字节雪球(Byteball):一个存储和传输价值的去中心化系统

Anton Churyumov tonych@byteball.org

原文网址：https://byteball.org/Byteball.pdf

翻译：Maxwell

摘要

Byteball(字节雪球)是一个去中心化的系统，允许任意数据的防篡改存储，包括可转移价值的数据，例如货币，产权，债务，股份等。这些存储单元彼此链接，每个存储单元包括一个或多个早期存储单元的散列值，既用于证实早期的单元又用于确立它们的偏序关系。链接单元之间形成 DAG（定向非循环图）。没有管理或协调新单元进入数据库的单一中心实体，允许每个人添加新的单元，只要他签署并支付的费用等于添加数据字节的大小。其他后来用户通过自己单元内的散列值来确认早期单元，并收取费用。随着新单元的添加，每个早期单元，包括其中的散列值，直接或间接的接收越来越多后来单元的确认。

被称为“bytes(字节币)”的内部货币，用于支付将数据添加到去中心化数据库中的费用。其他货币或资产也可以通过任何人自由发行，来代表产权，债务，股票等。用户彼此可以同时发送 bytes 和其他货币，以支付商品/服务或交换货币种类，转移价值的交易将作为存储单元添加到数据库。如果两笔交易尝试使用相同的输出（double –spend 双重支付）并且它们之间没有偏序关系，则两笔交易都被允许进入数据库，但只有在全序（Total order）中较早的一笔交易被认为是有效的。全序通过选择一条单链建立在 DAG（主链）上，吸引由被称为见证人的已知用户签署的单元。在主链上较早包含散列值的单元被认为在全序上较早。用户通过在每个存储单元中命名用户信任的证人来挑选见证人。见证人是具有现实世界身份的信誉良好的用户，这样命名他们的用户期望他们永远不要尝试双重支付

(double-spend)。只要大多数见证人表现得如预期的那样，所有的双重支付尝试会被及时检测到并且被标记。作为见证人 – 在用户的单元之后积累汇总被创作的单元，当用户单元的总次序被认为是最终不可更改时，存在确定性（非概率）的标准。

用户将其资金存储在可能需要多个签名的地址上（multisig 多重签名技术）。支出还可能需要满足其他条件，包括通过查找由其他用户（oracle）发布到数据库的特定数据来评估的条件。

用户可以发布新资产并定义以管理其可转移性规则。规则可以包括支出限制，例如每个转移的要求由资产的发行者联合签署，这是金融机构遵守现行条例的一种方式。 用户还可以发行其未传输至数据库的资产，因此对第三方是不可见的。相反，有关传输的信息在用户之间私下交换，并且只有这笔交易的散列值和支出证明（以防止双重支付）被发布到数据库。

1.介绍

在乔治·奥威尔的《1984》(政治讽刺小说)中，主角温斯顿·史密斯在真理部记录司作为编辑，从事篡改历史的工作，使历史符合不断变化的英格兰社会主义(故事中主角所在国家的政党)的路线，并删除涉及到的非人——已经“蒸发”的人，即不仅被国家杀害，而且甚至在历史或记忆中也被否认存在[1]。我们在这里提出的是不可重写的数据存储。 它是一个分布式去中心化的数据库，其中记录既不能修改也不能完全删除。

比特币[2]是第一个引入防篡改记录的系统，专门用于跟踪被称为比特币的电子货币单元的所有权。在比特币中，货币的所有转移都被表示为由当前币的所有者数字签名的交易，交易被捆绑成块，然后块被链接到由工作量证明机制（PoW）担保

的链（区块链）上，确保大量计算资源投入链的建设当中。因此 ，任何企图重写链中任何内容的行为，将需要消耗比已消耗计算资源更大的计算资源。

比特币出现后不久，它逐渐让人们意识到这不仅仅是一个去信任化的 P2P 电子货币。 它的技术成为解决其他问题所需要的新思路来源。同时，比特币的不足和局限性也同样变得清晰。 Byteball 旨在推广比特币，以成为任何数据的防篡改存储，而不仅仅是单一电子货币的转移，并消除阻碍比特币更广泛采用和增长的一些最紧迫的缺陷。

区块。在比特币中，交易被捆绑成块，然后块被链接到单链中。由于块是线性链接的，他们的时间间隔和大小使得节点之间的近同步性(near-synchrony)最优化，所以节点可以彼此共享一个新的块，比通常生成新块的速度快的多。这确保了节点最可能将相同的区块视为最后一个块，并且被孤立的可能性最小化。随着比特币的增长，块变得越来越笨拙。它们或者被限制大小，在这种情况下增长也同样被限制，或者它们传播到网络的所有节点花费时间过长，这样的话对于哪个块是最后一个，就存在更大的不确定性，并且更多的资源被浪费在扩展将被孤立的链上。在 Byteball 中，没有块，交易是他们自己的块，他们不需要连接到单链。相反，交易可以链接到多个先前的交易，并且整套交易不是链而是 DAG（定向非循环图）。基于 DAG 的设计最近备受关注[3-5]。

成本。 比特币交易是安全的，因为重做创建于交易之后的块中包含的所有 PoW 是非常昂贵的。但这也意味着，有必要对建立强大到足以抵御任何攻击者的合理 PoW 进行支付。 这笔款项用于支付建造 PoW 所需的电力。此处需要注意的是，这些钱从比特币生态系统流出到能源公司 - 意味着比特币持有者作为一个整体的社区，其资本正在失血。在 Byteball 中，没有 PoW，而是使用另一种基于“在比特币之前就已知的旧想法”的共识算法。

终结(finality)。比特币的交易终结是概率性的。对于你何时能说一个交易永远不会被逆转，没有严格和直观的标准。相反，你只能认为，随着更多的块被添加，交易被逆转的概率呈指数衰减。虽然这个概念对于那些精通数学的人来说是很清楚的，但是对于一个习惯在金钱所有权问题上期待黑白分明(a black-or-white picture)的普通人来说，这可能会是一次困难的销售。事情进一步复杂化，交易的终结也取决于其数量。如果金额很小，你可以合理地确信没有人会试图对你进行双重支付。然而，如果股权金额大于区块奖励（在写作时为 12.5 BTC），您可能推测，付款人可以临时租用哈希算力(hashpower)来挖掘不包含支付给您的交易的另一区块链。因此，在确定高价值交易是最终交易之前，您必须等待更多确认。在 Byteball 中，一个交易何时被认为是最终的，有一个确定性的标准，无论它有多么的大。

汇率。 比特币价格的非常不稳定是已知的。更大的问题是，这个价格不仅是波动的，它还不受任何约束。 股票和商品价格也很不稳定，但它们背后是有着一定基础原则的。股价主要是公司盈利，收入，债务资本比率等的函数。商品价格除其他因素外，还取决于与不同供应商的生产成本。 例如，如果石油价格长期低于一些供应商的生产成本，这些供应商最终将关闭，减少生产，并导致价格上涨。这存在着一个负反馈回路。 在比特币中，没有基础原则，没有负反馈。 一个比特币 500 美元的价格不比 5 万美元或 5 美元更合理。如果比特币价格从现在的位置发生变动，这一变动不会创造任何推动价格回升的经济力量。它只是处于狂热状态。在 Byteball 中，基本货币字节币(bytes)用于支付将数据添加到 Byteball 数据库中的费用。您添加 1Kb 的数据需要支付 1,000 字节币。它是衡量此数据库中存储效用的度量，并且实际用户将对什么是它的合理价格有他们的主张。如果字节币的价格高于你认为合理的价格，你会发现存储更少字节币的方法，因此你需要买更少的字

节币，需求减少，价格下降。这是负反馈，常见的所有商品/服务的需求都是由需求驱动，而不是投机。除了用字节币支付外，人们也可以发行其他资产并将其用作支付手段。这些资产可能代表例如以法定货币或自然单位为表示的债务（例如千瓦时-电功计量单位或油桶-石油计量单位）。这些资产的价格自然与基本货币或商品有关。

隐私性。 所有的 Bitcoin 交易和所有的地址余额在区块链上都可见。 虽然有办法混淆一个人的交易和余额，但这不是人们从一种货币中所期望看到的。Byteball 中的字节币（基本货币）同样可见，但有第二种货币（blackbytes，黑字节币），是明显较难追溯的。

合规性。 比特币被设计成一种人们对金钱有绝对控制权的匿名货币。 这个目标已经实现; 然而，这使得比特币不符合现有的法规，因此不适合在金融业使用。而在 Byteball 中，可以如比特币一般用任何规则发行资产来控制其可转移性，从没有任何限制，对每笔交易都需要由发行人签署（例如银行）或限制为有限的白名单用户。

2.数据库结构

当用户想要向数据库添加数据时，他创建一个新的存储单元并将其广播给他的对等节点。存储单元（除了别的以外）还包括：

•要存储的数据。一个单元可以包括多个数据包，称之为信息。 有许多不同类型的信息，且各有自己的结构。 其中一种信息类型是支付，用于向对等节点发送bytes(字节币)或其他资产。

•创建单元的一个或多个用户的签名。 用户由其地址标识。 个人用户可以（并且鼓励）拥有多个地址，就像比特币。最简单的情况，地址源于公钥，再次类似于比特币。

•引用由其哈希值标识的一个或多个先前的单元（父母单元）。

引用父母单元是建立单元的次序(目前为止只有部分次序)和推广区块链结构。由于我们不局限于连续块之间的单亲 - 单子关系，所以我们不必争取近同步(性)，并且可以安全地承受大的延迟和高吞吐量 : 每个单元只会有更多的父母单元和更多的子单元。如果我们沿着父子链在历史上前进，当同一单元被多个后来的单元引用时，我们将观察到许多分叉，并且当同一单元引用多个较早单元时，许多单元逐渐融合（开发者已经习惯看到这个动态）。这种结构在图论中称为有向无环图（DAG）。 单元是顶点，父子链是图的边缘。

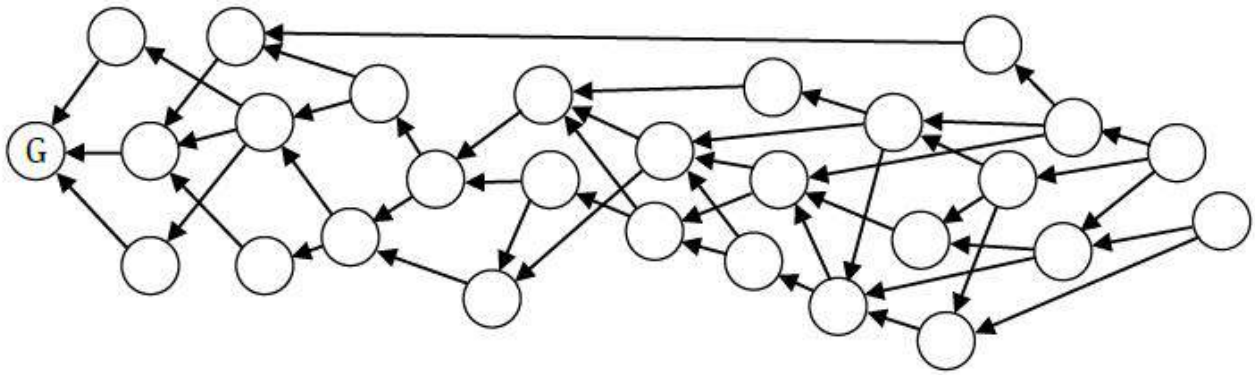

图 1，连接成一个 DAG 存储单元。箭头是从子单元到父母单元，G 是创始单元

在新的单元极少到来这种特殊情况下，DAG 将看起来几乎就像一个链，偶尔分叉而又快速融合。

类似于让每个新块确认先前所有块（以及其中的交易）的区块链中，DAG 中的每个新子单元确认其父母单元，父母单元的所有父母单元，父母单元的父母单元的父母单元等。如果有人尝试编辑一个单元，他也必须改变它的哈希值。 不可避

免地，这将破坏所有引用此单元哈希值的子单元，因为子节点的签名和哈希值取决于父哈希值。因此，不能在窃取其私钥或是不与其所有子单元达成合作的情况下修改单元。子单元们不能在没有与子单元合作的情况下修改他们的单元（原单元的孙子单元），等等这些。 一旦一个单元被广播到网络中，并且其他用户开始在它上面构建它们的单元（将其称为父单元），编辑这个单元所需的二次修改就如雪球一样增长。这就是为什么我们称之为 Byteball（我们的雪花是数据中的字节）。

与“发布一个块是一个罕见的事件，且只有特权阶层的用户在实践中从事这项活动”的区块链不同，在 Byteball 中，单元发布之后，立即开始累积确认，并且确认可以来自另一个新单元被任何人发布的任何时候。这里没有普通用户和矿工这样的两层系统。相反，用户互相帮助：通过添加一个新单元，发布者也确认了所有以前的单元。

如果尝试修改过去的交易，与需要大量计算工作的比特币不同，尝试修改 Byteball 中过去的记录需要与大量且越来越多的其他用户协调，其中大多数是匿名的陌生人。因此，过去记录的不变性是基于与如此大量的陌生人协调的复杂性，这些人难以达成一致，对合作没有兴趣，并且每一个人都可以否决修订。

通过引用其父单元，一个单元将包括其父单元。它不包括父母单元的全部内容; 相反，它的内容取决于父母单元哈希值的信息。同样，单位间接依赖于并因此包括父母的父母，其父母等，并且每个单元最终包括创世单元。

有一个协议规则，一个单元不能引用多余的父母单元 – 换言之，这些父母单元是一个父母单元包括另一个。 例如，如果单元 B 引用单元 A，则单元 C 不能同时引用单元 A 和 B. A 已经以某种方式包含在 B 中。此规则去除了，不能向 DAG 中添加任何新的有效连通性的不必要链接。

