Tezos — 自我进化的加密账本
白皮书

L.M Goodman

2014 年 9 月 2 日

"我们的论点并不是简单的循环
逻辑，其背后有着独到之处。"
— Willard van Orman Quine

摘要

本白皮书向您介绍 Tezos, 一个通用的且能够自我进化的加密数字账本。Tezos 的最大优势是可以吸收任何一种基于区块链的账本好的方面，其将常规区块链上的各种操作以单纯的功能模块的方式实现。通过网络壳（Shell）利用这些操作处理网络层任务。比特币，以太坊，Cryptonote 等等都可以在 Tezos 内通过网络层接口实现，进而被表征。

更重要的是，Tezos 支持元数据升级：即可以通过自我修正代码进化协议。为此，Tezos 从一个种子协议开始定义一整套流程来让持币的用户来对代码进行修正，以及修正这套流程所必须的投票体系本身。这和哲学家 Peter Suber 的 Nomic[3] 博弈观点不谋而和，该观点的博弈构建主要围绕一整套内省规则。

除此之外, Tezos 的种子协议被放在一个纯粹的股权证明系统（POS）上，支持图灵完备的智能合约。Tezos 通过 OCaml 语言进行实现，该语言是一套功能强大的函数式编程语言，提供高速，非歧义语义和语法以及整个生态系统。所有的这一切让 Tezos 成为一个形式化正确性证明的很好的候选者。

以下的白皮书要求读者要对比特币协议的一定程度的了解和并理解基本的加密学知识。

目录

1 简介 **4**

2 自动进化的加密数字账本 **4**

2.1 数学表达 4

2.2 网络 Shell 5

2.2.1 时钟 5

2.2.2 链选择算法 5

2.2.3 网络层防御 6

2.3 功能表述 6

2.3.1 验证链 6

2.3.2 协议进化 8

2.3.3 RPC 8

3 种子协议 **9**

3.1 经济 9

3.1.1 币量 9

3.1.2 挖矿和签名奖励 9

3.1.3 丢失的币 10

3.1.4 修改规则 11

3.2 股权证明机制 12

3.2.1 概览 12

3.2.2 时钟 12

3.2.3 随机种子生成 13

3.2.4 Follow-the-coin 过程 13

3.2.5 挖矿 15

3.2.6 区块签名 15

3.2.7 链的权重 16

3.2.8 公开谴责机制 16

3.3 智能合约 16

3.3.1 合约类型 16

3.3.2 起源 17

3.3.3 交易 17

3.3.4 存储费用 18
3.3.5 代码 18
3.3.6 交易费 18

4 总结 **18**

1 简介

在白皮书的第一部分，我们将讨论抽象区块链的概念以及自动修正的加密账本的实现。在第二部分，我们将具体展开描述我们提出的种子协议。

2 自动进化的加密数字账本

一个区块链协议包含三层不同的协议:

- 网络协议，发现并广播交易。
- 交易协议，定义有效交易。
- 共识协议，形成针对唯一链的共识。

Tezos 实现了一个一般性的 Shell。该 Shell 是对交易协议和共识协议是透明的。我们把交易协议和共识协议统称为区块链协议。我们将首先给区块链一个数学描述，再描述 Tezos 中区块链的实现选择。

2.1 数学表达

一个区块链的协议本质上是一个全局状态并发突变的单子 (monadic) 的实现，以区块为单位操作全局状态。区块的自由类群 (free monid) 从创始状态开始形成一个树状结构。一个全局标准状态就是这个树的满足一定顺序的最小叶。

以下是其抽象的表达:

- 定义 $(\mathbf{S}, \leq)$ 为一个完全排序的，可以被计数的，可能状态的子集。
- 定义 $\oslash \notin \mathbf{S}$ 为一个特殊的无效的状态
- 定义 $\mathbf{B} \subset \mathbf{S}^{\mathbf{S} \cup \{\oslash\}}$ 为区块集合。这个有效的区块集合是 $\mathbf{B} \cap \mathbf{S}^{\mathbf{S}}$.

