允许在区块链上以更隐私和更高效的方式使用智能合约。 2017年10月24日,Bitcoin Gold(BTG)通过改变比特币的挖矿算法,使得使用传统图形处理单元(GPU)的挖矿变得更加实用,而不是依 率,同时简化了智能合约的执行,使得复杂的比 MAST允许将多个交易条件嵌入到一个单一的交易中,但只有特定条件被满足时合约的具体细节才会被揭示。这提高了隐 🗗 赖于专用的ASIC硬件。这一变化旨在降低比特币挖矿的门槛,并使网络更加去中心化。25日,市场对 BTG 信息不足,引发抛售,BTG 价格 特币交易在区块链上的占用空间更小,费用更 私性,因为外部观察者无法看到未执行的交易路径。 暴跌 66%。 新的输出类型(Pay-to-Taproot, P2TR)允许比特币地址同时支持普通的支付和复杂的智能合约。无论交易类型如何复 2018年11月15日,BSV全称为Bitcoin Satoshi Vision从比特币现金(BCH)分叉而来,是比特币(BTC)的一个分叉版本,比特币现金已 · 杂,对外部观察者来说,所有交易都看起来像是标准的比特币交易。这增加了隐私性,并使复杂交易在区块链上的占用 🖒 · 经将区块大小从比特币原始的IMB提高到了32MB,但BSV的支持者认为这还不够。他们主张需要更大的区块来处理更多的交易,从而增强。 空间更小,费用更低 网络的可扩展性。 Bitcoin core 开发者团队 ——— Git 辽 - 版本号: 指示区块遵循的规则。 比特币PSBT(Partially Signed Bitcoin Transaction)是一种扩展的交易格式,它允许交易的不同部分在不同时间由不同的实体签署。这种格式对于多方签名和硬件钱包特别有用,因为它允许 在交易被最终广播到比特币网络之前,安全地传递交易数据以进行签名。PSBT格式使得交易的创建和签名过程可以更加灵活和安全,特别是在需要多方协作的情况下。 ✓ 上一个区块的哈希值:这是一个指向前一个区块的链接,形成了区块链的链式结构。 (PSBT) - 多方签名支持:PSBT允许在一个交易被完全签名并广播到比特币网络之前,多个参与方分步骤签署交易。这对于需要多重签名的复杂交易场景特别有用。 Merkle树根哈希:这是一个由区块中所有交易的哈希值构成的数据结构的根哈希值,用于快速和有效地验证区块中的交易数据 ─ 与硬件钱包的兼容性:PSBT格式方便与硬件钱包集成,允许安全地在设备间传递未完成的交易以供签名。 区块头 一 时间戳:标记区块生成的时间。 增强安全性:通过分离交易的创建和签名过程,PSBT减少了交易在签名过程中被篡改的风险。 UTXO集:比特币网络全节点动态管理和维护一个全局的UTXO集,数据并不直 - 难度目标(nBits):表示挖掘该区块所需的工作量或难度。 不存储在区块本身:重要的是要理解,UTXO集不是存储在任何单个区块或区块链的任何静态部分中。相 接存储在区块链本身的区块中。 反,它是一个动态的、实时更新的数据库,反映了所有比特币的当前所有权状态。 ➤ Nonce:一个32位的数字,矿工在挖矿过程中不断变化这个值,直到区块的哈希值满足网络难度要求。 这是一个基于区块链上的交易历史构建的,包含所有未花费输出的数据库。 全节点的内存中:比特币的全节点在内存中维护一个UTXO数据库。这个数据库是根据区块链上的交易历史 ☑ 区块结构 - 交易计数器:表示区块中包含的交易数量。 - 构建的。当节点同步新的区块时,它会更新其UTXO数据库,添加新的未花费的输出,并移除那些已经作为 当一个新的交易被发起时,网络会检查这个UTXO集来确认交易中的输入是否有 新交易的输入被花费的输出。 效(即确保交易中使用的比特币确实存在并且可用)。 输入列表:每个输入包括引用前一个交易的输出、解锁脚本(包括签名和公钥)、UTXO引用; 索引和数据库形式:不同的比特币实现(如Bitcoin Core)可能会以不同的方式存储UTXO集。通常,它 每当一个交易被确认并加入到区块链中,全节点就会解析这个交易,更新UTXO 们会使用专门的数据库格式(如LevelDB或BerkeleyDB),这些格式经过优化,可以高效地查询和更新 交易列表: 集以反映这些变化。 |输出列表:每个输出Output 包括接收的比特币数量、UTXO标识符(UTXO Identifier)、解锁脚本和锁定脚本(指定谁可 这是区块中 UTXO数据。 以使用这些比特币) 锁定脚本的最大长度为 80 字节。BRC20 协议中的 基础知识 最大的部 |铭文格式长度为 100 字节,因此无法直接存储在锁定脚本中。BRC20 协议使用 Witness 来解决这个问题。 交易列表中的输出:每个比特币交易包含一个或多个输出,这些输出定义了交易中比特币的接收者和接收的金额。每个输出都可以被视为一定数量的比 分,包含了 区块中所有 特币的锁定存款。 重要概念 一 交易的元数据,如版本号和锁定时间。 的交易。每 笔交易都包 一 UTXO的形成:当一个交易输出被创建且尚未被使用作为另一笔交易的输入时,它就被认为是UTXO,即这部分比特币尚未被再次花费。 🚽 🔾 Witness:在SegWit启用的交易中,这部分包含了验证每个输入的签名和脚本数据,但以不同的方式存储。 ▎UTXO与账户余额:在比特币中,账户的"余额"实际上是指向该账户的所有UTXO的总和。