LTO Network

Blockchain for Decentralized Workflows

www.lto.network

2019 年 1 月 2 日

摘要

数字化和业务流程自动化为降低成本和增加生产力方面提供了巨大的好处。然而，各机构组织仍很难为机构之间的工作流程将这些自动化的好处利用起来，部分原因是缺乏信任。比特币已经证明了区块链如何使用分布和加密技术来提供一个不依赖于信任的系统。

LTO 在此基础上构建了一个去中心化的工作流引擎，专注于特定及专有的协作。使用私有链（每个进程的新链）在各方之间共享信息，并在公链上进行哈希。这种混合型的方案使得各机构组织能够满足任何的数据保护法规。

前言

数字革命带来了许多变化，致力于提高我们的生活效率 [1]。这一波进展浪潮主要发生在面向消费者和

内部业务流程中。在各机构与组织之间的流程方面，我们必须承认这些变化并不是那么激烈。信件和传真在很大程度上被电子邮件所取代，打字机也已经被文字处理器所取代，但除了这些表面的变化之外，基础流程的执行几乎没有改变。

自动化未达成的主要原因是公司不愿意依赖由对手操作的外部系统 [2]，因为信息的分配在关系的结果中起着重要作用 [3]。当某处自然力量不平衡时，一方就可能掌握控制权，迫使其他人使用这一单方的集中管理系统。当我们在与政府打交道时，及在某种程度上与公司打交道时，很容易观察到这一点。当在没有任何人可以掌握控制权的情况下，自动化也就根本不会发生 [4]。

为了解决有关机构组织之间流程自动化的问题，二十多年以来人们一直在尝试去中心化的工作流程 [5]。在这些研究和实验中，假设的高度信任和公平竞争是默认存在的，而我们也主要侧重于解决技术挑战。而实际上，这是一个错误的假设，因为信任的缺乏阻碍了将小范围的试行成功投入生产的过程。

缺乏自动化的另一个原因是效率与腐败之间的关联性 [6]。按照一贯以来的传统，大公司和政府机构需要大量人员来执行流程。并且还需要相当数量的官僚机构来协调这些过程。如此以来，贿赂的代价增加了，执行自动化的激励力也降低了。但是，提高效率会抵消这种影响。

本文就将演示如何通过使用区块链解决这两个问题，以及提供一个所有各方都可以站在同一起跑线上的解决方案。

目录

第一部分　交互合约

业务流程建模是任何大中型组织的常见策略 [7]。通过工作流程过程的可视化可以对其进行分析、改进和自动化 (图 1) (figure 1)。这些模型人类与计算机都可以理解，与仅使用自然语言或编程语言编写的过程不同。

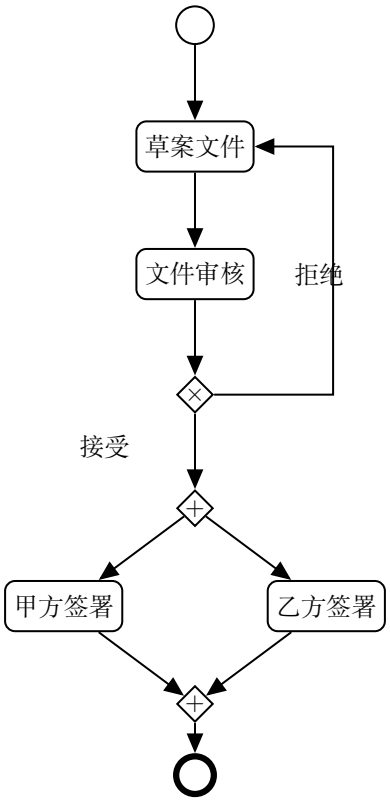


图 1: 业务流程建模和表示法图可以用来显示某工作流程。

对于组织之间的合作，建模不仅是为了改善沟通。各方必须具体说明工作程序，这将作为具有约束力的协议 [8]；这就是 LTO 平台所谓的 **交互合约**.

LTO 平台给每一个交互合约都会创建一条临时性的私有区块链。此链的目的并不是作为不可改变的账簿，而是保留各方组织都拥有最新的会签历史事件和共享状态。

1 智能 vs 交互合约

交互合约与实施于以太坊区块链的智能合约目标相似 [9]。两者都定义和巩固可在无信任和可验证的情况下施加的逻辑。

但是，这两种数字化合约背后的理念非常不同。以太坊将智能合约描述为包含价值的加密“盒子”，而这盒子只有在满足某些条件时才会解锁 [10]。

交互合约并不直接保留价值，而是描述两方或多方应如何互动。它们的意图更接近传统（纸质）合同的意图。

1.1 李嘉图合同

交互合约符合李嘉合同的定义[1][11]。最值得注意的是，人类和程序都可轻松阅读。这是从定义交互合约的方式获得的属性。既没有用于法律目的的自然语言版本，也没有用于编程执行的代码版本。

1.2 执行

链上执行不适用于很多现实生活的案例。智能合约依赖于主动执行，这意味着违反协议必须是不可行的或者得使任何一方必须退出 [12]。

就以保密协议为例。区块链不可能阻止一方透露信息，也无法强迫一方积极参与解决泄密行为。如果希望这种协议成为自我执行的合约 [13]，它必须将全部违约金扣住作为押金。这将确保每一方都会为自己的利益参与决议。

对于大多数组织而言，将大量资金用作合同的违约金存款是不切实际的 [14]。除此之外，罚款和类似措施的有效性仅限于智能合约所持有的押金。

大多数业务流程都要求通过权威机构链下解决争议。交互合约可以促进争议的删除过程。这可能包括冲突谈判，调解，甚至仲裁（通过仲裁者或法官）。

在 LTO 平台上运行流程会形成可验证的事件历史记录，从而减少不对称的信息。信息的分配会影响争议时的谈判 [3] ，并影响第三方权威机构进行的评估。

1.3 用户界面

以太坊提供了一种内置的图灵完备脚本语言，程序员可以使用它来构造任何数学方式定义的智能合约或交易 [10]。这使它们非常抽象，因正在运行的合约所包含的状态没有内在含义。

[1]李嘉合同可以定义为一份文件即 a) 发行人向持有人提供的合同，b) 由持有人持有并由发行人管理的有价值权利，c) 人们容易阅读（比如纸上合同），d) 程序可读（可解析为数据库），e) 拥有数字签名，f) 携带密钥和服务器信息，还有 g) 与独特且安全的标识符绑定。

要与此类合约进行交互的话，必须为特定的智能合约创建特定的用户界面，或者说，创建此类合约的界面 [15]。这些界面可以标准化，如 ERC-20[16] 与 ERC-721[16]，会把用户界面与合约逻辑分开。缺点是这也限制了设计合约的可能性。

通过交互合约，合约内的信息确实具有内在意义。虽然这限制了用例，但它确实能够纯粹基于合同及其过程提供的数据生成界面。因此，任何工作流程都可以在 LTO 平台上进行数字化和执行，而无需为每个工作流程创建特定的界面。

1.4 指令与整合

交互合约包含针对特定人员或节点的指令。一个节点可以根据指令行动或通知用户某个操作即将被执行。此操作可包括通过 HTTP API 调用从互联网获取信息。

在以太坊和 Hyperledger 中实施的智能合约逻辑只能改变区块链的状态。智能合约需要通过预言机将数据从外部源提交到区块链 [17]。

这个预言机可以根据智能合约的状态行事，但是这预言机的逻辑不会是合约的一部分。不过，这种逻辑在本平台是交互合约的一部分，因此各方参与者可能会对其进行验证，并可能引起争议。

2 有限状态机

交互合约将工作流定义为 **有限状态机** (FSM)[18]。这使我们可以将其显为流程图 (figure 2)。这样，人类和计算机都可以理解工作流程。

2.1 确定性有限状态机

任何区块链逻辑都需要是确定性的 [19]。计算机程序可能需要额外的努力来符合此标准，但是确定性有限状态机（DFSM）本身是确定性的。

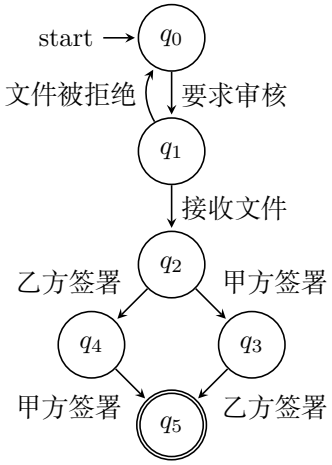


图 2: 化为流程图的有限状态机的示例

2.2 扩展有限状态机

图 2 显出了需要多数操作到达某个状态而它们发生的顺序是任意时，会出现问题。这可以被建模为每个可能顺序的过渡路径，如图 2所示。但是，通过这种方案解决，状态和状态转换的数量将随着操作的数量呈指数增长。这不仅使工作流的图形不太清晰，而且还会使定义工作流更加困难和容易出错。

因此，交互合约不使用常规有限状态机，而使用的是允许条件状态转换的 (EFSM) **扩展有限状态机** [20]。

图 3 用 EFSM（扩展有限状态机）视出与图 2 相同的工作流程。

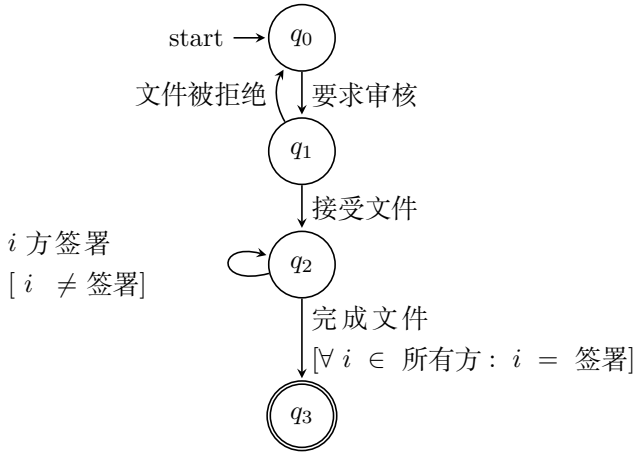


图 3: 扩展有限状态机的示例：括号中的条件必须为真才能使转换有效

2.3 交流性有限状态机

有限状态机仅限于顺序行为；而们不支持并发进程。若要表示具有并发性的工作流，每个并行指令序列需要画为独立的有限状态机。

交流性有限状态机 (CEFSMs) 允许通过组合各个事件序列来建模更复杂的过程。

事件链 (见第 9节) 可以作为两个 FSM 之间的交流通道。使用不同的事件链隔离两个进程会使整个系统不确定，因为通信通道是非确定性的 [21]。这可以通过确认事件来克服，如 13.1节所示。

2.4 作为自动机的合约

有限状态机可以作为参与者之间的协议用，通过规范义务，权限和禁止各方强加于另一方 [22]。金融协议等合同 [23] 和服务合同 [24] 可以完全数字化为 FSM。

然而，在本文章中所提供的表示图 [23, 24] 都不足以用作工作流程的条件，因为它们没有定义流程中的通信和编排。虽然可以在图中结合这些因素，它会使有限状态机成为指数级更复杂 [25]。