3.天然货币：Bytes

接下来，为了防止无用信息刷屏数据库，我们需要引入一些摩擦力。 入口的屏障应大致反映用户的存储效用和网络的存储成本。 对于这两者而言最简单的测量方法是测量存储单元的大小。因此，要将数据存储在去中心化的全球数据库中，您必须以内部货币（称为字节币）来做为支付费用，您支付的金额与你打算存储的数据大小相对等（包括所有标头，签名等）。就如同首次推出时等于一磅银的英镑，货币的名称反映了它的价值。

为了使激励措施与网络的利益保持一致，在大大小小的计算规则中存在一个例外。为了计算单元大小，假定单元就恰好具有两个父单元，不论真实数字。因此，两个父单元的哈希值的大小总是包含在单元大小中。此例外确保用户不会为了尽量降低成本，而尝试只包含一个父单元，因为无论包含多少父母单元，这成本都是相同的。

为了使 DAG 尽可能地精确，我们鼓励用户尽可能多地包含父母单元（如前所述，这不会对应付的大小产生负面影响），并且尽可能地通过那些最初作为父单元将其包含的单元费用的支付部分成为最近的父母单元。我们稍后将定义什么是“最初”。

字节币不仅可用于支付存储费用（也称为佣金），还可以发送给其他用户支付商品或服务或换取其他资产。为了发送付款，用户需创建包括诸如以下（从现在开始，我们使用 JSON 来描述数据结构）的支付信息的新单元：

```
{
    inputs: [
            {
                unit: "hash of input unit",
                message_index: 2, // index of message where this utxo
                was created
                output_index: 0 // index of output where this utxo
                was created
            },
            ...
    ],
    outputs: [
            {
                address: "RECEIVER ADDRESS",
                amount: 15000    // in bytes
            },
            ...
    ]
}
```

该信息包含：

• 输出数组：一个或多个接受字节币和金额的地址；

• 输入数组：一个或多个先前输出用于转移基金的引用，这些是过去被发送到作者地址且还未被使用的输出。

输入总和应等于输出加佣金的总和（输入金额从先前的输出读取，在支出时没有明确说明）。该单元用作者的私钥签名。

循环中的字节总数为 1015，这个数是常量。所有字节币都在创始单元发出，然后从用户转移到用户。费用由帮助维护网络健康的用户收取（更多细节过后补充），所以他们保持流通。之所以选择 1015 是因为在 JavaScript 中它代表最大的整数，数量只能是整数。货币的较大单位通过标准的前缀得出：1 千字节（Kb）是 1000 字节，1 兆字节（Mb）是 1 百万字节等等。

4.双花(双重支付)

如果用户尝试使用两次相同的输出，有两种可能的情况：

1、在尝试使用相同输出的两个单元有序，即一个单元（直接或间接）中包括另一个单元，并且在它之后。在这种情况下，我们显然可以安全地拒绝后面的单元。

2、他们之间无序关系，在这种情况下，我们都接受。我们建立单位之间的总序后，当他们隐藏在足够深的新单位里（见下文我们如何怎么做）。在总序较早出现的一方视为有效，另一方视为无效。

有一个简化定义总序的协议规则。我们要求，如果相同的地址发布超过一个单元，则它应当（直接或间接地）在每个后续单元中包含其所有先前单元，即来自相同地址的连续单元之间应当有序。换句话说，从同一作者的所有单元应连续。

如果有人违反这一规定并发布两个单元，使得它们无序（非序列单元），则这两个单元被视为双重支付，即使它们没有尝试使用相同的输出。这种非系列单元如上面情况 2 所述处理。

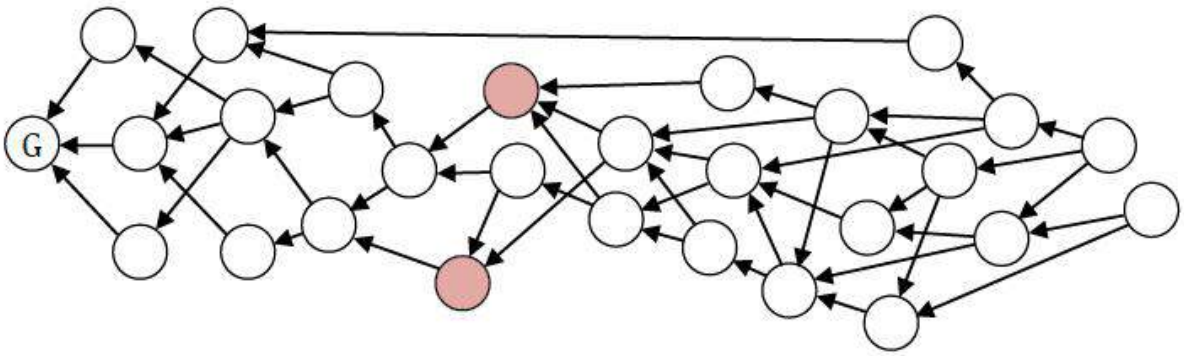

图 2 双花(双重支付) 它们之间无序

如果用户遵循这个规则，但仍尝试两次花费相同的输出，则双重支付变得明确有序，并且我们可以安全地拒绝后一种情况，如上面情况 1 所示。因此，同时不是非序列的双支出容易被过滤掉。

这个规则其实很自然。当用户组成一个新的单元时，他选择最近的其他单元作为其单元的母单元。通过把他们列在他的母名单上，他向外宣布他的图片，这意味着他已经看到了这些单元。因此，他看到了母单元里的所有母单元，母单元的母单

元等等，直到创世块。这个巨大的集合应该显然包括他自己已经产生的一切，并且他自己可见。

用户通过不包含一个单元（甚至是间接地通过父母单元）来否认他看到的这个单元。如果用户通过不包含他自己以前的单元来否认已经看到的这个单元，我们会说这很奇怪，这事有蹊跷。我们不鼓励这种行为。

5. 主链

我们的 DAG 是一个特殊的 DAG。在正常使用中，人们通常将他们的新单元链接到略小的最近单元上，这意味着 DAG 仅在一个方向上增长。人们可以把它画成一条里面带有许多交错线的粗线。该属性表明我们可以在 DAG 中沿着“子-父链”选择单个链，然后将所有单元关联到此链。所有的单元要么将直接位于这条我们称之为主链的链上，要么沿着 DAG 的边缘从相对少量的跃点到达。它就像一条连接着侧面道路的高速公路。

构建主链的方法之一是开发一种算法，在既定单元的所有父母单元中，选择一个作为“最佳父母单元”。选择算法应该仅基于所讨论的单元可用的知识，即选择算法应仅基于所讨论的单元可用的知识，即单元本身和其所有祖先中包含的数据。从 DAG 的任意顶端（无子单元）开始，然后沿着最佳父单元链接沿着历史向后进行。以这种方式移动，我们构建主链并最终到达创世单元。注意，从一个特定单元建立的主链不会因为新单元的增加而改变。这是因为，我们移动的每一步都是从子单元到父单元，而现存的单元永远不会获得新的父母单元。

如果我们从另一个顶端开始，我们将构建另一条主链。这里值得注意的是，如果这两条主链在历史返程中曾相互交叉，那么它们将在相交点之后沿着相同的路径行进。

在最坏的情况下，主链将仅在创始区相交。然而，考虑到单元生产的过程在用户之间不协调，人们期望可以找到一类主链，其聚集的位置离顶端不太远。

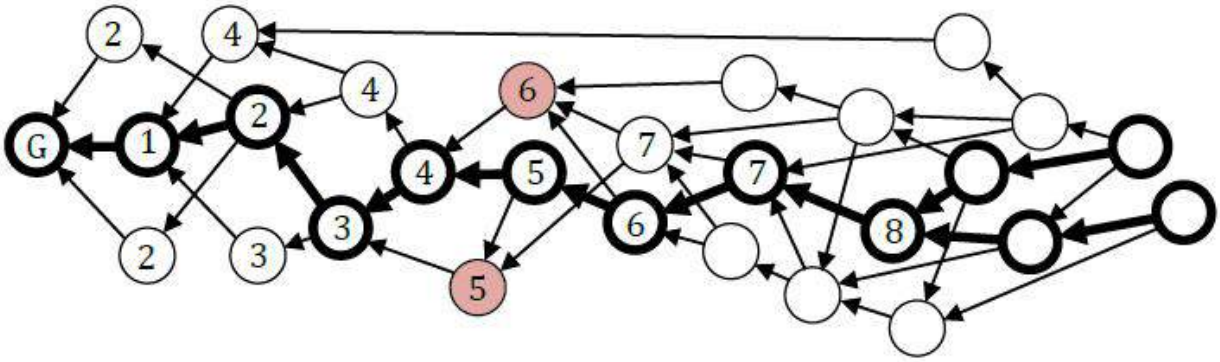

图 3，由不同的无子单元建立的主链相交，然后沿着相同的路径行进。在两个双重支付中，具有较低主链指数（5）的一条赢得胜利，而另一条（具有 MCI = 6）被认为无效。

一旦我们有一条主链，就可以在两个矛盾的、无序的单元之间建立总序。首先让我们索引直接位于主链上的单元。创世单元的索引值为 0，下一个 MC 单元是创世元的子单元，其索引值为 1，因此，沿着 MC 向前行进，我们为位于 MC 上的单元分配索引。对于不在 MC 上的单元，我们可以找到其中首先包括（直接或间接）该单元的 MC 索引。以这样的方式，我们可以给每一个单元指定一个 MC 指数（MCI）。

然后，在两个非序列中，具有较低 MCI 的那一个被认为是较早并且有效的序列，另一个则无效。如果两个非序列恰巧具有相同的 MCI，根据 tiebreaker 规则，具有较低散列值的单元（如基本 64 编码中所表示的）是有效的。请注意，我们保留所有版本的双重支付，包括最终损失的那些。DagCoin [3]是第一个发表的作品，建议存储所有冲突的交易并且决定哪一个被视为有效。

由一个特定的单元建立的 MC 指数告诉我们这个单元的作者对过去事件顺序的看法，即他对历史的观点。该顺序意味着要考虑哪个非序列单元有效，如上所述。

注意，我们通过选择给定单元的所有父母中的最佳父母单元，并在他们的 MC 之间进行选择 .所讨论的单元的 MC 指数将是最佳父单元通过一个链接向前延伸的 MC。

识别许多（甚至所有）可能由攻击者创建的父母单元，并记住最佳父母单元的选择本质上是历史版本的选择，我们应该要求从我们的最佳父母单元选择算法，该算法喜欢从子单元的角度来看“真实”的历史。因此，我们需要设计一个“现实测试”，我们的算法将对所有候选 MCs 运行，以选择得分最佳的一个。

6.见证人

为寻求一个“客观公正的测试”，通过观察我们网络的一些参与者，他们可能是非匿名长期参与社区并拥有良好信誉的人，或是主动维护网络健康发展的组织。我们称之为证人。虽然期望他们诚实行事是合理的，但完全信任任何一个证人也是不合理的。如果我们知道几个见证人的 Byteball 地址，并且期望他们频繁地发布，然后，为了测量候选 MC 的真实性，可以沿着 MC 及时返回，并统计出见证授权单元的个数（如果同一证人多次被遇到，那么他也不会被重复计数）。我们一遇到大量见证人，就会停止行进，然后，我们会从测量图上的创造起点开始量出最长路径的长度。我们将这个长度称为我们停止的单元级别，以及我们正在测试的见证父母级别的 MC。产生较大见证级别的候选 MC 被认为是更“真实”的，并且具有该 MC 的父母单元被选择为最佳父母单元。如果存在一些具有较大见证级别的竞争者，我们将选择其中最低级别的父母单元。如果连接持续，我们将选择具有最小单元散列值（在 base64 编码里）的父母单元。

这种算法允许吸引由见证人授权的单元的 MC 的选择，并且见证人被认为是现实的代表。

例如，如果攻击者从网络的真实数据分叉，并秘密地建立一个属于自己的单元（影子链）长链，其中一个包含双花 （双重支付），之后将他的分叉合并回诚实的 DAG 中，最佳父母单元选择算法在合并点处将选择把 MC 驱动到诚实 DAG 中的父母单元，因为这是证人活跃的地方。证人不能简单地投射到影子链中，因为在合并之前，他们无法看到它。这个 MC 的选择反映的是由证人和指定他们的用户所看到的事件的顺序。攻击结束后，整条影子链将着落在 MC 上的一点，并且影子链中包含的双花费（双重支付）将被视为无效，因为它的有效对手在点合并之前更早出现。