在 $\mathbf{S}$ 之上的顺序被延续，所以这里 $\forall s \in \mathbf{S}, \oslash < s$。这个顺序决定哪个叶在区块树上将被认可。$\mathbf{B}$ 中的区块可以看做是在这个状态上的操作。

总而言之，任何一个区块链协议[1]，不论是比特币，莱特币，点点币，以太坊，还是 Cryptonote，等等，都可以用以下元组来决定:

$$\left(\mathbf{S}, \leq, \oslash, \mathbf{B} \subset \mathbf{S}^{\mathbf{S} \cup \{\oslash\}}\right)$$

[1] GHOST 是一种可以基于树特性对叶子进行排序的方法。然而，这样的方式在理论和实践上都存在问题。与之相比，在主链添加挖矿证明几乎总是更好的解决方案。

这些区块链的网络层协议基本相同。其挖矿算法却发展迅速，激励着区块的创建。

在 Tezos 中，我们设计能够自省的区块链协议，让区块在协议层上运作。这样我们就能够递归地表示这组协议：

$$\mathcal{P} = \left\{\left(\mathbf{S}, \leq, \oslash, \mathbf{B} \subset \mathbf{S}^{(\mathbf{S}\times\mathcal{P})\cup\{\oslash\}}\right)\right\}$$

2.2 网络 Shell

仅有一个形式的数学表述并不足以让我们立刻建立一个区块树。我们还需要一个连接 Gossip 网络和协议的网络 Shell。

这个网络 Shell 通过维护客户端所知的最优链进而运作。它将从三种对象那里接受信息。前两个分别是交易和区块，确认有效后广播。第三个是协议，即用来修改现有协议的 OCaml 模块，对此我们稍后再进行详细描述。现在我们主要关注交易和区块。

网络 Shell 最艰巨的部分是保护节点免于遭受 DOS 攻击。

2.2.1 时钟

每个区块都有时间戳，这个时间戳只对网络外壳可视。如果一个区块是当前的，并且其时间戳在系统时间的几分钟之内，则该区块会被缓冲，否则将会被拒绝。这个协议设计必须能够容忍合理的时钟漂移，而且必须假设时间戳可能会被伪造。

2.2.2 链选择算法

该 Shell 维护一条单一的链，而不是一个完整的区块树。这个链只有在客户端获悉严格意义上的更优链时才会被覆盖。

从网络通信的角度，维护一个区块树会更加节省资源，但也会让网络更加容易被 DDOS 攻击，尤其是当一个攻击者产生大量的低得分但有效分叉的时候。

但仍然存在一种可能，一个节点谎报了链的得分，客户端可能会在处理大量的区块，发现和揭露该信息谎报,。随后这个欺骗节点将被忽略。

幸运的是，对于协议而言，低得分的链创建区块的等级越低。因此，客户端可能只需要查看一个弱分叉的几个区块，就可以判断它是不是某链的得分是不是欺骗。

2.2.3 网络层防御

此外，Shell 还具备防御性。它尝试连接不同 IP 端的对等节点，发现掉线的节点以及禁止恶意节点。

为防御某些 DOS 攻击，协议可提供 Shell 依赖区块大小和交易限制的环境。

2.3 功能表述

2.3.1 验证链

我们可以通过以下的 OCaml 类型有效捕捉几乎所有的抽象区块链结构的继承类。从定义区块头开始：

```
type raw_block_header = {
  pred: Block_hash.t;
  header: Bytes.t;
  operations: Operation_hash.t list;
  timestamp: float;
}
```

我们故意没有把区块头域强制类型化，以致于其能够表征任意内容。但我们有必要明确类型化 Shell 的相关操作。这些包括前区块的哈希，一个操作哈希的列表以及时间戳。在实践中，包含在一个区块内部的操作和区块一起在网络层上传播。操作本身可表征为任意的 blobs 结构.

```
type raw_operation = Bytes.t
```

Context 模块封装了一个基于磁盘的不可变化的键值对存储，在其帮助下表征状态。键值对结构十分通用，允许我们有效且广泛表征许多种状态。

```
module Context = sig
   type t
   type key = string list

   val get: t -> key -> Bytes.t option Lwt.t
   val set: t -> key -> Bytes.t -> t Lwt.t
   val del: t -> key -> t Lwt.t
   (*...*)
end
```

为避免磁盘操作的阻塞，函数使用异步单子 Lwt[4]。注意，这个操作在上下文中是纯函数式的：当 **set** 和 **del** 都返回一个新 **Context** 对象时，

get 操作使用 **option** 单子而非抛出异常。**Context** 对象模块将内存缓存和磁盘存储结合有效提供不可变化的存储环境。

现在我们能够定义任意区块链协议的模块类型：

```
type score = Bytes.t list
module type PROTOCOL = sig
  type operation
  val parse_block_header : raw_block_header -> block_header option
  val parse_operation :  Bytes.t -> operation option

  val apply :
    Context.t ->
    block_header option ->
    (Operation_hash.t * operation) list ->
    Context.t option Lwt.t

  val score : Context.t -> score Lwt.t
  (*...*)
end
```