实际上，FSM 最多仅能代表一份不完整的合同。然而，这不一定是一个问题，因为这些缺口可以用默认规则填充 [26]。该系统允许重新谈判交互合约内容，无论是解决特定情况或一般情况，如第 8节中所示。

另一件需要注意的事情是，在某过程中，不是每个操作都会构成一个约束

因素。例如，在图 1 中，接受文件不会构成具有约束力的协议；这仅在文档签署时发生。为了促进这种区分，我们可以将行动分类为信息性或执行性 [27]。

3 备择建模方法

交流性有限状态机通常用于描述电信系统和其他实时系统 [28]，而不用于业务流程的描述。

在对去中心化的组织之间的流程进行建模时，更常见的工作流注释会带来更多的挑战。

3.1 佩特里网 Petri

佩特里网 [29] 是一个系统的图形表示，其中可见到多个独立活动同时进行。Petri 和 FMS 之间的区别在于 Petri 能够对多个并发活动进行建模。在 FSM 中，始终都存在单个 “当前” 状态，该状态决定下一个可能发生的行动。在 Petri 网中，可能存在几种状态，其中任何一种状态都可能通过改变 Petri 网的状态而发展。某些，甚至所有状态可以并行发展，导致 Petri 网的一些独立变化同时发生 [30]。

可用于描述业务流程的工作流网（WF-net）[31]，是 Petri 网的子集。WF 网可以描述整个过程，而不仅仅是一个序列。

EFSM 使用综合状态来保存信息。此信息必须按顺序被隔离。若失败将数据化成不可改变可被利用，如第 5.1节所示。在 Petri 网是不可能用通用信息的。反而，信息必须流经工作流程。

通过 CEFSM 的方法，每个序列会被定义为单独的过程。这使将访问控制应用于一部分过程一项微不足道的任务。不过，当设计整个工作流程时，不能以相同的方式解决这个问题。

Petri 网有一个有趣的注释。多数研究 [31] 表明它们可用于表示业务工作流程。鉴于可以将 CEFSM 建模为 petri 网，使用 WF 网可能比使用单个 FSM 更合适。

由于 Petri 网和 FSM 之间的相似性，如本案文所述，切换到 WF 网并本质上不会改变我们的解决方案。

3.2 业务流程模型和标记法

业务流程模型和标记法 (BPMN) 是业务模型处理的行业标准，并且可能成为 LTO 的建模标记法候选之一。可是，BPMN 有多数尤其对于跨组织系统造

成不便的限制 [32]。

通常与 BPMN 相关联的业务过程执行语言 (BPEL)，是一种用于 Web 服务的非确定性图灵完备语言 [33]。这使得它不适合区块链的自动化。

建议的替代方案是将 BPMN 模型转换为 Petri 网，Petri 网又转换为智能合约 [34]。

虽然（图灵完备）智能合约转换步骤是多余的，从 BPMN 到 Petri 网（或 CEFSM）的转换还收有意义的，以支持当前的行业标准。

3.3 组织的设计和工程方法 DEMO

“DEMO” 是 “组织的设计和工程方法” 的缩写。这种用于描述组织及其业务流程的方法基于 “交流行为”。它使用四种模型来创建整体视图，即构造模型（CM），过程模型（PM），行动模型（AM）和事实模型（FM）[35].

DEMO 会给每一个执行的动作易建立一个通用的工作流程，而不是将过程中的每一步单独研究。此动作中有两个角色，就是发起者和执行者。标准序列如下进行：发起者发出请求，执行者做出承诺。执行者然后执行操作并对结果做出声明; 发起可以么接受这个，然后完成交易，或者拒绝它 [27]。

其他流程也可以被建模，例如执行者拒绝请求，发起者想要撤销请求，执行者无法履行承诺等。当使用其他建模方法时，这些替代流程通常不会或仅部分地建模，可在实践中，它们总是存在。

流程模型（PM）将这些事务组合在一起，以模拟完整的业务流程。这个模型和工作流程模型之间的区别在于，在这个模型中，每个人都尽可能并行工作，在需要的地方指定事务之间的依赖关系。这种设计选择意味着与其他建模方法相比时，在 DEMO 模型中我们无法清楚地了解我们在流程中的位置。此外，互相排斥信息（不要编辑准备签名的文件）并非身临其境。相反，它需要具体被指定。

DEMO 可能是制作高级模型的好方法，从而产生可以进行微调的工作流程。这应该会产生更完整的合同，减少对派生的依赖 (see Section 8.2)。

4 场景

工作流被定义为数据对象; 场景。它由以下元件组成：

- q_x 作为一个状态，有 q_0 为初始状态,
- Q 作为所有可能状态的集合 $Q = \{q_0, ..., q_{n-1}\}$,

- σ_x 作为某个操作,
- Σ 作为所有可能行动的集合 $\Sigma = \{\sigma_1, ..., \sigma_n\}$,
- δ 作为状态转移函数 $\delta : Q \times \Sigma \to Q$,
- F 作为最终状态的的集合 $F \subset Q \mid F = \emptyset$,
- $\bar{I}$ 作为所有角色定义,
- $\bar{A}$ 作为所有资产定义,
- D 作为嵌入数据对象的集合。

4.1 状态

集合在 Q 中的状态通常包含以下内容:

- 标题: 给某个状态设定的小标题,
- 带有交易的操作为 $\{(\overrightarrow{q_x}, \delta), ...\}$,
- 描述: 给状态的描述,
- 给特定角色的地图或指令。

每个状态描述了在此状态下可以执行的操作，包括状态转换。这允许不同操作可在不同的状态下使用。

4.2 操作

Σ 由场景定义，即可在工作流中执行的所有可能操作。此类操作的示例包括填写表单，查看某个文档以及执行 HTTP 调用。操作类型与物件属性是使用 JSON 模型定义的。

每个操作都会定义来自 $\bar{I}$ 套的哪些角色可以执行它而且可选附加的执行约束。这些约束使场景成为一个 *扩展* FSM。

当执行一个操作时，会触发状态转换。操作可被分类为需要执行人为干扰的手动操作，以及可以由系统自动执行的系统操作。

每个操作也可以下令，使用回复的数据去更新角色与资产。

4.3 角色

$\bar{I}$ 套定义可以在流程中发挥作用的所有角色。每个角色都使用 JSON 模型被定为一个物件。与进程相关的角色属性必须定义。

某个场景中的角色只是一个静态定义，可以在进程中实例化。

4.4 资产

$\bar{A}$ 定义流程中可用的所有资产。资产是可变数据对象。与流程相关的属性必须定义。

请注意，场景仅定义资产的结构。资产只能在流程中实例化。

5 数据对象

除了情景场景，还可以定义其他类型的数据对象。所有数据对象（包括场景）都使用 JSON 模型作为类型定义。常见示例包括表单，文档和模板。

数据对象可以嵌入到进程中，也可以独立链接和存储。

链接对象由其 JSON 表示的 SHA256 哈希标识。为了确保 JSON 编码数据始终产生相同的结果，链接时用到了确定性的 JSON 编码方法。

5.1 不变性

数据对象是不可变的，当数据对象被修改时，它产生一个新的数据对象。如果此数据对象作为资产嵌入到流程中，则旧对象将被修改后的对象替换。

值得注意的是，如果某一个数据对象在多数进程中用到，其更改将不会自动传播到其他进程。

实现不了不变性可能导致可利用的情况。在图 1，我们展示了谈判和签署文件的过程。很明显，在签名过程文档中不应该被修改。

5.2 表格

表格定义使用 JSON 模型来定义填写表格时应该产生的数据结构。可以使用附加的 UI 模式来指定如何呈现和显示相应的字段。

现实已有类似的措施 [36–39] 。我们的目标是与这些项目合作，形成统一的标准。

5.3 文件

数字工作流程可以大量消除纸质文档的需求。但是，法律合规性，向下兼容性和公司政策等可能仍需要使用文档。通过将模板定义为交互合约的一部分，可以使用流程收集的数据生成自然语言文档。

我们建议使用支持字段和条件部分的开放文档格式 [40] 来创建模板。

5.4 自定义类型

任何定义物件的 JSON 模型可以当成数据对象引用。自定义类型确实存在工作流无法正常运行的风险，因为其他方可能通过不支持该类型的节点参与。具有未知类型的数据将“按原样”存储，并且在该过程的上下文之外不可用。

6 参与方

参与者定义了交互合约中的个人，团队或组织。参与者始终包含以下信息：

- 身份标识，
- 节点统一资源标志符 URI，
- 自定义信息，
- 签署密钥，
- 加密密钥。

参与方与角色不同。角色是抽象的，例如“学生”；但是，参与方可能是“Bruce Willis”或“Acme Corp”。

签署密钥是具有与身份相关联的一个或多个公钥的映射。“用户”密钥是属于参与方的，所以只能由他/她用于签署任何操作。“系统”密钥由参与方使用的节点拥有，用于签署自动操作，除此之外还可以在流程中定义其他密钥类型。

公钥可用于加密数据，数据只有目的参与者能够解密。

6.1 邀请参与方

若希望添加新的参与方到某个流程的话，此场景该定义添加新参与者的操作。如果公钥在流程内是众所皆知的新参与者可以直接被添加。

当公钥是未知的，新参与者需要被邀请 (figure 4)。这可以通过任何足够安全的方案来完成，包括电子邮件。邀请系统会生成一次性密钥并将其发送到受邀方。被邀请方必须通过其自己的安全“用户”和“系统”密钥替换它。

在新参与方完全参与流程之前，可能需要进行其他身份验证。这可能是通过 SMS 验证到联合身份验证甚至公证批准。

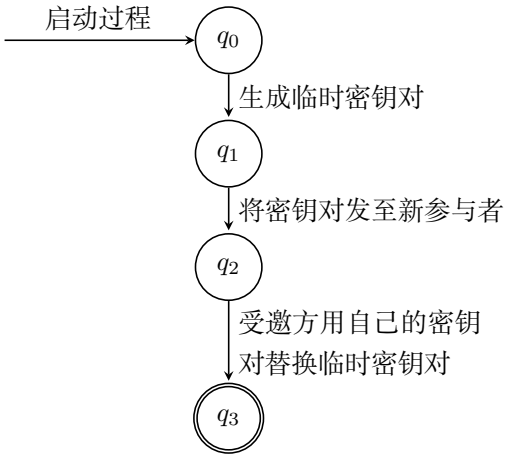


图 4: 邀请新参与方加入过程

6.2 更新参与方

除了身份标识符外，所有参与者可以自由修改自己的信息。这也允许某方切换到另一个节点。如果本过程不允许用户切换节点，则由参与者的节点拒绝该更改。

移除某个参与方可以通过清除签署密钥，加密密钥和节点 URI 来完成。

只有在场景中定义了此类操作并在当前状态下允许时，才能更新其他参与方。

7 过程

如果一个场景是工作流的无状态定义，则该过程是有状态实例化，包括以下内容：

- θ_x 作为回复，则 $f : q_x \mapsto \theta_x$
- Θ 作为所有回复的有序的集合 $\Theta = \{\theta_0, ...\theta_n\}$

