(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 118435558 A
(43)申请公布日 2024.08.02

(21)申请号 202280067476.X
(22)申请日 2022.09.06
(30)优先权数据
2114286.4 2021.10.06 GB
(85)PCT国际申请进入国家阶段日
2024.04.03
(86)PCT国际申请的申请数据
PCT/EP2022/074687 2022.09.06
(87)PCT国际申请的公布数据
WO2023/057149 EN 2023.04.13
(71)申请人 区块链许可股份公司
地址 瑞士楚格
(72)发明人 穆罕默德·萨比尔·基拉兹
杰克·欧文·戴维斯 刘卫
(74)专利代理机构 北京市竞天公诚律师事务所 11770
专利代理师 孙磊 徐民
(51)Int.Cl.
H04L 9/40(2022.01)
H04L 9/32(2006.01)
G06F 21/64(2013.01)
G06F 21/62(2013.01)

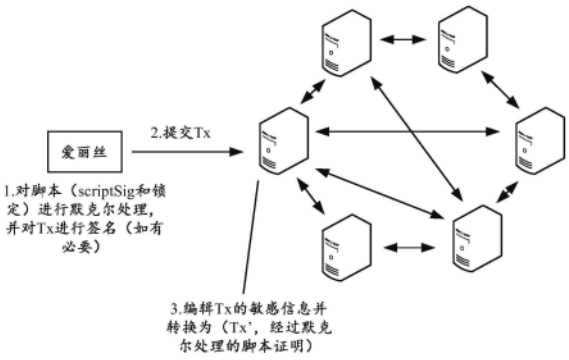
权利要求书3页 说明书32页 附图6页

(54)发明名称

编辑区块链事务的内容

(57)摘要

一种编辑区块链事务的数据的计算机实现的方法,其中所述方法包括:获取第一区块链事务,所述第一区块链事务包括一个或多个相应脚本,所述相应脚本包括待编辑的相应目标数据;对于所述一个或多个相应脚本中的至少一个相应脚本,基于所述相应脚本来构建相应默克尔树,其中所述相应目标数据在所述相应默克尔树的相应叶中的一个或多个相应叶之间划分;以及,通过使用所述相应默克尔树的相应默克尔根替换所述至少一个相应脚本,来生成所述第一区块链事务的编辑版本。



CN 118435558 A

1.一种编辑区块链事务的数据的计算机实现的方法,其中所述方法包括:

获取第一区块链事务,所述第一区块链事务包括一个或多个相应脚本,所述相应脚本包括待编辑的相应目标数据;

对于所述一个或多个相应脚本中的至少一个相应脚本,基于所述相应脚本来构建相应默克尔树,其中所述相应目标数据在所述相应默克尔树的相应叶中的一个或多个相应叶之间划分;以及

通过使用所述相应默克尔树的相应默克尔根替换所述至少一个相应脚本,来生成所述第一区块链事务的编辑版本。

2.根据权利要求1所述的方法,其中每个相应脚本包括一个或多个函数,并且其中所述的构建所述相应默克尔树包括:将一个或多个相应连续函数序列分组为所述默克尔树的相应叶。

3.根据权利要求2所述的方法,其中每个相应脚本包括相应数据函数,所述相应数据函数被配置为在执行时指示所述相应目标数据,并且其中所述的构建所述相应默克尔树包括:包括所述相应数据函数作为所述相应默克尔树的相应叶的一部分。

4.根据权利要求3所述的方法,其中每个相应脚本包括所述相应目标数据的相应数据长度,并且其中所述的构建所述相应默克尔树基于所述相应数据长度。

5.根据权利要求4所述的方法,其中所述的构建所述相应默克尔树包括:包括所述相应数据长度作为所述相应默克尔树的与所述相应数据函数相同的相应叶的一部分。

6.根据前述任一项权利要求所述的方法,其中所述相应目标数据在所述相应默克尔树的多个所述相应叶之间划分。

7.根据前述任一项权利要求所述的方法,其中所述第一区块链事务包括多个相应脚本,所述多个相应脚本包括相应目标数据,并且其中所述方法包括:

对于所述多个相应脚本中的每个相应脚本,基于所述相应脚本来构建相应默克尔树,其中所述相应目标数据在所述相应默克尔树的所述相应叶中的一个或多个相应叶之间划分;

通过使用所述相应默克尔树的相应默克尔根替换每个相应脚本来生成所述第一区块链事务的所述编辑版本。

8.根据前述任一项权利要求所述的方法,所述方法包括:通过对所述区块链事务的所述编辑版本进行哈希处理来生成所述第一区块链事务的所述编辑版本的初级事务标识符。

9.根据前述任一项权利要求所述的方法,所述方法包括:响应于对所述第一区块链事务的请求,将所述区块链事务的所述编辑版本传输给实体。

10.根据前述任一项权利要求所述的方法,所述方法包括:将所述第一区块链事务的所述编辑版本存储在存储器中。

11.根据前述任一项权利要求所述的方法,其中所述方法由事务生成者执行。

12.根据权利要求11所述的方法,所述方法包括:

为消息生成签名,其中所述消息的一个或多个部分基于所述第一区块链事务的所述编辑版本;

将所述签名包括在所述第一区块链事务的输入中;以及

将所述第一区块链事务提交给区块链网络进行核实。

13.根据权利要求12所述的方法,其中所述签名是椭圆曲线数字签名算法(ECDSA)签名。

14.根据权利要求12或13所述的方法,其中所述输入引用先前区块链事务的输出,所述先前区块链事务包括相应脚本,所述相应脚本包括相应目标数据,其中所述消息的一部分基于所述先前区块链事务的所述输出,并且其中基于所述先前区块链事务的所述输出的所述消息的所述部分基于默克尔树的默克尔根,所述默克尔树基于所述相应脚本来构建,其中所述相应目标数据在所述相应默克尔树的所述相应叶中的一个或多个相应叶之间划分。

15.根据权利要求1至10中任一项所述的方法,其中所述方法由事务核实者执行。

16.根据权利要求15所述的方法,所述方法包括:

从所述第一区块链事务的输入中提取签名;以及

验证消息的所述签名,其中所述消息的一个或多个部分基于所述第一区块链事务的所述编辑版本。

17.根据权利要求16所述的方法,其中所述输入引用先前区块链事务的输出,所述先前区块链事务包括相应脚本,所述相应脚本包括相应目标数据,其中所述消息的一部分基于所述先前区块链事务的所述输出,并且其中基于所述先前区块链事务的所述输出的所述消息的所述部分基于默克尔树的默克尔根,所述默克尔树基于所述相应脚本来构建,其中所述相应目标数据在所述相应默克尔树的所述相应叶中的一个或多个相应叶之间划分。

18.根据权利要求15或其任何从属权利要求所述的方法,所述方法包括:将所述第一区块链事务的所述编辑版本存储在区块链上。

19.根据权利要求8和权利要求15至18中任一项所述的方法,其中所述相应脚本中的至少一个相应脚本是所述第一区块链事务的第一输出的第一锁定脚本,并且其中所述方法包括:

在数据库中存储:相应默克尔根,所述相应默克尔根映射到所修改的初级事务,和所述第一输出的相应索引。

20.根据权利要求19所述的方法,所述方法包括:

获取第二区块链事务,其中所述第二区块链事务包括第一输入,所述第一输入引用所述第一区块链事务的所述修改的初级事务标识符,和所述第一输出的所述索引,其中所述第一输入包括第一解锁脚本,并且其中所述方法包括通过以下方式来核实所述第二区块链事务:

从所述数据库中检索:所述相应默克尔根,所述相应默克尔根映射到所述修改的初级事务标识符,和所述第一输出的所述相应索引;

基于所述第一输入的所述第一解锁脚本来确定所述第一输出的部分锁定脚本;

执行默克尔包含证明,以确认所述部分锁定脚本是与所述相应默克尔根对应的所述相应默克尔树的相应叶;以及

根据所述部分锁定脚本来核实所述第一解锁脚本。

21.根据权利要求8和权利要求15至18中任一项所述的方法,其中所述相应脚本中的至少一个相应脚本是所述第一区块链事务的第一输出的第一锁定脚本,并且其中所述方法包括:

在数据库中存储:部分锁定脚本,所述部分锁定脚本映射到所述修改的初级事务标识

符,和所述第一输出的相应索引,其中所述部分锁定脚本包括至少部分所述第一锁定脚本但不包括所述目标数据。

22.根据权利要求21所述的方法,所述方法包括:

获取第二区块链事务,其中所述第二区块链事务包括第一输入,所述第一输入引用所述第一区块链事务的所述修改的初级事务标识符,和所述第一输出的所述索引,其中所述第一输入包括第一解锁脚本,并且其中所述方法包括通过以下方式来核实所述第二区块链事务:

从所述数据库中检索:相应部分锁定脚本,所述相应部分锁定脚本映射到所述修改的初级事务标识符,和所述第一输出的所述相应索引;以及

根据所述部分锁定脚本来核实所述第一解锁脚本。

23.根据权利要求19和21所述的方法,所述方法包括:

从所述数据库中检索:所述相应默克尔根和所述部分锁定脚本,其映射到所述修改的初级事务标识符,和所述第一输出的所述相应索引;

基于所述第一输入的所述第一解锁脚本来确定所述第一输出的部分锁定脚本;

使用基于所述检索的部分锁定脚本生成的一个或多个相应叶来获取所述确定的部分锁定脚本的默克尔证明;

执行默克尔包含证明,以确认所述确定的部分锁定脚本是与所述相应默克尔根对应的所述相应默克尔树的相应叶;以及

根据所述确定的部分锁定脚本来核实所述第一解锁脚本。

24.根据前述任一项权利要求所述的方法,其中所述编辑的事务包括多个字段,并且其中所述方法包括:

生成事务默克尔树,其中所述事务默克尔树的相应多个相应叶由所述编辑的事务的一个或多个字段形成;以及

生成所述编辑的事务的二级事务标识符,其中所述二级事务标识符包括所述事务默克尔树的默克尔根。

25.根据权利要求24所述的方法,其中所述事务默克尔树的一个或多个叶由所述编辑的事务的多个字段形成。

26.根据权利要求24或25所述的方法,其中所述事务默克尔树的一个或多个叶由所述编辑的事务的单个字段形成。

27.根据权利要求24或其任何从属权利要求所述的方法,其中所述事务默克尔树的一个或多个叶由所述编辑的事务的相应字段的相应部分形成,而不是由完整的相应字段形成。

28.一种计算机设备,所述计算机设备包括:

存储器,所述存储器包括一个或多个存储器单元;以及

处理装置,所述处理装置包括一个或多个处理单元,其中所述存储器存储被设置在所述处理装置上运行的代码,所述代码被配置为当在所述处理装置上运行时,执行根据权利要求1至27中任一项所述的方法。

29.一种计算机程序,所述计算机程序包含在计算机可读存储器上并且被配置为当在一个或多个处理器上运行时,执行根据权利要求1至27中任一项所述的方法。

编辑区块链事务的内容

技术领域

[0001] 本公开涉及一种对存储在区块链事务中的内容进行编辑(redact)的方法。例如,该方法可以用于隐藏敏感或私有数据。

背景技术

[0002] 区块链是指一种分布式数据结构,其中在分布式对等(P2P)网络(以下称为“区块链网络”)中的多个节点中的每个节点处维护区块链的副本,并且广泛公开该副本。区块链包括一系列数据区块,其中每个区块包括一个或多个事务(transaction)。除所谓的“coinbase事务”外,每个事务都指向序列中的先前事务,该序列可以跨越一个或多个区块,回到一个或多个coinbase事务。coinbase事务将在下文进一步讨论。提交给区块链网络的事务包括在新区块中。新区块的创建过程通常称为“挖掘”,该过程涉及多个节点中的每个节点争相执行“工作证明”,即,基于等待被包括在区块链的新区块中的一组定义的有序且核实有效的未决事务的表示解决加密难题。应当注意的是,区块链可以在一些节点处被修剪(prune),并且区块的发布可以通过仅发布区块头来实现。

[0003] 区块链中的事务可用于以下目的中的一个或多个:传送数字资产(即,一定数量的数字令牌);对虚拟化分类账或注册表中的一组条目进行排序;接收和处理时间戳条目;和/或对索引指针按时间排序。也可利用区块链实现区块链上的层级附加功能。例如,区块链协议可允许在事务中存储附加的用户数据或数据索引。能够存储在单个事务中的最大数据容量没有预先指定的限制,因此可以并入越来越复杂的数据。例如,这可用于在区块链中存储电子文档、音频或视频数据。

[0004] 区块链网络的节点(通常称为“矿工”)执行分布式事务注册和验证过程,这将后续更详细地描述。总之,在该过程中,节点核实事务并将这些事务插入到区块模板中,这些事务尝试为该区块模板标识有效的工作证明解。一旦找到有效的解,新区块便会被传播到网络的其它节点,从而使得每个节点能够在区块链上记录新区块。为了将事务记录在区块链中,用户(例如,区块链客户端应用程序)将该事务发送到网络中的节点中的一个节点进行传播。接收该事务的节点可以争相寻找将核实有效的事务并入新区块的工作证明解。每个节点被配置为执行相同的节点协议,该协议将包括用于确认事务有效的一个或多个条件。无效事务将不会传播或并入到区块中。假定事务已经核实有效,从而在区块链上被接受,则该事务(包括任何用户数据)将因此在区块链网络中的每个节点上作为不可改变的公共记录进行注册和索引。

[0005] 成功解决工作证明难题可创建最新区块的节点通常被奖励一个称为“coinbase事务”的新事务,该事务分发数字资产数额,即令牌数量。无效事务的检测和拒绝是通过竞争节点的行动来执行的,这些竞争节点充当网络的代理并且通过激励报告和阻止不正当行为。信息的广泛发布使得用户可以连续地审计节点的性能。仅发布区块头使得参与者可以确保区块链具有持续完整性。

[0006] 在“基于输出的”模型(有时称为基于UTXO的模型)中,给定事务的数据结构包括一

个或多个输入和一个或多个输出。任何可花费输出包括指定数字资产数额的元素,该元素可从进行中的事务序列导出。可花费输出有时称为UTXO(“未花费事务输出”)。输出还可以包括锁定脚本,该锁定脚本指定输出的未来赎回条件。锁定脚本是限定核实和传送数字令牌或资产所必需的条件的谓词。事务(除coinbase事务之外)的每个输入包括指向先前事务中的此类输出的指针(即引用),并且还可以包括解锁脚本,用于解锁指向输出的锁定脚本。因此,考虑一对事务,将其称为第一事务和第二事务(或“目标”事务)。第一事务包括指定数字资产数额的至少一个输出,并且包括定义解锁该输出的一个或多个条件的锁定脚本。第二目标事务包括至少一个输入和解锁脚本,该至少一个输入包括指向第一事务的输出的指针;该解锁脚本用于解锁第一事务的输出。

[0007] 在此类模型中,当第二目标事务被发送到区块链网络以在区块链中传播和记录时,在每个节点处应用的有效性条件之一将是解锁脚本满足在第一事务的锁定脚本中定义的一个或多个条件中的所有条件。另一条件将是第一事务的输出尚未被另一早期有效事务赎回。根据这些条件中的任何一个条件发现目标事务无效的任何节点都不会传播该事务(作为有效事务,但可能注册无效事务),也不将该事务包括在要记录在区块链中的新区块中。

[0008] 另一种事务模型是基于账户的模型。在这种情况下,每个事务均不通过参考过去事务序列中先前事务的UTXO来定义转移的数额,而是通过参考绝对账户余额进行定义。所有账户的当前状态由节点单独存储到区块链中,并不断更新。

发明内容

[0009] 区块链被设计为不可变,这意味着一旦事务写入区块链,就会被视为不可能改变。然而,事务数据可能包含私有或敏感信息,可能需要加以保护,防止未经授权的访问。可能被视为私有或敏感的数据在本文中称为“秘密数据”,由于需要避免分发或授予对此类数据的访问权限,将这些数据称为“秘密”数据。因此,如果秘密数据已经写入所述区块链,就会出现关于如何处理所述秘密数据的问题。同样重要的是,防止在所述区块链上共享秘密数据,例如妥善保护私有数据并遵守数据保护法。所述问题还可以表述为如何删除或限制对存储在区块链事务中的所述秘密数据的访问,同时确保可以证明任何剩余(即,非秘密)数据属于所述事务,并且可以核实并在未来事务中使用所述事务。

[0010] 根据本文公开的一个方面,提供了一种编辑区块链事务的数据的计算机实现的方法,其中所述方法包括:获取第一区块链事务,所述第一区块链事务包括一个或多个相应脚本,所述相应脚本包括拟编辑的相应目标数据;对于所述一个或多个相应脚本中的至少一个相应脚本,基于所述相应脚本来构建相应默克尔树,其中所述相应目标数据被划分于所述相应默克尔树的相应叶中的一个或多个相应叶之间;以及,通过使用所述相应默克尔树的相应默克尔根替换所述至少一个相应脚本,来生成所述第一区块链事务的编辑版本(redacted version)。

[0011] 所述方法使得能够使用默克尔树来编辑(或删除或隐藏)区块链事务中的目标数据(例如,秘密数据)。所述事务包含至少一个脚本。所述脚本可以在事务的输入(解锁脚本)或输出(锁定脚本)中。所述脚本包含目标数据—待要编辑的数据。所述目标数据可以是如上所述的秘密数据,所述秘密数据是私有数据或敏感数据。通常,所述目标数据可以是要从

事务中删除的任何数据,而不必是私有数据或敏感数据。例如,所述目标数据因其大小可能需要从所述事务中删除,例如所述目标数据可能包含较大的图像或音频文件。又如,所述目标数据可能需要从所述事务中删除,因为所述目标数据包含错误或其他有害数据(在可能对存储或处理所述数据的计算机造成损害的意义上是有害的)。