不同于传统银行账户的余额概念,比特币的余额是通过计算所有指向用户地址的未花费输出来确定的。 ➤ 交易输出和 Witness 的关系是: 交易输出可以使用 Witness 来存储额外的元数据。 ➤ 作用 ———— 防止欺诈,例如 双重支付,还可以防止 伪造交易。 比特币脚本(Bitcoin Script)是一种简单的堆栈式编程语言,用于处理和验证交易。它是比特币交易的核心组成部分,使得比特币网络能够执行复杂的 BitCoin Script - 条件来控制交易的执行。脚本语言使用户能够设定条件,只有在满足这些条件时,比特币才能被转移。这包括但不限于多重签名交易、时间锁定等。比 特币脚本的设计目的是为了提供灵活性,同时保持足够的简单性以确保网络安全。 节点的总数约为16,245余个,最多的节点分布在美国、德国、法国、荷兰和加拿大等国家。这些节点的存在和运行是比特币网络去中心化和安全的关键 部分。 全节点:存储整个比特币区块链的历史并验证所有交易和区块。它们是网络的主要支撑,确保交易的正确性和安全性。 BTC网络节点 - 轻节点(SPV节点):不存储完整的区块链历史,而是依赖全节点来获取交易信息。它们适用于资源受限的设备,如智能手机。 ー 矿工节点:除了执行全节点的功能外,还参与挖矿过程,努力创建新的区块并获得比特币作为奖励。 - 监听节点:通常用于收集交易和区块数据,但不一定参与区块链的验证过程。 Robin Linus robin@zerosync.org 2023.12.12 BitVM的作者,同时也是ZeroSync的创始人和核心贡献 作者 Robin Linus BitVM 是一种计算范式,用于表达图灵完备的比特币合约。这不需要对网络的共识规则进行任何更改。与其在比特币上执行计算,不如仅进行验证,类似于乐观Rollup。证明者提出 —— 无需on chain的数据,先在链下发布并存储,链上只存放Commitment(承诺) 一个声明,称给定的函数对某些特定输入计算出某个特定输出。如果该声明是假的,那么验证者可以进行简洁的欺诈证明并惩罚证明者。使用这种机制,任何可计算的函数都可以在比 BitVM的思路 一 特币上验证。 发生挑战/欺诈证明时,我们只把需要上链的数据on chian,证明其与链上的Commitment存在关联。之后,BTC主网再校验这些on chian的数据是否有问题,数据生产者(处理交易的节点)是否 有作恶行为。这一切都遵循奥卡姆剃刀原则——"若非必要,勿增实体"(能少on chain,就少on chain)。 在 Taproot 地址中提交一个大型程序需要大量的链下计算和通信,然而产生的链上足迹却很小。只要双方合作,他们可以执行任意复杂的、有状态的链下计算,而不在链上留下任何 图灵机是由阿兰·图灵在1936年提出的抽象计算模型,用以定义什么是可计算的 痕迹。仅在发生争议时才需要链上执行。 图灵完备性(Turing Completeness)是一个与计算理论相关的概念。一个计算系统如果是图灵完备的,意味着它能够模拟任何图灵机的计算过程。换句话说,这样的系统能执行包括条件分支和循环等操作的任何算法,只要有足够的 图灵完备 🖸 时间和存储空间。 比特币区块链本身并不是图灵完备的。可以防止在比特币网络上运行恶意代码或造成无限循环,可能会威胁网络的安全和稳定性。 Optimistic rollups 是 Layer 2 扩展解决方案,旨在从其构建的底层 Layer 1 区块链中卸载一些交易处理,以提高交易吞吐量。它们之所以被称为"optimistic",是因为它们假设第 2 层上的所有交易在默认情况下都是有效的,并且只有 在验证者在争议期间质疑其有效性时才会验证交易。 Optimistic rollups 在链下执行交易,然后将许多交易进行打包,然后再将它们提交到基础链。与 ZK-Rollups 或状态通道等其他 2 层扩展解决方案相比,Optimistic Rollups 的优势包括相对简单和较低的交易成本,这是以延长争议 乐观Rollup 🖸 期或取款时间为代价的。 与零知识相比,Optimistic Rollups 的另一大优势是它们更全面,并且可以像它们所构建的底层区块链一样支持智能合约。对智能合约的原生支持意味着开发人员可以相对轻松地在 2 层网络上部署他们现有的去中心化应用程序,并 且只需进行少量代码调整。 比特币Taproot是比特币协议的一个重要升级,旨在提高比特币的隐私性、安全性和可扩展性。它于2021年11月被激活,是比特币自2017年SegWit升级以来的又一次重要升级Taproot地址通常以"bc1"开头,这是一种表示SegWit地址 Taproot 地址 🖸 —— 的格式,bc1包括更多的优点,例如更高的容量和更低的交易费用。 时间锁是对交易或输出的限制,只允许在一个时间点之后才能消费。比特币从一开始就有一个交易级时间锁定功能,它由交易中的nLocktime字段实现。在2015年底和2016年中期推出了两个新的时间锁功能,提供UTXO级别的时间锁功能, 按设计,比特币的智能合约功能仅限于基本操作,如签名、时间锁和哈希锁。BitVM 为更具表现力的比特币合约和链下计算创造了新的设计空间。