- q_t 作为前的状态
- I 作为在线的角色
- A 被创建的资产的集合。

7.1 操作

执行操作总是会产生响应。此响应必须由执行者签署并作为新事件提交。节点基于当前状态和刚执行的操作独立地确定新状态。

该场景被定义为确定性 FSM。然而，这仅涉及状态转换和投影。在诸如以太坊和 Hyperledger 之类的系统上，所有逻辑必须是确定性的，因为它由所有节点执行，并且需要在所有系统上产生相同的结果 [41]。

使用 LTO，只有单个节点或单个角色执行操作，这意味着操作不需要是确定性的。如本节所述 1.4，对于从外部源获取数据等操作，不需要预言机。

7.2 手动操作

在 LTO 平台上构建的应用程序必须告知人类参与者他们在当前状态下可能执行的操作。参与人类在将其提交给他的节点之前将签署他自己的响应事件，该节点将其分发给所有参与方。

7.3 系统操作

系统操作不需要人为干扰，因为节点会自动执行。因此，由节点签署响应操作，而不是人类用户。

这些操作始终由单个系统执行，而不必是确定性的。该流程的其他参与方验证响应，并可在需要时拒绝。由于系统操作中不涉及人为干扰，操作由系统本身而不是参与者签署。

系统操作也可以安排稍后执行的。这可以在场景中指定。它允许状态超时或以预定的频率轮询外部资源。

系统操作会自动执行，如果使用不当可能会产生错误或失败。对于这些操作，必须定义两个状态转换。一个用于成功执行的情况，一个用于失败的情况。

7.4 子进程

基于 FSM 的进程只能同时处于一种状态。子流程允许交互合约保持多个状态，并且可以并行执行不同的过程。这些流程共享一个事件链，但每个流程的数据仍然是孤立的。

为了促进子流程，交互合约可能包含可以从主场景实例化的子场景。

7.5 投影

除 FSM 状态外，该过程还包含其他有状态数据，例如资产和参与者。每个响应的有效负载可用于更新此数据。在场景中定义了有效负载该如何更新数据的规则。更新投影是确定性的，因此，对场景应用一组给定的响应将始终产生相同的投影。

投影可用于设置操作的参数以及为各个操作定义的约束 (见 4.2)。

7.6 数据运算符

在场景中可能会使用数据运算符来指定投影如何影响过程。这些运算符乃确定性函数，没有副作用。它们可用于算术或逻辑运算。这些操作的结果可以存储在投影中，并且可以用作例如状态转换的基础。

7.7 被动测试

包含仅由系统操作组成的循环的场景可能会导致无限循环，从而导致大量交易。在验证场景时，如果某个场景有这样的构造，我们要拒绝它。

确定某个程序是否可以永久运行被称为停机问题 [42]。停机问题在图灵机上是不可判定问题；但是，停机问题在 FSM 是可以解决的 [43]。由于 FSM 具有的转换路径数量有限，它们的循环可以有效的被检测到。

由于不可行的路径的存在，EFSM 模型的被动测试变得复杂，并且是一个开放的研究问题 [20, 44]。为了简单起见，我们忽略所有条件而可以假设通过 FSM 的任何路径都可行。我们承认这可能会导致误报。

8 自适应工作流程

场景将模拟流程最常见的案例。提前预见所有情况是不可能，并且无法对每个可能的边缘情况进行建模。采用代码即法律 (code-is-law) 方法会使系统僵化。相反，交互合约支持三种解决此类问题的方法。

- 批注
- 偏离
- 场景更新

8.1 批注

评论用于与求他参与者交流互动。例如，它们可用于解决冲突或在流程之外进行讨论。使用评论而不是链下沟通方法可确保会话记录在区块链上。它还让我们可以回溯检查在程序中何时发生某些对话。

评论不仅限于文字信息。也可以使用图像或文档来协助交流。批注不是过程的一部分，这意味着评论不会触发状态转换。因此，链上可以讨论进行关于在过程中未预定的主题。

8.2 偏离

任何一方都可以通过定义部分场景来建议偏离主流。此子流必须从现有场景的其中一个状态开始，并以该场景的某个状态结束。偏差流是一次性的，因为当流程返回到现有状态时，它们不再可用。

各方需要对偏离达成一致。请注意，偏离可能会导致只能通过手动解决方法来解决的分支。

偏离可用于解决争议。任何一方都可以建议它对先前事件的正确性提出异议，并对如何纠正这一各事件提出解决方案。

使用偏离的典型情况是进行支付安排。各组织当然都不希望在协议中显出该选项。预定义的子流程允许授予此类安排，同时保密。

8.3 场景更新

时常，正在运行的进程的场景会需要更改；例如，当协议受到更新或添加新法律时。

一方可以通过偏离流为给某过程提供新场景。此流程将状态移出过时的场景并进入新场景。

因为更新场景是事件序列的一部分，它不会破坏解决方案的确定性。

9 事件链

为了确定 FSM 和投影的状态，我们需要按给定的顺序处理响应集。插入或删除事件，更改事件的顺序或修改有效负载可能会导致完全不同的状态。

在中心化方案中，控制方负责数据完整性。各方都依赖这一方，因为它代表了唯一的事实来源。在去中心化的系统中，权力和责任由各方共享。

为了简化此过程，事件链就像一个特殊的私链。每个响应都包含在一个事件中，可以将其视为具有单个操作的区块。这些事件形成一条哈希链，由各方共享。共识算法确保各方就事件顺序达成一致。

9.1 非许可私有链

在一个场景中，人员和组织被定义为参与者。参与者可以自由选择其想要哪个节点参与本过程。

节点参与网络无需任何权限。启动任何程序甚至邀请其他参与者也无需任何权限。数据仅与进程的参与者之间共享。因此，事件链可以描述为非许可私有区块链。

9.2 加密签名

为确保没有人可以假造或伪造他人的事件，每个事件在使用非对称加密提交之前都会签名。签名事件也可作为收据，允许其他方证明该操作已由签名身份执行。

该平台使用 ED25519[45] 签名。这些椭圆曲线数字签名被 NIST[46] 与 ENISA[47] 等机构广泛使用，受到充分支持和认可，并被认为是安全的。椭圆曲线加密允许更快速的单签名验证和不失安全性的签名。它而且还减少了密钥和签名数据的大小。请注意，此方法本身并不提供完全的安全性，因为各方仍然可以假造或伪造自己的事件。换句话说，加密签名无法证明某事件有没有发生。

9.3 哈希链

每个事件都可以使用 SHA-2 256bit 哈希进行唯一标识。该行业标准算法可确保快速的可计算与抵抗原像和第二原像攻击以及碰撞 [48]。这是 NIST 推荐的算法 [49]。

将先前事件的哈希嵌入下一事件的哈希中会创建哈希链，该哈希链记录事件的时间顺序。当与加密签名结合使用时，哈希链提供足够的证据证明特定事件序列导致当前状态 [50]。

10 分配

与其要求各方从中心服务器或彼此提取信息，每一方负责将事件推送到所有其他参与方的系统。

系统需要始终可用，以便事件不会丢失。解耦和消息队列可以减少临时不可用的问题。在典型情况下，所有各方将连接到他们信任的节点，该节点将接收和处理它们的事件。该节点也是较大系统的一部分 (见章节 19.1)。

组织和政府可以自己运行节点。用户可以连接到其组织的节点或他们选择的公共可用节点。

CAP 定理意味着在存在网络分区的情况下，必须在一致性和可用性之间进行选择 [51]。通过使用解耦，我们牺牲了一致性，但是实现了更好的可用性。不在线或无法访问的节点不会中断其他参与者的进程，并且会在再次可用时同步。

10.1 私链

事件链是一个私有链，仅与参与者选择的节点之间共享。节点不会意识到他们未参与的私链。

节点同时存储和促进许多事件链。事件链是完全隔离的，与侧链不同。链不会直接相互影响。鉴于每个事件链的活动相当低，这允许水平缩放。

10.2 创始区块

任何人都可以随意创建一个新的事件链。此链的创始区块包含创建进程的用户的标识，后续块将包含场景。作为场景的一部分，其他参与者将被邀请到此私链。

11 共识机制

LTO 是一个分布式系统，所有方都可以通过自己的节点参与。节点将所有事件分发给其对等体，然后这处理这些事件。这意味着过程中存在一个节点之间的过程状态不同的短暂时刻。最终的一致性 [52] 保证最终，鉴于没有提交新事件，所有节点上的进程状态将是相同的。

但是，有时候，在达到一致性之前会有新事件提交。此时，两个或更多节点可能会将不同事件附加到事件链。在拜占庭故障期间 [53]，所有节点都认为他们的信息是正确的; 但是，整个系统处于不一致状态。在这种状态下，节点不再接受彼此的新事件; 他们需要能够达成共识，而不是停止。

分布式应用程序使用不同类型的共识算法。通常，这是拜占庭容错的情况。早期的拜占庭容错 (BFT) 方法不可地扩展 [54]。随着新的可扩展的共识算法的发明例如 工作量证明 *(PoW)* [55]，权益证明 *(PoS)* [56] 与权威证明 *(PoA)* [57]，使得具有大量参与者的分布式网络可被创建；这也称为分布式账簿技术。

虽然这些共识方法比传统的 BFT 方法更好地扩展，它们还是需要相对大量的参与者（PoW 多于 1000）才安全 [58]。传统的 BFT 方法依赖于不超过 $\lfloor \frac{n-1}{3} \rfloor$ 参与者是不良行为者的事实 [59]。这意味着如果有少于四个参与者，一个不良行为者可以影响系统，并且至少需要七个参与者保护他们免受两个不良行为者的影响。

事件链是一条私有链，参与者相对较少，通常少于 7 个，这意味着这些算法非常容易受到攻击。所以各个节点会认为它们的状态正确，除非另有证明，而不是信任多数投票。

11.1 冲突比率

事件链依赖于乐观性的并发控制; 大量的冲突会对共识算法造成压力，算法可能会相对较慢，因为它可能必须等待生成新的区块。

我们定义分布式事件链如下：

- 由 N 作为对事件链有贡献的实体。$\{n_1, n_2, n_3, \dots\}$。
- 由 C_n 作为事件链，由属于 n 实体的事件组成的序列 $(e_1, e_2, e_3, \dots)$，还由 C 作为事件链的所有副本的集合 $\{C_n | n \in N\}$。
- 由冲突或分支定义为 $\exists\, i, j \in N : i \neq j,\ C_{i_0} = C_{j_0}, C_i \not\subset C_j,\ C_j \not\subset C_i.$