[0012] 基于所述脚本,即使用所述脚本来构建默克尔树。所述脚本的不同部分用作所述默克尔树的不同叶。所述默克尔树的至少一个叶包括所述目标数据的一部分。在一些示例中,单个叶节点包括完整目标数据。在其他示例中,所述目标数据在多个叶之间划分。如本领域已知的,对所述默克尔树的叶进行哈希处理以形成叶哈希,对成对的叶哈希进行级联和哈希处理以形成相应内部哈希。重复执行对成对内部哈希进行级联和哈希处理的过程,直到保留单个哈希—所述默克尔根。使用所述默克尔根来替换所述脚本,以形成所述事务的编辑版本—“编辑的事务redacted transaction”。因此,使用所述默克尔根来替换包含所述目标数据的所述脚本,可以从所述事务中编辑或删除所述目标数据。

[0013] 可以存储和/或共享所述编辑的事务而不是所述事务的原始版本,从而防止存储和/或分发所述目标(例如,秘密)数据。

[0014] 所述方法可以由事务生成者(例如,用户)执行。例如,可以执行所述方法,以便基于所述编辑的事务来生成签名。所述方法还可以由事务核实者(例如,区块链节点)执行。例如,可以执行所述方法,以便基于所述编辑的事务来验证签名,为所述编辑的事务生成事务标识符,以及/或者核实所述编辑的事务。

[0015] 在一些实施例中,所述方法可以用于实现一种选择性公开机制,允许证明者(prover)选择要向验证者(verifier)公开的事务内容的子集,同时某些内容保持隐藏。所述机制与脚本无关,并且有助于选择性地公开任何脚本机制嵌入的内容。

附图说明

[0016] 为了帮助理解本公开的实施例并示出如何实施此类实施例,现将仅通过举例的方式参考附图进行说明,其中:

[0017] 图1是一种用于实现区块链的系统的示意性框图;

[0018] 图2示意性地示出了可记录在区块链中的事务的一些示例;

[0019] 图3示意性地示出了包含嵌入在不可花费输出中的数据的示例性区块链事务的结构;

[0020] 图4示意性地示出了对支付到公钥哈希(P2PKH)脚本进行默克尔处理的示例;

[0021] 图5示意性地示出对不可花费输出脚本进行默克尔处理的示例;

[0022] 图6是用于花费经过默克尔处理的事务的示例性高级架构的示意性框图,其中创建了花费先前经过默克尔处理的脚本的事务,广播了该事务,并且节点使用编辑的上一个事务根据先前经过默克尔处理的脚本来核实该事务;

[0023] 图7示意性地示出了编辑的事务示例,其中编辑的字段由虚实线框指示;

[0024] 图8示意性地示出了为编辑的事务生成二级标识符的示例。

具体实施方式

[0025] 1.示例性系统概述

[0026] 图1示出了一种用于实现区块链150的示例性系统100。系统100可以包括分组交换网络101,通常是诸如互联网的广域互联网。分组交换网络101包括多个区块链节点104,该多个区块链节点可以被设置成在分组交换网络101内形成对等(P2P)网络106。虽然未示出,但是区块链节点104可以被设置为近完全图。因此,每个区块链节点104高度连接到其它区块链节点104。

[0027] 每个区块链节点104包括对等体的计算机设备,不同的节点104属于不同的对等体。每个区块链节点104包括处理装置,该处理装置包括一个或多个处理器,例如一个或多个中央处理单元(CPU)、加速器处理器、专用处理器和/或现场可编程门阵列(FPGA),以及其它设备,例如专用集成电路(ASIC)。每个节点还包括存储器,即采用非暂时性计算机可读介质形式的计算机可读存储器。存储器可包括一个或多个存储器单元,其采用一个或多个存储器介质,例如诸如硬盘等磁介质、诸如固态硬盘(SSD)、闪存或电可擦可编程只读存储器(EEPROM)等电子媒介和/或诸如光盘驱动器等光学介质。

[0028] 区块链150包括一系列数据区块151,其中在分布式或区块链网络106中的多个区块链节点104中的每个节点处维护区块链150的相应副本。如上所述,维护区块链150的副本不一定意味着完全存储区块链150。相反,只要每个区块链节点150存储每个区块151的区块头(下面讨论),区块链150就可以进行数据修剪。区块链中的每个区块151均包括一个或多个事务152,其中该上下文中的事务是指一种数据结构。数据结构的性质将取决于用作事务模型或计划的一部分的事务协议类型。给定的区块链全程使用一个特定的事务协议。在一种常见的事务协议中,每个事务152的数据结构至少包括一个输入和至少一个输出。每个输出指定将数字资产的数量表示为财产的数额,其一个示例是输出被密码锁定到的用户103(需要该用户的签名或其它解进行解锁,从而进行赎回或花费)。每个输入指向先前事务152的输出,从而链接这些事务。

[0029] 每个区块151还包括区块指针155,其指向区块链中先前创建的区块151,以定义区块151的顺序。每个事务152(除coinbase事务之外)包括指向先前事务的指针,以定义事务序列的顺序(注:事务152的序列可进行分支)。区块151的区块链一直追溯到创始区块(Gb)153,该创始区块是区块链中的第一区块。区块链150中早期的一个或多个原始事务152指向创始区块153,而非先前事务。

[0030] 每个区块链节点104被配置为将事务152转发到其它区块链节点104,从而使得事务152在整个网络106中传播。每个区块链节点104被配置为创建区块151,并将相同区块链150的相应副本存储在其相应的存储器中。每个区块链节点104还维护等待并入到区块151中的事务152的有序集(或“池”)154。有序池154通常称为“内存池”。在本文中,该术语并不意在限制于任何特定的区块链、协议或模型。该术语是指节点104已接受为有效的有序事务集,并且对于该有序事务集,强制节点104不接受试图花费相同输出的任何其它事务。

[0031] 在给定的当前事务152j中,输入(或每个输入)包括指针,该指针引用事务序列中先前事务152i的输出,指定该输出将在当前事务152j中被赎回或“花费”。花费或赎回并不一定意味着转移金融资产,尽管这肯定是一种常见的应用。更一般地,可以将花费描述为消耗输出,或者将其分配给另一个后续事务中的一个或多个输出。通常,先前事务可以是有序集154或任何区块151中的任何事务。尽管为了确保当前事务有效,将需要存在先前事务152i并核实其有效,但是在创建当前事务152j甚至向网络106发送当前事务152j时,不必存

在先前事务152i。因此,在本文中,“先前”是指由指针链接的逻辑序列中的前任,而不一定是时间序列中的创建时间或发送时间,因此,不一定排除无序创建或发送事务152i、152j的情况(参见下面关于孤立事务的讨论)。先前事务152i同样可以称为先行事务或前任事务。

[0032] 当前事务152j的输入还包括输入授权,例如先前事务152i的输出被锁定到的用户103a的签名。反过来,当前事务152j的输出可以加密锁定到新用户或实体103b。因此,当前事务152j可将先前事务152i的输入中定义的数额转移到当前事务152j的输出中定义的新用户或实体103b。在某些情况下,事务152可具有多个输出,以在多个用户或实体间分割输入数额(其中一个可以是原始用户或实体103a,以便进行变更)。在某些情况下,事务还可以具有多个输入,将一个或多个先前事务的多个输出中的数额汇总在一起,并重新分配到当前事务的一个或多个输出。

[0033] 根据基于输出的事务协议,例如比特币,当诸如个体用户或组织这类的一方103希望颁布新的事务152j时(由该方采用的自动程序或人为地),该颁布方将该新事务从其计算机终端102发送到接收者。颁布方或接收者将最终向网络106的一个或多个区块链节点104(现在通常是服务器或数据中心,但原则上也可以是其它用户终端)发送该事务。另外还不排除颁布新事务152j的一方103可以将事务直接发送到一个或多个区块链节点104,并且在一些示例中,可以不将事务发送到接收者。接收事务的区块链节点104根据在每个区块链节点104处应用的区块链节点协议来检查事务是否有效。区块链节点协议通常要求区块链节点104检查新事务152j中的加密签名是否与预期签名相匹配,这取决于事务152的有序序列中的先前事务152i。在这种基于输出的事务协议中,这可以包括检查新事务152j的输入中包括的一方103的密码签名或其它授权是否与新事务花费(或“分配”)的先前事务152i的输出中定义的条件匹配,其中该条件通常包括至少检查新事务152j的输入中的密码签名或其它授权是否解锁新事务的输入所链接到的先前事务152i的输出。条件可以至少部分地由包括在先前事务152i的输出中的脚本来定义。或者,这可仅由区块链节点协议单独确定,或可通过其组合确定。无论采用哪种方式,如果新事务152j有效,区块链节点104会将其转发到区块链网络106中的一个或多个其它区块链节点104。这些其它区块链节点104根据相同的区块链节点协议应用相同的测试,并因此将新事务152j转发到一个或多个其它节点104等等。通过这种方式,新事务在区块链节点104的整个网络中进行传播。

[0034] 在基于输出的模型中,给定输出(例如,UTXO)是否分配(或“花费”)的定义是,根据区块链节点协议,其是否通过另一个随后事务152j的输入有效赎回。事务有效的另一个条件是其试图赎回的先前事务152i的输出尚未被另一个事务赎回。同样,如果无效,则事务152j将不会在区块链150中传播(除非被标记为无效并且被传播用于提醒)或记录。这可防止重复花费,即事务处理者对同一个事务的输出分配超过一次。另一方面,基于账户的模型通过保持账户余额防止重复花费。因为同样存在定义的事务顺序,账户余额在任何时候均具有单一定义的状态。

[0035] 除了核实事务有效之外,区块链节点104还争相成为在通常称为挖掘的过程中创建事务区块的第一个节点,而该过程由“工作证明”支持。在区块链节点104处,新事务被添加到尚未出现在记录在区块链150上的区块151中的有效事务的有序池154。然后,区块链节点争相通过尝试解决加密难题以组装有序事务集154中事务152的新有效事务区块151。通常情况下,这包括搜索“随机数”值,从而当随机数与未决事务有序池154的表示并置且进行

哈希处理时,哈希值的输出满足预定条件。例如,预定条件可以是哈希值的输出具有某个预定义的前导零数。注意,这仅仅是一种特定类型的工作证明难题,并且不排除其它类型。哈希函数的特性是,相对于其输入,其具有不可预测的输出。因此,该搜索只能通过强力执行,从而在试图解决难题的每个区块链节点104处消耗大量的处理资源。

[0036] 解决难题的第一区块链节点104在网络106上宣布难题解决,提供解决方案作为证明,然后网络中的其它区块链节点104则可以轻松检查该解决方案(一旦给出哈希值的解决方案,就可以直接检查该解决方案是否使哈希值的输出满足条件)。第一区块链节点104将一个区块传播到接受该区块的其它节点达成阈值共识,从而执行协议规则。然后,有序事务集154被每个区块链节点104记录为区块链150中的新区块151。区块指针155还分配给指向该区块链中先前创建的区块151n-1的新区块151n。创建工作证明解所需的大量工作(例如采用哈希的形式)发出信号通知第一节点104的意图以遵循区块链协议。这些规则包括如果它花费或分配与先前核实有效的事务相同的输出,则不接受事务为有效,否则称之为重复花费。一旦创建,区块151就不能修改,因为它在区块链网络106中的每个区块链节点104处进行标识和维护。区块指针155还向区块151施加顺序。由于事务152记录在网络106中每个区块链节点104处的有序区块中,因此提供了事务的不可改变公共分类账。

[0037] 应当注意的是,在任何给定时间争相解决难题的不同区块链节点104可以基于在任何给定时间尚未发布的事务的池154的不同快照来这样做,具体取决于它们何时开始搜索解或接收事务的顺序。解决相应难题的人员首先定义新区块151n中包括的事务152及其顺序,并且更新当前的未发布事务池154。然后,区块链节点104继续争相从新定义的未发布事务有序池154中创建区块,等等。此外,还存在解决可能出现的任何“分叉”的协议,其中两个区块链节点104彼此在很短的时间内解决难题,从而在节点104之间传播区块链的冲突视图。简言之,分叉方向最长的成为最终区块链150。应当注意的是,这不会影响网络的用户或代理,因为同一事务将出现在两个分叉中。

[0038] 根据比特币区块链(和大多数其它区块链),成功构造新区块104的节点被授予在分配附加限定数量数字资产的新特殊类型事务中新分配附加的、接受的数额的数字资产的能力(与代理间或用户间事务相反,该事务将一定数量的数字资产从一个代理或用户转移到另一个代理或用户)。这种特殊类型的事务通常称为“coinbase事务”,但是也可以称为“启动事务”或“产生事务”。它通常形成新区块151n的第一事务。工作证明发出信号通知构造新区块的节点的意图以遵循协议规则,从而允许稍后赎回该特定事务。在可以赎回该特殊事务之前,区块链协议规则可能需要成熟期,例如100个区块。通常,常规(非生成)事务152还将在其输出中的一个输出中指定附加事务费用,以进一步奖励创建其中发布该事务的区块151n的区块链节点104。该费用通常称为“事务费用”,并在下文中讨论。

[0039] 由于事务核实和发布中涉及的资源,通常至少每个区块链节点104采用包括一个或多个物理服务器单元的服务器的形式,或者甚至整个数据中心。但是,原则上来说,任何给定区块链节点104均可采用一个用户终端或联网在一起的一组用户终端的形式。

[0040] 每个区块链节点104的存储器均存储被配置为在区块链节点104的处理装置上运行的软件,以根据区块链节点协议执行其相应的角色并处理事务152。应当理解的是,在本文中归因于区块链节点104的任何动作均可通过在相应计算机设备的处理装置上运行的软件执行。节点软件可以在应用层或诸如操作系统层或协议层的较低层或这些层任意组合的

一个或多个应用中实现。

[0041] 扮演消费用户角色的多方103中的每一方的计算机设备102也连接到网络101。这些用户可以与区块链网络106交互,但不参与核实事务或构造区块。其中一些用户或代理103可以充当事务中的发送者和接收者。其它用户可以与区块链150交互,而不必充当发送者或接收者。例如,一些当事方可以充当存储区块链150的副本(例如,已经从区块链节点104获得区块链的副本)的存储实体。

[0042] 各方103中的一些或所有当事方可以作为不同网络的一部分连接,例如覆盖在区块链网络106之上的网络。区块链网络的用户(经常称为“客户端”)可以被称为是包含区块链网络106的系统的一部分;然而,这些用户不是区块链节点104,因为它们不执行区块链节点所需的角色。相反,每一方103可以与区块链网络106交互,从而通过连接到区块链节点106(即,与区块链节点106通信)来利用区块链150。出于说明目的,示出了双方103及其相应的设备102:第一方103a及其相应的计算机设备102a,以及第二方103b及其相应的计算机设备102b。应当理解的是,更多此类当事方103及其相应的计算机设备102可能存在并参与系统100,但为了方便起见,未进行说明。每一方103均可以是个人或组织。仅出于说明目的,在本文中,第一方103a称为爱丽丝,第二方103b称为鲍勃,但应当理解的是,这并不仅限于爱丽丝或鲍勃,且本文对爱丽丝或鲍勃的任何引用均可分别用“第一方”和“第二方”替换。

[0043] 每一方103的计算机设备102包括相应的处理装置,其包括一个或更多个处理器,例如一个或更多个CPU、图形处理单元(GPU)、其他加速器处理器、特定应用程序处理器和/或FPGA。每一方103的计算机设备102还包括存储器,即采用非暂时性计算机可读介质形式的计算机可读存储器。该存储器可包括一个或更多个存储器单元,其采用一个或更多个存储器介质,例如诸如硬盘等磁介质、诸如SSD、闪存或EEPROM等电子媒介和/或诸如光盘驱动器等的光学介质。每一方103的计算机设备102上的存储器存储软件,其包括被设置为在处理装置上运行的至少一个客户端应用程序105的相应实例。应当理解的是,在本文中归因于给定方103的任何行动均可通过在相应计算机设备102的处理装置上运行的软件执行。每一方103的计算机设备102包括至少一个用户终端,例如台式或笔记本电脑、平板电脑、智能手机或诸如智能手表等的可穿戴设备。给定方103的计算机设备102还可包括一个或更多个其他网络资源,诸如通过用户终端访问的云计算资源。

[0044] 客户端应用程序105最初可通过例如从服务器下载的适当计算机可读存储介质,或通过诸如可移动SSD、闪存密钥、可移动EEPROM、可移动磁盘驱动器、软盘或磁带等的可移动存储设备、诸如CD或DVD ROM等的光盘或可移动光驱等提供至任何给定方103的计算机设备102。

[0045] 客户端应用程序105至少包括“钱包”功能。这有两个主要功能。其中一个功能是使相应方103能够创建、授权(例如签名)事务152并将其发送到一个或多个比特币节点104,然后在区块链节点104的网络中传播,从而包括在区块链150中。另一个功能是向相应方汇报其目前拥有的数字资产数额。在基于输出的系统中,该第二功能包括整理分散在区块链150中属于相关方的各种事务152的输出中定义的数额。

[0046] 注意:虽然各种客户端功能可以描述为集成到给定客户端应用程序105中,但这不一定是限制性的,相反,在本文中所描述的任何客户端功能可以在由两个或更多个不同应用程序组成的套件中实现,例如经由API进行接口连接或一个应用程序作为另一个应用程

序的插件。更通俗地说,客户端功能可以在应用层或诸如操作系统的较低层或这些层的任意组合实现。下面将根据客户端应用程序105进行描述,但应当理解的是,这不是限制性的。

[0047] 每个计算机设备102上的客户端应用程序或软件105的实例可操作地耦合到网络106的区块链节点104中的至少一个。这可以启用客户端105的钱包功能,以将事务152发送至网络106。客户端105还可联络区块链节点104,以在区块链150中查询相应方103作为接收者的任何事务(或实际上在区块链150中检查其它方的事务,因为在实施例中,区块链150是在某种程度上通过其公开可见性提供事务信任的公共设施)。每个计算机设备102上的钱包功能被配置为根据事务协议制定和发送事务152。如上所述,每个区块链节点104运行软件,该软件被配置为根据区块链节点协议核实事务152并转发事务152以便在区块链网络106中传播。事务协议和节点协议相互对应,给定事务协议和给定节点协议一起实现给定的事务模型。相同的事务协议用于区块链150中的所有事务152。网络106中的所有节点104使用相同的节点协议。