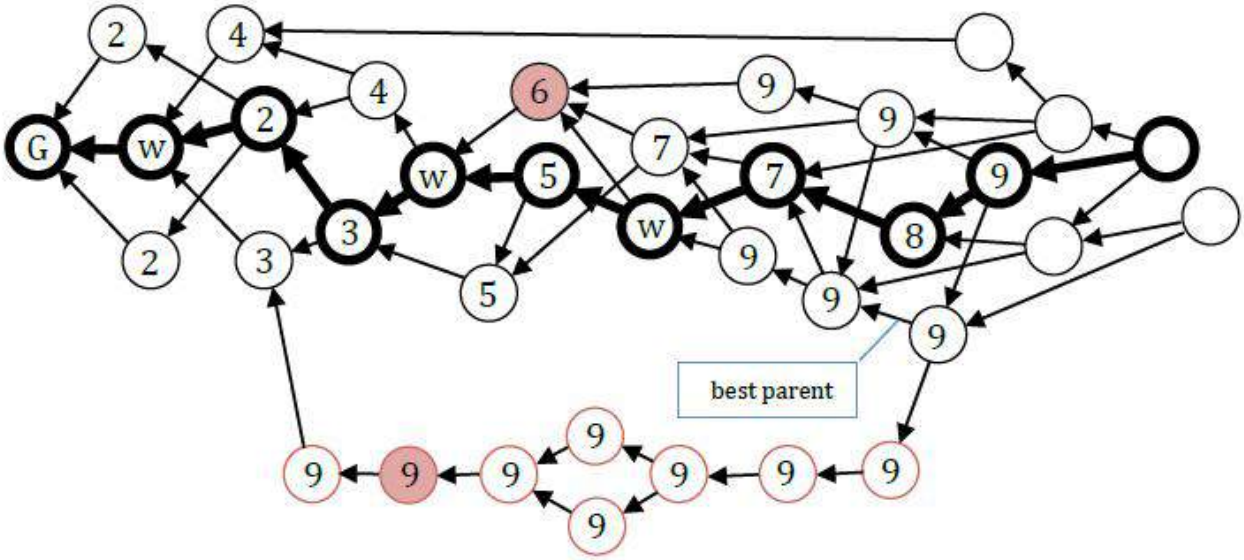

图 4。当攻击者将他的影子 DAG 重新加入发光的 DAG 中时，他的单元就失去成为最佳父母单元竞争的选择，因为这种选择偏爱拥有更多见证（标记为 w）的路径。

这个例子说明为什么大多数证人必须被信任才能连续发布。大多数见证者不应与攻击者勾结串通，并发布在他的影子链上。注意，我们相信见证人只是根据客观的事实，并且不在任何影子链上发布非序列单元。我们并没有给予他们任何超过网

络或相关部分的管理权。即使对于这小小的职责，指定见证人的用户可以随时改变他们的决定。

将一些已知的实体看作是现实标志的想法不是新的。很早就知道，一些公司从事这样的活动，为了证明一些数据在特定日期之前存在，可以在一些难以修改和广泛见证的媒体上分散并公布数据，如在印刷报纸[6]中发布哈希。 Byteball 的见证人与报纸有着同样的功能。像报纸一样，他们是众所周知并且可信的。至于可信的报纸内容被限制在相信他们公布那些得到的数据之上，byteball 的证人只有被信任才连续发布，而不是更多，就像报纸一样，证人不知道他们此刻见证的哈希值的背后是什么，而且几乎没有什么理由去关心。报纸很难被修改（但是具有修改的可能性，并且在《1984》中他们确实有这样做），然而证人产生的一切都受到数字签名的保护，这使得没有任何修改的可能性。考虑到可靠性，我们有一些证人，而不是只有一个，考虑到速度和便利，这些都是在线的。

决定了一系列的证人名单后，接着，我们可以选择最好的父母单元，以及如同“这些证人生活在哪里” 最符合现实定义的相应的历史。同时，父母单元本身也有不同的证人名单以及最终对现实的不同定义。我们渴望现实的定义，并希望历史继承从他们开始，覆盖到周围所有相同的事物。为了实现这一点，我们介绍以下附加协议规则。

“近乎一致的规则”：最好的父母单元必须从那些证人名单与孩子证人名单存在至多一人变动的父母单元中选择。该规则确保 MC 上相邻单元的证人名单差不多，因此他们的历史大多是相互一致的。见证列表相差 0 或 1 个突变的父母单元将被要求兼容（直接包括它们的单元），而其他的不兼容。不兼容的父母单元仍然被承认，但他们没有机会成为最好的父母单元。如果无子单元中不存在兼容的潜在的

父母单元（攻击者可以用携带完全不同的证人名单单元来泛滥网络），则应选择较老单元级别的父母单元。

上述意味着每个单元必须列出其证人名单，以便比较他们。我们要求证人的数量正好是 12.选择 12 这个数字是因为：

1.它足够大，以防范一些证人偶然间出错（他们可能证明不诚实，或被黑客攻击，或长时间离线，或丢失私钥并永远离线）；

2.它足够小，人们可以追踪所有证人的踪迹，了解谁是谁，并在必要时更改名单;

3.与那些未被更改的 11 名证人相比，允许突变的那一个得足够小。

如果用户认为任一见证人失去了他的信誉，或者刚好有更好的候选人，那么在他的名单中，用户可以用新的见证人更换那个见证人。考虑到他的证人名单可能与其他单元的名单在多个位置上没有不同。这意味着任何变化只能逐渐发生，对于大于一个位置的变化，需要一个普遍的共识。

7.结算

当新单元形成时，每个用户持续追踪其当前已被构建的 MC，好像他将要发布一个基于所有当前无子单元的新单元。当前的 MC 在不同节点处可能不同，因为它们可能看到不同的无子单元集合。我们需要构建当前的 MC，而不考虑见证列表，即用户自己的见证列表不重要，甚至不兼容的父母单元可被选择作为最好的父母单元。这意味着如果两个用户具有相同的无子单元集合，但具有不同的见证列表，那么它们当前的 MCs 将仍然是相同的。当前 MC 将不断随着新单元的形成而改变。然而，当我们将要示出时，一部分足够旧的当前 MC 将保持不变。

我们期望证人（或其中大多数人）诚实地行事，因此必须包括它们被任命的下一单元中的先前单元同样的证人。这意味着，当证人构建一个新单元，只有最近的单元是被选择作为父母单元的候选人。因此，我们可能预期，所有未来的当前 MCs 将不会比特定的稳定点更远地收敛（在回溯时间上）。实际上，创世单元是天然初始的稳定点。假设我们创建一个基于当前无子单元集的当前 MC，并且有一些点存在于预先相信其是稳定的 MC 上，即所有未来的当前 MC 被认为在该点处或之前收敛（再次，在回溯时间上），然后沿着相同的路线行进。如果我们可以找到向前推进此点（远离创世元）的方向，可以通过感应证明稳定点的存在。

注意，如果我们忘记除了最好的父母单元之外的所有父母单元，我们的 DAG 将被缩减为仅由最佳父链接组成的树。显然，所有 MCs 将沿着这棵树的分支行进。然后需要考虑两种情况 - 当树在当前稳定点中分支时和当它不分支时 - 并且决定我们是否可以将稳定点向前推进到下一个 MCI。

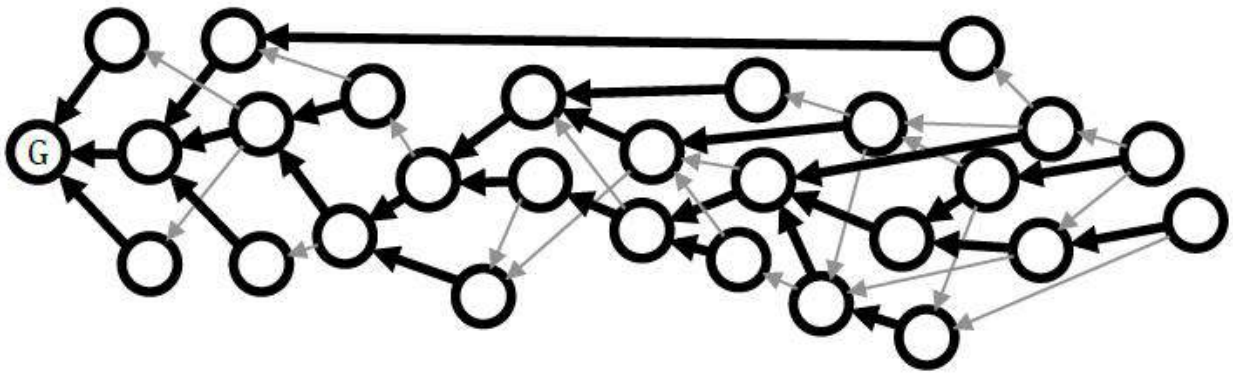

图 5.由最佳父链接组成的树。除了一个分支之外的所有在一些点后都停止增长。

首先，假设树不分支。那么我们需要考虑一种可能性，即一个新的分支将仍然被添加，并以某种方式由证人支持，使其超过了现有的分支。另一种可能性是，证人非常重视支持现有分支，要求包括某人先前的单元，让他们只能继续支持现有的

分支，而没有其他的选择，让我们量化后面的可能性。记住最好的父母单元被选择作为具有最大的见证水平的父母单元。让我们沿着当前的 MC 从尖端回溯到时间上，直到我们遇到很多见证人（我们指的是由位于当前稳定点上的单元定义的证人）。如果它们中的至少一个位于比当前稳定点更早的位置，那么我们不会尝试提前稳定点，否则我们会继续前进。在这种情况下，所有这些证人已经“投资”到当前的 MC。在这些当中证人，我们找到最小见证级别 min_wl。当这些见证人中的任何一个发布一个新的单元，这个单元可能拥有父母单元 MC 领导当前 MC 和导致竞争分叉， 并且具有最高见证级别的父母单元将会被选择作为最佳父母单元以及将会定义下一个当前 MC 的方向。由于证人不得不包括其以前的单元，领导当前 MC 的父母单元的见证级别将至少为 min_wl。任何导致替代分支父母单元的见证级别将永远不会超过当前稳定点的水平，即使所有剩余（少数）证人涌向替代分支。因此，如果当前 MC 增长足够远使得 min_wl 大于当前稳定点的水平，大多数证人将不得不增加对现有 MC 的支持，然后替代分支失去了所有赢的机会，并且我们可以推动稳定点向下一个 MCI 前进。

接下来，假设树做分支。我们需要找到一个替代分支将失去任何超过当前的 MC 机会的条件。让我们从前面的例子中定义 min_wl 开始。在替代分支的所有单元中，我们则选择那些增加见证水平的人，即他们自己的见证水平大于每个父母单元的见证水平。在这些中，我们找到最大水平，然后，即使所有剩余的（少数）证人聚集在替代分支上，替代分支上的见证等级将永远不会超过这个最大等级。因此，如果这个最大水平小于 min_wl，对于替代分支来说则游戏结束，并且我们可以沿着当前 MC 提前稳定点。

因此，在当前 MC 上存在一点，在该点之前 MC 将永远不会改变（假设大多数证人不发布非序列单元）。因此，相对于该 MC 定义的总顺序也是最终的。如果

我们有非序列单元，我们可以决定其中哪一个是有效的，以及最终的。如果新的非序列单元出现与已经稳定在 MC 上的任何内容发生冲突，则在旧对应单元之后，新的非序列单元将一定会被排序，并且新的非序列单元将被认为是无效的。因此，包括在稳定 MC 上的单元中进行的任何付款已经不可逆。与比特币不同，交易最终性只是概率性的，这是确定性交易的终极性。

每个用户基于他所看到的单元构建他自己的（主观的）当前 MC。由于新单元的传播不是即时的，并且它们可以以不同的顺序到达不同的用户，所以在任何给定时间内，用户将具有不同的当前 MCs 以及关于 MC 最后稳定点的不同看法。由于当前 MC 仅由用户已知的单元集合来定义，所以如果用户 B 尚未将其稳定点提升到与用户 A 相同的 MCI，那么，以后他将不可避免地那么做。（即一旦他接收与用户 A 相同的或者更多的单元。因此，不同用户关于任何给定单元的状态的意见最终是一致的。

8.非序列单元的储存

当我们决定一个单元是非序列单元时，我们仍然必须存储它。然而，其数据中的一部分被替换为数据哈希值。这个规则有两个目的。第一，为了减少因没有人支付单元的存储消耗（非序列单元的整个内容被认为无效，包括其佣金付款）。第二，为了减少用户发送非序列单元的效用，因为哈希值更替了对于作者来说想要（免费）存储的所有有用的数据。这可以防止攻击者滥用非序列单元作为免费存储大量数据的方式。

存储哈希值而不是所有内容对攻击者仍然有一些实用性，因为他可以自己存储原始数据，并使用哈希来证明数据存在。但请记住：

1.他仍然必须支付一个被认为有效的单元；

2.如果攻击者已在内部存储必要的元数据来翻译 byteball 数据 ,他可以通过将所有数据合并到 Merkle 树中并使用 Byteball 仅存储一个小单元成本的 Merkle 根来做同样好的工作。

在这样的设计下，因此，这里不存在试图发送非序列单元的自身利益。

应该提到的是，我们不能在第一次看到非序列单元时就拒绝它们。如果我们这样做，攻击者可以以不同的顺序将其非序列单元发送给不同的用户。那时，不同的用户就会坚持他们第一次接收的版本并且拒绝基于其他一切的版本，如此，攻击者将成功地划分网络。这就是为什么我们必须存储两个版本，然后决定他们顺序的原因。甚至，用户应该像任何其他单元一样将非序列单元转发给同辈们，因为同辈们越早了解非序列单元越好。

我们仍然尽可能避免包括非序列单元：只要它们是无子单元，父母单元选择算法就会排除非序列单元。出于这个原因，希望能帮助同辈们尽快了解非序列单元。

9.球

一个单元稳定后（即它包括在 MC 的稳定部分）我们基于该单元创建一个新的结构，我们称之为一个球：

```
ball: {
      unit: "hash of unit",
      parent_balls: [array of hashes of balls based on parent units],
      is_nonserial: true, // this field included only if the unit is
      nonserial
      skiplist_balls: [array of earlier balls used to build skiplist]
}
```

每个球包括关于它的所有祖先球的信息（通过父母单元），因此它依赖的增长的信息量的数量就像滚雪球。在这个球上，有一个标志可以告诉我们它是否是无效的（非序列），我们有引用到较老级别的球，以后我们将用于为轻客户建立证明。

当相应的单元变得稳定并且我们确定它是否是有序单元时，我们只能建立一个球。由于不同用户对于当前 MCs 的看法最终是一致的，因此它们将基于相同的单元构建完全相同的球。