潜在的应用包括国际象棋、围棋或 这就是CHECKLOCKTIMEVERIFY和CHECKSEQUENCEVERIFY。时间锁对于延期交易和将资金锁定到将来某个日期很有用。更重要的是,时间锁将比特币脚本扩展到时间的维度,为复杂的多步骤智能合约打开了大门。 扑克等游戏,特别是在比特币合约中验证有效性证明。此外,可能还能将 BTC 桥接到外部链条、构建预测市场或模拟新的操作码。 1简介 Introduction 哈希锁是限制一个输出花费的限制对象,其作用一直持续到指定数据片段公开透露。哈希锁有一个有用的属性,那就是一旦任意一个哈希锁被公开打开,任何其他使用相同密钥保护的哈希锁也可以被打开。这使得可能创建多 这里提出的模型的主要缺点是它仅限于具有证明者和验证者的两方设置。另一个限制是,对于证明者和验证者而言,执行程序需要大量的链下计算和通信。然而,这些问题似乎很可 个被同一哈希锁限制的输出,这些输出将在同一时间被花费。 能通过进一步的研究得到解决。在这项工作中,我们仅专注于两方 BitVM 的关键组成部分。 比特币脚本(Bitcoin Script)是一种简单的堆栈式编程语言,用于处理和验证交易。它是比特币交易的核心组成部分,使得比特币网络能够执行复杂的条件来控制交易的执行。脚本语言使用户能够设定条件,只有在满足这些条件 与乐观Rollup[1]和MATT提案(Merkelize All The Things)[2]类似,我们的系统基于欺诈证明和挑战-响应协议。然而,BitVM不需要对比特币的共识规则进行任何更改。底层原语: 时,比特币才能被转移。这包括但不限于多重签名交易、时间锁定等。比特币脚本的设计目的是为了提供灵活性,同时保持足够的简单性以确保网络安全。 相对简单。它主要基于哈希锁、时间锁和大型 Taproot 树。 证明者逐字节地承诺程序,但在链上验证所有这些将过于计算昂贵,因此验证者执行一系列精心设计的挑战,以简洁地驳斥证明者的虚假主张。证明者和验证者共同预签一系列挑战 /\*\* Otitle smart HelloWorld contract. \*/ contract HelloWorld { Smart Contract (Solidity程序) // the greeting variable, by default, it is set string public greeting: 比特币脚本 和响应交易,可用于解决以后的任何争议。 PUSH 60 PUSH 40 MSTORE PUSH 18 该模型旨在简单地说明,这种方法允许在比特币上进行通用计算。对于实际应用,我们应该考虑更高效的模型。 Bitcoin Script这种简陋的、由一堆独特操作码组成的、非图灵完备的编程语言。所以比特币脚本 不能直接相Solidity或Move等图灵完备的高级语言一样直接进行复杂运算;BitVM的方案是把 2 架构 指令译码 Instruction Dec 该协议很简单:首先,证明者和验证者将程序编译成一个巨大的二进制电路。证明者在一个 Taproot 地址中承诺该电路,该地址对电路中的每个逻辑门都有一个叶子脚本。此外,他 EVM/WASM/Javascript语言中操作逻辑转化成由Bitcoin Script的操作码组成的逻辑门电路; Architecture -ogic Gate (逻辑门) 们预签一系列交易,实现证明者和验证者之间的挑战-响应游戏。现在他们已经交换了所有必需的数据,所以他们可以对生成的 Taproot 地址进行链上存款。 这激活了合约,他们可以开始交换链下数据以触发电路中的状态变化。如果证明者提出任何不正确的声明,验证者可以拿走他们的存款。这保证了攻击者总是会损失他们的存款。 Script (比特而脚本) 🌑 公众号·极客 Web3 Taproot 是比特币的一个升级,旨在提高交易的隐私性和效率。Taproot 的主要目标是将多方签名交易(如多重签名交易)的交易脚本隐藏起来,使其看起来像是单方签名交 易,从而增加交易的隐私性 大型Taproot 树 大型Taproot树,涉及到更复杂、更大规模的多方签名结构或者更为庞大的交易树。这可能与构建更为复杂的智能合约或更复杂的交易结构有关。 承诺方案(Commitment Scheme)是一种用于确保信息不可更改和保密性的密码学工具。它允许一个实体(通常是发送者)承诺某个值,使其无法在承诺之后更改,同时另一方(通常是接收 比特值承诺是系统中最基本的组件之一。它允许证明者将特定比特的值设置为"O"或"I"。特别是,它允许证明者在不同的脚本和未使用的交易输出(UTXO)之间设置变量的值。这 者)可以在合适的时候验证这个承诺。 是关键的,因为它通过将执行运行时拆分成多个交易来扩展比特币的虚拟机(VM)。 1. commit:P(Prover)把一些数据分割成叶子节点,最后去构建整棵默克尔树。但整棵树不对外发布,它只是发布这个树的root 类似于Lamport签名[3],该承诺包含两个哈希,hashO和hash1。在稍后的某个时刻,证明者通过揭示preimageO,即hashO的原像,将比特的值设置为"O";或者证明者通过揭 <mark>- 2. reveal: P展现一个leaf及其branch(</mark>"branch" 通常指的是树的一个分支或路径,连接根节点和叶节点之间的所有中间节点) 示preimage1,即hash1的原像,将比特的值设置为"1"。