不是和数学模型做状态比对，相反，我们利用 **score** 函数将 **Context** 对象映射到一个字节列表。字节列表首先按长度排列，其次依照字母顺序。类似于软件的版本化，这是一个相当通用的结构，在表示不同排序时有着广泛应用。

为什么不在协议模块中定义比较函数？首先表征一个完全排序函数的需求很难执行。这经常可以从得分映射上得到验证（在上一个区块哈希的基础上破坏连接）。其次，原则上我们需要有对比不同协议状态的能力。具体的协议修改规则可能让这个发生的可能性变得非常低，但是网络 Shell 对此并不知晓。

在决定中继操作或者添加区块至本地区块树状数据库之前，**parse_block_header** 和 **parse_operation** 操作将暴露给网络 Shell，并允许其将完全类型化的操作和区块送至协议层，还要检测这些操作和区块是否经过严格封装。

协议的核心是 apply 函数：

- 当传一个区块头部以及附属的操作列表时，它会计算上下文中做出的变化，返回一个修改过的副本。在内部，只有变化才会保存，和版本系统中一样，类似版本处理一样使用区块的哈希。

- 当只是传操作列表时，它将贪心地试图尽可能生效更多的操作。对于协议自身而言该函数不是必须的，但对于试图形成有效区块的矿工来

说却有极大的用处。

2.3.2 协议进化

Tezos 最强大的特性是它的协议自我修正能力，主要通过暴露给协议两个过程函数实现:

- **set_test_protocol** 函数，使用新协议替代测试网络中使用的协议(通常是持币者投票决定的协议)。
- **promote_test_protocol** 函数，用当前通过测试的协议替代目前正在运行的协议。

这些函数通过改变目前相关联的协议来转换 Context 对象。当下一个区块在链上产生，新的协议开始生效。

```
module Context = sig
   type t
   (*...*)
   val set_test_protocol: t -> Protocol_hash.t Lwt.t
   val promote_test_protocol: t -> Protocol_hash.t -> t Lwt.t
end
```

protocol_hash 函数是**.ml** 和**.mli** 文件 sha256 哈希的 tarball 打包。这些文件在运行中编译，有获取较少的标准函数库权限，但被装在沙箱内，可能无法调用系统函数。

这些函数通过协议的 **apply** 函数调用，返回新的 **Context** 对象。

很多条件可能会触发协议的修改。在最初的简单版本上，持币者可以通过直接投票进行协议修改，而之后更复杂的协议可以通过逐步投票而获得接受。例如，当一个持币者希望一个修改案被通过，他会被要求提供计算机可以验证的证据来证明他的提案将会尊重协议的某些特性。这是对协议修改合规性在算法上的有效检测。

2.3.3 RPC

为了简易化图形化工作，这个协议暴露了 JSONRPC 的 API。该 API 自身可以被描述为一个代表各种过程类型的 json 模式。通常涞说，诸如 **get_balance** 的函数都可以通过 RPC 来实现。

```
type service = {
  name : string list ;
  input : json_schema option ;
  output : json_schema option ;
  implementation : Context.t -> json -> json option Lwt.t
}
```

这个名字是一个字符串列表，其目的是允许过程中出现命名空间。输入和输出是通过可选择性使用 json 模式刻画。

注意这个指令被用来在一个指定的上下文中，而这个上下文通常是最高得分叶子节点的最近一个祖先。例如，在最高得分叶子节点之上查询上下文 6 个区块，将会显示具有六个确认的账本状态。

界面本身可以裁剪至特定的协议版本，或者是由 JSON 数据继承而来。

3 种子协议

Much like blockchains start from a genesis hash, Tezos starts with a seed protocol. This protocol can be amended to reflect virtually any blockchain based algorithm. 更像是区块链开始于一个创世哈希，Tezos 开始于一个种子协议。这个协议可以被修改来反映几乎任何一个基于区块链的算法。

3.1 经济

3.1.1 币量

Tezos 一开始就会有一百亿个币，这些币小数点后保留两位。一个币被称作 1tez，而最小的单位是分。我们也建议使用 ꜩ (\ua729, “Latin small letter tz”) 拉丁字母来代表一个 tez。这样的话，一分等于 0.01tez，等于百分之一 tez。