意外冲突只有在两方各自，在从其它链受到更新之前，在自己的链上添加一个区块的情况下才会发生。

若称某人将更新传播到链的比率为 $P(x)$。此比率取决于在给定的时间范围内被添加到链中的区块数量，将该块传播到网络的其余部分所花费的时间，以及在该时间范围内对链做出贡献的实体的数量。假设每个人都对网络做出同等贡献的话，可以推导出以下函数 (1):

$$P(x) = \frac{f \cdot t}{n} \tag{1}$$

以：
$f =$ 交易总数量 / 时间范围
$n =$ 活跃参与者总数
$t =$ 将块传播到网络的其余部分所需的时间

这比率可以用来计算发生冲突的概率。通过从 1 减去不存在冲突的机会来导出该概率。若没有冲突的话，意味着当时没有人为链条做出贡献，其机会是使用

以下函数计算的 (2),

$$(1 - P(x))^n \tag{2}$$

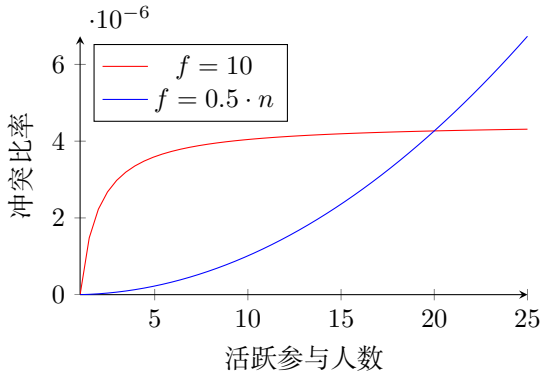
或者只有一个节点对链做出贡献，我们可以使用以下函数来计算其可能性 (3).

$$P(x) \cdot (1 - P(x))^{n-1} \cdot n. \tag{3}$$

因此，计算冲突的可能性得用 (4).

$$P(c) = 1 - (1 - P(x))^n - P(x) \cdot (1 - P(x))^{n-1} \cdot n. \tag{4}$$

若网络延迟为 1200 毫秒，我们发现冲突的可能性为 $< 0.0005\%$:



LTO 每条链上的活跃参与者极少；这减少了冲突的可能性。如果有五个以上活跃参与者，则该数字就此不重要。超过 10 名参与者，基于网络延迟和交易次数，冲突的可能基本上变得线性。

如果使用具有解耦消息队列的单独共享事件链，则交易频率将非常低。这使冲突的可能性边缘化。

11.2 分支验证

一个节点只能在某条链的最后一个事件以及之前的时间证明另一方知道其链的存在。任何方都可以在最后一次提交后进行分支。如果一方试图从最后一个事件之前的一个点开始分支链，那么所有（其他）方都会自动丢弃该分支，并记录该事件。

在接受来自冲突分支的新事件之前，它们将像任何收到的事件一样进行验证。该事件必须由其中一个参与者正确签名，并且必须正确锚定。如果事件的时间戳与锚点的时间戳相差超过 300 秒，则可以拒绝该事件。

11.3 锚定顺序

节点必须在 global 区块链上锚定事件。区块的顺序是通过挖掘设置的，挖掘区块内的事务顺序也是固定的。

这让我们可以使用 global 区块链上的锚点顺序来确定事件的顺序。如果发生冲突，必须接受首先锚定的块。私链的共识是通过公链的共识达成的。在公链上，共识是通过大量参与者之间的匿名协作使用 PoS 的变体的达成。

11.4 优先

某些情况下可能需要给出一个操作或角色优先权，所以即使它最后被锚定，它也是先排序的。参与者可以在场景中自由设置此类优先级。

某些事件类型（例如评论）自然具有较低的优先级。

使用优先级会启用抢先交易攻击，某人可以创建新的分支来回应一个事件，然后废止此事件。优先级当只有在没有问题的情况下才应使用。

11.5 未锚定事件

当收到尚未锚定的区块时，我们可决定无论如何都接受该块。当然，是在接受它就不会产生冲突的情况下。若某区块的锚固仅是被延迟，接受该块可避免给正在运行的过程带来多余的延误。另一方面，如果永远不锚定该块，则不会出现实际问题。如果所有人都接受该块，则该过程可以正常继续，并且该块可以稍后锚定。

11.6 合并分支

当出现分叉时，大多数区块链应用会选择一条链继续并忽略其他分支上发生的一切。使用像比特币这样的区块链，所有交易最终都会包含在每个分支的挖掘区块中。

在事件链上，事件本身构建了哈希链。挑选其中一个分叉会导致丢失有关已执行操作的信息。相反，当一个节点意识到另一个分叉优先于其自己的分叉时，它必须将它本地拥有的事件基于另一条链。这类似于使用 git 时的复位基底（rebase）操作 [60]。

11.7 分叉

即使有担保达成共识的方式，参与方也可能决定忽略其他链并保持分叉。对于大多数区块链应用来说，例如以太坊 [61]，没有理由去干扰分叉，因为价值仅

来自参与主链。

交互合约是一种用于数字化和部分自动化现有流程的工具。即使区块链允许分叉的存在，这些过程通常不会。在分叉的情况下，各方可以启动一个辅助过程以尝试手动解决冲突。

12 隐私

LTO 专为多方之间的运行流程而构建。除了这些参与方之外，没有人需要了解这个过程甚至是合作。

公有区块链容纳匿名帐户; 但是，这些帐户充当假名作用。任何交易都可能揭示帐户的身份，从而暴露完整的交易历史记录。智能合约要求数据是公开的，因为它需要可用于每个节点。在联盟链上，参与者彼此了解。

临时的私有链允许随机分类的参与者进行协作，而无需批准或公开信息。完成该过程后，可以完全清除这些链。

12.1 链结资料

每一方通过自己的节点或其信任的节点进行连接。每个节点都有一个私有存储服务，用户可以在其中存储数据。用户可以完全控制存储在此的数据，类似于存储在 DropBox 等服务上的数据。他们可以随时删除自己的个人数据。在未经有效协议和用户明确批准的情况下，数据不会被共享或处理。

当某个操作导致链结资料时，该数据不会直接与其他方共享; 只有一个哈希值会被添加到链中。LTO 通过将哈希与时间戳和一些随机数据一起放入信封中来防止哈希被用于验证接收数据之外的任何其他内容。为了形成信封而创建的哈希将永远不会在区块链上出现多次。

当某组织表明它想要执行需要链结资料的操作时，该组织的节点将自动发出请求。数据所有者的节点检查指定的操作是否有效，并且可能由当前状态的角色执行。

12.2 GDPR 通用数据保护条例

随着新的 GDPR 在欧洲的到来 [62] (通用数据保护条例) ，许多关于区块链应用程序不符合 GDPR 的事实已经引发了争论 [63]。造成这种情况的两个主要原因如下：

- 区块链的不变性与修改和删除数据的权利相冲突，

- 没有专用数据控制方的事实，因为区块链是一个分布式环境。

链结资料意味着参与方选择的节点将充当该用户的数据控制器。所有其他方始终充当数据处理器。数据请求自动格式化以创建适当的数据处理协议，其中包括处理数据所需的目的和时间 [64]。

临时区块链可以在需要时清除整个链。

简而言之，LegalThings 的隐私功能使得解决方案无需额外的努力符合 GDPR 政策。

12.3 零知识证明

某个场景可能要求一方向另一方证明其知道某个特定值。零知识证明（zk-proof）是一种方法，除了证明一方知道该值之外，不传达任何其他信息。

LTO 通过交互式证明系统支持 zk 证明。证明方与验证方两方交互目的为，证明方对验证方证实其诚实度（完整性）还有让验证方揭露证明方不诚实的证据（健全性）[65]。

交互合约的 zk 证明始终是两方之间的操作。合约中不需要非交互式的 zk 证明，例如 zkSNARKS，它仍然被认为是实验性的。

13 常用模式

13.1 链互动

某些流程可能必须与其他流程交互才能继续运行；例如，某各进程需要来自另一个进程的允许才能继续运行检索冲突解决过程的结果。从另一个进程请求数据是通过遵循 5中所示的模式完成的。

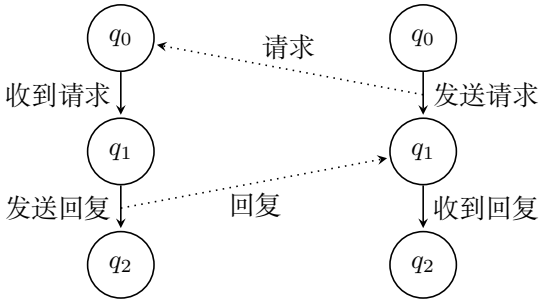


图 5: 两个进程交互时遵循模式。

13.2 显式同步

在某些情况下，如果事件传播得太晚，则会特别麻烦。当这种情况发生时，我们可以在场景中构建显式同步。这需要所有各方在继续该过程之前确认当前状态。这是 FSM 中的一种模式，旨在防止各方在此事件之前分叉事件链。如果发生争议或冲突，可以使用此类确认方式来决定应该用于继续该过程的分支。

这种显式同步仅在所有方都确认当前状态时才有效。如果某方缺乏继续的动力或有分叉流程的动力，则需要另一种解决方案。在这种情况下，想要继续该过程的一方向所有其他方宣布其意图。如果收到此公告的任何一方注意到所宣布的操作在自己的链上无效，他们可以传播此信息。这意味着链已经分叉并且必须应用常规冲突解决方案。为了防止此公告影响网络延迟会在假定没有人拒绝宣布的事件之前，等待一定时间。

第二部分　全局区块链

LTO 全局区块链是一个非许可的公有区块链，专门用于验证信息。存在目的是为了支持交互合约和私有的事件链。全局区块链具有可跨事件链，区块链和应用程序互操作的锚定和数字身份.

公证交易几乎可以在任何区块链上进行。然而，在针对金融交易或一般逻辑优化的区块链上，公证类型的交易是昂贵且低效的 [66]。此外，优化（例如剪枝和分片）可能会产生负面影响，因为会忽略相关信息 [67, 68]。

全局区块链属于 Nxt 系列 [69]。该区块链的一个独特特征是交易基于一系列核心交易类型，这些交易类型不需要网络节点上的任何脚本处理或交易输入/输出处理。这减少了区块链的大小，提高了效率，并允许对公证交易特别有利的聚合方法。

与其直接从 Nxt 开始，我们使用了 WAVES 平台 [70] 的分支作为基础。该平台已经落实了许多改进，例如 NG 协议 (节 15.6)，我们的网络将从中受益。专注于数字资产（包括彩色币）的现有交易类型将被删除或禁用，并由公证交易类型取代其位。

14　中心化与去中心化锚定

其他解决方案使用中心化的方法，一段时间其中所有哈希值被收集。从其交易创建一个梅克尔树，然后如一个单独交易添加到像比特币第三方区块链。因此，系统没有直接反馈，而且可能需要几个小时 [71] 才能从中心服务收集最终

收据。

LTO 全局区块链是一种去中心化的方案，在其每个节点都监督所有事务。当广播锚定交易时，它立即可见，并在大约 3 秒后使用 NG 预先批准 (节 15.6)。交易一分钟前就会在一个区块内。

节点可以跟踪所有锚定的哈希或仅跟踪它们自己的哈希值。如果需要，他们可以独立创建收据 (节 16.1)。不需要任何中心化服务。

15 共识算法

全局区块链作为典型的公有区块链运行。某个节点会被选择作为生产者以验证事务并伪造区块。为了测定生产者，我们使用 **Leased Proof of Importance**（LPoI）共识算法。生产者将获得伪造区块交易的费用。

重要性证明是权益证明（PoS）的变体，其中被选择伪造区块的机会是基于持有的代币数量和财产来确定的。在重要性证明，机会会根据节点的网络使用情况而增加 [72, 73]。