如果在某个时刻他们同时揭示了preimage0和preimage1,那么验证者可以将它们用作欺诈证明,并拿走证明者的存款。 · Commitment Scheme • 3. check: V (Verifier) 验证branch 这被称为equivocation(含糊其辞)。能够惩罚含糊其辞是使承诺具有约束力的原因 - 这是一种"基于激励的承诺"。 Merkle tree • 标签 2 1. binding:commit之后,所有可以reveal的leaf就已确定 将比特值承诺与时间锁相结合,允许验证者在某个给定的时间范围内强制证明者决定特定比特的值。 2. hiding:reveal时只展示了其中的一个叶子节点,并不会展示其他的叶子节点,所以其他叶子节点是隐藏的 Stack Elements 1. 各种hash: SHA256、Pedersen、MiMC 一些Commitment -- 2. 各种tree: Merkle、MPT 3. ZKP系列: KZG、FRI、IPA i. commit: P发布计算 f(x)=35 , 承诺知道其解 x 什么是 # $\text{Hipf}(x) = x^3 + x + 5 = 35$ OP IF Commitment OP HASH160 def f(x): y = x\*\*3assert y + x + 5 == 35OP\_EQUALVERIFY https://vitalik.ca/general/2016/12/10/qap.html ii. reveal: P公布解 x <0xb157bee96d62f6855392b9920385a834c3113d9a iii. check: V将解 x 代入计算 f(x)=35 进行验证 OP\_EQUALVERIFY iv. 解的变量 x 又称作计算的input,中间变量 y 又称作witness 1 OP\_ENDIF i. 链上运行的时候要减少发布的尺寸和验证的计算量 ii. ZK Rollup Commit一段计算 1. 发布:将计算编译为电路,发布电路的commitment(逻辑和变量在ZKP电路中又称 图1:一个用于1比特承诺的具体实现。为了解锁此脚本,证明者必须揭示hashO或hash1的原像中的一个。在此示例执行中,证明者揭示了hash1,并将比特的值设置为"1"。我们可以 作gate constraints和copy constraints) 拥有这个承诺的副本,以强制在不同脚本之间执行特定值。 为简单起见,从这里开始,我们假设存在一个名为OP\_BITCOMMITMENT的操作码,它是上述脚本的缩写。该操作码消耗两个哈希和其中一个哈希的原像。根据原像匹配的哈希,它 将比特值放在堆栈上。 3 比特值承诺 Bit Value Commitment https://vitalik.ca/general/201 9/09/22/plonk.html 2. 验证: 用一个基于ZKP的validity proof做简洁的验证 输入值:0x47c3le6lla3bd2f3a7a422076l3046703fa27496 每个Bit Value Commiement只能被 ·打开时为0或1,相当于一个二进制的 进入OP\_IF分支。 变量,只有0或1两种情况 对0x47c3le6lla3bd2f3a7a422076l3046703fa27496 做OP\_HASH160之后的结果(等于 - 逻辑 🖸 多个Bit Value Commiement引用 0xf592e757267b7f307324fle78b34472f8b6f46f3) 的话就形成了一连串电路 Opening this bit commitment to the value "1" 与0xf592e757267b7f307324f1e78b34472f8b6f46f3 做OP\_EQUALVERIFY,若相等返回1 Witness Script - 若不等则返回O 这段Bitcoin script可简记为 简记 ——— `<hash O> <hash 1> OP BITCOMMITMENT` Bit Value Commitment OP EQUALVERIFY unlock的时候,输入preimage0则返回0,输入preimage1则返回1 6 OP\_ELSE 1. commit: P发布`<hash 0> <hash 1> OP BITCOMMITMENT` OP\_HASH160 Commitment Scheme —— - 2. reveal:P展现`<preimage>` 3. check: V验证`<preimage> <hash\_0> <hash\_1> OP\_BITCOMMITMENT` OP EQUALVERIFY 1. binding:commit之后,所有可以reveal的preimage就已确定 OP ENDIF 2. hiding: reveal时只展示了其中的一个preimage Lamport 签名的基本思想是,对于每个比特位,生成一对密钥(私钥和公钥),通过哈希函数将私钥映射成相应的公钥。