10. 最后的球

为了保护球（最重要的是，非序列标识）不被修改，我们要求每个新单元包含作者所知的最后一个球的哈希值（这是从最后一个稳定单元建立的球，并且它位于 MC 上）。这样，最后一个球将受到作者签名的保护。之后，证人将（直接或间接）计入新单元本身。

如果不曾拥有完整 byteball 数据库的人想知道一个特定的单元是否是连续的，那么，他得给予我们他所信任的具有诚实行为的一系列见证人名单，然后，我们将会创建一条最近的包含之前提到的多数见证人的单元链，然后从链条的最旧单元读取最后一个球，并使用球来构建一个哈希树，哈希树的顶部包含最后一个球，并且在树下面的某处包含所请求的单元。该哈希树类似于非常高的 Merkle 树，其中每个节点都有额外的数据。哈希树可以使用 skiplist（跳跃表）进行优化。

参考最后一个球也让用户看到其他人对 MC 上稳定点的看法，并将其与自己的愿景进行对比。

我们还要求最后一个球不要早于每个父母单元的最后一个球。这样确保最后一个球要么沿 MC 前进，要么停留在相同位置，而不会退后。

为了进一步降低妨害者的自由度，我们添加了一个要求：

单元的见证名单必须与位于 MC 单元最后部分的每个单元和最后一个球的单元之间的证明名单相一致。此要求确保在尝试其他改变之前，证人名单的所有更改首先达到稳定点。否则，攻击者可能会将明显已被修改的证人名单注入 MC 上，并

停止从新证人的地址发布。攻击者可能会在 MC 上插入大量修改的证人列表，并阻止新证人发布地址。在这种情况下，稳定点将无法超越攻击者证人所占据的范围。

当代单元的所有证人名单大多相似的要求意味着所有用户对于当前可以信任的社区作为灯塔的用户几乎有相似的看法。所有用户对当前可以充当社区灯塔的人拥有相似的观点。这类似于生物学，同一物种的生物体必须具有大致相同的基因。证人名单的小差异允许进行仍然保持系统的完整性的变化发展。

11.见证列表单元

预计许多用户希望准确地使用相同的见证列表。 在这种情况下，为了节省空间，它们不会列出 12 个所有见证人的地址。相反，他们会参考另一个先前的单元，这个单元会明确地列出这些证人。从参考单元的角度来说，见证列表单元必须是稳定的，即它必须包括在最后一个球的单元中。

12.单元结构

这是一个单元例子：

{

version: '1.0',

alt: '1',

```
messages: [ {
      app: 'payment',
      payload_location: 'inline',
      payload_hash:
      'AegecfpDzh8xvdyIABdynrcP6CTd4Pt42gvRiv0Ftjg=',
      payload: {
            inputs: [{
                  unit:
                  '7yctnKyuAk5P+mFgFQDdDLza88nkceXYjsTs4e3doQA=',
                  message_index: 0,
                  output_index: 1
            } ],
            outputs: [
                  { address: 'DJ6LV5GPCLMGRW7ZB55IVGJRPDJPOQU6',
                  amount: 208 },
                  { address: 'Z36JFFX2AH7X5JQ2V2C6AQUUOWFESKZ2',
                  amount: 3505 }
            ]
      }
} ],
authors: [ {
      address: 'DJ6LV5GPCLMGRW7ZB55IVGJRPDJPOQU6',
      authentifiers: {
            r:
            '3eQPIFiPVLRwBwEzxUR5thqn+zlFfLXUrzAmgemAqOk35UvDpa4h
            79Fd6TbPbGfb8VMiJzqdNGHCKyAjl786mw=='
      }
} ],
parent_units: [
      'B63mnJ4yNNAE+6J+L6AhQ3EY7EO1Lj7QmAM9PS8X0pg=',
      'D6O1/D9L8vCMhv+8f70JecF93UoLKDp3e2+b92Yh2mI=',
      'ZxqzWP6q6hDNF50Wax8HUK212lH/KSIRdW5a6T9h3DM='
],
last_ball: '8S2ya9lULt5abF1Z4lIJ4x5zYY9MtEALCl+jPDLsnsw=',
last_ball_unit: 'bhdxFqVUut6V3N2D6Tyt+/YD6X0W+QnC95dMcJJWdtw=',
witness_list_unit:
'f252ZI2MN3xu8wFJ+LktVDGsay2Udzi/AUauE9ZaifY='
```

这里：

•version（版本）是协议版本号。将根据这个版本的协议解释该单元；

•alt 是替代货币的标识符，稍后我们将会讨论;

•messages（消息）是包含实际数据的一个或多个消息的数组;

o app 是一种信息类型，比如说“payment”代表支付，“text”代表任意文本信息等等

o payload_location 说明在哪里找到消息的有效载荷。 如果有效载荷包括在消息中，则可以是“内联”，如果在互联网地址可用有效载荷，则是“uri”，如果

未发布有效载荷，则是“无”，被私有地存储和/或共享，并且 payload_hash 用于证明它存在于特定的时间；

o payload_hash 是 base64 编码中的有效载荷散列;

o payload 就是真实的荷载量（因为这是这个例子里的“内联”）。 有效载荷结构是特定的应用程序。付款描述如下：

. 输入代表了一串由支付（指令）消耗的输入货币。输入货币所有者必须在单元的签名者（作者）之中;

. 单元是指生产货币的哈希单元。单元必须计入在最后一个球的单元才可消费;

•message_index（消息索引）是输入单元的消息数组的索引。它表明生产货币的消息;

. •output_index 是指进入输入单元 message_index 消息的输出数组的索引。它表示货币产生的输出;

. 输出是一组输出，表明谁收到钱；

. •address（地址）是指收件人的 Byteball 地址;

. •金额是他收到的金额;

• •作者是指创建和签署本单元的作者数组。所有输入货币必须属于作者;

• o 地址是作者的 Byteball 地址;

• o 认证是证明作者真实性的数据结构。最常见的是 ECDSA（椭圆曲线数字签名算法）签名;

•parent_units 是父母单元的散列数组。它必须按照字母顺序排列;

•last_ball 和 last_ball_unit 分别是最后一个球及其单元的哈希值。

•witness_list_unit 是可以找到见证列表的单元的哈希值。

所有哈希值都采用 base64 编码。

请注意，在单元结构中没有时间戳字段。在 Byteball 中，没有依赖于系统时间的协议规则。它根本不需要，因为只要依靠事件的顺序就足够了。

当时间戳从节点转发到节点时，仍然被添加到单元。然而，这只是建议性的，并且是由轻钱包用来在钱包中显示一个单元被生成的适当时间，这可能与单元被接收的时间存在显著的差异，因为轻钱包可能会长时间离线。

13.佣金

如前所述，存储单元的成本是以字节大小为单位的。 该佣金分为两部分：标头佣金和有效载荷佣金。有效载荷佣金等同于消息的大小; 标头佣金是其他一切的大小。两种类型的佣金是不同的分配。

标头佣金转到其中一个将付款人单元作为父母的未来单元中。 只有在付款人单元的 MCI 和下一个 MCI 变得稳定之后才选择接收者。为了确定接收者，我们选取那些 MCI 等于或大于付款人 MCI 的子单元。这些子单元的每个哈希值与位于下一个 MCI 上（相对于付款人）的单元的哈希值相连接，并且具有最小哈希值（十六进制）的子单元赢得标头佣金。

设计下一个 MC 单元的哈希是为了引入不可预测性（下一个 MC 单元不是预先知道的），并且没有任何尝试或提高接收佣金的机会或通过使用自己的单元哈希来实现无效。同时，将候选人限定为那些 MCI 不大于 1 的人，这好于 MCI 不大于 1 的付款人。鼓励选择最近的单元作为父母。这对于保持 DAG 尽可能的精确是有用的。

我们只支付标头佣金，而不是将全部佣金给那些很快选择我们单元作为父母的人，原因如下：如果我们支付了整个佣金，我们会激励滥用行为：将一个数据分成几个块，并构建一个自身单元的长链来存储每单元的一个块。在前一个单元支付的所有佣金，将立即由下一个单元的同一用户收集起来。由于我们只支付标头佣金，这样的行为是无利可图的，因为要生产链上的额外元素，必须花费额外的标头佣金-大致等于收入的一个费用。我们使用剩余（有效载荷）佣金来激励对于保持网络健康很重要的活动。

有效载荷佣金转给证人。为了激励证人频繁活动，我们将有效载荷佣金平均分配给所有的证人，在支付单元后他们迅速发布 100 个 MC 索引（发布速度越快，该单位变得越稳定）。如果在此间隔内，12 名证人都发布索引，则每个人收到 1 / 12 的有效载荷佣金。如果只有一个证人发布，他会收到所有的有效载荷佣金。在特定情况下，没有证人在间隔内发布，他们都会收到 1/12 的有效载荷佣金。如果除法产生小数，则根据数学规则四舍五入。由于四舍五入，支付给证人的总佣金可能不等于从该单元的作者中收到的总有效载荷佣金，所以总货币供应量也会略有变化。显然，仅在 MCI + 100 变得稳定之后才分发，其中 MCI 是付费单元的 MCI。

要花费获得的标头佣金或证人佣金，使用以下输入：

```
inputs: [
        {
                type: "headers_commission",
                from_main_chain_index: 123,
                to_main_chain_index: 196
        },
        {
                type: "witnessing",
                from_main_chain_index: 60,
                to_main_chain_index: 142
        },
        …
]
```

这样的输入扫描了由作者从佣金支付单元中赚取的所有标头佣金或证人佣金，而佣金支付单元从主链索引 from_main_chain_index 和 to_main_chain_index 之间发出。

当一个由多个作者签名的单元获得标头佣金时，到目前为止，如何在作者之间分割佣金尚未明确。 要删除模糊性，由多个作者签名的每个单元都必须包含一个描述收入分成比例的数据结构：

```
unit: {
        …
        earned_headers_commission_recipients: [
                {address: "ADDRESS1", earned_headers_commission_share: 30},
                {address: "ADDRESS2", earned_headers_commission_share: 70}
        ],
        …
}
```

收到佣金的地址不需要与作者地址相同 - 佣金可以发送到任何地址。 即使该单元由单个作者签名，它也可以包括此字段，向其他地方重新发送标头佣金。

14.确认时间

确认时间是指单元进入数据库达到稳定性的时间。这取决于证人发布的频繁程度，因为为了达到稳定，在新增加的单元之后，我们需要在 MC 上积累足够的见证作者单元。为使确认期最小化，证人应该频繁地发布（通过佣金发放规则已经激励他们这么做了），但也不能太频繁。如果两个或两个以上的证人几乎同时发出它们的单元（比通常需要将新单元传播给其他证人更快），这可能导致由最佳父链接组成的树的不必要的分支，这将延迟稳定性。因此，当证人连接良好并快速在机器上运行，此时达到最佳确认时间，因此他们能够快速验证新单元。我们估计最佳确认时间约为 30 秒;这只有在新单元的流量足够大才能达到，因此证人从见证佣金中所获得的收入超过他们发布自己单元的费用。

尽管充分确认的时间是相当长的，信任其对等体以传递所有新单元而不进行过滤的节点可以合理确定，一旦一个单元被至少一个证人所包含，再加上一个典型的延迟时间已经过去(新单元从点到点的时间)，该单元很有可能达到终局，并被视为有效单元。即使之后出现双花（双重支付），在本单元之后它也有可能被排序。

15.分区风险

没有命令，Byteball 节点的网络永远不能被划分为持续运营的两个部分。即使在全球网络中断的情况下，例如亚大西洋大鼠切断了连接欧洲和美洲的电缆，至少断裂的其中一方将注意到它已经失去了大多数的见证人，这意味着它不能推进稳定点，而且没有人可以花费卡在 MC 不稳定部分的输出。即使有人试图发送双花，直到恢复连接前它将持续不稳定（因此无法识别）。断裂的另外一方碰巧是大多数见证人，将继续正常运营。

16.审查

在设计上，在 byteball 上已经无法修改或删除任何过去的记录。阻止任何特定类型的数据进入数据库也是相当困难的。

首先，数据本身是可以被隐藏的，并且只有实际发布到数据库的哈希值才能证明数据的存在。只有在哈希值被存储以及该单元被其他单元包含之后，才可能揭露数据，这使得它变得不可修改。

其次，即使当数据是开放时，将数据包含或不包含在数据库中的决定委托给可能（并且实际上被激励）以新单元作为父母的许多匿名用户。试图审查不良单元的人，不仅要避免直接包含该单元（作为父母），还要间接避免通过其他单元包含该单元。（这与比特币不同，矿工或采矿池可以直接过滤个别交易，此外，比特币用

户对谁成为矿工没有任何意见）。由于包含“违规”单元雪球的单元数，任何企图避免这种情况的都需要审查自己。只有大多数见证人才能有效地施加禁止内容规则-如果用户选择这样的见证人。

17. 选择见证人

依赖见证人是 Byteball 植根于现实世界的原因。 同时，使得 byteball 高度依赖于人们的决定。 健全的系统取决于用户负责地设置他们信任的见证人名单。这个过程不能安全地自动化，例如，如果大多数用户只是为了兼容，开始自动更新他们的见证列表来匹配最新观察的单元列表，用最自己单元冲击网络的攻击者很容易利用这个，来逐渐改变主要的见证名单并到其选择的东西。

虽然最大化的建议可以是“仅手动编辑见证列表”，这对于大多数用户来说是有负担的，但是对于管理见证列表更实际的方法是追踪并以某种方式平均几个“行业领袖”的见证人列表，他们关心网络健全或在不一定与 Byteball 相关的活动中已获得良好声誉。 其中一些人可能自身是见证人。与见证人列表不同，行业领袖列表不必兼容，并且无法频繁更新列表他们不会有任何直接的负面影响，例如找不到兼容的父母和发布新单元。 我们期望大多数用户使用最受欢迎并数量相对较小的钱包之一，该钱包将默认设置为遵循钱包供应商的见证人列表，而钱包供应商又可以看到其他主要用户的见证人列表。