在 “LTO 代币经济” 论文中详细描述了从该共识算法中产生的代币经济 [74]。

15.1 租赁

NXT, Waves, 以及其他该系列的区块链都使用权益证明租赁 [69, 75]。通过租用代币，代币持有者将锻造区块的权限传给所选节点。这些节点可以像挖掘池一样运作，在租赁者之间按比例分享奖励。

NXT 样式网络在当某方控制至少 1/3 的所有活跃余额时，容易受到攻击 [76]。这是一个问题，已实施 LPoS 算法的项目往往具有高度中心化。前二名 Nxt 节点控制着 50% 以上的网络 [77].。在 Waves，前两名节点控制着网络的 1/3 而前五名节点控制 50%[78] 以上。

在 18节中，我们将讨论大量去中心化对该网络的重要性。为了防止拥有大量租赁股权的节点，租赁代币存在限制。每个节点都必需要拥有至少 10% 持有的代币。POI 也抵消了这种影响，因为它使被动的节点处于劣势状态。

15.2 抽奖因素

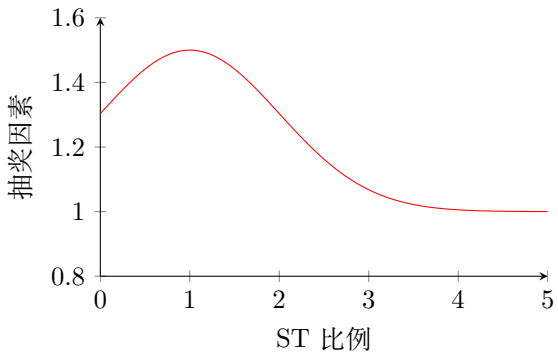
用于计算节点在网络中的使用，我们将使用影响造块比率的 ST 比例，此比例由锁仓代币总额对节点助于的交易算出。

$$\text{ST 比率} = \frac{\text{锁仓代币\% 总量}}{\text{助于的交易\% 总量}}$$

S /T 比率与“抽奖因素”连在一起。抽奖因素是一个数学函数，它影响选择节点生成新块的机会。ST 比率越平衡（🄵接近 1.0），筹建因素越高。如果 ST 比率不平衡（节点不提供任何交易），则抽奖因素将为 1.0。

$$\text{抽奖因素 } r = 1 + (0.5 \cdot e^{-0.5 \cdot (\text{ST 比例} - 1)^2})$$

该公式导致钟形曲线的标准偏差较大。

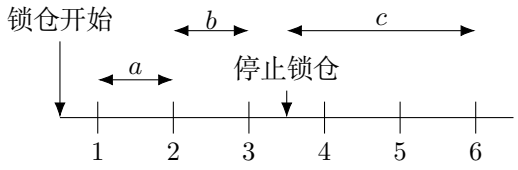


最高的抽奖系数是 1.5，这是最低与最高 2.0 之间的一半。通过锁额外代币 X 来增加自己的重要性的话，则需要花费 $2 \cdot X$ 倍的交易费用。重要性虚假增值在 18.1章节中详细解释。

要完全理解抽奖因素的概念及其对代币经济的影响，请阅读“*LTO* 代币经济”论文 *[74]*.

15.3 锻造概率

被选中锻造（forge）的机会是 $P(forge) = S \cdot r$。贡献的交易 T 是随时间计算的。在计算 $P(forge)$ 时，S 必须在同一时间段内保持不变，以防止滥用的可能性。



$a =$ 计算 P(forge) 的时刻

$b =$ 锻造新区块的时刻

$c =$ 代币仍被锁定的时刻

图 6: 锁仓代币和锻造块的时间线。时间以汇总块的数量来衡量。

15.4 公平的 PoS

决定哪个节点有资格锻造新块的函数基于 Waves 创建的 Fair Proof of Stake 算法 [79]。这是对原始 Nxt PoS 算法的改进，该算法偏袒高估大量锁仓的节点。

有关基础算法的进一步理解，请阅读*"Fair Proof of Stake"* 论文 *[79].*

要将此算法从 PoS 转换为 PoI，该函数将抽奖因素应用于锁仓余额，从而产生为 $b_i \cdot r$.

- T_i 作为第 i 到 th 个帐户的区块生成时间，
- X_n 生成签署，
- r 抽奖因素，
- b_i 节点锁仓与锁仓总额的百分比，
- Λ_n 为基准目标
- $T_{min} = 5$ 给区块之间的延迟的常数，
- $C_1 = 70$, 用来延迟分配的形状的常数，
- $C_2 = 5E17$ 调整基准目标的常数。

$$T_i = T_{min} + C_1 \cdot log(1 - C_2 \cdot \frac{log(X_n/X_{max})}{r \cdot b_i \cdot \Lambda_n})$$

如果在超过分配的时间之前收到新块，则必须将此块添加到链中，并且必须计算新的延迟。之前的 T_i 从此不重要。每个节点都会自己算 T_i ； 此信息不是由生成者提供的。这意味着没有必要在计算中使用不正确的锁仓额。

15.5 生成者签署

在确定时间延迟 T_i 时，是不会考虑区块哈希的。该哈希基于区块的内容，而该内容由锻造区块的节点定。如果这个哈希在决定谁可以锻造下一个哈希中起任何作用，那就很容易操纵 T_i。某个节点可以创建多个不同的区块，并且仅广播具有最低时间的区块。

为防止这种情况，我们仅使用生成签名。此签名是一条辅助哈希链，仅使用上一代签署和生成者的公钥。在 Nxt，这是完全确定性的，使其易于被利用 [76]。

为了解决这个问题，以及减少分叉的机会，Fair PoS 使用 100 个区块前使用的生成签署。节点或租约中的余额发生任何变化会导致 T_i 对于给定的 X_n 的改变。尽管如此，控制至少 1/3 代币的任何组都有 30% 的优势。

PoI 需要锁仓余额和抽奖因素在固定的区块数量内保持不变。这会使它更容易受到这种攻击。

作为解决方案，生成不是可以公开计算的哈希。节点必须对上一代生成签名进行哈希处理，并使用其私钥对该哈希进行签名。这可以作为生成签名。与 Nxt 和 Waves 相反，一个节点只能为自己计算 T_i，而无法提前确定下面的生成者。

15.6 NG（next generation - 新一代）协议

NG 协议最先为了减少比特币的扩展问题而提出。虽然它从未在比特币上实施，但自 2017 年 12 月以来，Waves NG 一直在 Waves 主网上运行。

在 NG，由两种类型的区块，微块和密钥块。先前被选的节点可以继续验证交易，平均每 3 秒创建一个微块。当选择新节点时，它会从未完成的微块创建一个密钥块。微块中的交易可以在某种程度上被认为是安全的，对于诸如锚定之类的低风险交易。

交易费用的奖励在锻造微块的节点和锻造密钥块的节点之间分配为 40% - 60%。此分配必须始终有利于密钥块锻造者。否则，就会有动力忽视已经锻造的微块并创建新的微块。

在公开压力测试中，Waves NG 证明能够处理高达 6000 交易/分钟，峰值为 17,000 交易/分钟 [80]. NG 协议极有可能可以处理高达 1000 交易/秒或者 60,000 交易/分钟。

NG 减少了实际延迟并且是其他优化的关键组件。

16 交易类型

LTO 全局区块链使用预定义的交易类型。这允许更紧凑的块并且不需要脚本。如果需要的话，将来可以扩展交易类型列表。目前可以进行的交易类型已列出如下：

- 锚定：用于验证来自私有链的交易，
- 颁发证书：用于声明参与者之间的关系，
- 扩展/撤销证书：用来扩展或删除某种关系，
- 代币转账：用于将代币发送到另一位参与者，
- 代币锁仓：让参与者锁仓或租赁代币，
- 取消锁仓：用于停止锁仓或租赁代币，
- 设置脚本事务：用于配置智能帐户。

16.1 锚定

锚定是获取某本文档或其他数据的哈希值并将其存储在一条区块链上的交易。目标是让任何人，包括创建者在内，都无法对其文档进行回溯或把日期填迟[81]。

私有事件链的每个事件都锚定在全局区块链上。第三方应用程序可以使用全局链来锚定文档以进行存在证明。我们估计 99% 全局链上的交易可能是锚定事件。考虑到大多数交易都是用于锚定，聚合这些事务会减少区块链所需的磁盘空间。

在锻造区块时，节点按照列表中显示的顺序从事务创建二叉哈希树 [82]。只有哈希树根被添加到区块链中。作为验证过程的一部分，每个节点都会重新创建此哈希树。

节点能够索引每个锚点哈希值。但是，为了减小磁盘大小，大多数节点应该选择提取自己的锚事务的 Merkle 路径。该路径形成可与原始数据一起存储的收据（就如事件）。

16.2 验证和授权

挑战/回应认证方法（例如用户名和密码验证）需要中心化系统操控。在完全去中心化的系统中，我们依靠加密签名来提供身份验证。虽然事件链之间不共享信息，但鉴于用于签名的密钥对，仍可以跨链识别各方参与者。

从更广泛的意义上讲，各方可以通过这种方式签署任何类型的信息。这是PKI 证书现在提供的类似用例。依赖中心机关颁发和撤销证书阻碍了将其作为挑战/响应认证的替代。

使用公有区块链，可以创建和使用公钥/私钥对，而无需中心权限。密钥对形成独一的身份标识，可以通过从公钥的散列派生成的地址来引用该标识。

16.3 证书

证书事务允许每个参与者通过引用其地址来传达有关另一个参与者的信息。与代币不同，授予和撤销帐户完全在发行人的掌控之中。

证书可以被指定特定类型，该类型由发行证书的一方选择。虽然不是必需的，但是我们还是建议在显示给其他人之前由收件人帐户确认证书。

16.4 信任之链

虽然具有私钥对的公共地址是一种身份验证方法，它并不提供授权解决方案。证书可用于指定参与者之间的关系。

这种途径类似于信任网（WoT）。WoT 与 PKI 有许多缺点，但是我们平台上没有这些缺点。

在区块链上，建立和撤销关系或将参与者标记为受损是非常简单，即时和不可撤销的。区块链交易具有时间戳，允许我们在某个时间点验证关系的存在。

我们不是简单地建立身份，而是建立特定的关系。除了存在关系外，交易不会确认或否认有关参与者的任何其他信息，因此各方无需与另一方进行实际会面。

在给定的情况中，我们只关心基于这种关系在两个参与者之间找到信任链。这模仿了 PKI 验证所做的信任链，但没有中心机构。Rather than an absolute root certificate, the blockchain address of either our organization or an organization we do business with functions as trust root.