签名时,将消息的每个比特位与相应私钥中的位进行匹 Lamport签名 配,然后使用匹配的公钥中的哈希值作为签名的一部分。验证时,使用相同的哈希函数对签名进行验证。 任何可计算函数都可以表示为布尔电路。与非门(NAND)是一种通用逻辑门,所以任何布尔函数都可以由它组成。为保持模型简单,展示了我们的方法适用于简单的与非门。此外, 我们展示了如何任意组合门。这一起证明了 BitVM 可以表达任何电路。 与非门承诺的实现很简单。它包含代表两个输入的两个位承诺和代表输出的第三个位承诺。脚本计算两个输入的与非值,以确保它与承诺的输出位相匹配。 栈顶口 ... we put it to the altstack for now // Reveal preimage of hash Cl or hash C0 2 <0xC468A29472CACF3EF179BA2352F88587B91E3E15> 3 <0x829923B22B9E831822E0A783F92687D27128157B> 4 OP BITCOMMITMENT 5 // Now the bit value of "C" is on the stack 6 // ... we put it to the altstack for now 最后一个进的,先出去 7 OP TOALTSTACK Bit Value Commitment C 入栈 / Reveal preimage of hash A0 or hash A1 Bit Value Commitment B 入栈 代码解释 —— 9 // Reveal preimage of hash B1 or hash B0 Bit Value Commitment A 未入栈 OP\_BITCOMMITMENT Bit Value C <0x34F0132278E874836DA82F8A6C1E10A21A153D17> 比较 A NAND B的值和C是否相等 Now the bit value of "A" is on the stack 1 <0xF9FCE46CEFE9D9392108480AD42B4CE69557D27D> 2 OP BITCOMMITMENT - 形成的电路 -.3 // Now there's the bit value of "B" on the stack 4 // ... we put it to the altstack for now 将Bit Value Commitment作为输入和输出组装成logic gate 5 OP\_TOALTSTACK 可简记为<hash\_c0> <hash\_c1> <hash\_b0> <hash\_b1> <hash\_a0> <hash\_a1> · Logic gate 7 // Reveal preimage of hash A0 or hash A1 OP\_GATECOMMITMENT,对应的unlocking script是criptEcriptEcriptEcriptEcriptEcriptEcriptEcriptE 8 < 0x5ACFDE72A8E37111CBA96D3DD705BA983F47AF4D> 9 <0xA0172816A2D1B20EF0D5A1093958E9564E590BAF> 发布一个Logic Gate Commitment表示其承诺3个Bit Value Commitment打开后符合gate的逻辑约束 OP BITCOMMITMENT OP\_EQUALVERIFY 比较 21 // Now the bit value of "A" is on the stack 4逻辑门承诺 **Logic Gate** 程序转成逻辑门计算的例子 — 24 // Verify that "A NAND B == C" Commitment 发布计算f(x)=35, 承诺知道其解x 将计算编译为电路 7 // Read "B" from altstack 计算机/处理器,是一个由大量逻辑门电路组合成的输入-输出系统。BitVM的核心思路之一,是用Bitcoin Script(比特币脚本),模拟出逻辑门电路的输入-输出效果。只要能模拟出逻辑门电 OP FROMALTSTACK 🖊 路,理论上就可以实现图灵机,完成所有可计算任务。也就是说,只要你人多钱多,就可以召集一帮工程师,帮你用功能简陋的Bitcoin Script代码,先模拟出逻辑门电路,再用巨量的逻辑门电 路实现EVM或是WASM的功能。 ) // Compute "A NAND B" OP NAND 3 // Read "C" from altstack 4 OP FROMALTSTACK 5 // ... and check that it is correct 6 OP EQUALVERIFY 解释 图2: NAND操作的逻辑门承诺。执行此脚本需要揭示比特承诺A、B和C的值,使得A NAND B = C成立。 (此截图来自于一款 教学游戏: 《图灵完备》,其中最核心的内容,就是用逻辑门电路尤其是与非门,搭建出完整的CPU处理器) (在这里,我们为简单起见假设存在用于OP\_NAND的操作码。