见证人也有他们的见证人列表，并建议用户选择他们信任的见证人，以保持他们见证人列表代表普通用户的信念。这是非常重要的，因为未经大多数现有证人的批准，主要的证人列表无法改变。建议见证人和未来见证人公开宣布他们的见证人名单政策（例如遵循和平均其他声誉良好的用户的证人列表），以及用户根据此政策以及其他因素评估他们对工作的适应性。 任何违反声明的政策行为都会立即显

现，并可能触发更换见证人的行为。对政策的无理修改也是如此。 该政策约束证人，并使他遵守公众舆论，即使它反对见证人本人或他的朋友。

如前所述，我们的协议规则要求：

1.只有在见证列表不超过 1 个突变的父母中选择最优父亲

2.相对于最后球单元的见证人列表，应该不超过 1 个突变

3 相对于最后球单元的所有 MC 单元不稳定的证人名单，应该不超过 1 个突变

4. 只有在当前稳定点之后当前见证人发布足够的单元，稳定点才能提前。

这些规则旨在防止恶意和意外的分叉。 同时，它们意味着主要见证人列表的任何变动都必须是渐进的，每一步都必须得到大多数目前证人的批准。 首先单方面的变动需要达到稳定，并且得到大多数老证人的认可，然后才能进行另一次变动。如果社区突然决定需要立即替换两个见证人，则在一个变动进入 MC 之后，以上的规则 3 将阻止第二个变动，直到第一个变动达到稳定。

除了以上所有的建议，仍然可能的是，由于行业领导者的疏忽，被选的见证人后来形成卡特尔，并集体阻止替换他们中任何一个人的尝试，来保持他们从见证佣金中赚取的利润。如果他们这样做，该行为将是有目共睹的，因为他们的见证列表将保持不变，而大多数其他行业领导的见证列表将有一个突变（允许保持兼容的最大值）。

如果老证人不屈服于这样的压力，改变用户唯一的办法是“革命”–即在旧资产的某些方面上，开办新资产来继承所有余额，用户地址等，但是从新见证列表开始，添加了特殊的协议规则，用以分裂时处理这种不兼容的变动。为了与旧资产区分开来，他们会为'alt'字段（这是什么'alt'）分配一个新值，并在新资产下发行的所有单元中使用它。结果，用户将持有两个资产（旧的 alt =“1”，新的例如 alt =“2”），并且能够独立地花费。如果分裂合理，可能会抛弃旧资产，但当新资产

是正常时，分裂之前积累的所有数据是可获得的。由于协议几乎是相同的（除了处理分裂和变动 alt 的规则），在所有用户和商业设备上的软件将更容易地更新安装。

如果有人只是想创建新资产来尝试另一个协议规则，他也可以使用'alt'字段继承来自旧资产的一切，为用户转换为合适的币种，并从第一天开始有大量的用户结余。

18. Skiplist(跳表)

有些球含有跳表阵列，加快轻客户端证明的建设（见下文）。 只有那些直接在 MC 上的球有一个跳表，其 MC 指数可以被 10 整除，跳表列出最接近之前 MC 的球，其索引在末端具有相同或较少数量的零点。 例如，MCI 190 处的球具有参照 MCI 180 处的跳表。在 MCI 3000 处的球具有参照 MCI2990,2900 和 2000 处的跳表。

19.轻客户端

轻客户端不存储整个 Byteball 数据库。 相反，他们将感兴趣的一子集的数据下载下来，例如仅是在任一用户的地址上进行的支出或被资助的交易。

轻客户端连接到完整节点来下载他们感兴趣的单元。轻客户端向完整节点告知其信任的见证人列表(不一定是用于创建新单元的见证人)以及其自身地址的列表。完整节点搜索轻客户端感兴趣的单元，并按照以下方式为每个单元构建一个证明链：

1.沿着 MC 及时返回，直到满足大多数见证人的请求。 收集所有这些 MC 单元。

2.从这个集合中的最后一个单元（它也是最早的时间），读取最后一个球。

3.从这最后一个球开始，沿着 MC 及时返回，直到遇到有跳表的任何一个球。收集所有的球。

4.使用跳表，跳到更早引用跳表的球。这个球也有一个跳表，再次跳跃。 在跳表数组中有几个球，它们总是跳过最大的距离，所以我们首先加速跳跃 10 个索引，然后是 100，再是 1000 等等。

5.如果跳表的下一次跳跃将我们抛在目标球后面，通过更小距离的跳跃减速。最后，索引离开跳表，每次仅使用父母链沿着 MC 走。

这条链一开始就是证人授权的单元，从轻客户端的角度来看是可信的。 通过父母单元链接（同时累积见证人）或通过引用最后的球，或通过父母球链接或通过跳表链接来连接链上的所有元素。在链的结尾，我们有一个需要证明其存在性的单元。

20.多重签名

一个单元可以由多方签署。 在这种情况下，单元中的作者数组有两个或多个元素。

这是有用的，例如。 如果两个或多个当事人想签署一份合同（一份普通固定的合同，不是灵活的合同）。 他们将签署包含文本消息（app='text'）的同一单元。 他们不必在公共数据库中存储合同的全文，支付它 - 一个哈希就足够了（payload_location ='none'），并且各方可以私下存储文本。

多重签名的另一个应用是资产交换。假设用户 A 想要将资产 X 发送给用户 B，以换取资产 Y（本地货币“字节”也是资产 - 基本资产）。 然后，他们将组成一个包含两个付款消息的单元：一个付款将资产 X 从 A 发送到 B，另一个付款将资产 Y 从 B 发送到 A.他们都签署双重创作单元并发布它。这个交换是极其微小的 - 也

就是说，两个付款同时执行或两者都失败。 如果其中一笔付款出现双花，则整个单元将被视为无效，并且其他付款也被视为无效。

这种简单的结构允许用户直接交换资产，而不需要将他们的钱交给任何中心化交易所。

21.地址

用户由其地址来确认，交易输出发送到地址，并且像在比特币里，建议用户具有多个地址，并避免重复使用它们。 然而，在某些情况下，重复使用是正常的。 例如，见证人应该从同一地址来重复发布。

一个地址表示一个定义，它是一个布尔表达式（远类似于比特币脚本）。 当用户签名一个单元时，他还提供一组认证器（通常是 ECDSA 签名），当应用于定义时，为了证明该用户有权签名本单元，必须将其评估为真。 我们在 JSON 中写定义。 例如，将需要 ECDSA 签名来签名的地址进行定义： ["sig"，{"pubkey":"Ald9tkgiUZQQ1djpZgv2ez7xf1ZvYAsTLhudhvn0931w"}]

此定义指示地址的所有者在定义中（在 base64 编码中）具有给定其公共对等体的私钥，并且他将使用该私钥来签署所有小区。 如果在相应的认证器中给出的签名有效，则上述定义评估为真，否则为假。 除了验证器之外，对小区的所有数据计算签名。

给定一个定义对象，相应的地址只是初始定义对象的一个哈希值加上一个校验和。 添加校验和以避免输入错误。 然而，与一般的校验和设计不同，该校验和位不仅仅被附加到未校验数据的末尾。相反，它们被插入到数据内的多个位置。 这种设计使得在期望地址的字段中插入长串非法数据变得困难。该地址用 base32 编码。上述定义对应于地址 A2WWHN7755YZVMXCBLMFWRSLKSZJN3FU.

当资金存入改地址时，付款发送者知道并仅指定付款输出中的该地址（定义的校验和散列）。 该定义没有显示，除了所有者之外，它仍然是未知的，直到输出消耗完。

当用户从地址中发送他的第一单元时，他必须在作者数组中揭示其定义（为了使签名验证成为可能）：

```
unit: {
      …
      authors: [ {
            address: 'DJ6LV5GPCLMGRW7ZB55IVGJRPDJPOQU6',
            definition: [
                  "sig",
                  {"pubkey":"AsnvZ3w7N1lZGJ+P+bDZU0DgOwJcGJ51bjsWpEqfqB
                  g6"}
            ],
            authentifiers: {
                  r:
                  '3eQPIFiPVLRwBwEzxUR5thqn+zlFfLXUrzAmgemAqOk35UvDpa4h
                  79Fd6TbPbGfb8VMiJzqdNGHCKyAjl786mw=='
            }
      } ],
      …
}
```

如果用户从同一地址中发送第二个单元，他必须省略定义（在 Byteball 上，它已经被了解）。 只有在定义变得稳定之后，他才可以发送第二个单元，即显示定义的单元必须包括在第二单元的最后一个球单元中。

用户可以更新其地址的定义，同时保留旧地址。例如，要旋转链接到地址的私钥，用户需要发布一个包含消息的单元，例如：

```
unit: {
      …
      messages: [
            …
            {
                  app: "address_definition_change",
                  definition_chash: "I4Z7KFNIYTPHPJ5CA5OFC273JQFSZPOX"
            },
            …
      ],
      …
}
```

这里，definition_chash 表示新地址定义的哈希校验和（其在以后不被公开），并且单元本身必须由旧的私密签名。 该地址的下一个单元必须：

•将此 address_definition_change 单元包含在其最后一个球单元中，即它必须已经达到稳定;

•与地址上第一条消息相同的方式，在作者数组中显示新定义;

地址更改后，不再等同于其当前定义的哈希校验和。 相反，它保持等同于其初始定义的哈希校验和。

如果用户想要改变密钥（例如当迁移到新设备时）同时保持旧地址（例如，用户名），则定义变化是有用的，例如，如果此地址已经参与其他长期定义（见下文）。

21.1. 定义语法

21.1.1.逻辑运算符

定义可以包括“and”条件，例如：

```
["and", [
["sig", {pubkey: "one pubkey in base64"}],
["sig", {pubkey: "another pubkey in base64"}]
]]
```

为了签名交易，例如，手提电脑和智能手机需要来自两个独立设备的签名时，这是有用的。

“或”条件，如：

```
["and", [
["or", [
["sig", {pubkey: "laptop pubkey"}],
```

["sig", {pubkey: "tablet pubkey"}]

]],

["sig", {pubkey: "smartphone pubkey"}]

]]

定义可以要求最小数量的条件成为一个真的更大的集合，如 2-of-3 签名

["r of set", {

required: 2,

set: [

["sig", {pubkey: "laptop pubkey"}],

["sig", {pubkey: "smartphone pubkey"}],

["sig", {pubkey: "tablet pubkey"}]

]]

（“r”代表“需要”），其特征在于两个强制签名的安全性和可靠性，因此在其中一个私钥丢失的情况下，地址仍然可用，并且可以用于改变其定义和用新私钥替换丢失的第三个私钥。

此外，不同的条件可以给予不同的权重，这是最低的要求：

["weighted and", {

required: 50,

set: [

{weight: 40, value: ["sig", {pubkey: "CEO pubkey"}] },

{weight: 20, value: ["sig", {pubkey: "COO pubkey"}] },

{weight: 20, value: ["sig", {pubkey: "CFO pubkey"}] },

{weight: 20, value: ["sig", {pubkey: "CTO pubkey"}] }

]]

21.1.2.委派到其他地址

地址可以包含对另一个地址的引用：

["and", [

["address", "ADDRESS 1 IN BASE32"],

["address", "ADDRESS 2 IN BASE32"]

]]

它将签名委托给另一地址，并且对于构建共享控制地址（由若干用户控制的地址）是有用的。 这种语法使用户能够随时随意更改自己组件地址的定义，而无需打扰其他用户。

21.1.3. 签名和认证

在大多数情况下，定义将包括至少一个签名（直接或间接）：

["sig", {pubkey: "pubkey inbase64"}]

代替签名，定义可能需要提供哈希的前映像：

["hash",{"hash":"valueof sha256 hash in base64"}]

这可以用于交叉链交换算法[7]。 在这种情况下，进入的哈希前映像作为认证器之一。

默认签名算法是曲线 secp256k1 上的 ECDSA（与比特币相同）。最初，它是唯一支持的算法。在将来如果添加其他算法，则算法标识符将用于定义的相应部分，例如用于量子安全的 NTRU 算法：

["sig", {algo: "ntru",pubkey: "NTRU public key in base64"}]

当人们在与更常规的签名相结合时，多重签名定义允许对未经验证的签名方案进行安全实验。

在单元标头中的认证对象包含签名者或其他数据（例如哈希前映像）,它们在地址定义中由认证者子定义路径锁定。对于单字地址，例如

["sig", {pubkey: "pubkey inbase64"}]

路径只是“r”（r 代表根）。 如果另一个定义中包括认证者所要求的子定义（例如 and/or），则该路径通过索引扩展到包含自定义的数组中，并且路径组件由点定界。 例如，对于地址定义：

["and", [

["sig", {pubkey: "one pubkey in base64"}],

["sig", {pubkey: "another pubkey in base64"}]

]]

路径是“r.0”和“r.1”。 对于更深层的嵌套定义：

["and", [

["or", [

["sig", {pubkey: "laptop pubkey"}],

["sig", {pubkey: "tablet pubkey"}]

]],

["sig", {pubkey: "smartphone pubkey"}]

]]

路径是“r.0.0”，“r.0.1”和“r.1”。 当有可选签名（例如 2 的 3）时，路径告诉我们实际使用了哪些私钥。

21.1.4. 定义模板

一个定义也可以引用定义模板:

["definition template", [

"hash of unit where thetemplate was defined",

{param1: "value1",param2: "value2"}

]]