16.5 智能帐户

默认情况下，某帐户的任何交易将有其帐户的密钥对签署。Waves 在其平台实施了智能帐户的概念，允许任何人自定义此逻辑 [83]。

为此，可以使用非图灵完整语言来编写此逻辑。此脚本仅用于验证或拒绝特定帐户的事务。它不能触发其他交易。因此，这种智能合约不会阻止聚合操作。以确保这一点 LTO 智能帐户而且进一步被限制。

LTO 没有数据交易，并且无法从脚本访问其他事务。

通过指定需要用于签署事务的备用公钥，可以使用智能帐户创建多签名帐户。它不是直接指定这些密钥，而是指定任何拥有特定证书的人都可以签署一笔交易。

用户也可能还会考虑其他一些限制。帐户可以被锁定，只被允许在多个块之后转移代币，要求在帐户上保留最少数量的代币，或者只允许将代币转移到特定帐户。

当运行节点时，我们建议使用多重签名。与节点相关联的帐户通常包含大量代币以便锁仓和锚定。该帐户的私钥对于节点是已知的。使用多重签名的话，获取该键不会给这些代币的直接访问。使用多重签名，获取该密钥不会让用户直接访问代币。

锚定交易不受智能帐户的影响。他们总是需要使用帐户的私钥进行签名。该逻辑适用于允许未来可能的优化，例如全局区块链节点的水平缩放。

17 汇总区块

锚定是一种低影响，无中断，可为区块链带来额外的安全性的方案。我们预计其他不使用交互合约的应用程序也会使用锚定功能。

一个让区块链难以扩张的反面就是每条链会无限的继续增长 [84]。链的大小给保留副本的节点需要的硬件产生压力。它还给必须回放整个链的新节点带来了负担。为了降低链的增长速度，我们使用了汇总区块。

17.1 密钥块大小

表格 1显示区块链上的密钥块的结构。全球链应该可以扩展到 5000 万/日以上笔交易。这大约是预期使用量的五倍。这种密钥块的大小由区块数据和交易事务数据定夺 (5)。

$$Keyblock\ size = d + t \tag{5}$$

有:
$d =$ 区块数据
$t =$ 交易事件数据

在计算预期的块大小之前，我们会假设:

- 99.98% 的交易是锚定交易 (Table 5)。这是全球区块链的主要用途，
- 其余的 0.02% 是证书交易 (Table 7)，

- 其他类型交易都很少发生，可以忽略不计，
- 所有交易均匀分布在区块上，
- 平均而言，每分钟生成一个关键块，
- 每一天创建 1440 个关键块。

区块数据大小是 277 bytes (Table 1)。在先前做出的假设下，可以使用等方程算交易数据的大小 (6).

$$Transaction\ data\ size = n \cdot (0.9998 \cdot a + 0.0002 \cdot c) \tag{6}$$

有
$a =$ 锚定交易的大小 (Table 5),
$c =$ 发行证书交易的大小 (table 7),
$n =$ 每个区块的交易数量。

这使得密钥块的总大小约为 3.8MB.

17.2 无聚合的增长

有每个区块大小 3.8MB 还有每一天会生成的 1440 个区块，如果它持续满负荷运行，本区块链每天将增加 5.47GB 数据 / 每年 2TB。

预期用量大约一千万笔交易，每一天给区块链增加 1.1GB 数据。这导致每年约 36.5 亿笔交易，或者换句话来说每年增加 400GB 数据。

对于共有 3.40 交易的比特币 [85] 来讲，看网络和硬件速度，从创世起同步大约需要 7 天。

在拥有数十亿笔交易的情况下，硬碰硬同步可能意味着等待数周甚至数月才能使全球区块链同步。

聚合交易的目标之一是每年只需要 20 分钟的同步，当然再次取决于网络和硬件速度。一年 365 个汇总块，节点应该能够在 3 秒内处理每一个汇总块。

17.3 隔离见证

隔离见证是比特币采用的策略通过将交易分成需要处理的数据和用于验证交易的数据来减少每个区块中的数据 [86]，这种称为见证数据，其中包含签名。

确定性是一种保证，足够深的区块永远不会从区块链中移除。无论概率终结性或协议终结性如何，如果能不恢复该块，则见证数据再无用处。节点可以自由删除达到确定性的区块的见证数据，从而节省磁盘空间。

我们建立在隔离见证的逻辑基础上，产生了汇总块的概念。

17.4 聚合

这些是每 1440 个块（大约每天一次）创建的特殊区块。它们包含自上一个汇总块以来所有新区块的聚合值。重放链时，只需要应用汇总块即可接近当前状态。然后，只需要重放在最后一个汇总块之后创建的块。这显然减少了重放时间。

倒数第二个汇总块和之前的所有块都是最终块。节点在这个点之前不会考虑接受任何分叉，无论最长链状况如何。这意味着只需要存储先前 1440 到 2880 密钥块的交易。在将密钥块存储在汇总块中之后从密钥块中删除事务数据会将密钥块大小从 11.3MB 减少到 277 bytes，与汇总块相比，它们可以忽略不计。

17.5 与剪枝的区别

乍一看，这种方案似乎与区块链剪枝类似，因为我们只保留了有限的一组交易。剪枝的危险在于当有伪造的信息被引入时，它会威胁到不可变的性质。

危险来自分配区块链的状态。通过隔离见证，交易在没有签名验证的情况下通过，依赖于终结性的概念。但是，每个节点仍然需要应用来自创始的所有交易来计算当前状态。

实际的交易数据不用于计算块的签名，而是存储为下一个块的附件 (Table 2)。作为没有交易的事件，密钥块是区块链的一部分，不能被忽略。如果在未经验证的情况下通过交易，则聚合它们几乎没有风险。

17.6 汇总块大小

汇总块包含有关非锚定交易的所有信息以及所有交易费用和其他代币转账的聚合版本。这种区块相当大的块，特别是与密钥块相比时。

为了减少被使用的内存量，交易费用和代币转账交易会被缩小到每个参与者的余额变化 (Table 4)。汇总还包含不可聚合的交易，如证书，锁仓和脚本交易。

若要预算汇总块的大概大小，我们进行了以下假设：

- 总共有 20 万人参加，
- 每天会创建一个汇总块，
- 根据前一节中的假设，我们可以认定余额变化是汇总快的唯一重要部分。

在使用这些假设的情况下，我们可以计算汇总块的大小。当使用方程 (7)号来计算大小时，结果约为 10.3MB。

$$Summary\ block\ size = Transaction\ summary = p \cdot e \tag{7}$$

$p =$ 过去 1500 个区块的参与者数量，
$e =$ 余额更改总结条目的大小 (table 4).

17.7 总大小

区块链的总大小由两部分组成，静态部分和成长部分。静态部分由最后 1000 个密钥块组成。那些块将依然包含附件数据(5)。

$$Static\ part = n \cdot k \tag{8}$$

$n =$ 有交易数据存储在内的密钥块数量，
$k =$ 密钥块大小 (5)。

当我们使用方程 8 用于计算概率的大小, 它表明总的大小是 11.3GB 左右。由于只有最后 1000 个区块被完整存储，包括交易，因此这个大小可能略有不同，但不会明显增加。

成长部分包括汇总块 (7) 以及删除交易数据后密钥块的剩余部分。我们将使用方程 (9)将大小定义为每年的增长率。

$$Growing\ part = n \cdot k + m \cdot s \tag{9}$$

with:
$n =$ 每年生成的密钥块数量，
$m =$ 每年汇总块的数量，
$k =$ 没有交易数据的密钥块的大小，
$s =$ 汇总块的大小 (7).

按照先前做出的假设，该方程表明区块链每年增长约 1 GB。3.7GB。

17.8 历史节点

节点不会被要求删除旧的交易。通过保持链中所有交易，历史节点可以在需要时证明区块链的正确性。历史节点无法传递伪造的历史记录，因为历史节点的区块需要符合其他节点的。网络可以依赖于相对少量的历史节点。

运行一个历史节点没有在链上的好处。它不会增加锻造新块的机会。运行这种节点必须出自社区的兴趣或者二次收入。

18 网络漏洞

18.1 重要性虚假增值

对于重要性证明 POI 一个忧虑就是某人通过虚拟交易来提升自己的重要性的。我们可以将垃圾交易的利润/损失计算为最大抽奖因素的公式，如下所示：

- 抽奖因素; r,
- staked tokens 比例; b_i,
- 交易成本; c,
- 网络上的总交易量; n,
- 垃圾交易; τ,
- 奖励; p,
- 垃圾交易的利润/损失; $\Delta p = p_{r_{max}} - p_{r=1}$.

$$p = (r \cdot b_i \cdot n \cdot c) - (\tau \cdot c) \tag{10}$$

$$r = 1, \tau = 0 \rightarrow p = b_i \cdot n \cdot c \tag{11}$$

$$r = r_{max}, \tau = b_i \cdot n \rightarrow (r_{max} - 1) \cdot b_i \cdot n \cdot c \tag{12}$$

这给出

$$\Delta p = ((r_{max} - 2) \cdot b_i \cdot n \cdot c) \tag{13}$$

鉴于

$$b_i > 0, n > 0, c > 0, \Delta p < 0 \rightarrow (r_{max} - 2) < 0 \tag{14}$$

$$r_{max} < 2 \tag{15}$$

方程 15 证明了是不可能直接从垃圾交易，由最大抽奖系数等于或小于 2，得到利润的。

接近 2 的抽奖因素将使垃圾交易几乎免费。通过低成本增加重要性乃下策，因为它可以帮助攻击者用 51% 攻击试图破坏网络。1.5 的最大抽奖系数确保了夸大重要性的高成本。

18.2 无利益攻击

无利益攻击原理假设节点将继续在任何分支上构建，而不是选择最长的链，因为这样做没有任何风险 [87]。如果所有节点都示出这种不良行为，攻击者只需要一小部分代币就可以强制网络切换到另一个链；就是 1% 攻击。

这种情况被称为公地悲剧 [88]。各方都试图通过滥用该系统来获得个人利益。但是，如果每个人都这样做，将没有人从中受益。相反，它只会导致破坏网络，导致底层代币的价值下降。https://v2.overleaf.com/project/5b95e00fe0915f6ac7833767 Waves Fair PoS 使未发布的孤立分支非常难以赶上主支 [79]。利用只占一小部分代币的坏行为者从这种行为中获利的可能性微不足道。

恶意参加者需要创建和保持节点的更改版本。成本代价加上从中获益的机会很少，应该足以抑制这种行为 [89]。

18.3 LPoS 中心化

已实施 LPoS 算法的项目往往具有高度中心化。

这种效果可以通过代币奖励来解释。具有接近 100％正常运行时间的专业设置不会错过任何锻造机会，在设定数量的代币被锁仓时产生更多奖励。这会使代币持有者租赁到这些节点并具有增强效果，因为减少的开销允许向出租人提供更高的支出。

限制每个节点可持有的的代币数量是一种有缺陷的方法，为女巫攻击创造了机会 [90]。在无权限信誉系统中，一个节点可以将自己作为多个假名身份进行，从而避免此限制。

由于我们的解决方案的性质，大多数交易将由与节点有关联的密钥对签名。网络还将包含相对大量的节点。这对 PoS 没有影响; 但是，在 PoI 上，它为平台用户提供了优于《非用户-代币持有者》的优势。

另一项措施是利用租赁代币的限制; 每个节点都需要拥有至少 10% 的代币所有权。

18.4 阻断服务器攻击

由于可扩展性有限，使用大量交易过载任何公有区块链都相当容易。交易费用是抵御这类攻击的主要防御手段，但交易仍需要进行验证，以确定资金不足。此外，凭借相当数量的资金，可以通过垃圾交易使网络超载，如同以太坊于 2018 年 7 月所见。