实际上它并不存在,然而,可以使用OP\_BOOLAND和OP\_NOT轻松实现。) ZZZZZZZZZZZZ 有人曾将BitVM的思路比作:在《我的世界》里,用红石电路做一个MI处理器。或者说,相当于用积木搭出来纽约帝国大厦。 上一节,定义了与非门承诺。通过组合门承诺来表达任何电路。执行的每一步都在一个 Tapleaf 中提交。它们都合并到同一个 Taproot 地址中,使得证明者可以执行电路中的任 何门。执行一个门需要证明者打开相应的门承诺,并为其输入和输出位设置值。 Taptree 可能变得非常庞大,拥有十亿个 Tapleaf 脚本,但它在链上的足迹却很小。 - A, B, C, D是input, E, F, G, H, I, J, L是witness(过程变量) Prover(证明者)发布一个Binary Circuit Commitment 即发布电路中所有的Logic Gate Commitment 1. <hash c0> <hash c1> <hash b0> <hash b1> <hash a0> <hash a1> OP GATECOMMITMENT; 其中一些包含相同的<hash 0> <hash 1>则表示电路相连 2. 表示其承诺电路中所有Bit Value Commitment打开后符合电路的逻辑约束 每个NAND门都代表着一段Bitcoin Opcode Script 的可执行代码 图3 解释 图3:一个随机示例电路,有8个不同的与非门和4个输入A、B、C和D。使用数十亿个门将允许我们定义基本上任何函数。 Gate Commitments 5二进制电路承诺 **Binary Circuit** (1) A NAND B = E Gate Commitments Commitment C NAND D = F(1) A NAND B = E最后不会去发布整个这个电路,会去发布它的一个 commitment 电路。把电路的每 (3) A NAND C = GC NAND D = F-个逻辑门依次排开,做成了一棵树Taproot树,只发布 Taproot这个树根 root 就 E NAND F = HCircuit (3) A NAND C = GTaproot 计算→Binary Circuit→opcode script→taproot (5) G NAND B = I 实现了发布流程 E NAND F = Hcommitment,编译结果是链上可执行的 Circuit H NAND I = J**Taproot** (5) G NAND B = I J NAND A = LH NAND I = JL NAND D = True (7) J NAND A = L L NAND D = True 图4 图4:对于每个门,证明者的 Taproot 地址包含一个带有相应门承诺的叶脚本。这使证明者可以在以后的任何时候设置电路的输入值(这里是A、B、C和D)。 承诺一个电路是不够的。为了驳斥一个不正确的主张,验证者必须能够挑战证明者的声明。这是通过他们在设置期间预先签署一系列交易来实现的。这些交易像挑战 → 响应 → 挑战 → − 1. 交易的input包含P和V的联合签名,确保交易的inputs和outputs符合各个场景下(challenge、response等)的格式要求 响应……这样链接起来。如果双方中的一方停止参与,那么在一定的超时后,另一方将赢得挑战并可以拿走双方的存款。只要双方合作,他们可以用2-of-2签名共同解决任何合约。以 下机制仅在欺诈情况下需要。 1. output只展示树根,树的每一个节点是一个比特币的Locking Pre-signed off-chain transaction script, 解锁的时候input展示任一叶子(locking script)及其对 P&V sig check + locking script 1 应的unlocking script,执行成功则解锁output;Script3 的 Pre-sign -· 其余unlocking script locking script是2-of-2(A和B)的Schnorr签名;那解锁脚本就 TX 2 **TX 3** TX 1 TX 4 P&V sig check + locking script n 是A的签名+B的签名;运行成功之后就可以解锁整个output 2. output 使用 Taproot技术,只展 示树根,即满足多 2. 既能将复杂计算上链(可以包含很多并行的解锁条件),又 Vicky reveals icky reveals Paul reveals Paul reveals deposit 1 BTC a challenge a challenge a response 节省链上空间和保护隐私(只展示其中的一个解锁条件) 个条件中的任意一 个即可unlock这个 output 3. Taproot也是一种Commitment Scheme icky slashes Vicky takes equivocation Paul takes Paul takes Taproot - challenge 类型的交易 签名格式规范都通过Prover和Verifier的共同签名来保证符合规范。