[0048] 当给定方103(比方说爱丽丝)希望发送拟包含在区块链150中的新事务152j时,她将根据相关事务协议(使用其客户端应用程序105中的钱包功能)制定新事务。然后,她将事务152从客户端应用程序105发送到她所连接的一个或多个区块链节点104。例如,这可能是与爱丽丝的计算机102最佳连接的区块链节点104。当任何给定区块链节点104接收新事务152j时,其将根据区块链节点协议及其相应的角色进行处理。这包括首先检查新接收的事务152j是否满足变为“有效”的特定条件,具体示例稍后将详细讨论。在一些事务协议中,有效条件可通过事务152中包含的脚本在每个事务的基础上进行配置。或者,条件可仅仅是节点协议的内置功能,或通过组合脚本和节点协议进行定义。

[0049] 如果新接收的事务152j通过有效性测试(即:“有效”的条件下),接收事务152j的任何区块链节点104将向在区块链节点104处维护的有序事务集154中添加新的核实有效事务152。进一步地,接收事务152j的任何区块链节点104随后将核实有效事务152传播至网络106中的一个或多个其它区块链节点104。由于每个区块链节点104应用相同的协议,因此假定事务152j有效,这意味着事务很快将在整个网络106中传播。

[0050] 一旦进入在给定区块链节点104处维护的未决事务有序池154,该区块链节点104将开始争相解决其各自的包含新事务152的池154的最新版本上的工作证明难题(请记住,其它区块链节点104可以尝试基于不同的事务池154来解决难题。但是,首先解决难题的人将定义包括在最新区块151中的事务集合。最终,区块链节点104将解决有序池154的一部分的难题,该有序集154包括爱丽丝的事务152j)。一旦包括新事务152j的池154完成工作证明,其将不可变地成为区块链150中区块151中的一个区块的一部分。每个事务152包括指向早前事务的指针,因此事务的顺序也被不可变地记录下来。

[0051] 不同的区块链节点104可以首先接收给定事务的不同实例,并且因此在一个实例被发布到新区块151中之前具有关于哪个实例“有效”的冲突视图,此时所有区块链节点104同意所发布的实例是唯一的有效实例。如果区块链节点104将一个实例接受为有效实例,然后发现第二实例已记录在区块链150中,则区块链节点104必须接受这一点,并将丢弃(即,视为无效)其最初接受的实例(即,在区块151中尚未公布的实例)。

[0052] 作为基于账户的事务模型的一部分,由一些区块链网络操作的另一种类型的事务协议可称为“基于账户的”协议。在基于账户的情况下,每个事务均不通过参考过去事务序

列中先前事务的UTXO来定义转移的数额,而是通过参考绝对账户余额进行定义。所有账户的当前状态由网络的节点单独存储到区块链中,并不断更新。在此类系统中,事务使用账户的运行事务记录(也称为“头寸”)进行排序。该值由发送者签名作为其加密签名的一部分,并作为事务引用计算的一部分进行哈希处理。此外,可选的数据字段也可以在事务中签名。例如,如果数据字段中包含先前事务的ID,该数据字段可指向先前事务。

[0053] 2.基于UTXO的模型

[0054] 图2示出了示例性事务协议。这是基于UTXO的协议的示例。事务152(简称“Tx”)是区块链150的基本数据结构(每个区块151包括一个或多个事务152)。下面将通过参考基于输出或基于“UTXO”的协议进行描述。但这并不限于所有可能的实施例。应当注意的是,虽然参考比特币描述了示例性基于UTXO的协议,但是它同样可以在其它示例区块链网络上实现。

[0055] 在基于UTXO的模型中,每个事务(“Tx”)152包括数据结构,其包括一个或多个输入202和一个或多个输出203。每个输出203可包括未花费事务输出(UTXO),其可用作另一新事务的输入202的来源(如果UTXO尚未赎回)。UTXO包括指定数字资产数额的值。这表示分布式分类账上的一组令牌。UTXO还可包含其来源事务的事务ID以及其它信息。事务数据结构还可包括标头201,其可包括输入字段202和输出字段203的大小指示符。标头201还可包括事务的ID。在实施例中,事务ID是事务数据(不含事务ID本身)的哈希值,且存储在提交至节点104的原始事务152的标头201中。

[0056] 比方说爱丽丝103a希望创建转移相关数字资产数额至鲍勃103b的事务152j。在图2中,爱丽丝的新事务152j标记为“Tx_1”。该新事务获取在序列中先前事务152i的输出203中锁定至爱丽丝的数字资产数额,并至少将此类数额中的一部分转移至鲍勃。在图2中,先前事务152i标记为“Tx_0”。Tx_0和Tx_1只是任意的标记,其不一定意味着Tx_0指区块链151中的第一事务且Tx_1指池154中的后续事务。Tx_1可指向仍具有锁定至爱丽丝的未花费输出203的任何先前(即先行)事务。

[0057] 当爱丽丝创建其新事务Tx_1时,或至少在她将该新事务发送至网络106时,先前事务Tx_0可能已经有效并包括在区块链150的区块151中。该事务此时可能已包括在区块151中的一个区块中,或者可能仍在有序集154中等待,在这种情况下,该事务将很快包括在新区块151中。或者,Tx_0和Tx_1可以创建并一起发送至网络106;或者,如果节点协议允许缓冲“孤立”事务,Tx_0甚至可以在Tx_1之后发送。本文事务序列上下文中使用的“先前”和“后续”一词是指由事务中指定的事务指针定义的序列中的事务顺序(哪个事务指向哪个其他事务等等)。它们同样可以替换为“前任”和“继任”、“先行”和“后代”或“父项”和“子项”等。这不一定指其创建、发送至网络106或到达任何给定区块链节点104的顺序。然而,指向先前事务(先行事务或“父事务”)的后续事务(后代事务或“子事务”)不会有效除非父事务有效。在父事务之前到达区块链节点104的子事务被视为孤立事务。根据节点协议和/或节点行为,其可被丢弃或缓冲一段时间,以等待父事务。

[0058] 先前事务Tx_0的一个或更多个输出203中的一个包括特定的UTXO,标记为$UTXO_0$。每个UTXO包括指定UTXO表示的数字资产数额的值以及锁定脚本,该锁定脚本定义后续事务的输入202中的解锁脚本必须满足的条件,以使后续事务有效,从而成功赎回UTXO。通常情况下,锁定脚本将数额锁定至特定方(该数额的事务的受益人)。即,锁定脚本定义解锁条件,

该解锁条件通常包括以下条件:后续事务的输入中的解锁脚本包括先前事务被锁定到的一方的加密签名。

[0059] 锁定脚本(亦称scriptPubKey)是节点协议识别的域特定语言中写入的一段代码。此类语言的特定示例称为“脚本(Script)”(S大写),其可由区块链网络所使用。锁定脚本指定花费事务输出203所需的信息,例如爱丽丝签名的要求。解锁脚本出现在事务的输出中。解锁脚本(亦称scriptSig)是提供满足锁定脚本标准所需信息的域特定语言中写入的一段代码。例如,其可包含鲍勃的签名。解锁脚本出现在事务的输入202中。

[0060] 因此在示出的示例中,Tx_0的输出203中的$UTXO_0$包括锁定脚本[Checksig P_A],该锁定脚本需要爱丽丝的签名Sig P_A,以赎回$UTXO_0$(严格来说,是为了使试图赎回$UTXO_0$的后续事务有效)。[Checksig P_A]包含爱丽丝的公私密钥对中的公钥P_A的表示(即哈希)。Tx_1的输入202包括指向Tx_1的指针(例如,通过其事务ID($TxID_0$),其在实施例中是整个事务Tx_0的哈希值)。Tx_1的输入202包括在Tx_0中标识$UTXO_0$的索引,以在Tx_0的任何其他可能输出中对其进行标识。Tx_1的输入202进一步包括解锁脚本<Sig P_A>,该解锁脚本包括爱丽丝的加密签名,该签名由爱丽丝通过将其密钥对中的私钥应用于预定的部分数据(有时在密码学中称为“消息”)创建。爱丽丝需要签名以提供有效签名的数据(或“消息”)可通过锁定脚本、节点协议或其组合进行定义。

[0061] 当新事务Tx_1到达区块链节点104时,该节点应用节点协议。这包括一起运行锁定脚本和解锁脚本,以检查解锁脚本是否满足锁定脚本中定义的条件(其中该条件可包括一个或更多个标准)。在实施例中,这涉及并置两个脚本:

[0062] <Sig PA><PA>||[Checksig PA]

[0063] 其中“||”表示并置,“<…>”表示将数据放在堆栈上,“[…]”表示由锁定脚本组成的函数(在该示例中指基于堆栈的语言)。同样,脚本可以使用公共堆栈一个接一个地运行,而不是并置脚本。无论采用哪种方式,当一起运行时,脚本使用爱丽丝的公钥P_A(包括在Tx_0的输出的锁定脚本中),以认证Tx_1的输入中的解锁脚本是否包含爱丽丝签名预期部分的数据时的签名。也需要包括预期的部分数据本身(“消息”),以便执行此认证。在实施例中,签名的数据包括整个Tx_1(因此不需要包括一个单独的元素来明文指定签名的部分数据,因为其本身便已存在)。

[0064] 本领域技术人员将熟悉通过公私密码进行验证的细节。基本上而言,如果爱丽丝已使用其私钥加密签署消息,则给定爱丽丝的公钥和明文中的消息,诸如节点104等其它实体可验证消息必须已经由爱丽丝签名。签署通常包括对消息进行哈希,签署哈希值和将此标记到消息作为签名,从而使公钥的任何持有者能够验证签名。因此,应当注意的是,在实施例中,在本文中对签名特定数据片段或事务部分等的任何引用可以意味着对该数据片段或事务部分的哈希值进行签名。

[0065] 如果Tx_1中的解锁脚本满足Tx_0的锁定脚本中指定的一个或多个条件(因此,在所示示例中,如果在Tx_1中提供了爱丽丝的签名并进行验证),则区块链节点104认为Tx_1有效。这意味着区块链节点104会将Tx_1添加到待定事务有序池154。区块链节点104还会将事务Tx_1转发到网络106中的一个或多个其它区块链节点104,以便其会在整个网络106中传播。一旦Tx_1有效并包括在区块链150中,这会将$UTXO_0$从Tx_0定义为已花费。应当注意的是,Tx_1仅在花费未花费事务输出203时才有效。如果其试图花费另一事务152已经花费的输出,则即使满

足所有其它条件，Tx_1也将无效。因此，区块链节点104还需要检查先前事务Tx_0中引用的UTXO是否已经花费(即，其是否已经形成另一有效事务的有效输入)。这是为何区块链150对事务152施加定义的顺序很重要的原因之一。在实践中，给定区块链节点104可维护单独的数据库，标记已花费事务152的UTXO 203，但最终定义UTXO是否已花费取决于是否在区块链150中形成了另一有效事务的有效输入。

[0066] 如果给定事务152的所有输出203中指定的总数额大于其所有输入202所指向的总数额，则这是大多数事务模型中的另一失效依据。因此，此类事务不会传播或包括在区块151中。

[0067] 请注意，在基于UTXO的事务模型中，给定UTXO需要作为一个整体使用。不能“留下”UTXO中定义为已花费的一部分数额，而同时又花费另一部分。但UTXO的数额可以在后续事务的多个输出之间分割。例如，Tx_0的$UTXO_0$中定义的数额可以在Tx_1中的多个UTXO之间分割。因此，如果爱丽丝不想将$UTXO_0$中定义的所有数额都给鲍勃，她可以使用剩余部分在Tx_1的第二输出中自己找零，或者支付给另一方。

[0068] 在实践中，爱丽丝通常还需要包括用于比特币节点104的费用，该比特币节点104在区块151中成功包含爱丽丝的事务104。如果爱丽丝未包括此类费用，则Tx_0可能会被区块链节点104拒绝，并且因此尽管在技术上有效，但可能不会传播并且包括在区块链150中(如果区块链节点104不希望接受事务152，节点协议不强迫区块链节点104接受)。在一些协议中，事务费用不需要其自身的单独输出203(即不需要单独的UTXO)。相反，输入202指向的总数额与给定事务152的输出203指定的总数额之间的任何差额都将自动提供给发布事务的区块链节点104。例如，假设指向$UTXO_0$的指针是Tx_1的唯一输入，并且Tx_1仅具有一个输出$UTXO_1$。如果在$UTXO_0$中指定的数字资产数额大于在$UTXO_1$中指定的数额，则可以由赢得工作证明竞赛以创建包含$UTXO_1$的区块的节点104分配(或花费)该差值。替代地或附加地，这不一定排除可以在其自身事务152的其中一个UTXO 203中明确指定事务费用。

[0069] 爱丽丝和鲍勃的数字资产由区块链150中任何位置的任何事务152中的锁定至他们的UTXO组成。因此，通常情况下，给定方103的资产分散在整个区块链150的各种事务152的UTXO中。区块链150中的任何位置均未存储定义给定方103的总余额的一个数字。客户端应用程序105的钱包功能的作用是将锁定至相应方且在其它随后事务中尚未花费的各种UTXO值整理在一起。为实现这一点，其可以查询存储在任何一个比特币节点104处的区块链150的副本。

[0070] 应当注意的是，脚本代码通常用示意图表示(即使用非精确语言)。例如，可以使用操作码(opcode)来表示特定功能。“OP_...”是指脚本语言的特定操作码。举例来说，OP_RETURN是脚本语言操作码，当在锁定脚本的开始处在操作码前加上OP_FALSE时，操作码创建事务的不可花费输出，该输出可以在事务内存储数据，从而将数据不可改变地记录在区块链150中。例如，数据可包括需存储在区块链中的文件。

[0071] 通常，事务的输入包含对应于公钥PA的数字签名。在实施例中，这基于使用椭圆曲线secp256k1的ECDSA。数字签名对特定的数据段进行签名。在实施例中，对于给定事务，签名将对部分事务输入以及部分或全部事务输出进行签名。对输出的特定部分进行签名取决于SIGHASH标志。SIGHASH标志通常是包含在签名末尾的4字节代码，用于选择签名的输出(并因此在签名时固定)。

[0072] 锁定脚本有时称为“scriptPubKey”,指其通常包括相应事务被锁定到的当事方的公钥。解锁脚本有时称为“scriptSig”,指其通常提供相应的签名。但是更通俗地说,在区块链150的所有应用中,UTXO赎回的条件并不一定包括对签名进行验证。更通俗地说,脚本语言可用于定义任何一个或多个条件。因此,可以优选更为通用的术语“锁定脚本”和“解锁脚本”。

[0073] 3.侧信道

[0074] 如图1所示,爱丽丝和鲍勃的计算机设备102a、120b中的每个计算机设备上的客户端应用程序都可以包括附加通信功能。此附加功能可使爱丽丝103a建立与鲍勃103b的单独侧信道107(在任何一方或第三方的鼓动下)。侧信道107使得能够脱离区块链网络交换数据。此类通信有时称为“链下”通信。例如,这可用于在爱丽丝与鲍勃之间交换事务152,而不将该事务(尚未)注册到区块链网络106上或将其发布到链150上,直到其中一方选择将其广播到网络106上。以这种方式共享事务有时称为共享“事务模板”。事务模板可能缺少形成完整事务所需的一个或多个输入和/或输出。替代地或附加地,侧信道107可用于交换任何其它事务相关数据,例如密钥、议付数额或条款、数据内容等。

[0075] 通过与区块链网络106相同的分组交换网络101可建立侧信道107。替代地或附加地,侧信道301可以经由诸如移动蜂窝网络的不同网络或者诸如无线局域网络的局域网建立,甚至经由爱丽丝和鲍勃的设备102a、102b之间的直接有线或无线链路建立。通常,在本文中任何地方所指的侧信道107可以包括经由一项或多项联网技术或通信介质的任何一条或多条链路,这些链路用于“链下”交换数据,即脱离区块链网络106交换数据。在使用多条链路的情况下,链下链路束或集合整体上可以称为侧信道107。因此,应当注意的是,如果说爱丽丝和鲍勃通过侧信道107交换某些信息或数据等,则这不一定意味着所有这些数据都必须通过完全相同的链路或甚至相同类型的网络发送。

[0076] 4.编辑事务内容

[0077] 区块链事务包括输入和输出,这些输入和输出中的每个输入和输出包括脚本。区块链脚本(例如,支付到公钥(P2PK)、支付到公钥哈希(P2PKH)、多重签名、OP_RETURN)可以用于在事务中存储任意数据。输出包含可以对复杂的锁定条件进行编码的脚本,以及存储可能是也可能不是锁定条件的一部分的附加数据项。例如,可以使用称为OP_FALSE OP_RETURN输出的不可花费脚本模式将任意数据添加到事务中。用于在账本上存储任意数据的脚本操作码OP_FALSE OP_RETURN的组合会导致区块链的大小增加,并消耗希望存储区块链的全部数据的区块链节点104的磁盘空间。存储在区块链上的一些数据可能是敏感数据或私有数据等。一些数据在记录到区块链上时或稍后可能属于这些类别中的一种类别。一些数据在一些司法管辖区可能属于这些类别中的一种类别,但在其他司法管辖区则不属于。因此,除了其他原因之外,还需要能够对事务的数据进行编辑(redact),以便减小事务规模,以及/或者遵守(取决于司法管辖区的)规则和法规。图3示出了具有OP_FALSE OP_RETURN有效载荷的示例性区块链事务的结构。

[0078] 本公开的实施例使得能够编辑存储在区块链事务中的数据,即,作为脚本的一部分包括在事务的输入或输出中的数据。待编辑的数据在本文中称为“目标数据”。目标数据可以作为输入(解锁)脚本的一部分包括在内。例如,图3所示的事务包括解锁脚本,该脚本包括数据<Sig_A><P_A>—签名和公钥。目标数据可以作为输出(锁定)脚本的一部分包括在内。