节点可以选择在这种情况下自动增加交易费用。理想情况下，节点从锁仓中获得的收益与在交易上花费的数量相同。随着垃圾代币增加交易费用的回报，

这些费用将用于自动抵御攻击。

18.5 SHA-2 漏洞

在 2017 年，当谷歌研究实验室发现两个不同文档导致相同哈希的冲突时，SHA-1 被证明是脆弱的 [91]。如果 SHA-2 256bit 同样容易受到攻击，那么对于基于二叉树的锚定来说，这可能是致命的。

如果发现冲突，某人可以声称冲突文件已经过公证。更糟糕的是，给定二进制哈希树根和随机哈希，某人可能能够生成有效的二叉路径。这将允许黑客验证任何文档。但是，这仍然不是一件容易的事，因为对于树中的每个分支，黑客需要组合两个长度正好为 32 个字节的哈希。

虽然使用暴力算法去破 SHA-1 仍然需要在一辈子难以实现的计算，生日悖论导致发现冲突所需的计算要少得多。生日悖论也适用于 LTO 公共链，因为有许多二进制哈希树根，每个都有最大数量的二叉路径。

为了解决这个问题，在验证有争议的情况下，验证可能会基于历史节点。然而，这减少了锚定节点的总体使用。

反而，我们可以添加辅助哈希树，其中 SHA-2 哈希再用另一算法（例如 SHA-3 或 Blake2）进行哈希处理。当可能伪造哈希树时，简单地双重哈希是没有用的，因为攻击者只需算法中的漏洞。但是，如果有两个哈希树的话，则需要破解两种算法。然而，即使这样，也需要为单个块的两个哈希树找到冲突，从而消除了生日悖论的优势。

第三部分　平台

19　架构

19.1 微体系架构

LTO 节点是使用微服务架构模式开发的。这意味着节点的所有功能都被分成微服务，每个服务只负责整个节点的一小部分。这种模式有几个优点，已列出如下：

- 失事故隔离：如果服务失败，则不一定会干扰其他服务。
- 可扩展性：节点内的所有服务都是分离的，因此，它们可以在不同的机器上运行。这使它非常适合水平缩放。缩放是自动化的，如第 24节中的说明中所述。

- 灵活性：用某些编程语言能够提升某些功能。每种服务都可以用不同的编程语言开发。
- 代码质量：通过将节点拆分为小型且定义明确的模块，开发人员可以更轻松地阅读和查看。这样可以提高代码质量。

微服务被聚集在一个 docker 箱器。所有这些容器都使用 Kubernetes 容器编排平台运行；本过程将在第 24节中描述。每个微服务设计都是独立运行。这意味着它没有共享依赖项，因此每个容器都有自己的数据库或事件队列。微服务还设计为无状态运行，以便它们易于扩展。

19.2 应用层与服务

就如在第 19.1节中描述，节点会被分成多数服务。这些服务分为四个不同的层。一个节点由以下层组成：

- 用户界面层：这包括与应用程序层交互的 UI 应用程序。
- 应用层：这包含处理由事件链中的事件触发的操作的所有服务，
- 私连层：这负责节点的解耦，
- 公链层：这管理公有连的服务。我们的全球公有区块链针对存储哈希进行了优化。每个节点都对所有哈希值进行索引，以便可以轻松验证它们。

20 用户界面层

UI 层包含两个前端应用程序，使用户可以轻松地开发和调试他们的交互合约。首先是链查看器，它允许用户连接到特定节点并列出所有链。用户只能列出和查看他所属的链。第二个应用程序是 playground 应用程序，它允许用户开发交互合约中的场景。它在状态图中可视化场景，并包含其他可视化和验证工具。

21 应用层

21.1 网络服务器

Web 服务器应用程序充当前端和节点内应用程序之间的代理。Web 服务器有两个作用，如下所示：

- 验证对服务的所有请求

- 将所有请求代理到正确的服务

21.2 工作流程引擎

交互合约的实际创建和执行由工作流服务完成。如果事件链服务收到的事件包含交互合约中的操作,则它将被发送到工作流服务。然后,工作流服务将执行将导致工作流中的状态转换和工作流的新投影的动作。此投影存储在 MongoDB 数据库中。

22 私链层

私有链层将节点解耦。即使在连接不良或负载较高的情况下,解耦也可确保稳定的系统。消息队列是私链的通信层。提供消息队列的技术是 RabbitMQ，这是一个轻量级消息代理，非常适合在节点内以及至其他节点上传递消息。RabbitMQ 有个叫作 Shovel 的功能，该功能能够建立与另一个 RabbitMQ 代理的连接并交换消息。此机制用于将事件从一个节点传输到另一个节点。

三个服务管理所有入站和出站事件。

22.1 事件链服务

管理私有链的服务是事件链服务。此服务处理所有传入事件。在处理这些事件时会遵循以下步骤:

- 验证传入事件：通过检查传入事件是否正确签名以及链是否已损坏来。
- 验证区块链是否与本地存储的链匹配。如果不是，需要需要尽早执行冲突解决。
- 如果事件是由属于该节点的参与者发送的，该节点会执行所接收的事件。否则，它只会将它们存储在数据库中。
- 如果从不同节点添加新参与者，则整个链将转到此节点
- 所有新事件都转发到相关节点。

对于存储，事件链服务使用 MongoDB 数据库。

22.2 事件队列服务

事件入队服务就是将事件放在事件队列上的小任务。它既为节点内的服务（例如工作流服务和事件链服务）以及外部用户执行此操作。

22.3 事件派遣服务

事件队列中的所有消息都是由事件派遣服务处理。它获取所有消息并将其分发给事件服务。如果事件服务处理该消息，它将被标记为已处理; 否则，它将被移动到死信队列。

23 公链层

23.1 锚定服务

锚定服务服务是公共链的核心。锚服务将是使用具有 NG 协议的 NXT 平台的分叉。锚服务将被扩展，以便它可以处理“正常”事务和数据事务。这些数据交易将用于存储来自私有链的事件的哈希值，如第 16.1节中所述。这些哈希值将每天收集并确定性地合并到二叉哈希树中。这样，可以从存储中删除数据交易以减少存储空间，但人们仍然可以验证某哈希是否存在。

为了能够验证是否存储了某个哈希，所有哈希将对进行索引。这使验证更快，因为不需要搜索所有数据交易。

24 容器编排

由于节点包括多个微服务，因此需要容器编排平台来管理容器的运行。容器编排平台负责一些任务，例如配置主机，实例化容器，重新启动失败的容器以及通过添加或删除容器来扩展群集。此任务可用于不同的容器编排工具完成，例如 Docker Swarm，Mesos，Nomad 或 Kubernetes。起初，Kubernetes 将包含一个配置文件。每个服务都将配置为使用自己的负载均衡器在自己的 Pod 中运行。这样做是为了使各个服务可以彼此独立地扩展。Pod 水平自动伸缩用于管理服务的扩展 [92]。

References

[1] Hannah Ritchie Max Roser. *Technological Progress*. `https://www.ft.com/content/cb56d86c-88d6-11e7-afd2-74b8ecd34d3b`. Accessed: 03-06-2018. 2017.

[2] Christine Legner and Kristin Wende. "The challenges of inter-organizational business process design –a research agenda". In: (2007).

[3] Benjamin E. Hermalin and Michael L. Katz. "Moral Hazard and Verifiability: The Effects of Renegotiation in Agency". In: (1990).

[4] Audun Jøsang. "The right type of trust for distributed systems". In: *Proceedings of the 1996 workshop on New security paradigms*. ACM. 1996, pp. 119–131.

[5] Israel Z Ben-Shaul and Gail E Kaiser. "A paradigm for decentralized process modeling and its realization in the oz environment". In: *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press. 1994, pp. 179–188.

[6] Ray Fisman and Roberta Gatti. "Bargaining for bribes: The role of institutions". In: *International handbook on the economics of corruption* (2006), pp. 127–139.

[7] Jörg Becker, Michael Rosemann, and Christoph Von Uthmann. "Guidelines of business process modeling". In: *Business Process Management*. Springer, 2000, pp. 30–49.

[8] Manfred Reichert, Thomas Bauer, and Peter Dadam. "Enterprise-wide and cross-enterprise workflow management: Challenges and research issues for adaptive workflows". In: (1999).

[9] Vitalik Buterin. "Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform". In: (2013).

[10] Vitalik Buterin and Karthik Gollapudi. *A Next-Generation Smart Contract and Decentralized Application Platform*. `https://github.com/ethereum/wiki/wiki/White-Paper/f18902f4e7fb21dc92b37e8a0963eec4b3f4793a`. Accessed: 22-05-2018.

[11] Ian Grigg. "The Ricardian Contract". In: (2004).

[12] Nick Szabo. “Formalizing and Securing Relationships on Public Networks”. In: (1997).

[13] Telser. “A Theory of Self-Enforcing Agreements”. In: (1980).

[14] David Joulfaian Douglas Holtz-Eakin and Harvey S. Rosen. “Sticking it out: entrepreneurial survival and liquidity constraints”. In: (1993).

[15] *Toshi wallet now supports ERC20 tokens and ERC721 collectibles*. `https://blog.toshi.org/toshi-wallet-now-supports-erc20-tokens-and-erc721-collectibles-e718775895aa`. Accessed: 30-08-2018.

[16] *ERC-20 Token Standard*. `https://eips.ethereum.org/EIPS/eip-20`. Accessed: 30-08-2018.

[17] Oraclize. “A Scalable Architecture for On-Demand, Untrusted Delivery of Entropy”. In: (2015).

[18] Daniel IA Cohen and Daniel IA Cohen. *Introduction to computer theory*. Vol. 2. Wiley New York, 1991.

[19] Marko Vukolić. “Rethinking permissioned blockchains”. In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM. 2017, pp. 3–7.

[20] AbdulSalam Kalaji, Rob Mark Hierons, and Stephen Swift. “A search-based approach for automatic test generation from extended finite state machine (EFSM)”. In: *Testing: Academic and Industrial Conference-Practice and Research Techniques, 2009. TAIC PART'09.* IEEE. 2009, pp. 131–132.

[21] M.G. Gouda, E.G. Manning, and Y.T. Yu. “On the Progress of Communication between Two Finite State Machines”. In: (1984).

[22] G Pace and J Schapachnik. “Contracts for Interacting Two-Party Systems”. In: (2012).

[23] Mark D. Flood and Oliver R Goodenough. “Contract as Automaton: The Computational Representation of Financial Agreements”. In: (2015).

[24] Davide Basile, Pierpaolo Degano, and Gian-Luigi Ferrari. “Automata for Service Contracts”. In: (2014).

[25] Pierpaolo Degano Davide Basile and Gian-Luigi Ferrari. “From Orchestration to Choreography through Contract Automata”. In: (2014).

[26] Ian Ayres and Robert Gertner. “Filling Gaps in Incomplete Contracts: An Economic Theory of Default Rules”. In: (1989).

[27] Jan L.G. Dietz. “Understanding and Modeling Business Processes with DEMO”. In: (1999).

[28] YoungJoon Byun, Beverly A. Sanders, and Chang-Sup Keum. “Design Patterns of Communicating Extended Finite State Machines in SDL”. In: (2001).