3.1.2 挖矿和签名奖励

原则 我们认为任何一个去中心化的货币要保证安全性都需要给予参与者金钱上的奖励。正如在我们的定位白皮书中提到的，仅仅依赖于交易费进行激励会受到公共地悲剧影响。在 Tezos 中，我们提供一个债券和现金奖励的二元激励体系。

债券是矿工所购买的一年期的安全存款。当出现双重签名的情况下，这些债券将会被收回。

一年以后，矿工除了债券以外还将获得另外的奖励，作为他们机会成本的补偿。债券和奖励的价值是系统安全的最主要保证，而它们的价值将仅占全部价值的一小部分。

债券的真正目的是减少奖励的总量，并利用人对损失的普遍抗拒心理来提高网络的安全。

细则 根据种子协议，每挖一个块将获得 512Tezos 币现金奖励以及需要 1536 币的债券。而每个区块签名将获得 $32\Delta T^{-1}$ 个 tez 的奖励。这里 ΔT 代表以分钟为单位的区块和先前区块之间签名的时间间隔。每个区块至多有 16 个签名，但签名不需要持有债券。

假设出块率为每分钟一个区块，那么有 8% 的最初货币供应量在第一年后应该以安全债券的形式被持有。

我们的奖励的计划被设定为 5.4% 的通货膨胀率。这个名义上的通货膨胀应该是中性的，它不会让有的人变富，而让其他人变穷。[2].

请注意这里的一年周期是根据区块的时间戳而不是区块的数量决定的，这是为了抵消矿工挖矿算力不同导致的不确定性。

展望 这个奖励机制将给矿工 33% 的债券回报。这个回报在初期必须要足够高，因为矿工和签名者需要共同持有一整年易波动的债券资产。

但是，随着 Tezos 的成熟，这个回报可以被逐渐降低到一个同期的主流利率水平，并长期维持低于 1% 的形式通胀率，但是这样做的合理性还需要进一步论证。

3.1.3 丢失的币

为了减少货币基数变化所带来的不确定性，所有的没有任何动作的地址和地址上所包含的币超过一年（由时间戳决定）都会被销毁。

[2] 与之相比，比特币的挖矿的通货膨胀让比特币的持有者整体变穷，而中央银行又让金融界变得有钱，而让储户变穷。

3.1.4 修改规则

协议修改的生效时间取决于选举周期，每个周期的长度为 $N = 2^{17} = 131\,072$ 个区块时间。考虑到区块间隔是一分钟，所以这个间隔是三个月。这个选举的周期将 $2^{15} = 32\,768$ 个区块分成四阶段。这个周期相对比较短，这主要是为了激励早期的协议的改善。随着未来的修改，这个周期最终将变长。选举结果的生效需要满足一定的法定人数。法定人数开始于 $Q = 80\%$，但会为了反映平均参与度而动态适应。当且仅当需要处理丢失的币时，这些才是必须的。

第一阶段 建议修改协议需要提交一个代表新协议的.ml 和.mli 文件的 tarball 打包的哈希。持币人可能会批准任意数量的所提交协议。这也就是批准投票，是一个强健的投票过程。

第二阶段 在第一阶段获得最多批准的修改案基础上进行投票。投票人可以投票反对或者弃权。投弃权的人也按法定人数计算。

第三阶段 如果达到法定人数 (包括明确的弃权)，并且修改案达到了 80% 的支持率，那么修改案就会被批准，并替代测试协议。否则它将被否决。假设法定人数达到值为 q，按如下方式更新最小法定人数 Q：

$$Q \leftarrow 0.8Q + 0.2q.$$

做出以上更新是为了避免因为丢失币的问题延缓投票过程。最小法定人数是达到前面选举法定人数以上的指数滑动平均。

第四阶段 假定修改案被批准，在第三阶段开始的时候开始测试网络运行该修改案。持币者将进行第二次投票来确认他们确实想要支持测试协议替代主协议。这也要求一定的法定人数，以及 80% 的绝大多数投票者支持。

我们特地选择了一个保守的方法做出修改决定。但是投票人可以自由地去选择自己认为对有利的修改提议，并来对这个修改案进行适度的放松或者收紧。

3.2 股权证明机制

3.2.1 概览

我们的 POS 机制集成了几个不同的观念，其中包括 Slasher[1]，chain-of-activity[2] 以及 proof-of-burn。以下的是对算法的简要概览。算法组件将在以下被详细地介绍。