例如,同一事务包括两个锁定脚本,这两个脚本都包含数据。第一锁定脚本包括数据P_B—公钥。第二锁定脚本包括数据<data>—任何任意数据,例如媒体内容、个人详细信息(例如,姓名和地址)、发票信息等。目标数据可以可花费(即,可转让和可分配)输出或不可花费(即,不可转让和不可分配)输出的一部分包括在内。

[0079] 目标数据的编辑可以由能够访问原始区块链事务的任何实体执行,该原始区块链事务可以是包含数据的任何事务。为了简洁起见,执行编辑过程的实体将称为"编辑者redactor"。在一些实施例中,编辑者可以是诸如用户之类的一方,例如爱丽丝103a或鲍勃103b,或者更确切地说他们的相应计算机设备102a、102b。在其他实施例中,编辑者包括区块链节点104。这些实施例将在下面进行讨论。

[0080] 编辑者获取目标事务,即包含要作为脚本的一部分进行编辑的目标数据的事务。编辑者可以生成目标事务。例如,编辑者可以是生成事务的用户(例如,爱丽丝103a)。在这种情况下,编辑者还可以生成目标数据,或者目标数据可以从别处获取(例如,接收)。在其他示例中,编辑者可能不是生成目标事务的实体。例如,目标事务可以从区块链获取,或者从区块链节点104接收,或者从事务生成者接收。例如,编辑者可以是从生成目标事务的爱丽丝103a接收目标事务的区块链节点104。

[0081] 在获取目标事务之后,编辑者使用包括目标数据的脚本(例如,解锁脚本或锁定脚本)来生成默克尔树。整个脚本用于生成默克尔树。目标数据以其原始形式形成默克尔树的一个或多个叶。也就是说,默克尔树的一个或多个叶包括目标数据的至少一部分(或由其组成)。在一些示例中,单个叶可以包括目标数据(或由其组成),这意味着完整目标数据作为该叶的一部分包括在内。或者,目标数据可以在默克尔树的多个叶之间划分,这意味着不同的叶包括目标数据的不同部分。例如,目标数据可以被划分为相等大小的块,也可以被划分为预定数量的块。相反,目标数据可以任意划分。

[0082] 应当注意的是,术语"默克尔树"通常是指二叉哈希树。然而,一般而言,本公开的实施例可以使用任何类型的哈希树,例如三级哈希树。因此,对"默克尔树"的任何引用都可以替换为更一般的"哈希树"。

[0083] 在一些示例中,脚本可以仅包括目标数据。例如,解锁脚本可以仅包括用于解锁先前锁定脚本的解锁脚本的目标数据。在这种情况下,默克尔树可以完全基于目标数据。然而,在实践中,脚本更有可能包含其他组件,而不仅仅是目标数据。例如,脚本可以包括被配置为执行相应操作的一个或多个函数。在比特币区块链上,这些函数称为操作码。附加地或替代地,脚本可以包括将不被编辑的数据。例如,脚本可以包括可能不需要编辑的公钥哈希和确实需要编辑的图像文件。在这些示例中,目标数据仅构成默克尔树的一部分,即默克尔树的一个或多个叶。函数和/或其他数据也构成默克尔树的一部分。也就是说,默克尔树的一个或多个叶可以包括一个或多个函数(例如,由其组成),并且默克尔树的一个或多个叶可以包括非目标数据(例如,由其组成)。

[0084] 脚本的不同组件(例如,函数、目标数据、其他数据等)具有例如从左到右的顺序,在构建默克尔树时遵循该顺序。默克尔树的叶是使用脚本的组件按照它们出现的顺序形成的。例如,脚本可以包括第一组一个或多个函数,然后是目标数据,之后是第二组一个或多个函数。第一组函数形成默克尔树的第一组叶,目标数据形成默克尔树的第二组叶,第二组函数形成默克尔树的第三组叶。

[0085] 在一些示例中,可以将一个或多个连续函数分组在一起,以形成默克尔树的单个叶。在其他示例中,每个函数可以形成默克尔树的单个叶。

[0086] 在一些示例中,脚本可以包括被配置为在执行时指示目标数据的数据函数(例如,OP_PUSHDATA、OP_RETURN等)。指示目标数据可以包括在执行时将目标数据输出到存储器(例如,基于堆栈的存储器)。在这些示例中,数据函数可以与其他函数分开,使得它是默克尔树的相应叶中包括的唯一函数。包含数据函数的叶可以包括其他数据,例如目标数据的长度(例如,以字节为单位)。或者,目标数据的长度可以作为默克尔树的不同叶包括在内。数据的长度可以用于确定默克尔树的叶数。

[0087] 下面将更详细地讨论图4和图5,这些附图示出了基于包含待要编辑的目标数据的脚本来生成默克尔树的过程。如图4所示,从左到右,第一组函数(操作码)形成第一叶,数据函数和目标数据(<pubkeyhash>)形成第二叶,第二组函数形成第三叶,第四叶通过复制第三叶形成。在图5中,第一数据函数(OP_RETURN)形成第一叶的一部分,第二数据函数(OP_PUSHDATAX)连同数据长度({X-bytes])一起形成第二叶的一部分,目标数据在六个叶节点之间划分。

[0088] 在基于脚本生成默克尔树之后,通过使用默克尔树的默克尔根替换脚本来生成目标事务的编辑版本。因此,使用默克尔根对目标数据进行编码(和隐藏)。

[0089] 目标事务可以包括多个相应脚本,该多个相应脚本包括相应目标数据。可以针对相应脚本中的一个、部分或全部相应脚本执行上述基于脚本来生成默克尔树的过程。然后,可以使用相应默克尔树的相应默克尔根来替换相应脚本。

[0090] 编辑者可以将被编辑的事务发送给一个或多个实体,例如用户103或区块链节点104。例如,编辑者可以是从用户(例如,鲍勃103b)接收对目标事务(不一定是其编辑版本)的请求的区块链节点104。

[0091] 编辑者可以生成所编辑的事务的初级事务标识符。在一些区块链上,初级事务标识符(TxID)被生成为序列化事务(例如,32位版本、32位锁定时间、事务输入列表、事务输出列表等)的双重SHA256哈希(或SHA256d=SHA256(SHA256(.)))。可以通过对所编辑的事务进行哈希处理,以类似的方式生成所编辑的事务的初级事务标识符TxID'。用于生成初级事务标识符TxID'的哈希函数可以是SHA256或双重SHA256哈希函数。可以使用其他哈希函数,例如SHA512。

[0092] 在编辑者是事务生成者(例如,用户、钱包应用程序等)的情况下,编辑者可以基于被编辑的事务来生成签名。具体地,签名基于消息(即,是消息的函数),其中该消息基于所编辑的事务。例如,该消息可以是事务的序列化版本。通常,签名基于事务的脚本(无论这些脚本采用原始形式还是哈希形式)。现在,签名基于默克尔根,该默克尔根替换一个或多个脚本。签名对所编辑的事务的一个或多个输入和/或输出进行签名,并且包括在所编辑的事务的输入中。签名可以是一个或多个附加项(例如,私钥)的函数。例如,签名可以是ECDSA签名。可以使用替代的签名方案。所编辑的事务,签名被包括在目标事务(即,事务的原始未编辑版本)中,并提交给区块链网络106进行核实。也就是说,通过使用相应默克尔根替换一个或多个脚本来生成所编辑的事务,基于所编辑的事务来生成签名并其包括在事务的原始版本中,然后将已签名的原始事务发送到区块链网络106。

[0093] 如上所述,签名包括在目标事务的输入中。输入引用上一个事务的输出(“先前输

出”)。输出可以包括脚本,该脚本包括目标数据。通常,要签名的消息的一部分基于先前输出。现在,在构建消息时,编辑者基于先前输出的脚本(使用上面描述的方法)来构建默克尔树,并且使用默克尔根来替换该脚本。然后,要签名的消息基于默克尔根,而不是先前输出的脚本。

[0094] 在编辑者是事务核实者(例如,区块链节点104)的情况下,事务核实者可以获取包括签名的目标事务,并且基于所编辑的事务来验证签名对消息有效。该消息以与上述相同的方式生成。

[0095] 目标事务的脚本中的至少一个脚本可以是锁定脚本—第一锁定脚本。这里,“第一”用作标签,并不一定表示锁定脚本列表中的第一个锁定脚本。在编辑者是事务核实者的情况下,编辑者可以将基于第一锁定脚本生成的默克尔根存储在数据库中,该数据库映射目标事务的修改的初级事务标识符。例如,所修改的初级事务标识符可以是密钥值对(key-value pair)中的密钥,其中第一锁定脚本的默克尔根是该值。如果目标事务具有多个输出,则第一锁定脚本所属的输出的索引可以构成密钥的一部分。编辑者可以存储多个密钥值对,其中每个密钥与相应事务的相应输出相关。数据库可以用于核实事务,如下所述。

[0096] 事务核实者可以获取第二区块链事务,该第二区块链事务包括输入—第一输入,该输入引用目标事务的输出(例如,包括第一锁定脚本的第一输出)。同样,“第一”在这里仅用作引用目标事务的输出的输入的标签。第二事务的第一输入包括第一解锁脚本。目标事务的第一输出使用目标事务的所修改的初级事务标识符和第一输出的索引来引用。事务核实者可以通过多种方式中的一种方式来核实第二事务。

[0097] 作为第一选项,事务核实者从数据库获取映射到所引用的输出的默克尔根,并基于第一解锁脚本来确定第一锁定脚本的部分锁定脚本。部分锁定脚本包括锁定脚本的一部分但不是全部,例如直到目标数据的所有锁定脚本。部分锁定脚本还包括与默克尔根对应的默克尔树的一个或多个哈希。哈希可以是叶哈希和/或内部哈希。每个哈希直接(在叶哈希的情况下)或间接(在内部哈希的情况下)基于目标数据。例如,参考图3,部分锁定脚本可以采用以下形式:OP_DUP OP_HASH160 H_1 H_{23}。事务核实者可以生成部分锁定脚本或从别处获取该部分锁定脚本,例如区块链节点104或向区块链提交第二事务的实体。

[0098] 事务核实者使用第一解锁脚本中包含的信息(例如,数据和/或函数)来确定部分锁定脚本的形式,例如,部分锁定脚本中必须存在(或至少可能存在)的函数。部分锁定脚本可以基于第一解锁脚本的句法分析来确定。例如,如果第一解锁脚本包括紧跟有公钥的签名,则事务核实者可以假设第一锁定脚本包括支付到公钥哈希脚本。部分锁定脚本可以设置为支付到公钥哈希脚本。又如,第一解锁脚本可以包含关于部分锁定脚本的形式的指示。事务核实者可以执行包含证明,以确认部分锁定脚本属于与默克尔根对应的默克尔树。在确认部分锁定脚本确实属于默克尔树并且因此是第一锁定脚本的一部分之后,事务核实者根据部分锁定脚本来核实第一解锁脚本。如果核实失败,则拒绝第二事务。

[0099] 作为第二选项,事务核实者可以将第一输出的部分锁定脚本存储为映射到所修改的事务标识符和第一输出的索引的密钥值对中的值,而不是将第一输出的默克尔根存储为密钥值对中的值。如上所述,部分锁定脚本包括原始形式的第一锁定脚本的一部分(不包括目标数据)和基于第一锁定脚本生成的默克尔树的一个或多个叶或内部哈希。例如,部分锁定脚本可以包括一组函数和/或非目标数据以及目标数据的哈希。在获取引用目标事务的

第一输出的第二区块链之后,事务核实者从数据库中检索部分锁定脚本,并根据部分锁定脚本来核实第一解锁脚本。换句话说,剩余原始形式的第一锁定脚本与第一解锁脚本一起执行。如果核实失败,则拒绝第二事务。

[0100] 作为第三选项,可以组合第一选项和第二选项。也就是说,事务核实者可以将默克尔根和部分锁定脚本两者存储在数据库中,其映射到所修改的初级事务标识符和目标事务的第一输出的索引。响应于接收到第二事务,事务核实者可以确定部分锁定脚本的原始部分(例如,基于对第一解锁脚本的分析),并使用从数据库中检索的部分锁定脚本来执行默克尔包含证明,以确认该原始部分确实是第一锁定脚本的一部分。然后根据第一解锁脚本来核实部分锁定脚本的原始部分。如果核实失败,则拒绝第二事务。

[0101] 上面的描述主要侧重于基于目标事务的各个脚本来生成默克尔树,以生成编辑的事务。编辑者(无论是用户103、区块链节点104还是其他实体)可以基于所编辑的事务来生成默克尔树。也就是说,生成事务层级的默克尔树,其中叶由所编辑的事务的各部分形成。图8中示出了一个示例,其中所编辑的事务的字段中的至少一个字段包括默克尔根,例如图3或图4的默克尔根。事务默克尔树的默克尔根用作所编辑的事务的第二事务标识符。所编辑的事务包括多个字段,例如版本号、锁定时间、一个或多个输入、一个或多个输出等。一个或多个叶可以包括单个相应字段的部分或全部。一个或多个叶可以包括多个相应字段的部分或全部。

[0102] 因此,本公开认识到可以利用默克尔树来实现对所编辑的事务的各个数据字段的验证的另一种方式。所编辑事务被拆分(或解析)为用于生成默克尔树的一组数据字段,即事务的不同部分被标识为单独的数据字段。默克尔根用作事务的新颖二级标识符。查询用户(例如,可以仅操作轻量级客户端)可以使用该二级标识符来执行证明,以证明事务中存在(小)数据字段,而无需获取完整事务数据并对其进行哈希处理。

[0103] 5.使用经过默克尔处理的脚本的选择性披露

[0104] 本节提供了关于如何实现上述实施例以编辑区块链内容的示例。

[0105] 在本节中,提供了一种新机制,这种新机制可以用于选择性地披露利用区块链脚本的默克尔处理的事务数据。在较高层级,为事务中的特定脚本计算默克尔树,其中脚本可以是输入或输出。以这种方式从左到右解析脚本,即使用哈希函数(例如,SHA256)对每个连续操作码序列进行哈希处理以创建叶节点。每当遇到<data push>时,可以类似地对数据进行哈希处理以创建叶节点,或者可以将数据划分为n个块,对每个块进行哈希处理以创建n个叶节点。在已经创建所有叶节点后,可以计算表示脚本的默克尔树根。

[0106] 由于默克尔树的特征,现在可以通过哈希计算而不是成本高昂的非对称操作(例如,签名、安全多方计算、不经意传输和零知识证明)来非常高效地实现选择性地披露信息。

[0107] 5.1经过默克尔处理的脚本

[0108] 在实施例中,事务中的所有脚本以经过默克尔处理的形式表示为“脚本默克尔树”。这种方法可以概括为,通过将每个连续操作码串视为叶节点、将每个数据推送视为叶节点,并根据默克尔树的叶节点在脚本中出现的顺序对其进行排序来从脚本生成脚本默克尔树。用于对脚本scriptMerklize(SCRIPT)进行默克尔处理的示例性算法描述如下:

[0109] 1.从左到右解析脚本:

[0110] a.对每个连续操作码序列进行分组,并且将以字节为单位的底层原始表示索引为

叶。

[0111] b.单独处理采用<Data>＝OP_PUSHDATAX[X bytes][Data]形式的每个数据推送，并将其索引为叶，其中OP_PUSHDATAX是以下操作码中的一个操作码:OP_PUSHDATA1、OP_PUSHDATA2、OP_PUSHDATA4。X的大小为[X bytes]。

[0112] c.根据叶的索引对其进行排序，最左边的叶对应于脚本的最左边部分。

[0113] 2.对每个叶进行哈希处理，以在默克尔树中创建叶节点。或者，可以取每个叶的双重哈希(SHA256d＝SHA256(SHA256(x))，其中x是前置消息)。

[0114] 3.构建默克尔树T，并从叶计算根R，其中填充以通常的方式完成。

[0115] a.树始终具有偶数个叶。如果存在奇数个叶，可以复制最后一个叶。例如，如果树中的最后一个叶是左分支，则最后一个叶被复制到右边叶位置。

[0116] OP_PUSHDATAX和[X bytes]分别是第4节中描述的数据函数和数据长度的示例。数据长度用于显示所编辑的数据的长度。

[0117] 除了上面所示的步骤之外，还可以进一步细分脚本的<Data>元素，以便允许在特定脚本中更精细地选择性披露每个数据推送的不同部分。例如，数据包含固定大小帧的媒体内容，这些帧自然可以被划分为更小的块。在这种情况下，只需在scriptMerklize()算法中添加以下附加内容:

[0118] 每当遇到<data push>时，可以首先将其划分为n个块，并计算每个块的哈希以创建n个叶。n值可以是固定的，甚至可能因脚本而异。此外，应当注意的是，一般而言，这些块中的每个块不需要大小相等。

[0119] 如果不划分数据

[0120] Leaf＝OP_PUSHDATAX[X-Bytes]DATA

[0121] 则OP_PUSHDATAX[X-bytes]也可以从<Data>中分离出来，作为n个数据叶中的一个单独的叶，如下所示，其中一个<data push>现在成为树中的n+1个叶。

[0122] $Leaf_0$＝OP_PUSHDATAX[X-Bytes]

[0123] $Leaf_1$＝Data[1]

[0124] …

[0125] Leaf_n＝DATA[n]

[0126] 备注:将数据划分为更小的块需要权衡，但是由于增加了哈希函数压缩和默克尔树计算，因此需要付出计算成本。假设内容数据为4480位。SHA256(内容数据)需要9个压缩函数，但只有1个叶。然而，如果将内容数据划分为448位，则有10个数据块，每个块需要1个压缩函数，因此总共需要10个压缩函数。第二选项还会增加树深度。

[0127] 这种算法从任何给定的脚本生成默克尔树。该默克尔树的根R可以用于下面描述的更新的TxID生成和SIGHASH算法。

[0128] 现在考虑对一些示例性脚本应用上述脚本默克尔处理算法。

[0129] 5.1.1实施例

[0130] 实施例1:支付到公钥哈希(P2PKH)

[0131] 考虑简单的“支付到公钥哈希”输出，其中可以将其脚本写为:

[0132] SCRIPT＝OP_DUP OP_HASH160 OP_PUSHDATA<pubkeyhash>OP_EQUALVERIFY OP_CHECKSIG

[0133] 其中,SCRIPT被分块为叶,如下所示:

[0134] $Leaf_0$=OP_DUP OP_HASH160

[0135] $Leaf_1$=OP_PUSHDATA<pubkeyhash>

[0136] $Leaf_2$=OP_EQUALVERIFY OP_CHECKSIG

[0137] $Leaf_3$=$Leaf_2$

[0138] 图4中示出了该示例。

[0139] 应当注意的是,作为树计算的一部分,复制最后一个叶。此外,应当注意的是,对脚本进行默克尔处理不会从事务中删除或"隐藏"脚本。相反,使用术语"替换"意味着使用默克尔根进行计算,而不是脚本本身。需要强调的是,需要在默克尔树的叶中保留脚本及其数据元素的顺序,因为要对花费这一点的事务进行后续核实。应当注意的是,这里的"后续核实"是指对任何未来事务进行核实,该事务花费已经编辑的事务的输出,并将其脚本中的一个脚本替换为Merklization。

[0140] 这里的微妙之处在于,除了获取和执行脚本的未编辑部分之外,核实者可能还需要执行完整性检查(即,默克尔包含证明),以证明未编辑的脚本部分在被部分编辑之前是原始脚本的一部分。下面详细介绍了这种类型的后续事务核实的示例性流程。

[0141] 作为可能的编辑的示例,图4中的P2PKH输出已经在其他事务中花费(即,分配)。之后,可以对该P2PKH输出进行编辑(例如,对$Leaf_1$的脚本进行编辑)。应当理解的是,由于H($leaf_1$),因此所编校的事务是有效的,因为可以计算脚本的默克尔根和与已经挖掘的事务匹配的TxID。此外,甚至可以通过获取公钥并手动重新创建$leaf_1$、OP_PUSHDATA公钥并对其进行哈希处理来计算与H(leaf1)匹配的哈希值,从而证明花费事务的scriptSig中的公钥是有效的。

[0142] 需要强调的是,与实现相同效果的更多替代方法相比,这使得能够高效地核实编辑的事务的后续花费,这可能会调用基于零知识的协议来证明完整性。默克尔树验证往往比执行零知识证明更高效,这意味着该方法将允许更高效地核实所编辑的事务。

[0143] 实施例2:OP_FALSE OP_RETURN

[0144] 假设某些数据内容已经插入到具有OP_RETURN操作码的事务中,并且包含一些敏感信息(例如,个人的),并且要求选择性地披露一些部分,同时完全隐藏其他部分。假设

[0145] SCRIPT=OP_FALSE OP_RETURN<DATA>

[0146] 其中<DATA>=OP_PUSHDATAX[X-bytes][Data],其中[X-bytes]是[DATA]的长度(以字节为单位)。

[0147] 然后,构建树的叶,其中每个叶(从左到右)是一组操作码或数据推送。这意味着脚本中的每个数据推送都是叶,每个操作码集群也是推送。例如,SCRIPT被分块为叶(如下所示),数据推送被划分为如上所述的精细块:

[0148] $Leaf_0$=OP_FALSE OP_RETURN

[0149] $Leaf_1$=OP_PUSHDATAX[X-bytes]

[0150] $Leaf_2$=$Data_0$

[0151] …

[0152] $Leaf_{n+1}$=$Data_{n-1}$

[0153] 其中[DATA]=[$Data_0$||$Data_1$||…$Data_{n-1}$]。

[0154] 数据可以具有不同的大小。使用该技术的系统可以根据不同的标准来配置n。假设数据的大小为4000位。然后,可以使用以下任意规则对数据进行划分:

[0155] ●系统确定并预定义n:假设系统使用n=6。然后,每个数据叶将是$\lfloor 4000/6 \rfloor = 666$位(将舍入余数加到最后一个块中)。为了使用SHA256对此进行哈希处理,将每个数据叶输入填充到1024位(以填充大小为512位的2个区块)。

[0156] ●系统限制与叶相关联的数据量:如果系统使用SHA256作为哈希函数,则可以使用零位填充数据以由大小为512位的k个块组成[3]。在数据为4000位的情况下,由于每个块为512位,因此需要填充数据,并且有$n = \lceil 4000/512 \rceil = 8$个叶节点需要压缩。

[0157] 图5示出了具有8个叶的经过默克尔处理的脚本的示例,其中第一叶是操作码"OP_FALSE OP_RETURN"序列,其余叶是被划分为6个块的数据。

[0158] 5.2安全分析

[0159] 哈希函数容易遭受离线(暴力)攻击,使用情况取决于原像是否有足够的熵。如果事务中存在秘密数据,则可能需要确保秘密部分具有足够的熵,使得不会遭受暴力攻击。更具体地,攻击者不应该能够从混淆(哈希或加密)的消息中检索秘密。这里的熵是指在需要随机数据的算法中使用的系统收集的随机性。缺乏良好的熵可能会使安全系统易受攻击,并且无法安全地加密/混淆数据。作为一般的经验规则,除了最敏感的应用程序外,128位可能被视为足以满足所有应用程序的需求。

[0160] 下面,通过分别考虑每个用例来对机制的使用方式进行分类。

[0161] ●如果秘密数据内容的熵大于128位:

[0162] o检查该秘密内容的各个块(即,默克尔树的叶)是否也具有足够的熵。隐私仍然可以通过级联相邻数据来保持,直到有足够的熵,然后仅披露那些块的默克尔根。

[0163] 1.例如,如果$Leaf_2$没有足够的熵,但与$Leaf_1$和$Leaf_3$结合时有足够的熵,则仅披露H_{4567}和$Leaf_4$,…,Leafn、(即,不共享$Leaf_0$,$Leaf_1$,$Leaf_2$和$Leaf_3$)。

[0164] ●如果秘密数据内容的熵小于128位,但完整数据内容的熵大于128位:

[0165] o如果仅隐藏秘密数据内容,则被阻止(隐藏)的块容易遭受离线攻击。

[0166] o隐私仍然可以按照如下方式保持:

[0167] 1.提取这128位(公共部分以及秘密部分)并将它们级联以增加熵;

[0168] 2.将该新值保密;以及

[0169] 3.公开其余公共部分。

[0170] ●如果整个脚本的熵小于128位:

[0171] o它容易遭受离线攻击。

[0172] o然而,为了完整起见,

[0173] 除了数据的默克尔根之外,请勿披露数据的任何部分,因为这足以核实事务。

[0174] 5.3经过默克尔处理的脚本的第1层应用

[0175] 现在考虑如何将经过默克尔处理的脚本的概念应用于第1层区块链系统(例如,比特币核心)的设计。本节的目标是研究如何在(类似比特币的)区块链的协议层级使用脚本Merklization,以允许节点104从其区块链150的副本编辑事务信息,而不牺牲验证与已编辑的那些事务相关的未来事务的能力。

[0176] 在上一节中,从一般意义上讨论了如何为给定脚本生成默克尔根。然后,可以在现有计算中使用该默克尔根来代替脚本,这些计算通常涉及对脚本本身进行哈希处理,例如SIGHASH算法或TxID计算。本节中的第1层建议会影响从事务数据计算TxID的方式,但不会影响事务消息本身中包含的内容。如后一节所述,TxID的构建将使用脚本默克尔根来代替原始操作码,但事务本身仍将包含完整的脚本。

[0177] 应当注意的是,脚本的默克尔处理通常在原始事务广播之后进行,此时需要编辑一些数据。

[0178] 该方法需要对现有区块链实现方式的以下方面进行更改:

[0179] ●事务ID(TxID)构建

[0180] ●SIGHASH算法

[0181] ●事务核实

[0182] 这些更改允许在没有完整的原始事务脚本数据的情况下核实事务。这在事务已经包括在区块151中之后的某个时刻一些事务数据被标识为秘密的情况下是有用的,因为它允许节点104核实包含秘密数据的事务输出点的后续花费(如果编辑脚本核实失败,例如在引导节点时,如果事务在区块链中足够深,则可以假设事务是有效的),并且它允许终端用户对事务中剩余的非秘密数据执行存在证明。

[0183] 图6示出了事务的示例性流程。在步骤1中,对目标事务的脚本进行默克尔处理。例如,可以使用下面描述的修改的SIGHASH算法对事务进行签名。在步骤2中,将原始目标事务(其可以包括在步骤1中生成的签名)提交给区块链150。在步骤3中,通过对脚本进行默克尔处理从事务中编辑目标数据。

[0184] 5.3.1修改TxID构建协议

[0185] 在(比特币的)当前实现方式中,将事务标识符(TxID)计算为原始事务数据Tx的SHA256d,其可以写为:

[0186] TxID:=SHA256d(Tx)

[0187] 然而,计算TxID需要完整的原始事务数据,在处理可能不希望处理的秘密或大型数据的情况下,希望避免使用这些数据。为了适应经过默克尔处理的脚本,可以使用新算法computeTxID()来计算所有事务的TxID,如下所示:

[0188] 1.获取事务Tx的完整事务数据。

[0189] 2.清空表示为Tx′的简化事务的每个脚本的脚本数据。

[0190] 3.代替每个清空的脚本,放置该脚本的默克尔根,如通过算法scriptMerklize(SCRIPT)先前计算的那样。结果是Tx″。

[0191] 4.计算TxID″=SHA256d(Tx″)。

[0192] 这定义了新的(初级)事务ID,它可以用于唯一地引用事务,而不存在重新构建依赖于编辑的脚本数据的TxID的风险。

[0193] 另一种方式是,即使脚本的各部分之后被编辑,事务也具有相同的TxID。

[0194] 应当注意的是,在该示例中选择使用SHA256哈希函数,这意味着所有脚本都将被32字节的哈希摘要替换,而不考虑要替换的脚本的大小。如果为scriptMerklize()算法选择不同的哈希函数,则替换摘要的长度可能有所不同。

[0195] 5.3.2修改SIGHASH算法

[0196] (比特币的)现有实现方式包含SIGHASH算法,该算法定义了将比特币事务序列化为消息,这些消息将由ECDSA签名进行签名,并用于解锁脚本。SIGHASH算法的一个关键特征在于,当生成用于对特定事务输入进行签名的序列化消息时,这确保该消息也包含在该输入中使用的先前输出点和先前锁定脚本。这意味着要签名的序列化消息明显地依赖于先前锁定脚本。至关重要的是,必须确保该先前锁定脚本确实是上一个事务的一部分,这可以通过验证先前锁定脚本确实是原始事务Tx_{Prev}的一部分来实现,该原始事务的双重哈希产生$TxID_{Prev}$。在括号中给出数据类型的完整SIGHASH算法写为:

[0197] 1.$Tx_{current}$中的nVersion(4字节小端)

[0198] 2.所有输入输出点的序列化的SHA256d(32字节哈希)

[0199] ●如果设置了ANYONECANPAY标志,则这应该是一个32字节的零。

[0200] 3.所有输入的nSequence的序列化的SHA256d(32字节哈希)

[0201] ●如果设置了ANYONECANPAY标志,则这应该是一个32字节的零。

[0202] 4.正在花费的输出点(用于事务ID的32字节+用于索引的4字节小端)

[0203] 5.subScript的长度(以字节为单位)(大端)

[0204] 6.subScript(定义如下)

[0205] 7.输出的amount(以聪为单位)(8字节小端)

[0206] 8.该输出点的nSequence(4字节小端)

[0207] 9.所有输出amount和scriptPubKey的序列化的SHA256d。这些数据取自$Tx_{current}$中的输出。

[0208] ●如果设置了SINGLE标志,并且输入索引小于输出的数量,则这应该是输出(具有与输入相同的索引的scriptPubKey)的双重SHA256。

[0209] ●如果设置了NONE标志,这应该是一个32字节的零。

[0210] 10.事务$Tx_{current}$的nLocktime(4字节小端)

[0211] 11.签名的sighash类型(4字节小端)

[0212] 上述算法中的步骤6依赖于subScript,其是使用先前锁定脚本和prevOuts生成的。上一个事务之间的关系如下:

[0213] “根据previousScriptPubKey创建一个新的subScript。该subScript从最近的OP_CODESEPARATOR(在[being executed]的OP_CHECKSIG之前的一个)开始,到previousScriptPubKey结束。如果没有OP_CODESEPARATOR,则previousScriptPubKey变为subScript”。

[0214] 如果正在花费的先前锁定脚本可能包含不希望共享的数据(例如,秘密数据),则SIGHASH算法对原始previousScriptPubKey的这种依赖性会出现问题。因此,如果经过编辑,则不能再使用,并且步骤6失败。

[0215] 使用所编辑的脚本进行后续核实的示例:

[0216] 使用事务Tx_1的术语“后续核实”或“后续花费”来指代任何后续事务Tx_2的核实,该后续事务花费Tx_1的任何输出。

[0217] 例如,假设Tx_1和Tx_2是已经在区块链中挖掘的事务:

[0218]

Tx_1	Tx_2	
OUTPUT	INPUT	OUTPUT
LOCKING	UNLOCKING	LOCKING
ScriptPubKey -->	scriptSig -->	OP_RETURN CONTENT

[0219] 以下场景强调了现有SIGHASH算法如何无法处理编辑的脚本数据:

[0220] ●核实Tx_2,其中Tx_2正在花费Tx_1的输出,该输出之后被编辑,因为SIGHASH涉及对所编辑的输出scriptPubKey进行哈希处理,该scriptPubKey现在缺少数据。

[0221] ●核实Tx_2,其中Tx_2正在发送到输出,该输出之后被编辑,因为SIGHASH涉及对OP_RETURN输出进行哈希处理,该OP_RETURN输出现在缺少数据。

[0222] 该问题通过将SIGHASH算法中使用的previousScriptPubKey替换为锁定脚本的默克尔根来解决,该锁定脚本是使用scriptMerklize(previousScriptPubKey)生成的。这意味着能够运行整个SIGHASH算法,从而仅使用先前锁定脚本的默克尔根来核实签名。更一般地,在对所有脚本进行签名时,可以将所有脚本替换为它们的scriptMerklize()根。这是可以在新版本的SIGHASH算法中实现的规则,使得在消息构建过程中,由ECDSA签名进行签名的任何脚本被替换为默克尔根。

[0223] 5.3.3后续核实事务

[0224] 如上所述对事务进行签名时,SIGHASH算法(取决于SIGHASH标志)通常涉及将输出脚本(和其他数据)哈希到摘要中,然后对该摘要进行签名。在新构建中,在签名之前使用新的消息序列化算法(例如,ECDSA)。不同之处在于,SIGHASH算法现在使用默克尔根,而不是脚本本身。这使得相关各方能够对签名进行后续核实,这些相关各方仅接收到签名可能已经对其进行签名的所编辑的事务。应当注意的是,该修改的SIGHASH算法仅在通常情况下调用,即当调用OP_CHECKSIG、OP_CHECKSIGVERIFY、OP_CHECKMULTISIG和/或OP_CHECKMULTISIGVERIFY操作时。

[0225] 例如,考虑具有包含OP_FALSE OP_RETURN<data>的输出的事务的情况,其中OP_RETURN数据被编辑。在原始(比特币)事务中,需要UTXO的输出脚本来进行哈希处理,但是对于所编辑的事务,这些数据不可用,无法重现。因此,无法重现已签名的摘要。在新构建中,如果向必须核实事务的接收方提供了脚本的默克尔根,以便实现进行签名的哈希过程,则可以进行后续核实。通过执行相同的SIGHASH算法来生成摘要,可以验证事务中的签名。

[0226] 应当注意的是,对SIGHASH算法的这种提议修改是专门设计的,以确保ECDSA签名的创建和核实不会受到脚本编辑的阻碍。对于所有签名,修改都可以实现这一点,而不管这些签名是否试图花费编辑的输出。

[0227] 在下一节中,描述了在这种情况下执行事务核实所需的步骤,使得一个事务的一部分的编辑不会阻止未来事务花费其输出。

[0228] 5.3.4更新事务核实

[0229] 在上一节中,展示了如何更新SIGHASH算法,使得它不依赖于原始形式的输入的先前锁定脚本。这样可以在不知道完整先前原始锁定脚本的情况下验证签名。然而,在核实事务时,验证解锁脚本中的ECDSA签名只是该过程的一部分。当花费事务试图花费已被部分编辑的锁定脚本时,还必须确认花费事务所满足的先前锁定脚本的部分实际上是原始锁定脚本的一部分。这可以通过将先前锁定脚本映射到先前TxIDPrev来实现,后者现在还对经过

默克尔处理的脚本进行编码。

[0230] 要实现这一点,一种方法是将采用($TxID_{Prev}$,i_{Prev})形式的输出点映射到锁定脚本,如下所示:$(TxID_{Prev}, i_{Prev}) \mapsto [Locking\ script]$。然后,可以使用[Locking script]来检索正被花费的部分脚本,并使用脚本Merklization来证明正被花费的部分确实是完整脚本的一部分,而不仅仅是从存储器中检索。例如,对于采用[$ChecksigP_A$]<Data>OP_DROP形式的脚本,可能希望在不知道<Data>组件的情况下花费输出。这意味着需要确认[$ChecksigP_A$]组件确实是完整原始锁定脚本的一部分,而无需访问<Data>组件。

[0231] 在现有系统中,UTXO数据库的行为类似于采用($TxID_{Prev}$,i_{Prev})形式的密钥值存储:[Locking Script],其中密钥是由现有事务标识符$TxID_{Prev}$和输出索引i_{Prev}组成的输出点,针对该输出点存储的值是与其对应的锁定脚本。在核实该输出的未来花费时,对节点软件中实现的过程的高级描述如下:

[0232] 1.解析花费事务$Tx_{current}$,并获取该花费事务的输入引用的输出点$TxID_{Prev}$,i_{Prev}。

[0233] 2.对于所获取的每个输出点j,从UTXO数据库中检索对应的[Locking script]。

[0234] 3.对于$Tx_{Current}$的每个输入k,根据所检索的对应[Lcoking script]来核实其[Unlocking script]。

[0235] 4.