[29] Petri. “Kommunikation mit Automaten”. In: (1962).

[30] Dennis Kafura. *Notes on Petri Nets*. `http://people.cs.vt.edu/kafura/ComputationalThinking/Class-Notes/Petri-Net-Notes-Expanded.pdf`. Accessed: 01-09-2018.

[31] Wil M.P. van der Aalst. “The Application of Petri Nets to Workflow Management”. In: (1998).

[32] Jan Recker et al. “How good is bpmn really? Insights from theory and practice”. In: (2006).

[33] Wil M.P. van der Aalst et al. “Life After BPEL?” In: (2005).

[34] Luciano García-Bañuelos et al. “Optimized Execution of Business Processes on Blockchain”. In: (2017).

[35] Jan L.G. Dietz. “DEMO: Towards a discipline of organisation engineering”. In: (1999).

[36] *JSONForms - React*. `https://jsonforms.io/`. Accessed: 30-08-2018.

[37] *JSONForm - Bootstrap 3*. `https://github.com/jsonform/jsonform`. Accessed: 30-08-2018.

[38] *Mozilla react-jsonschema-form*. `https://github.com/mozilla-services/react-jsonschema-form`. Accessed: 30-08-2018.

[39] *Angular Schema Form*. `http://schemaform.io/`. Accessed: 30-08-2018.

[40] *Open Document Format*. `http://www.opendocumentformat.org/`. Accessed: 30-08-2018.

[41] Sindhu Sajana and Sethumadhavan. “On Blockchain Applications: Hyperledger Fabric And Ethereum”. In: (2018).

[42] Stephen A Cook. "The complexity of theorem-proving procedures". In: *Proceedings of the third annual ACM symposium on Theory of computing*. ACM. 1971, pp. 151–158.

[43] Daniel Brand and Pitro Zafiropulo. "On communicating finite-state machines". In: *Journal of the ACM (JACM)* 30.2 (1983), pp. 323–342.

[44] Robert M Hierons AbdulSalam Kalaji and Stephen Swift. "New approaches for passive testing using an Extended Finite State Machine specification". In: (2003).

[45] O Sury and R Edmonds. *Edwards-Curve Digital Security Algorithm (EdDSA) for DNSSEC*. Tech. rep. 2017.

[46] NIST. *Transition Plans for Key Establishment Schemes using Public Key Cryptography*. `https://csrc.nist.gov/News/2017/Transition-Plans-for-Key-Establishment-Schemes`. Accessed: 13-07-2018. 2017.

[47] Daniel J. Bernstein et al. "High-speed high-security signatures". In: *Journal of Cryptographic Engineering* 2.2 (2012), pp. 77–89. ISSN: 2190-8516. DOI: `10.1007/s13389-012-0027-1`. URL: `https://doi.org/10.1007/s13389-012-0027-1`.

[48] Henri Gilbert and Helena Handschuh. "Security analysis of SHA-256 and sisters". In: *International workshop on selected areas in cryptography*. Springer. 2003, pp. 175–193.

[49] NIST. *NIST Policy on Hash Functions*. `https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions`. Accessed: 13-07-2018. 2015.

[50] Bruce Schneier and John Kelsey. "Cryptographic Support for Secure Logs on Untrusted Machines." In: *USENIX Security Symposium*. Vol. 98. 1998, pp. 53–62.

[51] E. Brewer. "Towards Robust Distributed System. Symposium on Principles of Distributed". In: (2000).

[52] Peter Bailis and Ali Ghodsi. "Eventual consistency today: Limitations, extensions, and beyond". In: *Queue* 11.3 (2013), p. 20.

[53] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[54] Miguel Castro and Barbara Liskov. *Byzantine fault tolerance*. US Patent 6,671,821. 2003.

[55] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. `https://bitcoin.org/bitcoin.pdf`. Accessed: 17-05-2018.

[56] Aggelos Kiayias et al. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 357–388.

[57] POA Network. *Proof of Authority: consensus model with Identity at Stake*. `https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256`. Accessed: 17-05-2018.

[58] Marko Vukolic. "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication". In: (2016).

[59] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[60] Git Documentation. *Git Branching - Rebasing*. `https://git-scm.com/book/en/v2/Git-Branching-Rebasing`. Accessed: 09-08-2018.

[61] Aaron van Wirdum. "Rejecting Today's Hard Fork, the Ethereum Classic Project Continues on the Original Chain: Here's Why". In: *Bitcoin Magazine* 20 (2016).

[62] European Parliament. *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL: on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN`. Accessed: 12-07-2018. 2016.

[63] Olly Jackson. "Is it possible to comply with GDPR using blockchain?" In: *International Financial Law Review* (2018).

[64] *Art. 28 GDPR - Processor*. `https://gdpr-info.eu/art-28-gdpr/`. Accessed: 15-09-2018.

[65] S Goldwasser, S Micali, and C Rackoff. "The knowledge complexity of interactive proof systems". In: (1989).

[66] *Blockchain costs per transaction*. https://www.blockchain.com/charts/cost-per-transaction. Accessed: 05-09-2018.

[67] Emanuel Palm. *Implications and Impact of Blockchain Transaction Pruning*. 2017.

[68] Wenting Li et al. "Towards scalable and private industrial blockchains". In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM. 2017, pp. 9–14.

[69] Serguei Popov. "A probabilistic analysis of the nxt forging algorithm". In: *Ledger* 1 (2016), pp. 69–83.

[70] Waves platform. *WAVES whitepaper*. https://blog.wavesplatform.com/waves-whitepaper-164dd6ca6a23. Accessed: 16-07-2018. 2016.

[71] *Chainpoint Node API: How to Create a Chainpoint Proof*. https://github.com/chainpoint/chainpoint-node/wiki/Chainpoint-Node-API:-How-to-Create-a-Chainpoint-Proof. Accessed: 05-09-2018.

[72] Gleb Kostarev. *Review of blockchain consensus mechanisms*. https://blog.wavesplatform.com/review-of-blockchain-consensus-mechanisms-f575afae38f2. Accessed: 13-07-2018. 2017.

[73] NEM. *NEM technical reference*. https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf. Accessed: 13-07-2018. 2018.

[74] LTO. "LTO Token Economy". In: (2018).

[75] Waves Platform. *Blockchain Leasing For Proof Of Stake*. https://blog.wavesplatform.com/blockchain-leasing-for-proof-of-stake-bac5335de049. Accessed: 13-07-2018. 2018.

[76] mthcl. "The math of Nxt forging". In: (2014).

[77] *Waves generators*. http://dev.pywaves.org/generators/. Accessed: 28-08-2018.

[78] *Nxt Blockchain Explorer*. https://nxtportal.org/monitor/. Accessed: 28-08-2018.

[79] Kofman Begicheva. "Fair Proof of Stake". In: (2018).

[80] *Waves-NG stress test: results in!* https://blog.wavesplatform.com/waves-ng-stress-test-results-in-44090f59bb15. Accessed: 05-09-2018.

[81] S. Haber and W.S. J Stornetta. “How to time-stamp a digital document”. In: (1991).

[82] Ralph C. Merkle. “Method of providing digital signatures”. U.S. pat. 4309569. Jan. 5, 1982.

[83] A. Begicheva and I. Smagin. “RIDE: a Smart Contract Language for Waves”. Pat. 2018.

[84] Saifedean Ammous. “Blockchain Technology: What is it good for?” In: (2016).

[85] *Blockchain number of transaction*. `https://www.blockchain.com/en/charts/n-transactions-total`. Accessed: 05-09-2018.

[86] *BIP 141: Segregated Witness (Consensus layer)*. `https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki`. Accessed: 05-09-2018.

[87] *Problems Ethereum*. `https://github.com/ethereum/wiki/wiki/Problems`. Accessed: 30-08-2018.

[88] G Hardin. “The Tragedy of the Common”. In: (1969).

[89] *Nothing considered a look at nothing at stake vulnerability for cryptocurrencies*. `https://pivx.org/nothing-considered-a-look-at-nothing-at-stake-vulnerability-for-cryptocurrencies/`. Accessed: 30-08-2018.

[90] John R Douceur. “The sybil attack”. In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[91] Marc Stevens et al. “The first collision for full SHA-1”. In: (2017).

[92] Kubernetes. *Kubernetes, Horizontal Pod Autoscaling*. `https://github.com/kubernetes/community/blob/master/contributors/design-proposals/autoscaling/horizontal-pod-autoscaler.md`. Accessed: 03-08-2018.

#	**Field Name**	**Length**
1	Version	1
2	Timestamp	8
3	Parent block signature	64
4	Consensus block length	4
5	Base target	8
6	Generation Signature	32
7	Transaction list Hash	32
8	Anchor Merkle root	32
9	Generator public key	32
10	Block's signature	64

表 1: Key block structure

#	**Field Name**	**Length**
1	Amount of transactions (x)	4
2	Transaction #1 bytes	113
...	...	...
2 + (K - 1)	Transaction #K bytes	113

表 2: Key block attachment

#	**Field Name**	**Length**
1	Version	1
2	Timestamp	8
3	Parent block signature	64
4	Consensus block length	4
5	Base target	8
6	Generation Signature	32
7	Transaction list Hash	32
8	Transaction #1 bytes	TODO
...	...	...
$8 + (K - 1)$	Transaction #K bytes	TODO
$9 + (K - 1)$	Balance change summary entry #1	40 (Table 4)
...	...	...
$9 + (K - 1) + (N - 1)$	Balance change summary entry #N	40 (Table 4)
$10 + (K - 1) + (N - 1)$	Generator public key	32
$11 + (K - 1) + (N - 1)$	Block's signature	64

表 3: Summary Block structure

#	**Field Name**	**Length**
2	Wallet address	32
3	Balance change	8

表 4: Balance summary entry

#	**Field Name**	**Length**
1	Transaction type	1
2	Anchor hash	32
3	Fee	8
4	Timestamp	8
5	Signature	64

表 5: Anchor transactions structure

#	**Field Name**	**Length**
1	Transaction type	1
2	Sending address	32
3	Receiving address	32
4	Amount	8
5	Fee	8
6	Timestamp	8
7	Signature	64

表 6: Transfer transaction structure

#	**Field Name**	**Length**
1	Transaction type	1
2	Id	32
3	Sending address	32
4	Receiving address	32
5	Expiration Date	8
6	Certificate Type	32
7	Fee	8
8	Timestamp	8
9	Signature	64

表 7: Issue certificate transaction structure

#	**Field Name**	**Length**
1	Transaction type	1
2	Id	32
3	New expiration Date	8
4	Fee	8
5	Timestamp	8
6	Signature	64

表 8: Update certificate transaction structure

#	**Field Name**	**Length**
1	Transaction type	1
2	Sending address	32
3	Receiving address	32
4	Amount	8
5	Fee	8
6	Timestamp	8
7	Signature	64

表 9: Lease transaction structure

#	**Field Name**	**Length**
1	Transaction type	1
2	Sending address	32
3	Receiving address	32
4	Amount	8
5	Fee	8
6	Timestamp	8
7	Signature	64

表 10: Cancel Lease transaction structure