在模型中，参数指定要替换的变量值。模型需要在一个指定的消息模型 app=“definition-template”（和往常一样，使用稳定）中保存起来，模型本身是用于处理信息的，在模型中看起来是正常的定义，可能包括引用变量@param1 @param2。定义模型可以使代码被重新使用。在其他模型中也可能被引用。

21.1.5. 会签

要求附属定义放在另一个地址：

["cosigned by", "ANOTHER ADDRESS IN BASE32"]

21.1.6. 查询是否使用了地址

附属定义的其他可能要求：一个地址被看作是至少包含在最后一个球单位中的一个单位的作者：

["seen address", "ANOTHERADDRESS IN BASE32"]

21.1.7. 数据输入

一个非常有用的条件,可以用来查询预先存储在 byteball 的数据：

["in data feed", [

["ADDRESS1", "ADDRESS2", ...],

"data feed name",

"=",

"expected value"

如果至少有一个消息像“datafeed name”等于“expected value”在地址”ADDRESS1”，“ADDRESS2”..（预测）的数据之间，则该条件为真。数据库是一种看起来像这样消息类型：

unit: {

…

messages: [

…

{

app: "data_feed",

payload_location: "inline",

payload_hash: "hash of payload",

payload: {

"data feed name": "value",

"another data feed name": "value2",

…

}

},

…

],

…

}

数据字段可以设计包含预测的定义。如果两个或两个以上当事人信任指定实体（预测）提供的真实数据，他们可以建立一个根据不同预测权利发布的数据共享控制地址给当事人。例如，这个地址定义表示二进制选项：

["or", [

["and", [

["address", "ADDRESS 1"],

["in data feed", [["EXCHANGE ADDRESS"],"EURUSD", ">",

]],

["and", [

["address", "ADDRESS 2"],

["in data feed", [["TIMESTAMPER ADDRESS"],"datetime", ">",

最初，甲乙双方资金按此定义地址（删除任何信任的要求，他们利用多边签署和发送他们双方签字的合约）。如果通过交换地址交换的欧元/美元汇率超过 1.1500，甲方可以收回资金。如果在 2016 年 10 月 1 日以前和以后的任何日期截止时间未超过 1.1500，乙方可以从这个地址中转走全部资金。如果满足所有条件，且平衡地址仍然是非空的，甲乙双方可以把钱同时转回，两边花费将像往常一样处理。

比较运算符号可以是“=”、“！=”、“>”、“>=”、“><”，和“<=”。该数据库的消息必须像平常一样是最后一个单元球之前送达。为防止任何单个预测突然失效而增加风险的情形出现，必须提供多个数据地址。

另外一个例子，一个客户从一个商人买商品，但他不完全相信商人，要求在商人没有发货的情况下，钱自动回到客户手上。客户支付给公共地址的定义为：

["or", [

["and", [

["address", "MERCHANT ADDRESS"],

["in data feed", [["FEDEX ADDRESS"],"tracking", "=",

]],

["and", [

["address", "BUYER ADDRESS"],

["in data feed", [["TIMESTAMPER ADDRESS"],"datetime", ">",

定义取决于发布跟踪所有成功交付货物数量的 FedEx 预测。货物交付是商家收到货款的先决条件。如果不在指定日期前交付，客户可以把他的钱取回。

这个例子有点夸张，因为它要求 FedEx 标记每笔交易。

21.1.8. 可信的数据

一个更简单的方式来达到同样的目的，还有一个语法：

["in merkle", [

["ADDRESS1", "ADDRESS2", ...],

"data feed name",

"hash of expected value"

如果指定预期值的哈希是包含在任何一个从”地址 1”、“地址 2”...中输入的可信根值，则评估为真。使用这种定义方式，FedEx 只会从以前完成发布的所有货物交易中定期发布可信根。为了使用这个地址，商家会提供可信路径，证明指定的值确实是包含在相应的可信树。证明是可信的地址将会发送给使用者。

21.1.9. 自我检查

一个定义也可以包括单元本身查询。这附属定义

['has', {

what: 'input'|'output',

asset: 'assetID in base64 or "base" for bytes',

type: 'transfer'|'issue',

own_funds: true,

amount_at_least: 123,

amount_at_most: 123,

amount: 123,

address: 'INPUT OR OUTPUT ADDRESS IN BASE32'

对于所有可选择的选择程序，如果具有至少一个输入或输出（取决于“什么”的领域）单元，那么确定为真实。

类似的情况，“有一个”通过选择程序正确的输入或输出的要求。

分散交换可以通过“有”的程序实现。在此之前，我们讨论了通过多边签署实现资产交换。然而，多边签署并不包括任何价格协商机制。假定用户想买 1200 单位的其他资产，他愿意支付不超过 1000bytes。同时，他也不愿意一直在线等待卖家。他宁愿下一个交易指令等到出现匹配的卖家自动成交。他可以通过创建一个限制指令，发送 1000 个字节到通过以下定义的地址：

["or", [

["address", "USER ADDRESS"],

["and", [

["address", "EXCHANGE ADDRESS"],

["has", {

what: "output",

asset: "ID of alternative asset",

amount_at_least: 1200,

address: "USER ADDRESS"

第一种选择，只要用户 喜欢可以选择取回他的 bytes，从而取消订单。第二种选择，可选择花费资金交换的权利，提供在同一单元另一个输出指令至少支付 1200 其他资产到用户地址。交易公开地列在指令上，卖家会发现，构成交易资产单位，在交易过程中多边签署。

你也可以使用“has”条件的抵押贷款。假设借款人持有一些非流动性资产，需要一些 bytes（或其他流动资产）。借款人和贷款人可以共同签署一份合约。单位的一部分发送他需要的字节给借款人，该单元的其他部分锁住缺乏流动性的资产进入一个地址，该地址被定义为：

["or", [

["and", [

["address", "LENDER ADDRESS"],

["in data feed", [["TIMESTAMPER ADDRESS"],"datetime", ">",

]],

["and", [

["address", "BORROWER ADDRESS"],

["has", {

what: "output",

asset: "base",

amount: 10000,

address: "LENDER ADDRESS"

]],

["and", [

["address", "LENDER ADDRESS"],

["address", "BORROWER ADDRESS"]

第一种选择，如果贷款不能按时偿还，允许贷款人没收抵押品。第二选择，如果贷款人支付 10000bytes（约定的贷款金额包括利息）给借款人，允许借款人收回抵押品，第三种选择，如果当事人都同意，当事人可以修改条款。

以下要求也可以包含在一个附属定义内：

['has equal', {

equal_fields: ['address', 'amount'],

search_criteria: [

{what: 'output', asset: 'asset1', address: 'BASE32'},

{what: 'input', asset: 'asset2', type: 'issue', own_funds:

true, address: 'ANOTHERBASE32'}

如果有满足至少一对搜索条件的输入或输出（第一组滤波器的第一个元素被搜索；第二个由第二个），则确认为真实的；它们的一些字段相等。

类似的情形，“has oneequal”要求恰好有一个这样的条件。

另一个附属定义根据一定的标准对目标值比较输入或输出和筛选：

['sum', {

filter: {

what: 'input'|'output',

asset: 'asset or base',

type: 'transfer'|'issue',

own_funds: true,

address: 'ADDRESS IN BASE32'

},

at_least: 120,

at_most: 130,

equals: 123

21.1.10. 否定

任何不包括“信号”、“散”、“地址”、“连署”，或“可信”的情况都可以否定：

["not", ["in data feed", [["NOAAADDRESS"], "wind_speed", ">", "200"]]]

既然选择原始数据是合法的（在没有看到新的数据时），通常结合如上所述的负条件，截止时间是一种在特定日期的要求。

21.2. 一般要求

定义一个地址必须包含至少一个明确或隐含（如通过一个“地址”）“信号”。

为了避免消耗太多的资源进行验证，每个定义的操作总次数限制为 100 次，包括引用定义中的操作，如“地址”和“定义模板”。

这个数字是我们有的 byteball 的 9 个任意常数的一个，其他 8 个是：证人总数：12；允许变动的最大范围：1；一个得到 MC 认证指标的最大数量：100；一些原始数据为标题的大小：2；每个单位最大信息量：128；每个消息编号最大输入或输出的：128；每个单位最大数：16；货币供应总量：10 的 15 次方。相比之下，比特币至少有 17 个常数[8]，而以太坊单定义收费就有 30 个常数[9]。

请注意，上述定义语言是陈述性和完全布尔语句组成，使其更接近传统的法律合同语言。然而，在它的表达项目的权利，语言不接近任何以太坊智能合约语言。事实上，它不是简单的“HELLO WORLD”。这不是我们的目标。byteball 定义

语言不是简单的设计，而是综合的；相反，它被设计成尽可能多样化的，不仅程序员自己能理解的语言。简单明了的语法允许每个人在没有开发人员的帮助（“律师”对于智能合同的时代）下理解和撰写简单的定义，并使出现错误的几率最小。

22.简介

如果用户愿意，可以在 byteball 存储他们的文件。他们用这样的一个信息：

unit: {

...

messages: [

.

{

app: "profile",

payload_location: "inline",

payload_hash: "hash of payload",

payload: {

name: "Joe Average",

emails: ["joe@example.com", "joe@domain.com"],

twitter: "joe"

}

},

...

],

...

}

他们透露，自己的数据量和真实性都依赖于用户自己。为了保证任何特别有关用户的信息是真实，系统必须寻找证据来证明。

23. 认证

认证确认签发认证的用户（认证者）验证用户一些关于签发认证人（主角）的数据。认证是存储像以下这样的信息：

unit: {

…

messages: [

…

{

app: "attestation",

payload_location: "inline",

payload_hash: "hash of payload",

payload: {

address: "ADDRESS OF THE SUBJECT"

profile: {

name: "Joe Average",

emails: ["joe@example.com"]

}

}

},

…

],

…

}

信息包含在认证中，不需要与用户自己发布的信息一样。事实上，公布的文件可能根本不存在。

虚假广告荐证人的工作类似于现代的证书颁发机构验证对象的真实身份并证明一个特定的公共密钥（或 byteball 地址）确实属于一个人或组织。我们希望他们在 byteball 继续相同的活动和那些想证明自己真实世界和 byteball 联系的真实身份之间收费。目击者和潜在证人可能会希望得到一些证明来增加他们的信任。某些资产类型可能需要与资产交易的证明（见下文）。

认证的应用程序是必需的但主题的名字是不重要，可以省略的验证配置文件的名称或其他个人身份信息是可能的。证实文件可能不包括任何关于那个主题的有意义的信息，从而让他向大家除了证人匿名，证人将仍然保持有关主体的记录和揭示它们在某些情况下，如在证人的条款或者如果在法律要求的规定。

24. 资产

我们设计了一个数据库，允许任何数据不变的存储各类数据。所有的数据库，最有趣的存储在一个公共数据库是那些具有社会价值，即对一个或两个以上的用户是有价值的数据。其中一类是资产。资产可以归属于任何人，和不变性和事件的全序的性质，我们在 byteball 建立股权转让长链的有效性非常重要。资产 byteball 可以发行，转让，交换，和他们的行为类似于本国货币'bytes’ 。他们可以代表任

何有价值，例如，债券，股票，loyalty point，航线时间，商品，其他信用或者加密货币。

要定义一个新的资产，定义用户发送这样的消息：

unit: {

...

messages: [

...

{

app: "asset",

payload_location: "inline",

payload_hash: "hash of payload",

payload: {

cap: 1000000,

is_private: false,

is_transferrable: true,

auto_destroy: false,

fixed_denominations: false,

issued_by_definer_only: true,

cosigned_by_definer: false,

spender_name_attested: true,

attestors: [

"2QLYLKHMUG237QG36Z6AWLVH4KQ4MEY6",

"X5ZHWBYBF4TUYS35HU3ROVDQJC772ZMG"

}

},

…

],

…

}

Here:

•可发行的上限是最大金额。与预定义的本地货币字节比较，字节上限为 10 的 15 次方；

•个人的表示如果资产转让私下或公开（见下文）。字节是公开的；

•可转让的表示如果资产可以在不通过资产定义的第三方之间转让。如果不可转让的，定义必须是唯一的发件人或每个传输接收机。字节转换；

•自动销毁表示如果资产被送到字节定义者时被破坏。字节不自动销毁；

•fixed_denominations 表示如果资产可以是任何整数量派（任意数量）或只在固定的面值（如 1，2，5，10，20，等），这是纸币和硬币的情况下任意数量的字节；

•只由定义者发布表示如果资产可以只由定义者颁发。字节，整个货币供应量是在创世记单位出具；

•定义者认证表明如果每个转让必须签署通过资产的定义，规定资产是有用的。传输字节不必任何人签署；

•花费证明表明如果富豪已被证实为花。如果他碰巧收到资产但尚未证实，为了能使用这些资产，他必须通过一个上市的定义下的证人认证。这个要求对于可调节要资产也是有用的。字节不需要认证；

•证人是被资产定义者推荐的一系列证人地址（如果花销证明是真的）。这个名单可以被发送证人资产消息代替证人名单的定义者修改；

•教派（未显示在这个例子中，仅用于 fixed_denominations 资产）列出了所有可以发出的允许 denonmination 和各面额硬币的总数；

•转让条件是一个允许资产转让的条件。定义在语言与地址的定义相同，除了它不能引用任何需要 authentifier，如“信号”。默认情况下，除了其他字段定义的外，没有限制.；

•问题条件与交易条件一样，除了发行交易。

每个信息单位可以有不超过 1'资产 '。在资产被定义后，它是通过哈希单位那里定义的（因此每个 信息范围要求 1 资产单位）。

一个资产的转移看起来像一个字节的转移，不同的是，有一个额外的字段的资产 ID：

```
unit: {
...
messages: [
...
{
app: "payment",
payload_location: "inline",
payload_hash: "hash of payload",
payload: {
asset: "hash of unit where the asset was
defined",
```

inputs: [

{

unit: "hash of source unit",

message_index: 0,

output_index: 1

},

...