[0236] 上述过程的第二步是将先前锁定脚本连接到所引用的输出点,假设UTXO数据库已被正确维护,使得先前锁定脚本的完整性可靠。

[0237] 在节点104不希望存储完整先前锁定脚本,而是实现所提出的经过默克尔处理的脚本系统的情况下,可以更改上述过程以确保$Tx_{Current}$中正被花费的部分锁定脚本的完整性,而无需知道其所有数据。部分锁定脚本可以提供一些数据叶,并隐藏其他数据叶,并且可能仅包括所隐藏的数据叶的哈希。例如,如果先前锁定脚本是OP_CHECKSIG OP_FALSE OP_RETURN<data>,则节点104可以改为仅存储OP_CHECKSIG和用于该脚本的采用经过默克尔处理的形式的OP_RETURN数据叶的对应叶哈希,而不是存储该完整脚本。

[0238] 为此,可以通过以下三种方式来更改UTXO数据库实现方式:1)存储经过默克尔处理的脚本根;2)所需部分脚本;或3)两者的组合。

[0239] 5.3.4.1默克尔根存储

[0240] 在UTXO数据库存储脚本默克尔根的情况下,操作方式类似于采用($TxID_{Prev}$,i_{Prev})形式的密钥值存储:[Merkle root],其中[Merkle root]是将scriptMerklize()算法应用于[Locking script]所产生的根R。

[0241] 然后,可以通过以下方式更改事务核实过程,以确保在$Tx_{Current}$中使用的先前锁定脚本的完整性:

[0242] 1.解析花费事务Tx_{curren}t,并获取该花费事务的输入引用的输出点($TxID_{Prev}$,i_{Prev})。

[0243] 2.对于所获取的每个输出点j:

[0244] 2.1从UTXO数据库中检索对应的[Merkleroot]。

[0245] 2.2解析$Tx_{Current}$的[Unlocking script],确定[Partial Locking Script]正被事务花费。

[0246] 2.3执行默克尔包含证明,以确认[Partial Locking Script]是[Merkle Root]下

集合的成员。

[0247] 3.对于$Tx_{Current}$的每个输入k,根据所检索且经过完整性检查的对应[Partial Locking script]来核实其[Unlocking script]。

[0248] 应当注意的是,在步骤2.2中,使用语法分析来确定先前部分解锁脚本。然而,这可以通过多种方式在实现方式中替换,例如,如果$Tx_{Current}$的输入包含关于部分脚本被满足的明确指示。该部分脚本也可以作为事务提交者与核实者之间的消息传输协议的一部分传递。

[0249] 5.3.4.2部分脚本存储

[0250] 在UTXO数据库存储部分脚本的情况下,操作方式类似于采用以下形式的密钥值存储,即"$TxID_{Prev}$,$Index_{Prev}$:[Partial locking script]",其中[Partial locking script]是在其他脚本数据被修剪和删除时,(比特币)节点选择保留的锁定脚本的部分。

[0251] 然后,可以通过以下方式更改事务核实过程,以确保在$Tx_{Current}$中使用的先前锁定脚本的完整性:

[0252] 1.解析花费事务$Tx_{current}$,并获取该花费事务的输入引用的输出点$TxID_{Prev}$,$Index_{Prev}$。

[0253] 2.对于所获取的每个输出点j,从UTXO数据库中检索对应的[Partiallocking script]。

[0254] 3.对于TxCurrent的每个输入k,根据所检索的对应[Partial locking script]来核实其[Unlocking script]。

[0255] 这种情况与现有实现方式相同,不同之处在于所检索的脚本只是完整先前锁定脚本的一部分。同样,使用以下假设,即UTXO数据库以及因此[Partial locking script]的完整性由先前已经处理区块链的整个历史记录的节点104来确保。

[0256] 应当注意的是,可能存在以不同方式阻止的多个版本的先前锁定脚本,因为已被要求阻止内容的节点可能会阻止不同的数据,取决于司法管辖权、法院命令和偏好等。如果UTXO数据库存储针对同一输出点的多个部分先前锁定脚本,例如"$TxID_{Prev}$,$Index_{Prev}$:[Partialscript1][Partialscript2]",则可能还需要指示应该检索哪个部分脚本,以便对应于$Tx_{Current}$上的花费解锁脚本。例如,假设事务包含具有多个不同字段A、B、C的医疗记录。可能需要在一个司法管辖区(例如,美国)阻止字段A和字段C,而在另一个司法管辖区(例如,英国)仅阻止字段A和字段B。因此,同时为美国和英国提供服务的矿工需要根据不同的阻止标准拥有该事务的两个视图(view)。

[0257] 这在先前解锁脚本可能具有可以独立满足的多个不同分支的情况下尤其相关。也就是说,可以使用非常不同的解锁脚本来核实具有OP_IF分支的事务,可能一个矿工将保留OP_IF…部分,而其他矿工将保留OP_ELSE…部分。每个矿工只能根据TX的两个"视图"中的一个视图来核实该脚本的花费,具体取决于被阻止的内容。与默克尔根存储情况一样,这可以通过提供关于$Tx_{Current}$的输入满足那个先前锁定脚本部分的指示来完成。

[0258] 5.3.4.3组合存储

[0259] 在最后一种情况下,节点104可能希望通过针对UTXO集合中的输出点存储脚本默克尔根和部分脚本来实现UTXO数据库。这将允许节点重新生成即时核实花费事务所需的默克尔证明,而不是从外部源获取这些默克尔证明。

[0260] 然后,可以通过以下方式更改事务核实过程,以确保在$Tx_{Current}$中使用的先前锁定脚本的完整性:

[0261] 1.解析花费事务$Tx_{Current}$,并获取该花费事务的输入引用的输出点($TxID_{Prev}$,i_{Prev})。

[0262] 2.对于所获取的每个输出点j:

[0263] 2.1从UTXO数据库中检索对应的[Merkle root]。

[0264] 2.2解析$Tx_{Current}$的[Unlocking script],确定[Partial Locking Script]正被事务花费。

[0265] 2.3使用剩余部分锁定脚本生成的叶来获取特定[Partial Locking Script]的默克尔证明。

[0266] 2.4执行默克尔包含证明,以确认[Partial Locking Script]是[Merkle Root]下集合的成员。

[0267] 3.对于TxCurrent的每个输入k,根据所检索且经过完整性检查的对应[Partial Locking script]来核实其[Unlocking script]。

[0268] 应当注意的是,除了添加步骤2.3之外,该更新的核实方法与节点仅存储脚本默克尔根的情况相同,其中默克尔证明在这里从UTXO数据库中存储的其他部分脚本数据生成。

[0269] 5.3.5通过选择性披露数据来阻止事务内容

[0270] 在本节中,描述了要被阻止的敏感数据的包含证明,在给定默克尔根、叶(即,采用明文形式的公共数据)和(被阻止数据的)叶的哈希值的情况下,可以很容易地核实事务的正确性。

[0271] 在下文中,提供了特定脚本(OP_FALSE OP_RETURN),但是可以很容易地将其推广到任何(比特币)脚本。假设

[0272] $Leaf_0$=OP_FALSE OP_RETURN OP_PUSHDATA

[0273] $Leaf_1$=$Data_0$

[0274] $Leaf_2$=$Data_1$

[0275] …

[0276] $Leaf_n$=$Data_{n-1}$

[0277] 假设OP_RETURN数据包含秘密内容。在这种情况下,将以此类方式共享最大数量的数据块和子哈希,即它们不会披露关于秘密内容的任何信息,同时仍然可以计算默克尔根。例如,假设事务TxID包含

[0278] $Data=Leaf_1||Leaf_1||\cdots||Leaf_n$

[0279] 其中$Leaf_2$和$Leaf_5$包含不可共享的秘密内容。为了证明存在非秘密部分,同时阻止私有部分,在给定TxID的情况下,如果共享

[0280] $Leaf_1$,SHA256($Leaf_2$),$Leaf_3$,$Leaf_4$,SHA256($Leaf_5$),$Leaf_6$,…,$Leaf_n$,则仍有可能计算数据的默克尔根并(通过TxID)核实事务,而无需披露秘密内容。应当注意的是,当缺少$Leaf_2$和$Leaf_5$时,验证者无法计算OP_RETURN数据的默克尔根。如果没有OP_RETURN数据的默克尔根,则无法计算TxID,因为新的TxID格式需要默克尔根。然而,如果提供H($Leaf_2$)和H($Leaf_5$),则验证者可以很容易地计算默克尔根。因此,H($Leaf_2$)、H($leaf_5$)是执行验证所需的部分锁定脚本的部分。

[0281] 图7示出了包含可花费输出和不可花费输出的示例性事务。该示例示出了如何从TxID计算和事务核实的角度来查看事务。

[0282] 5.4事务默克尔树

[0283] 如上所述,可以使用所编辑的事务的字段来生成默克尔树,以便生成所编辑的事务的二级事务标识符,例如用于确定该事务是否包括称为“候选数据字段”的特定数据。由于在大多数区块链协议中,每个事务与通常通过对完整事务进行哈希处理(或双重哈希处理)生成的初级事务标识符相关联,因此称为二级事务标识符。

[0284] 一种示例性技术包括:将事务字段逐个拆分为一组数据分组(或数据字段),这些数据分组(或数据字段)可以用作默克尔树的叶,其中默克尔树的根对应于二级事务标识符。在本文中,拆分相当于标识事务的数据字段。换句话说,事务实际上不必被“分割”。相反,可以将事务的不同部分标识(例如,分配)为该组数据字段中的不同数据字段。二级事务标识符对于给定事务将是唯一的(例如,给定事务的唯一256位数字表示(哈希值))。该哈希值可用于验证任何单独字段是否是默克尔树的有效叶,而无需获取整组(即,完整事务)。

[0285] 在给定事务Tx的情况下,验证其是否已经包括在区块链的区块中的一种方式是验证其对应的TxID(初级事务标识符)是否出现在区块中。该检查可以通过执行默克尔树证明(例如,默克尔证明)来验证对应于TxID的事务是由区块头中的哈希根表示的事务集的一部分来完成。然而,该检查要求验证者首先获取完整的事务消息m=Tx,并确认TxID=H(Tx)实际上对给定Tx和假设TxID的保持,其中H是哈希函数。这对于区块链的一些用户来说可能是有问题的,在用户实现轻量级客户端和/或消息m较大时尤为如此。

[0286] 在示例中,可以生成符合以下定义的二级事务标识符MTxID:MTxID:=F(Tx,TxID)。算法F用作从两个输入消息Tx和TxID生成二级事务标识符的单向函数。假设TxID可以作为事务消息Tx的函数编写

[0287] TxID:=H^2(Tx),这意味着MTxID可以按照以下方式作为单个消息m=Tx的函数编写:MTxID=F(Tx,H^2(Tx)):=F(Tx)。算法F包括默克尔树生成者。该算法将完整事务作为输入Tx并返回哈希摘要MTxID,该摘要可以用作事务的二级标识符。二级标识符可以是256位哈希摘要。应当理解的是,在该方案中,二级标识符MTxID不替换初级标识符TxID。为了加强这一点,算法F的设计包括例如使用典型的双哈希函数H^2(Tx)生成TxID,该函数将二级标识符绑定到初级标识符。

[0288] 用于生成二级事务标识符MTxID的方法可以分为三个主要阶段,如下所述。

[0289] 输入:Tx

[0290] F(Tx):

[0291] 1)计算TxID:=H^2(Tx)(应当注意的是,这是一个可选步骤)。

[0292] 2)将Tx分为一组$\mathcal{D}$ $N = 2^k$个有序的数据分组$\mathcal{D} \coloneqq \{D_1, D_2, \ldots, D_N\}$。

[0293] 3)使用该组$\mathcal{D}$的分组作为叶生成二进制默克尔树T,并计算其根R。

[0294] 输出:MTxID=R

[0295] 5.4.1阶段1:计算TxID

[0296] 该方法可以包括通过对事务进行哈希处理来生成初级事务标识符。第一阶段是对Tx执行的SHA-256双哈希计算,生成初级事务标识符TxID。可以使用其他哈希函数,并且在一些示例中仅执行单个哈希计算。通常,Tx消息本身不会显式包括TxID。因此,后续阶段需

要对TxID进行该显式预计算,从而可以显式编码该初级标识符的方式构建默克尔树。

[0297] 5.4.2阶段2:将Tx分为有序数据集$\mathcal{D}$

[0298] 将包括消息Tx的事务数据分为离散分组,该离散分组可以用作默克尔树T的叶。可以将事务拆分为其现有字段。这些字段中的大多数包含简单的数字数据,其大小通常非常小-通常介于1字节和32字节之间。因此,大多数事务与sigScript和scriptPubKey字段相关,其分别与输入(解锁)和输出(锁定)相关。可以使用三个类别来区分事务字段:输入字段、输出字段和其他字段。在一些示例中,将事务拆分为这三个类别,例如,一个数据字段包括所有输入字段,一个数据字段包括所有输出字段,一个数据字段包括其他字段。可以将输入字段和输出字段分为非脚本字段和脚本字段,非脚本字段包括数字数据,脚本字段包括脚本数据。

[0299] 下表显示了可以将事务拆分为一组数据字段的方式。

[0300]

字段	大小(字节)	输入、输出或其他?	脚本或非脚本?
版本	4	其他	
txin_count	1-9	其他	
tx_in []			
txid_prev	32	输入	非脚本
vout	4	输入	非脚本
scriptSigLen	可变	输入	非脚本
scriptSig	可变	输入	脚本
sequence	4	输入	非脚本
txout_count	1-9	其他	
txout []			
值	8	输出	非脚本
scriptPubKeyLen	可变	输出	非脚本
scriptPubKey	可变	输出	脚本
locktime	4	其他	

[0301] 该表表明,可以通过几种方式将Tx消息拆分为其组件字段以形成一组分组$\mathcal{D}$。

[0302] 在一些示例中,可以将事务拆分为:至少一个包括该事务的输入数据的数据字段(例如,txid_prev);至少一个包括该事务的输出数据的数据字段(例如,value);至少一个包括该事务的非输入数据和非输出数据(即其他数据,例如version)的数据字段。每个数据字段可以仅由一种类型的数据组成,例如仅由输入数据组成。

[0303] 在其他示例中,可以将事务拆分为更多的数据字段。例如,该组数据字段可以包括:至少一个包括该事务的脚本输入数据的数据字段(例如,scriptSig);至少一个包括该事务的非脚本输入数据的数据字段(例如,vout);至少一个包括该事务的脚本输出数据的数据字段(例如,scriptPubkey);至少一个包括该事务的非脚本输出数据的数据字段(例如,scriptPubKeyLen)。

[0304] 应当注意的是,根据本公开,可以使用基于相应脚本生成的默克尔树的对应默克尔根来替换每个scriptSig和scriptPubKey。

[0305] 可以将事务Tx拆分为以下一组有序数据分组：

[0306] D_1=<version>；

[0307] D_2=<txin_count>；

[0308] D_3=<txout_count>；

[0309] D_4=<locktime>；

[0310] D_5=<txid_prev>//<vout>//<scriptSigLen>//<sequence>；

[0311] D_6=<scriptSig>的默克尔根；

[0312] D_7=<value>//<scriptPubKeyLen>；

[0313] D_8=<scriptPubKey>的默克尔根。

[0314] 如果每个分组D_i是二进制默克尔树的叶，选择以这种方式拆分数据具有几个好处。首先，级联所有非脚本输入以形成分组D_5，并且类似地，级联所有非脚本输出以形成D_7。这意味着，将事务的每个输入和输出准确地拆分两个部分-非脚本组件和脚本组件(或者更确切地说，它们的默克尔根)。这是有利的，因为这意味着每个输入和输出可以作为同级叶进行配对。因此，输入和输出可以与其他字段完全分离，并且每个输入或输出的脚本组件和非脚本组件也可以分离。

[0315] 除了这些字段之外，TxID还可以作为叶包括在默克尔树T中。根据上述示例，这意味着包括数据字段D_9=<TxID>。通常，事务中可以有许多输入和许多输出，每个输入和输出可以通过算法F拆分为两个组件。因此，该树T中数据块$N = |\mathcal{D}|$的总数将由以下方程式给出

[0316] $$N = \underbrace{4}_{\substack{Other\\fields}} + \underbrace{(2 \times n_{in})}_{input\ fields} + \underbrace{(2 \times n_{out})}_{output\ fields} + \underbrace{1}_{TxID},$$

[0317] 其中n_{in}和n_{out}分别是事务输入和输出的数量。在一些示例中，可以级联所有其他数据字段，以形成单个数据字段。这可将叶的总数减少到$N=2+2(n_{in}+n_{out})$。

[0318] 在因没有$k \in \mathbb{Z}^*$而使得$N=2^k$的情况下，可以添加空填充数据的$2^k-N>0$个数据分组，以确保二进制默克尔树有足够的叶。该填充可以是空数据，也可以是TxID的2^k-N个副本，以便加强初级事务标识符与最终二级标识符MTxID之间的链接。

[0319] 5.4.3阶段3：使用$\mathcal{D}$生成默克尔树T并计算根R

[0320] 该方法包括：使用该组数据字段作为事务默克尔树的叶。事务默克尔树包括叶层、一个或多个内部层和根层。叶层包括多个叶节点(也称为叶哈希，因为每个节点是哈希摘要)。每个叶节点是通过对事务的相应数据字段进行哈希处理生成的。至少一个叶哈希基于初级事务标识符。在一些示例中，一个或多个叶哈希是通过对初级事务标识符进行哈希处理生成的。在一些示例中，级联该事务的一个或多个字段与初级事务标识符，然后进行哈希处理以生成相应的叶哈希。每个内部层包括多个内部节点(或内部哈希)。给定内部层中的每个内部哈希是通过对来自较低层的至少两个哈希的级联进行哈希处理生成的。例如，第一或最低内部层(即直接连接到叶层的内部层)包括通过对至少两个叶哈希的级联进行哈希处理生成的内部节点。根层包括事务树的默克尔根，即二级事务标识符。二级事务标识符是通过对一个或多个内部层(即直接连接到根层的内部层)中最高内部层的内部哈希的级联进行哈希处理生成的。