真正需要 Interactive proving 的时候将这些交易上链,那就可以 Pri-sign预签名内容 保证 Interactive proving 按照既定的我们事先约定好的这个流程去进行它。 - esponse 类型 hash3 and NAND3 P&V fund Challenges Taproot hash4 and NAND4 P fund P&V sig check hash5 and NAND5 circuit inputs check hash6 and NAND6 利益驱使下, P会提供有效的reveal, V会challenge其 V fund timelock hash7 and NAND7 有效性,如果通过check则P拿走所有赌注,否则V拿走 + V sig check -个Taproot 代表2解锁条件 - 否则V可以在一段时间后转走赌注 P&V fund 图5:在设置期间生成的一系列预签交易,用于执行多轮挑战和响应。 P reveal Vicky通过打开她的Tapscript叶中的一个哈希锁来选择一个挑战。这为Paul解锁了一个特定的 Tapscript,并迫使他执行它。脚本迫使 Paul 揭示 Vicky 所挑战的门承诺。通过重 P&V sig check 复这个程序几轮查询,任何不一致的主张都可以迅速被驳斥。 reveal circuit inputs + challenge 6 挑战与响应 输入 ——— 电路所有的input 如果证明者停止与验证者的链下合作,验证者需要一种方法在链上强迫他的行动。验证者通过解锁一个哈希锁来做到这一点:证明者的UTXO中的每个与非 Tapleaf 只有在证明者知 Challenges P&V sig check 道验证者持有的一个原像时才能被消费。因此,证明者可以通过揭示其输入和输出来证明给定的 Tapleaf 正确执行,但仅当验证者通过揭示守卫该 Tapleaf 的哈希的原像为他"解 1. 允许V挑战P reveal的所有内容 and Responses equivocation 锁"时才能这样做。应用二分查找,验证者在log(N)轮挑战和响应后就可以快速识别证明者的错误。 2. V如果发现P reveal的内容带入到电路中是错,则可以 输出3中情况 执行Equivocation挑战 Conflicting Values + P sig check 3. 若V根据P reveal的内容没发现异常,则一段时间后P A0 and A1 可以转走赌注 P reveal B0 and B1 Verifier 会去 challenge 具体某一段计算,让这 - CO and C1 个 Prover 去把这段计算上链去运行一下,那这 → D0 and D1 个话则这个 P response 代表 Prover 把所 • E0 and E1 challenge 的那段计算放到链上去运行。 P和V生成这些交易的过程即为commit,因为其中包含了Circuit Taproot。每笔交 对之间的hashlock不同,script一样 GO and G1 易生成的过程需要通过Prover和Verifier的联合签名才能让这个交易的生效;从而保 证在浙西额交易在使用时都符合Interactive Proving的流程规范 **Equivocations** pair内每一行有同样的hashlock Taproot IO and I1 使用两个成对的Taproot树组成 如果challenge Hash7,则Prover需 要提供Hash7的Preimage 和 K0 and K1 P和V interactive proving NADN7的commitment(inputs) L0 and L1 - 如果P无法在链上正确执行challenged gate,V可以在一段时间后转走赌注 图6:在每次响应后,Vicky可以惩罚含糊其辞。如果 Paul 曾经为变量揭示两个矛盾的值,那么 Vicky 立即赢得挑战,并被允许拿走他的存款。Vicky 通过揭示 Paul 的任何一个位 - 如果V没有新的challenge,P可以在一段时间后转走赌注 承诺的两个原像来证明 Paul 的含糊其辞。 P在reveal和response阶段都会reveal一些Bit Value Commitment C0 and C1 D0 and D1 V punish 此时如果V发现P存在equivocation, 即:发现对一 V sig check · 个 Bit value commitment,我现在得拿到了它两 reveal a equivocation 个 PreImage,可以立刻punish \_\_\_ I0 and I1 \_\_\_\_ Kθ and K1 V punish 用电路中任一Bit Value Commitment的两个 preimage解锁 过程 证明者可以通过揭示相应的比特承诺来设置输入。理想情况下,他们会在链外揭示这些承诺,以最小化在链上的占用。在非合作情况下,验证者可以强制证明者在链上揭示他们的 7输入与输出 Inputs and 通过预先交换数据但加密的方式,可以处理大量数据。这样证明者可以在之后的某个时刻揭示解密密钥。 Outputs 多方输入也是可能的。门可以有来自双方的位承诺。