],

outputs: [

{

address: "BENEFICIARY ADDRESS",

amount: 12345

},

.....

任意数量的资产覆盖整个供应必须在单笔交易中发出的。特别是，所有字节都在发生单位发放。如果资产达到上限，这一系列问题序号必须是 1。如果是没有达到上限的，一系列不同问题序号编号必须是唯一的。

一项资产只定义一次，不能被修改后，只有证人名单可以修改。

这个资产代表资产的定义者。如果是发行人的债务，发行人证明或放弃匿名获得债权人的信任这是合理的预期。

然而最终用户免费使用或不使用一种资产，资产者可以涉及的资产交易施加任何要求。

通过组合不同的资产定义者可以设计出满足各种需求的资产，包括那些受监管的金融机构必须遵循。例如，通过要求每个传递联署签名，金融机构可以有效地否决了所有任何抵触法规或合同规则的款项。会签各支付之前，金融机构（也就是定义者和发行人）将会检查用户确实是它的客户，资金接受者也是一个客户，那个客户已经通过了所有认识你的客户（KYC）程序，使资金不被法院命令冻结，以及进行不断要求的任何其他检查改变法律，法规和内部规则，包括那些被定义后的资产。

24.1. 银行发行的资产

有被完全符合安全（并保证在所有转账熟悉确定性终结），银行可以发行，将本国货币和银行资产支持的（这是正确的审计和监控的中央银行），此类资产的任何业务的法律性质是与所有其他银行的钱相同，和大家都很熟悉。唯一新奇的是，存款和转账在 byteball 而不是银行的内部数据库是可追踪的。在 byteball 数据库跟踪有两个结果：

•（不太受欢迎的）所有的操作都是公开的，这是熟悉的比特币和减轻使用多个半匿名地址，其中只有银行知道背后的真实地址的人。另一个保护隐私是私人支付更有效的办法，这个我们以后会讨论；

•（一个好的）银行发行的资产可以在对等的方式交换的字节或其他资产，而不必信任任何第三方，如交易所。

这里的银行类似于纹波网关。

在上面的交换的情况下，交换的一段是从一个用户在一个银行发行的资产的另一个用户支付。如果用户是同一个银行的客户，这个过程很简单。当用户持有的账户在不同的银行，银行可以方便跨行转账给对方。假设用户 U1 想钱转移到用户 U2 的情况下，用户 U1 的开户银行 B1 和银行用户 U2 账户 B2。银行 B2 也在银行 B1 开立账户。U1 转账给银行 B2 在银行 B1 开立的账户，同时，B2 转账给用户

U2.所有这些交易必须自动的。三个参与者（U1,B1,B2）必须因此在单个单位签名，转账者 B1 的钱从 U1 到 B2，转账者 B2 的钱转给 U2.

最终的结果是，U1 下降他的平衡在 B1，U2 增加他的平衡在 B2 和 B2 增加了平衡在 B1。B1 在 B2 银行也将在一个相应的帐户，其余额将作为反向支付处理来自用户的用户 B1 B2 的相互义务（B1 生长。在 B2 和 B2 B1）。可以部分取消的银行相互签订交易，将等量的各自的发行人（很方便的把钱自动发送给发行方破坏）。可以通过传统的银行同业支付定期结算是不取消的。触发结算，以积极的净余额的银行将他的平衡至发卡银行，而由于没有启在同一事务中分散转移，这引发了一个传统法定货币支付发行人持有银行。

当有许多银行 ,建立与各同行的银行直接对应关系可能很麻烦。在这种情况下，银行同意中央对手方 C（大的会员银行或一个新的机构），只能完全通过中央银行统一结算。

从 U1 到 U2 相同的转移将由 3 个交易：

1.U1 寄钱到 C 在 B1 的帐户；

2.C 赚钱给 B2（或 C 破坏 B2 的钱被返回到 B2）；

3.B2 转账给 U2。

将 3 笔交易都是捆绑成一个单一的单位签字 U1，B1（为 U1 的交易需要担保人），C 和 B2。

24.2. 非金融资产

其他应用程序，不必要的金融可以直接用 byteball 资产。例如，可靠计划可以发布可靠点子作为资产，使用 byteball 已经存在的基础设施允许人们使用这些可靠的点子交易，包括点对点（如果程序规则允许）。对于可以跟踪 byteball 追踪资产的程序开发者也是一样的。

24.3. 债券

企业可以在 byteball 发行债券。法律结构问题与普通债券一样，唯一不同的是，存款将现在使用的 byteball 而不是内部数据库追踪债券所有权（类似于银行）。在 byteball 上使用债券直接交易，没有集中的交换。当银行的钱也在 byteball，即时付款交货（在这种情况下，fiat 支付）成为可能，没有交易对手风险和没有任何中央机构。对债券和支付的标注是同时方签署相同的单元进行传输。

债券，如果足够的流动性，也可以使用第三方作为支付手段。

当债券发行时，发行人和投资者将多边签署一个普通单位，将新发行的债券对投资者同时发送从投资者的借款人的信息（或其他资产用于购买债券，如银行发行的背书资产）。赎回时，他们签署的另一个多边单元逆转交换（最有可能的是，在不同的汇率），期间支付赎回债券的价格是票面价值，而价格是卖发行时在其有效期限内必须低于面值反映利息（假设为简单的零息债券）。债券的二手市场的价格低于票面价值，逐渐接近它。

经济增长中有很多项目融资、债券发行 byteball 投资将发行的比赎回的多。当经济放缓，所有债券的总供应减少，因为有较少的项目来融资，因此，如果他们积极使用作为支付手段，债券的总供给自我调节是非常重要的。

如果两家企业在 net-30 条款进行交易，买方和卖方有权证券化交易的信用期在 30 天。例如，买方可以发行 30 天债券和利用它们立即支付卖方。卖方可以等待 30 天要么通过赎回债券，要么使用债券作为支付其供应商的一种手段。在这种情况下，当他们成熟，它将供应商赎回债券。

24.4 商品债券

债券可以以单位发行，而不仅仅是以货币发行，例如，100 桶债券赋予其持有人在债券到期时接收 100 桶石油的权利，1KWh 的债券持有人有权获得 1kWh 的

电力。持有人还可以选择以到期日当前的价格接收 100 桶石油或 1kWh 的货币等价物。

这种债券实际在对冲风险上非常有用，考虑一个新的石油项目，在开始商业运营前需要多年时间和大量的资金投资，如果仅以本国的货币进行融资，则该项目可能永远不会被资助，因为待新工厂开始出售石油时，未来的石油的价格不确定，债权人必须考虑到价格过低的风险，这将导致借款人可能违约。债权人要将风险定价在利率里，这意味这利率变得太高，这个项目将不可能发生。

然后，如果项目运营商可以用桶借款，违约风险将大大降低。现在，该项目将可能按计划启动，并将有可能产生计划的石油量。因此，它将能够在规定的时间内生产和偿还所有借用的桶，还有其他的风险，但一个巨大风险，市场风险被移除。它被从借款人中删除，但转移到贷款人，贷款人必须考虑油价下跌，他们收入比投资者更少。从另一个方面，如果价格上涨，贷款人从价格差额中获得额外的利润（注意，通过借用桶，借款人放弃这种上升潜力），并且这是有投资者愿意在商品中占据位置。由于债券实在 Byteball 中交易的，贷款人只要他们喜欢可以轻松地卖掉它。不像油期货，其交易是一个零和博弈，商品债券的投资确实为行业带来了融资。此外，石油期货是一中短期工具，而商品债券允许买入和持有，这更适合长期投资者。

还有另一类潜在的贷款人，那些针对相反风险的人。例如，航空公司希望抑制油价价格的上涨，一种方法是通过购买债券期望与石油价格相关联的石油生产公司的商品债券。

以上适用于任何商品，例如，电，铁矿石，金，其他金属，作物等。

从借款人的角度来看，商品债券可以被认为是以今天的价格销售未来生产的一中方式，对于贷款人，它是以今天的价格购买未来的供应方式。

如果大部分经济运行于商品债券，即使没有政府的干预，杠杆周期自然平滑，因为在衰退期间商品价格自动降低债务。

24.5 基金

对于个人用户，可能难以跟踪市场上可购得的大量的债券，相反，他们宁愿选择投资于专业管理的基金，并持有大量多元化债券组合。基金发行自己的资产，跟踪基金投资组合的总价值。每当投资者购买新发行的基金资产时，基金将使用收益购买债券。当用户退出时，基金出售其持有的一些债券，并销毁由用户退回的基金发行的资产。基金资产没有上限，其总供应因投资者进入和退出而异。因基金持有的所有债券在 Byteball 上可见，所有其价值易于审计。由于比基础债券更加流动，基金的资产更有可能成为支付手段。

24.6 结算（金钱或财产的转让：契约）

银联可以使用资产进行银行间的结算，一些大型银行发行的法定资产只能被银联成员认证的用户使用，该法定资产由发行银行的储备金背书（保障），当一家小银行想与另一家的银行结算时，银行只需发送资产。接收银行可以使用该资产与其他银行结算，或向该资产发行银行兑换成法定货币。银行也可将美元兑换成欧元或其他类似法定货币。所有这些转让和交易是即时结算，他们是最终的和不可撤销。在全世界银行间金融电信学会（传统银行），银行间结算时交换的是有关支付信息，而实际货币转移是一个单独的步骤，在 Byteball，银行间的信息结算，即完成了信息和货币价值传输。

25.私人支付

到目前为止，我们只考虑了公开化发送的付款（支付），即他们包含内联的有效载荷，且对每人可见。Byteball 允许记录（公布）私人有效荷载，用户可以保持

有效荷载私有，但只记录其哈希值，以能够证明有效荷载存在于特定的时间。为了适用于支付，资金发送方还需通过专用通信信道将私人有效荷载发送给接收方。接受方将需要在 Byteball 查找有效荷载哈希值，确认他存在。然而，这还是不足够隐藏来自其他已经被删除 Byteball 节点的有效载荷内容的功能，验证相同的输出没有花费两次。要回复此功能，我们在该单元添加了额外的公共字段。这个字段被称为支出证明，它的结构如下：

它完全取决于所消耗的输出，因此，试图在次使用相同的输出将产生相同的支出证明；

它不会显示任何关于输出的支出；

很容易看出这种结构满足上诉的要求。

spend_proof = hash({

asset:payload.asset,

unit:input.unit,

message_index:input.message_index,

output_index:input.output_index,

address: src_output.address,

amount:src_output.amount,

blinding:src_output.blinding

这里，payload.asset 是私人转移的资产的 ID，输入指的是消耗一个输出 src-output 前的输入，私人输出应该有一个额外的称为“盲目”的字段，一个设计为随机字符串，使它不可能预知支出证明的消费输出（所有其他字段来自一个相当狭窄的可能值集合，可以在合理的时间范围内迭代）

以上支出证明适用于转让，问题如下：

spend_proof= hash({

asset:payload.asset,

address:"ISSUER ADDRESS",

serial_number:input.serial_number, // always 1 for capped assets

amount:input.amount, // issue amount

denomination:1 // always 1 for arbitrary-amounts payments

请注意，发行交易的支出证明不包括任何盲目因数，因此，有可能知道发行一枚货币，但这一枚货币的接受者任然被第三方隐藏。此外，对于转账交易，由于付款人知道致盲因数，他可以计算货币被使用时被公布的支付证明，这就意味着，他可以知道收款人何时支付货币，但他看不到收款人或新的盲目因数，因此，无法进一步跟踪货币。

支出证明添加到单位：

unit: {

...

spend_proofs:[{ spend_proof: "the above hash in base64",

address:"SPENDING ADDRESS" // only if multi-authored

},

...

],

...