[0321] 二级事务标识符(事务默克尔树的默克尔根)可以包括在区块的生成事务中。然后,可以将该区块记录在区块链中。或者,如下所述,二级事务标识符可以包括在经由网络的一个或多个节点传输给节点的事务。

[0322] 算法F采用一组有序的数据分组$\mathcal{D}$,并通过使用这些分组作为叶构建默克尔树T。图8示出了二进制默克尔树的示例性结构。在该示例中,前四个数据字段$D_1,\cdots,D_4$是事务的其他字段,随后的$2\times(n_{in}+n_{out})$个数据字段是表示为脚本D_5、D_6字段数据和非脚本D_7、D_8字段数据对的输入字段和输出字段。剩余的数据字段包括至少一个字段D_9、D_{10},该字段包含TxID和达到某个整数k的$N=2^k$所需的任何填充D_N。

[0323] 6.进一步评论

[0324] 一旦给出本文的公开内容,所公开技术的其它变体或用例对于本领域技术人员可能变得显而易见。本公开的范围不受所描述的实施例限制,而仅受随附权利要求限制。

[0325] 例如,上面的一些实施例已经根据比特币网络106、比特币区块链150和比特币节点104进行了描述。然而,应当理解的是,比特币区块链是区块链150的一个特定示例,并且上述描述通常可以应用于任何区块链。也就是说,本发明决不限于比特币区块链。更一般地,以上对比特币网络106、比特币区块链150和比特币节点104的任何引用可以分别参考区块链网络106、区块链150和区块链节点104来替换。区块链、区块链网络和/或区块链节点可以共享如上所述的比特币区块链150、比特币网络106和比特币节点104的部分或全部所述特性。

[0326] 在本发明的优选实施例中,区块链网络106是比特币网络,并且比特币节点104至少执行对区块链150的区块151进行创建、发布、传播和存储中的所有所述功能。不排除可能存在仅执行这些功能中的一个或部分功能但不是全部功能的其它网络实体(或网络元件)。也就是说,网络实体可以执行传播和/或存储区块的功能,而不创建和发布区块(请记住,这些实体不被认为是优选的比特币网络106的节点)。

[0327] 在本发明的其他实施例中,区块链网络106可以不是比特币网络。在这些实施例中,不排除节点可以执行对区块链150的区块151进行创建、发布、传播和存储中的至少一个或部分功能但不是所有功能。例如,在这些其它区块链网络上,“节点”可用于指被配置为创建和发布区块151但不存储和/或传播这些区块151到其它节点的网络实体。

[0328] 甚至更通俗地说,上面对术语“比特币节点”104的任何引用可以用术语“网络实体”或“网络元件”代替,其中这样的实体/元件被配置为执行对区块进行创建、发布、传播和存储中的一些或全部角色。这种网络实体/元件的功能可以在硬件中实现,方法与上面参照区块链节点104所述的方式相同。

[0329] 应当理解的是,上述实施例仅通过示例的方式进行描述。更通俗地说,可根据下述任何一个或多个语句提供一种方法、装置或程序。

[0330] 语句1.一种编辑区块链事务的数据的计算机实现的方法,其中所述方法包括:

[0331] 获取第一区块链事务,所述第一区块链事务包括一个或多个相应脚本,所述相应脚本包括待要编辑的相应目标数据;

[0332] 对于所述一个或多个相应脚本中的至少一个相应脚本,基于所述相应脚本来构建相应默克尔树,其中所述相应目标数据在所述相应默克尔树的相应叶中的一个或多个相应叶之间划分;以及,

[0333] 通过使用所述相应默克尔树的相应默克尔根替换所述至少一个相应脚本,来生成所述第一区块链事务的编辑版本。

[0334] 语句2.根据语句1所述的方法,其中每个相应脚本包括一个或多个函数,并且其中所述的构建所述相应默克尔树包括:将一个或多个相应连续函数序列分组为所述默克尔树的相应叶。

[0335] 语句3.根据语句2所述的方法,其中每个相应脚本包括相应数据函数,所述相应数据函数被配置为在执行时指示所述相应目标数据,并且其中所述的构建所述相应默克尔树包括:包括所述相应数据函数作为所述相应默克尔树的相应叶的一部分。

[0336] 语句4.根据语句3所述的方法,其中每个相应脚本包括所述相应目标数据的相应数据长度,并且其中所述的构建所述相应默克尔树是基于所述相应数据长度进行的。

[0337] 语句5.根据语句4所述的方法,其中所述的构建所述相应默克尔树包括:包括所述相应数据长度作为所述相应默克尔树的与所述相应数据函数相同的相应叶的一部分。

[0338] 语句6.根据前述任一项语句所述的方法,其中所述相应目标数据在所述相应默克尔树的多个所述相应叶之间划分。

[0339] 语句7.根据前述任一项语句所述的方法,其中所述第一区块链事务包括多个相应脚本,所述多个相应脚本包括相应目标数据,并且其中所述方法包括:

[0340] 对于所述多个相应脚本中的每个相应脚本,基于所述相应脚本来构建相应默克尔树,其中所述相应目标数据在所述相应默克尔树的所述相应叶中的一个或多个相应叶之间划分;

[0341] 通过使用所述相应默克尔树的相应默克尔根替换每个相应脚本来生成所述第一区块链事务的所述编辑版本。

[0342] 语句8.根据前述任一项语句所述的方法,所述方法包括:通过对所述区块链事务的所述编辑版本进行哈希处理来生成所述第一区块链事务的所述编辑版本的初级事务标识符。

[0343] 语句9.根据前述任一项语句所述的方法,所述方法包括:响应于对所述第一区块链事务的请求,将所述区块链事务的所述编辑版本传输给实体。

[0344] 语句10.根据前述任一项语句所述的方法,所述方法包括:将所述第一区块链事务的所述编辑版本存储在存储器中。

[0345] 语句11.根据前述任一项语句所述的方法,其中所述方法由事务生成者执行。

[0346] 语句12.根据语句11所述的方法,所述方法包括:

[0347] 为消息生成签名,其中所述消息的一个或多个部分基于所述第一区块链事务的所述编辑版本;

[0348] 将所述签名包括在所述第一区块链事务的输入中;以及,

[0349] 将所述第一区块链事务提交给区块链网络进行核实。

[0350] 语句13.根据语句12所述的方法,其中所述签名是椭圆曲线数字签名算法(ECDSA)签名。

[0351] 语句14.根据语句12或13所述的方法,其中所述输入引用先前区块链事务的输出,所述先前区块链事务包括相应脚本,所述相应脚本包括相应目标数据,其中所述消息的一部分基于所述先前区块链事务的所述输出,并且其中基于所述先前区块链事务的所述输出

的所述消息的所述部分基于默克尔树的默克尔根,所述默克尔树基于所述相应脚本来构建,其中所述相应目标数据在所述相应默克尔树的所述相应叶中的一个或多个相应叶之间划分。

[0352] 语句15.根据语句1至10中任一项所述的方法,其中所述方法由事务核实者执行。

[0353] 语句16.根据语句15所述的方法,所述方法包括:

[0354] 从所述第一区块链事务的输入中提取签名;以及,

[0355] 验证消息的所述签名,其中所述消息的一个或多个部分基于所述第一区块链事务的所述编辑版本。

[0356] 语句17.根据语句16所述的方法,其中所述输入引用先前区块链事务的输出,所述先前区块链事务包括相应脚本,所述相应脚本包括相应目标数据,其中所述消息的一部分基于所述先前区块链事务的所述输出,并且其中基于所述先前区块链事务的所述输出的所述消息的所述部分基于默克尔树的默克尔根,所述默克尔树基于所述相应脚本来构建,其中所述相应目标数据在所述相应默克尔树的所述相应叶中的一个或多个相应叶之间划分。

[0357] 语句18.根据语句15或其任何从属语句所述的方法,所述方法包括:将所述第一区块链事务的所述编辑版本存储在区块链上。

[0358] 语句19.根据语句8和语句15至18中任一项所述的方法,其中所述相应脚本中的至少一个相应脚本是所述第一区块链事务的第一输出的第一锁定脚本,并且其中所述方法包括:

[0359] 在数据库中存储:相应默克尔根,所述相应默克尔根映射到所修改的初级事务,和所述第一输出的相应索引。

[0360] 语句20.根据语句19所述的方法,所述方法包括:

[0361] 获取第二区块链事务,其中所述第二区块链事务包括第一输入,所述第一输入引用所述第一区块链事务的所述修改的初级事务标识符,和所述第一输出的所述索引,其中所述第一输入包括第一解锁脚本,并且其中所述方法包括通过以下方式来核实所述第二区块链事务:

[0362] 从所述数据库中检索:所述相应默克尔根,所述相应默克尔根映射到所述修改的初级事务标识符,和所述第一输出的所述相应索引;

[0363] 基于所述第一输入的所述第一解锁脚本来确定所述第一输出的部分锁定脚本;

[0364] 执行默克尔包含证明,以确认所述部分锁定脚本是与所述相应默克尔根对应的所述相应默克尔树的相应叶;以及,

[0365] 根据所述部分锁定脚本来核实所述第一解锁脚本。

[0366] 语句21.根据语句8和语句15至18中任一项所述的方法,其中所述相应脚本中的至少一个相应脚本是所述第一区块链事务的第一输出的第一锁定脚本,并且其中所述方法包括:

[0367] 在数据库中存储:部分锁定脚本,所述部分锁定脚本映射到所述修改的初级事务标识符,和所述第一输出的相应索引,其中所述部分锁定脚本包括至少部分所述第一锁定脚本但不包括所述目标数据。

[0368] 语句22.根据语句21所述的方法,所述方法包括:

[0369] 获取第二区块链事务,其中所述第二区块链事务包括第一输入,所述第一输入引

用所述第一区块链事务的所述修改的初级事务标识符,和所述第一输出的所述索引,其中所述第一输入包括第一解锁脚本,并且其中所述方法包括通过以下方式来核实所述第二区块链事务:

[0370] 从所述数据库中检索:相应部分锁定脚本,所述相应部分锁定脚本映射到所述修改的初级事务标识符,和所述第一输出的所述相应索引;以及,

[0371] 根据所述部分锁定脚本来核实所述第一解锁脚本。

[0372] 语句23.根据语句19和21所述的方法,所述方法包括:

[0373] 从所述数据库中检索:所述相应默克尔根和所述部分锁定脚本,其映射到所述修改的初级事务标识符,和所述第一输出的所述相应索引;

[0374] 基于所述第一输入的所述第一解锁脚本来确定所述第一输出的部分锁定脚本;

[0375] 使用基于所述检索的部分锁定脚本生成的一个或多个相应叶来获取所述确定的部分锁定脚本的默克尔证明;

[0376] 执行默克尔包含证明,以确认所述确定的部分锁定脚本是与所述相应默克尔根对应的所述相应默克尔树的相应叶;以及,

[0377] 根据所述确定的部分锁定脚本来核实所述第一解锁脚本。

[0378] 语句24.根据前述任一项语句所述的方法,其中所述编辑的事务包括多个字段,并且其中所述方法包括:

[0379] 生成事务默克尔树,其中所述事务默克尔树的相应多个相应叶由所述编辑的事务的一个或多个字段形成;以及,

[0380] 生成所述编辑的事务的二级事务标识符,其中所述二级事务标识符包括所述事务默克尔树的默克尔根。

[0381] 语句25.根据语句24所述的方法,其中所述事务默克尔树的一个或多个叶由所述编辑的事务的多个字段形成。

[0382] 语句26.根据语句24或25所述的方法,其中所述事务默克尔树的一个或多个叶由所述编辑的事务的单个字段形成。

[0383] 语句27.根据语句24或其任何从属语句所述的方法,其中所述事务默克尔树的一个或多个叶由所述编辑的事务的相应字段的相应部分形成,而不是由完整的相应字段形成。

[0384] 语句28.一种计算机设备,所述计算机设备包括:

[0385] 存储器,所述存储器包括一个或多个存储器单元;以及,

[0386] 处理装置,所述处理装置包括一个或多个处理单元,其中所述存储器存储被设置在所述处理装置上运行的代码,所述代码被配置为当在所述处理装置上运行时,执行根据语句1至27中任一项所述的方法。

[0387] 语句29.一种计算机程序,所述计算机程序包含在计算机可读存储器上并且被配置为当在一个或多个处理器上运行时,执行根据语句1至27中任一项所述的方法。

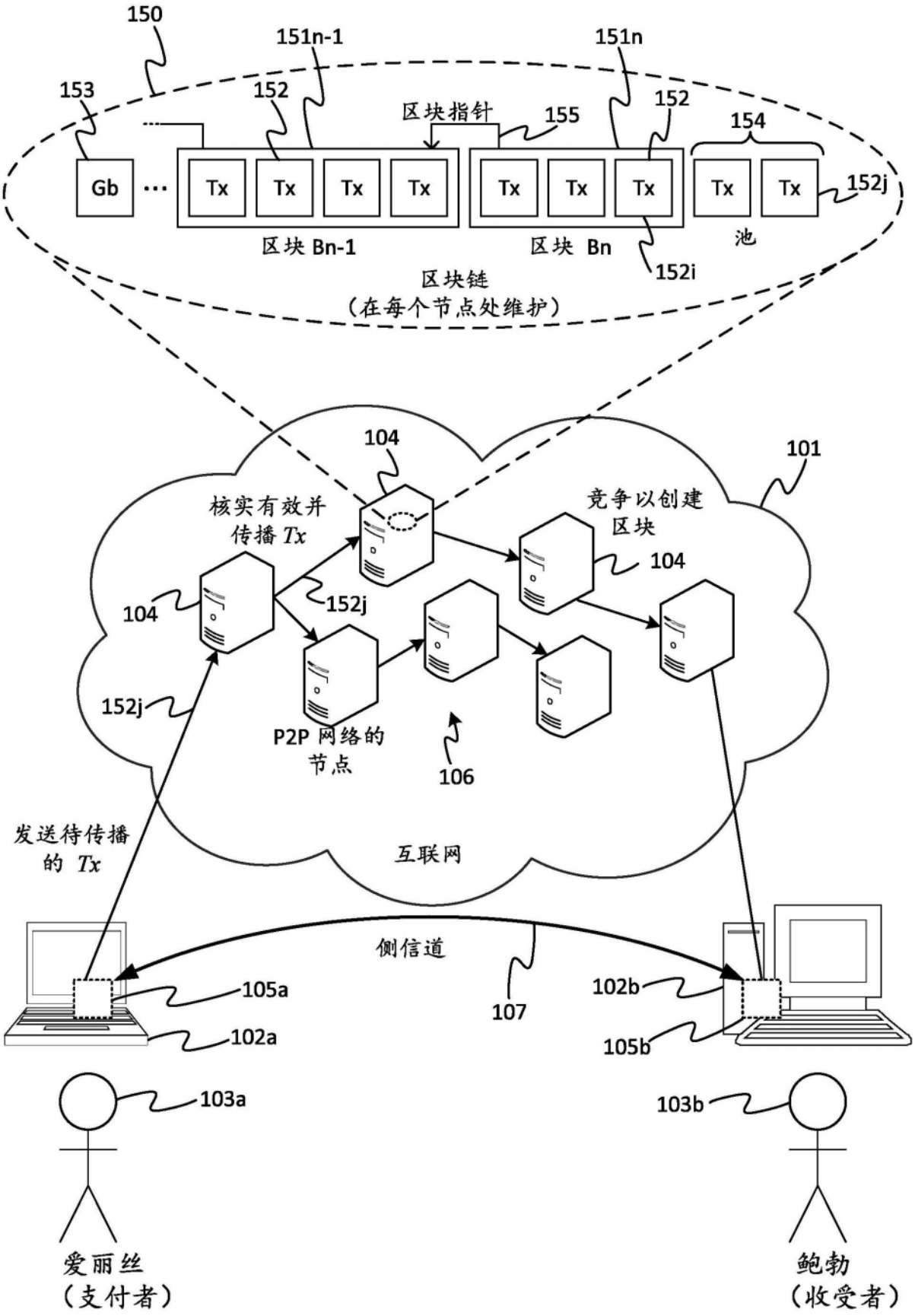

150
151n-1
151n
153
152
区块指针
155
152
154
Gb
Tx
Tx
Tx
Tx
Tx
Tx
Tx
Tx
Tx
152j
区块 Bn-1
区块 Bn
池
152i
区块链
（在每个节点处维护）
104
101
核实有效并
传播 Tx
竞争以创建
区块
104
104
152j
152j
P2P 网络的
节点
106
发送待传播
的 Tx
互联网
侧信道
107
105a
102a
102b
105b
103a
103b
爱丽丝
（支付者）
鲍勃
（收受者）