难度

P&V联合pre-sign并不能达到预期的效果。

Optimistic Rollup的实现

- 1. 交易的input只有运行时才能确定,所以无法pre-sign这些交易

把计算编译成Binary Circuit,gate可能太多 ———— 要challenge-response很多次才能找到fraud

3.1. 允许任何人可以生成交易参与interactive proving

3. 感觉改成可信第三方sign更合理

Prover reveal的时候,交易的input的空间可能不够

- 2. 其次,P&V联合sign的话,受利益驱使,P可以不签名以阻止V challenge,V也可以不签名以阻止P response

·计算输入的每个bit都会转化成一个preimage,占用空间很大

Bitcoin一个block中的input用SegWit的话最多支持4MB

3.2. 交易必须经过第三方签名,第三方只保证交易的格式符合流程要求,不保证交易本身的正确性

在简单的NAND电路中表达函数是低效的。通过使用更高级的操作码,程序可以更有效地表达。例如,比特币脚本支持添加32位数字,因此我们无需为此使用二进制电路。我们还

这里提出的模型仅限于两方。然而,可能可以建立双向通道,并将它们链式连接以形成类似于闪电网络的网络。探索双方设置可能会产生一些有趣的泛化可能性。例如,我们可以探 索网络的1对n星型拓扑结构。另一个研究问题是我们是否可以将我们的模型应用于n-of-n设置,并创建更复杂的通道工厂。此外,我们还可以将我们的系统与不同的链下协议结合使

其他研究方向包括跨应用内存、如何对刻在链上的任意数据进行陈述,或链下可编程电路,即链下虚拟机。还有可能应用更复杂的采样技术,类似于STARKs,以在单一轮中检查

比特币在编码欺诈证明的大型 Taptrees 中可以验证任何程序的执行,因此在某种意义上是图灵完备的。这里概述的模型的一个主要限制是它仅限于两方设置。希望这可以在后续工

可以拥有更大的比特承诺,例如,可以在单个哈希中承诺32位。此外,脚本的大小可以达到约4 MB。因此,我们可以在每个叶子脚本中实现远远超过一个NAND指令。

下一个重要的里程碑是完成具体的BitVM设计和实现,以及Tree++,一个用于编写和调试比特币合约的高级语言。

[1] Ethereum Research. Optimistic rollups. https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/, 2022.

[3] Jeremy Rubin. CheckSigFromStack for 5 Byte Values. https://rubin.io/blog/2021/07/02/signing-5-bytes, 2021.

[2] Salvatore Ingala. Merkleize all the things. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2022-November/021182.html, 2022.

8 局限性与展望

Outlook

9 结论

致谢

参考文献

References

Conclusion

Limitations and

用,例如闪电网络或rollups。

电路。

作中得到泛化。

特别感谢Super Testnet和Sam Parker,他始终

拒绝相信比特币不会是图灵完备的。