}

因此，要发送私人付款，发送用户应该：

1 为每个输出添加随机盲目因子

2 不公布有效载荷，而是将其可以找到的有效荷载的哈希单元，私人发送给收款人

3 对于每个输入，将相应的支出证明添加到单元中

如果所有的验证器都看到从同一个地址发布的相同的支出证明，则应该拒绝该单元（前提是地址连续发送），收款人应检查：1、他私人收到的有效载荷是由付款人哈希到 payload-hash 发布到 Byteball 的；2、来自私人有效荷载输入的支出证明与单元中支出证明数据相匹配。

当接收到私人支付的用户想要支付，他必须将他已经接受的私人有效荷载转发给新的收款人，因此新的收款人可以回到资产发行节点，验证整个所有权转移（历史），链的长度将随着每次转移而增长。

请注意，根据我们迄今为止所考虑的付款格式，每个单元可以合并先前几个单元的输出数据并产生若干个新的输出单元数据，每个先前单元取决与几个更早的单元，并且每个输出单元随后将被分割成若干新的输出，因此，未来单元的数量像初始单位的“血液细胞”随时间呈指数增长。相反，对单元输入有贡献的原始数据与历史上的步数呈指数增长。为了避免这种数据快速增长，我们需要限制货币的可分割性，且这证明将具有固定面额属性的资产类型设置为“true”是非常有用的。

26 固定面值资产

固定面额资产作为一组不可分割的货币，非常类似与所有人都熟悉的铸造硬币和纸币。每个硬币的面额必须是一组允许面额中的一种，其选择的面额在实际应用中符合最高精度和最小数量的硬币，大多数现代货币体系的面额遵循 1-2-5 模式，

即 1,2,5,10,20,50,100,200,500 等。这种模式也被推荐用于 Byteball 的固定面额资产。

货币最初被分组为包，类似于纸币包，包可以分成更小的子包或单个货币，但不重新合并，这意味着每次传输只有一个输入（因为不允许合并），并且输出量必须是货币面额的倍数（因为面额是最小的不可分割的量）。

每次交易，发行或转移，处理只有一个面额的货币，它不能同时发行或转移不同面额的货币（但每个存储单元可以包括多个这样的交易），固定面额交易和任意量资产交易具有相同的格式，不同不同之处在于仅允许一个输入，该金额必须是一个面额的倍数，并且添加面额字段。

payload: {

asset: "hash of unit where the asset wasdefined",

denomination: 100,

inputs: [// exactly one input

{

type:"issue",

amount:1000000,

serial_number:1, // always 1 for capped assets

address:"ISSUER ADDRESS" // only when multi-authored

}

],

outputs: [

{

address: "BENEFICIARYADDRESS",

amount: 800 // multiple of 100

},

{

address: "CHANGE ADDRESS",

amount: 999200 // multiple of 100

}

]

}

如果资产有上限，则每个面额的整个供应必须在单个交易中发出，因此，如果资产具有 16 个面额，需要 16 笔交易才能完全发放资产。如果资产没有上限，则相同地址的相同面额的不同发行序列号必须是唯一的。

如果需要发放或转移几个货币（通常是这种情况），付款人在同一单元中包含几个这样的信息。对于转移，货币通过单元，消息索引，输入索引来确认该货币由先前所有者转移到当前所有者。

对于私人支付 ,有效荷载单独地 ,并隐藏在除了收款人的所有输出者的接收者。

payload: {

asset: "hashof unit where the asset was defined",

denomination:200,

inputs: [{

unit:"hash of source unit",

message_index: 2,

output_index: 0

}],

outputs: [

{

output_hash: "hash of hidden partof output that

includes address and blindingfactor",

amount: 800

},

…

]

}

输出中开源的信息允许接收者验证所有输出的总和与输入匹配，为收款人制定的单一输出显示如下：

output: {

address:"BENEFICIARY ADDRESS",

blinding:"some random string"

}

这使得收款人能够验证 output-hash ,同时当他决定支付时构成未来的支付证明。

在 Byteball，我们有一个私人固定面额资产 blackbyte 定义这些属性。

cap: 2,111,100,000,000,000,

is_private:true,

is_transferrable:true,

auto_destroy:false,

fixed_denominations:true,

issued_by_definer_only:true,

cosigned_by_definer:false,

spender_name_attested:false,

denominations:[

{denomination:1, count_coins: 10,000,000,000},

{denomination:2, count_coins: 20,000,000,000},

{denomination:5, count_coins: 10,000,000,000},

{denomination:10, count_coins: 10,000,000,000},

{denomination:20, count_coins: 20,000,000,000},

{denomination:50, count_coins: 10,000,000,000},

{denomination:100, count_coins: 10,000,000,000},

{denomination:200, count_coins: 20,000,000,000},

{denomination:500, count_coins: 10,000,000,000},

{denomination:1000, count_coins: 10,000,000,000},

{denomination:2000, count_coins: 20,000,000,000},

{denomination:5000, count_coins: 10,000,000,000},

{denomination:10000, count_coins: 10,000,000,000},

{denomination:20000, count_coins: 20,000,000,000},

{denomination:50000, count_coins: 10,000,000,000},

{denomination:100000, count_coins: 10,000,000,000}

]

}

请注意我们需要面额为 2 的货币发行双倍的数量，因为我们需要更频繁的使用他们，例如，数据 4（2+2）和 9（5+2+2）都需要 2 个面额为 2 的货币进行相加。

用于支付证明的传输和私人不可分割资产的问题在任意数量的资产除了发行面额不一定是 1 以外是完全相同的。

与可分割付款不同，每个固定面额的货币不会与其他货币合并，因此，当货币私人转移时，货币历史数据随时间线性增长而非指数增长，仍是可控的。（假定计算资源，如存储，宽带和 CPU 功率在可预见的未来继续成指数级增长）

随着历史的发展，私人有效荷载在货币所有者的第三方中公开，如前所述，增长相当缓慢，私人有效荷载的价值相对于对方随时间而减少。然而，应该记住每天发送和接收大量的商家和交易所的付款，将积累非常大的历史数据。因此，应避免地址重用，即使是私人支付。

请注意在某些情况下，第三方可以推断出重要的信息，即使来自私人支付，例如，在大多数数据包被分割成单个货币后，当用户在同一个单元发送大量的个人支付消息，由于发送的金额显著大于货币最大面额，可能发送了多个最大面额的货币，因此观察者可能认为用户发送最大面额的货币，由此，观察者可能推断出转移的大致数量（但是其实没有），为了避免泄露这样的信息，建议大量的资金分散在多个地址上，并以单独的单元发送。

我们选择的支付证明方法不是唯一的方法。为了证明接收者收到的钱没被发送者发送给其他人，付款人可以向收款人发送曾经从他的地址发送的所有私人有效荷载。然后收款人可以检查并验证没有双重支出。我们选择不采取这种方式，因为这会涉及不必要的隐私泄露，增加了客户端代码的复杂性。相反我们选择稍微增加空间使用，但使验证更简单。

27 文本

运行“text”消息类型可以存储任意文本。

unit: {

… messages: [

…

{

app:"text",

payload_location:"inline",

payload_hash:"hash of payload",

payload: "any text"

},

…

],

…

}

对文本的解释取决于作者和需要查看数据的合作伙伴，Byteball 节点除了检查它是一个字符串以外，并不验证它。可以使用此消息的类型，例如，发送不可信的微博。有效载荷可以是私有的，并且是有用的，例如，用于存储用户的知识产权的哈希，或用于存储仅少数方需要知道的合同文本的哈希。

28 任意结构化数据

运行“date”消息类型可以存储任意结构化数据。

unit: {

```
...
 messages: [
 ...
 {
app:"data",
payload_location:"inline",
payload_hash: "hash of payload",
payload: {
key:"value",
another_key: {
subkey:"other value",
another_subkey: 232
}
 }
 },
 ...
 ],
 ...
 }
```

这个数据的解释取决于作者和他的需要查看数据的合作伙伴，Byteball 节点除了检查它是一个对象，并不验证它。例如，理解消息类型节点的一个子集被用于发布以太坊代码，但请记住，即使以太坊规则对代码是无效的，消息类型也不能拒绝该单元。

像“付款”和“文本”，“数据”消息是可以被私有的，在这种情况下只存储哈希值。继续我们的以太坊例子，如果相应的支付证明在必要的地方设计时，以太坊合约可以私下运行。

29 投票

任何人可以通过发送带有 app=“poll”的消息来设置投票：

```
unit: {
...
messages: [
...
{
app:"poll",
payload_location:"inline",
payload_hash: "hash of payload",
payload: {
question:"Should the United Kingdom remain a member of the European Union or leavethe European Union?",
choices:["Leave", "Remain"] }
},
...
],
...
}
```

用户发送“投票”消息，投出选票：

unit: {

…

messages: [

…

{

app:"vote",

payload_location:"inline",

payload_hash: "hash of payload",

payload: {

unit: "hash of the unit where the pollwas defined",

choice:"Leave"

}

},

…

],

…

确定谁具有投票资格取决于投票组织者，Byteball 不强制任何规定除了选项在允许的范围内，例如，组织者可以接受来自验证的用户或来自预定的白名单用户的投票，不合格的投票任然会被记录，但应由组织者在计票时排除。

投票权重和结果解释也取决于投票的组织者，如果用户通过他们的余额投票，他们可以将余额转移到另一个地址然后在次投票，这种投票应当被妥善处理。

30 私人消息

对于私人付款，用户需要一个安全的途径传送私人的有效荷载到对方，用户或者他们的设备也需要进行通信以组合多信号地址的签名。

由于我们不能期望用户的设备一直在线并且容易连接，我们需要一个始终在线的、易于访问、能够临时存储任何发送到用户设备数据的存储转发中介。

在 Byteball 中，这样的中介被成为集线器（hub），其作用类似于电子邮件。中枢是提供存储和转发私人消息到与之连接设备服务的 Byteball 节点，可以有许多的集线器。运行钱包代码的每个设备订阅其选择的集线器，并可以经该集线器到达。选择的家庭集线器（home hub）可以随时更改。每个设备都有一个对设备唯一的永久性私钥。相应的公钥的哈希值（更准确的说，基于该公钥的单一信号哈希值）被称为设备的地址，并且以基于 32 数像支付地址。完整设备地址，包括当前的中枢名称，可以写为 DEVICEADDRESSINBASE32@hubdomainname.com，如果设备转移到另一个集线器，@之前的完整地址保持不变，与电子邮件不同，名称不能是“已采取”。

每个设备使用 websockets 连接其家庭集线器。设备与集线器保持连接，如果新消息一到达，集线器向连接的设备立即发送新消息。集线器不保留设备成功发送的消息副本。与集线器的连接是通过 TLS 加密的。

当设备想要其他设备时发送内容时，它会连接到收件人的集线器并发送消息。与电子邮件不同，没有中转环节，发件人直接连接到收件人的集线器。设备之间的所有通信都是终端到终端的加密和数字签名，因此即使中心也无法查看或修改它。我们使用 ECDSA 进行签名，使用 ECDH+AES 进行加密。

在交换加密消息之前，设备必须进行配对，即获悉彼此的公钥。这可以通过各种方式实现，例如，通过扫描其中设备里公钥和中心域名编码的 QR 代码，通过电子邮件发送此信息，或通过点击安全网站上的 byteball:// 链接。

为了以上的安全 ,每个设备生产临时私钥 ,并将相应的公钥上传到其家庭中枢，之后，设备会不时地变换密钥，但保留上一个密钥的副本，以防有人向上一个密钥发送消息，集线器更换密钥。集线器仅保留设备订阅临时公钥的一个版本。发送设备按照以下步骤发送消息：

1.连接到收件人的集线器

2.从集线器接收接受者的当前临时公钥

3.生产自己的一次性临时密钥对

4.从接受者的临时公钥和自己的临时私钥中导出 ECDH 共享密钥

5.使用此共享秘钥对消息进行 AES 加密

6.添加自己的临时公钥

7.用自己的永久秘钥签署包裹

8.将其发送到集线器

接受者设备验证签名，使用对等体的临时公钥和自己拥有的临时私钥导出 ECDH 秘钥，并解密信息。

如果发送设备无法连接到收件人的集线器，它会将消息加密到接受者的永久密钥，并将加密的消息存储在本地以供将来重试。此加密的目的是避免周围不加密的信息。在连接到收件人集线器成功后，设备发送加密的信息，从而在次加密，因此此消息是双重加密。请注意，这不是单个加密不足，而是因为我们不想在不确定的时间重试连接时存储未加密的内容。

注意，通信实在设备之间，而不是用户，用户可以拥有多个设备，比如笔记本电脑，智能电话，平板电脑，并设置几个多站点地址，地址取决于存储于多个设备的钥匙。当用户需要签署交易时，他在他的一个设备上启动他即可。然后，此设备使用私人信息将部分签名交易发送到其他设备，收集所有签名并发布交易，存储在每个设备上的私钥不应该离开该设备。当用户在三分之二地址中替换他的一个设备是 ,他仅使用其他 2 个设备来改变定义地址 ,并用新设备的密钥替换旧设备的密钥。

私人消息还可以在设备之间发加密信息，这信息是严格对等的，不会进入 Byteball 数据库，并且可在在读取后安全的丢弃。

当用户以 blackbytes 或其他私人资产付费时，他们必须发送私人有效荷载，并且确保设备可以通信，他们需要知道对方的设备地址，才能彼此获悉对方的付款地址。一旦他们的设备建立通信，收款人可以通过聊天消息将他的付款地址发送到付款人。这样的支付方案很容易为每个传入的付款生成一个唯一的支付地址。商家可以运行聊天机器人用文本消息与用户通信，当用户准备支付时，机器人产生新的支付地址，并在聊天消息中将其发送给用户。

31 结论

我们提出了一中用于任意数据分散不变的存储系统，包括诸如金钱等社会价值数据。每一个新的数据单元隐含地确认了先前所有单元的存在。因为每个新单元也隐含地保护所有以前的单元不被修改和移除，使修订过去的记录类似 1984 的数据成为不可能。有一种内部货币，用于支付将数据包含在分散数据库中的费用。付款等于要存储的数据的大小，并且除了这种付款之外，对数据库的访问没有限制。在数据库上还可以发行其他的资产，并跟踪其所有权。当使用内部货币和其他资产跟踪付款时，通过选择由知名且信誉良好的用户所目睹的历史版本来解决双重支出。

最终清算是确定的。可以发行在任何规则下可转让的资产，允许监管机构发行符合监管要求的资产。同时，传输可以从隐藏在第三方发送私人的内容下，直接从付款人发送到收款人，发布支出证明，确保每个硬币值发送一次。

参考文献：

1. Quoted fromWikipedia https://en.wikipedia.org/wiki/Nineteen_EightyFour.

2. SatoshiNakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System,https://bitcoin.org/bitcoin.pdf, 2008.

3. SergioDemian Lerner. DagCoin,https://bitslog.files.wordpress.com/2015/09/dagcoin-v41.pdf, 2015.

4. Serguei Popov. The Tangle,http://iotatoken.com/IOTA_Whitepaper.pdf, 2016.

5. TomHolden.Transaction-Directed Acyclic Graphs, https://bitcointalk.org/index.php?topic=1504649.0,2016.

6. Linkedtimestamping, https://en.wikipedia.org/wiki/Linked_timestamping.

7. Atomiccross-chain trading, https://en.bitcoin.it/wiki/Atomic_crosschain_trading.

8.https://github.com/bitcoin/bitcoin 9. Gavin Wood. Ethereum: A SecureDecentralised Generalised Transaction Ledger,http://gavwood.com/Paper.pdf.