图1

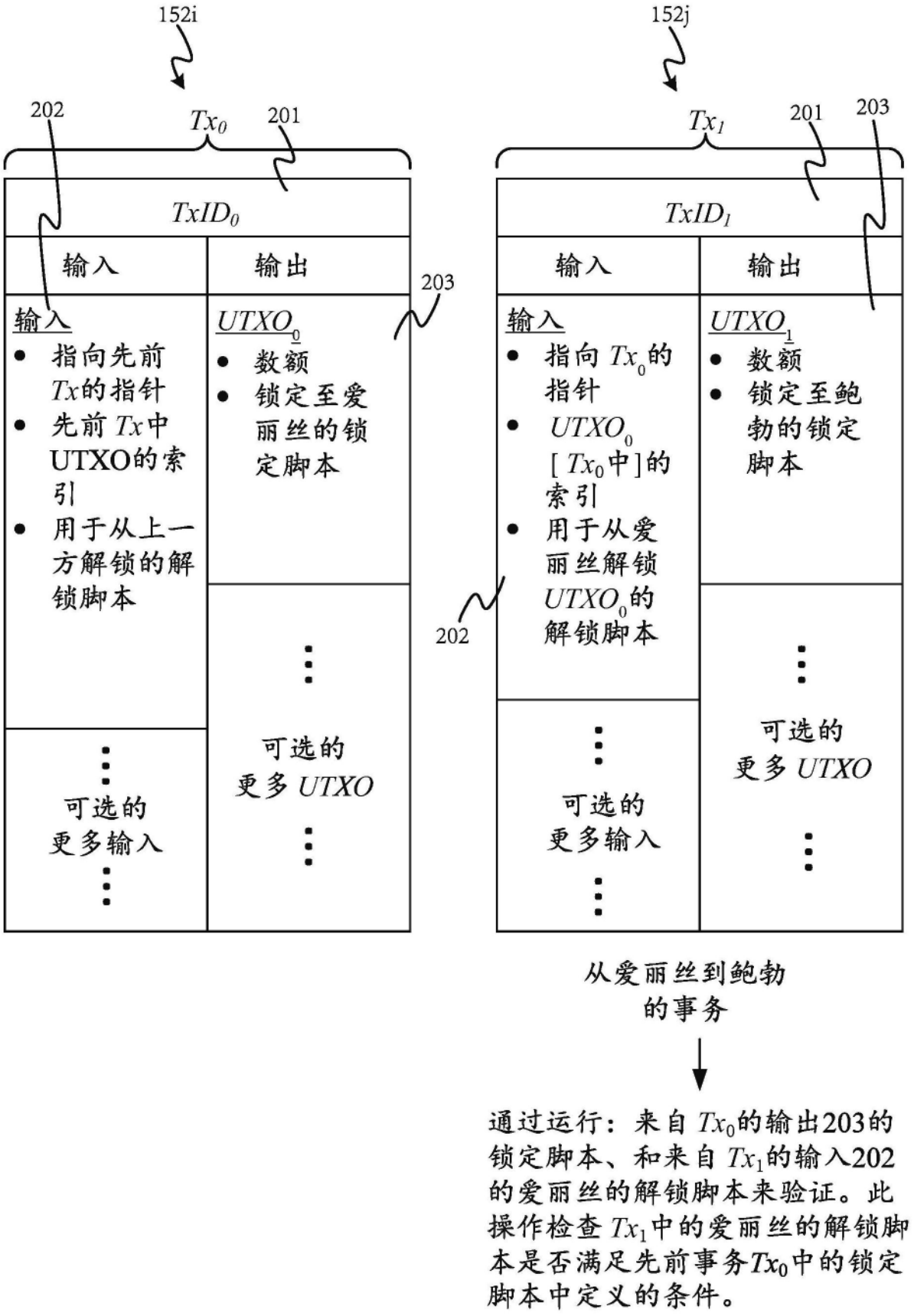

152i
152j
202
Tx_0
201
203
Tx_1
$TxID_0$
$TxID_1$
输入
输出
输入
• 指向先前Tx的指针
• 先前Tx中UTXO的索引
• 用于从上一方解锁的解锁脚本
$UTXO_0$
• 数额
• 锁定至爱丽丝的锁定脚本
可选的更多输入
可选的更多 UTXO
• 指向 Tx_0 的指针
• $UTXO_0$ [Tx_0中]的索引
• 用于从爱丽丝解锁 $UTXO_0$ 的解锁脚本
$UTXO_1$
• 数额
• 锁定至鲍勃的锁定脚本
从爱丽丝到鲍勃的事务
通过运行：来自 Tx_0 的输出203的锁定脚本、和来自 Tx_1 的输入202的爱丽丝的解锁脚本来验证。此操作检查 Tx_1 中的爱丽丝的解锁脚本是否满足先前事务 Tx_0 中的锁定脚本中定义的条件。

图2

$TxID_1$			
版本	1	锁定时间	0
输入计数	1	输出计数	2
输入列表		输出列表	
输出点	解锁脚本	值	锁定脚本
TxID、索引	$< Sig_A >< P_A >$	x 聪	[P2PKH P_B]
		0	OP_FALSE OP_RETURN <data>

图3

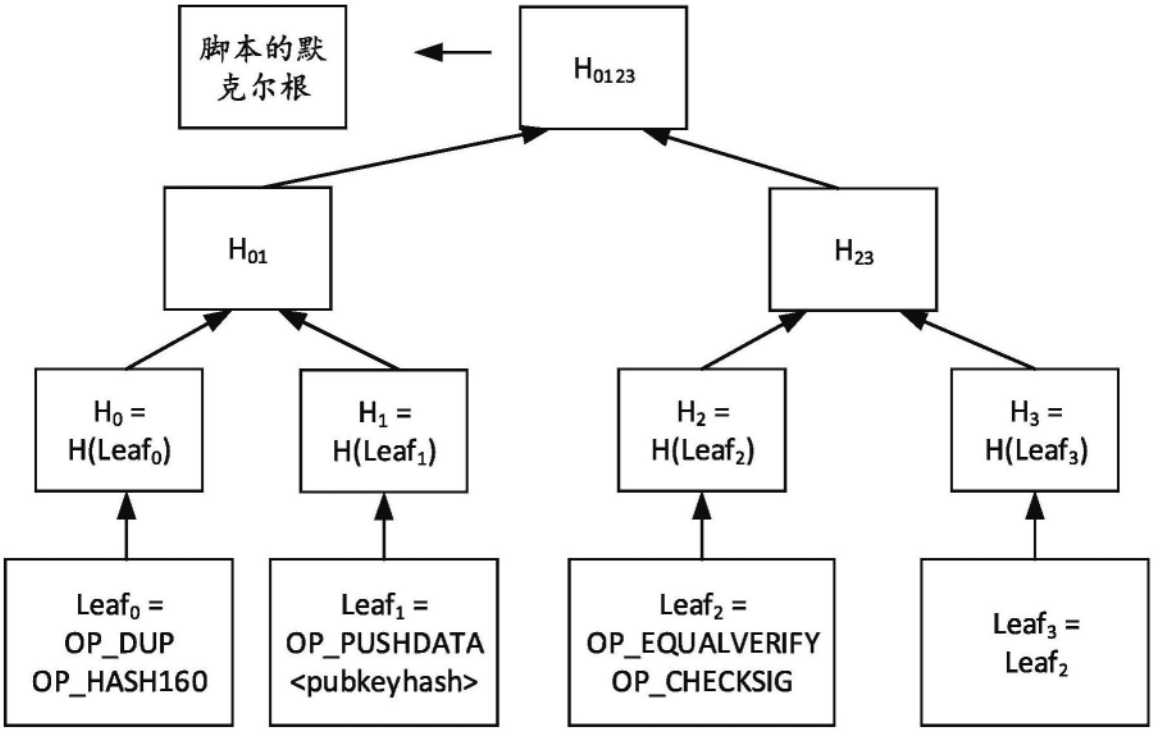


图4

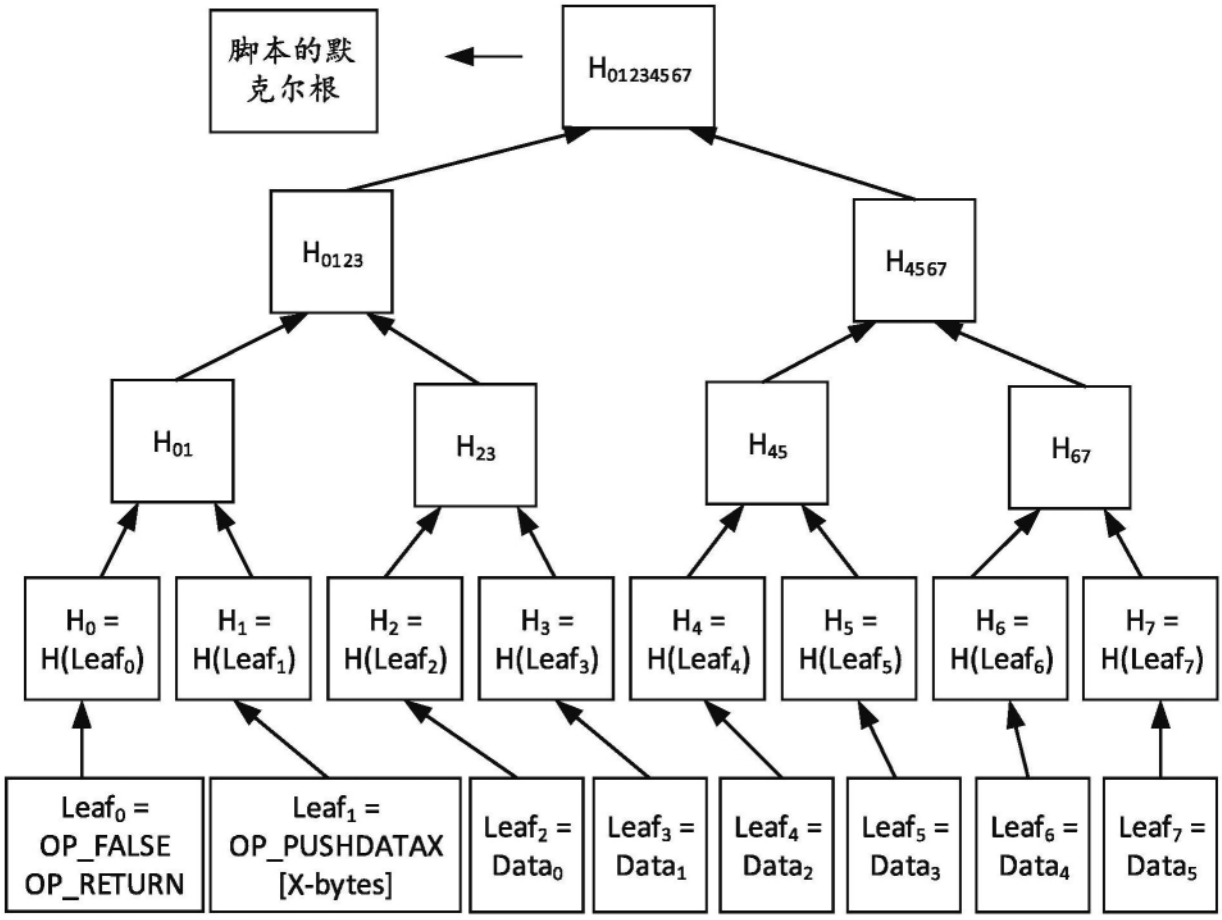

脚本的默克尔根
$H_{01234567}$
H_{0123}
H_{4567}
H_{01}
H_{23}
H_{45}
H_{67}
$H_0 = H(Leaf_0)$
$H_1 = H(Leaf_1)$
$H_2 = H(Leaf_2)$
$H_3 = H(Leaf_3)$
$H_4 = H(Leaf_4)$
$H_5 = H(Leaf_5)$
$H_6 = H(Leaf_6)$
$H_7 = H(Leaf_7)$
$Leaf_0$ = OP_FALSE OP_RETURN
$Leaf_1$ = OP_PUSHDATAX [X-bytes]
$Leaf_2 = Data_0$
$Leaf_3 = Data_1$
$Leaf_4 = Data_2$
$Leaf_5 = Data_3$
$Leaf_6 = Data_4$
$Leaf_7 = Data_5$

图5

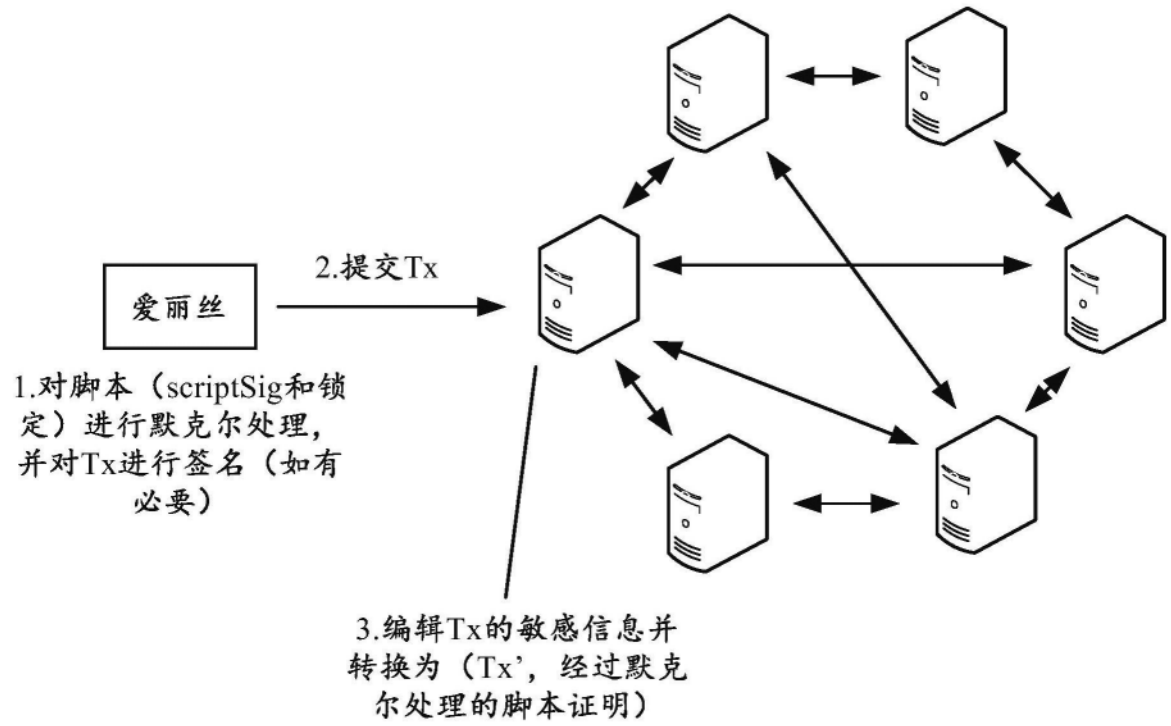


图6

$TxID_1$			
版本	1	锁定时间	0
输入计数	1	输出计数	2
输入列表		输出列表	
输出点	解锁脚本	值	锁定脚本
TxID、*索引*	Merkle Root of $< Sig_A >< P_A >$	*x* 聪	Merkle Root of [P2PKH P_B]
		0	Merkle Root of OP_FALSE OP_RETURN <data>

图7

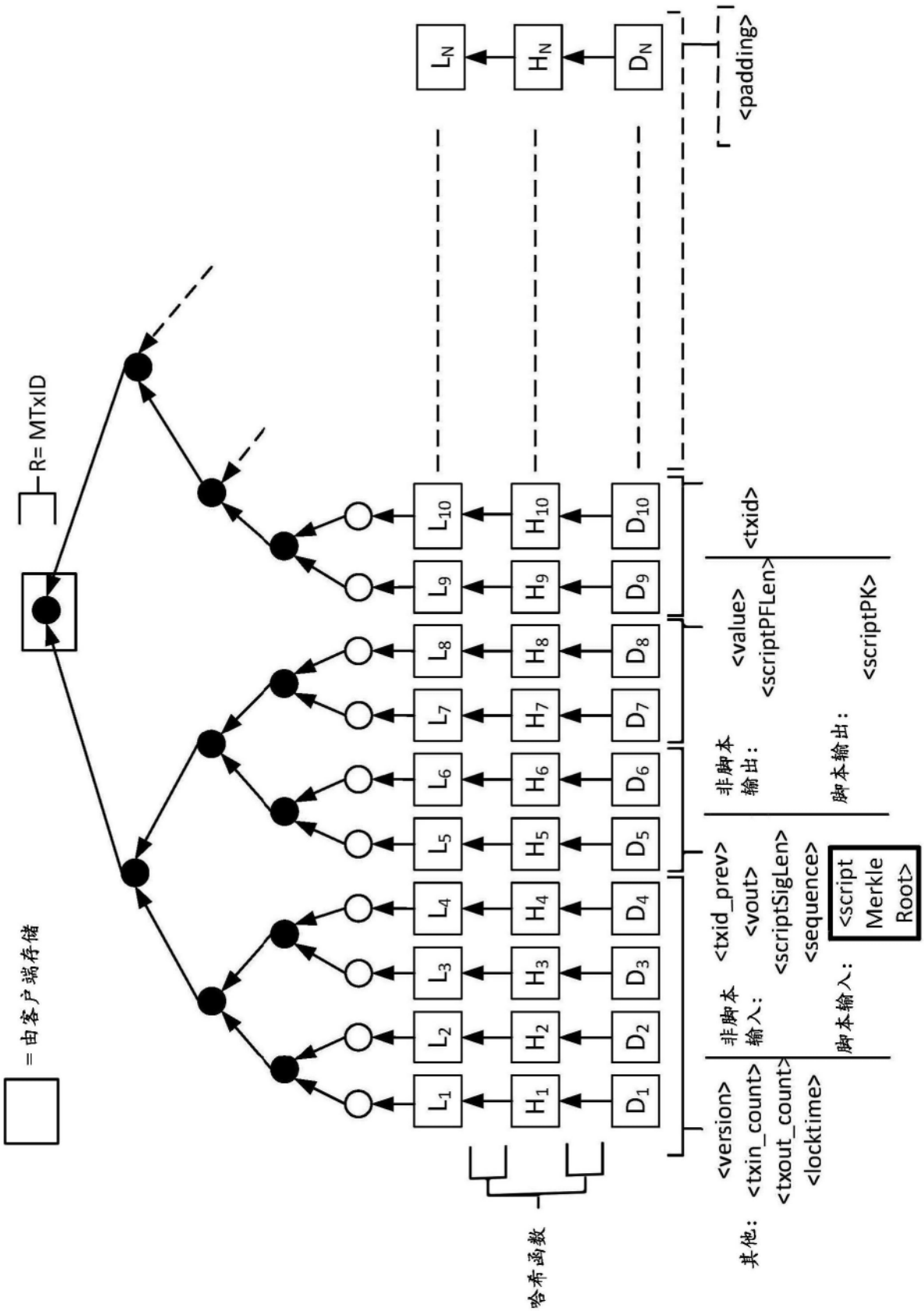

= 由客户端存储
R= MTxID
哈希函数
其他：
<version>
<txin_count>
<txout_count>
<locktime>
非脚本输入：
<txid_prev>
<vout>
<scriptSigLen>
<sequence>
脚本输入：
<script Merkle Root>
非脚本输出：
<value>
<scriptPFLen>
脚本输出：
<scriptPK>
<txid>
<padding>
L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 LN
H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 HN
D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 DN

图8