区块由随机的持币人（矿工）挖出，并包括随机的持币人（签名人）提供的前一个块的多签。挖矿和签名都将获得一笔小奖励，但是也要求存一笔一年期的押金，如果期间出现双挖或者双签，那么这笔押金将被没收。

这个协议以2048个区块为周期运行。在每个周期的开始，矿工选择数字做成随机种子，提交至倒数第二个周期，并在最后一个周期展现出来。通过使用随机种子，“跟随币”策略分配给指定地址的下一个周期的挖矿权以及签名权，见下图1。

3.2.2 时钟

协议在区块之间增加了最小延迟。原则上，任何一个持币人都可能挖出块。但是，就一个特定的块而言，每个持币者都将受到一个随机的最低延期的制约。最高优先级的持币人很可能会在上一个块出现后一分钟后挖出下一个。优先级第二的持币者可能在两分钟后挖到下一个块，以此类推，优先级第三的会在三分钟后挖到下一个块。

这将保证在一个分叉中持有股权较少的持币人拥有较低的出块率。否则，一个针对 CPU 的 DDos 攻击将可能欺骗节点，导致其确认一个较长的自称高得分的链。

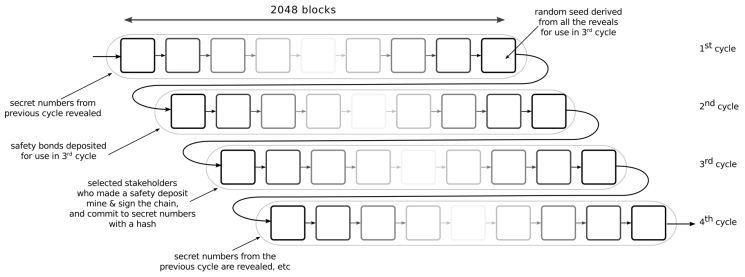

图 1: 四阶段 POS 机制

3.2.3 随机种子生成

每一个被挖的块都会携带一个由矿工选择的随机数的哈希值。这些数字必须在下一个周期的担保金没收时间前被公开。这个严厉的惩罚措施可以防止因矿工拒不提供随机数而有可能导致的对种子进行的熵攻击。

恶意的矿工在下一个周期会试图阻止该随机数被公开，但是因为多个随机数可能会在单个区块中被公开，这样的企图很难成功。

在一个周期中所有的被公开的数字都将合并到一个哈希列表中，并且这个种子将从根部通过使用scrypt 的密钥衍生函数得出。这个密钥的衍生需要经过调试，使得衍生种子所需的时间和一台普通桌面电脑确认区块所需平均时间百分之一的一小部分的量级相当。

3.2.4 Follow-the-coin 过程

为了随机选取持币者，我们引入一个跟随币机制。

原理 这个方法最初出现在比特币中，称为 follow-the-satoshi。该过程假设每一个锻造出来的聪单位的比特币都有唯一的序列号，聪按照创建时间隐式排序，通过区块链对其操作和追踪。当然，单个分不能够直接追踪。但是可以通过使用一些规则描述当输入集成起并通过多个输出花出时发生的情况。

最后, 算法持续追踪每一组和密钥相关联的间隔。每个间隔表示聪区间的范围。不幸的是，随着时间的延长，数据库变得越来越零碎，使得客户端膨胀。

币卷 通过建立大的由一万个 tez 组成的币卷，我们优化之前的算法。总共会有大约一百万个卷存在。数据库把每个卷地图映射到其当前的所有者。

每一个地址都带有一个特定卷的集合，以及一些零钱。当我们花掉只占一个卷一小部分的币时，整个卷就会被破坏，而它的序列号会发送至卷的 LIFO 队列，该队列有点像放置废弃物的场所。每笔交易都应该以最小化破坏卷的方式处理。每当一个地址有足够的币来组成一个卷，序列号将从队列中拉出，再次生成卷。

LIFO 队列优先级保证了在一个秘密分叉上挖矿的攻击者无法通过账户间混洗改变的方法，达到修改其拥有的币数的目的。

这样做会有一个很小的问题，那就是股份数被近似为卷值最接近的整数。但是相对于 follow-the-satoshi 方法，该方法在效率方面提供了很大的改善。

虽然币卷被数字化，这个方法并不能阻碍那些诸如零币那样的使用可替代性保持的协议。这些协议可以使用相同的 limbo 队列技术。

动机 上述流程和仅仅按照余额权重衡量选取随机地址的机制在功能上有很大的不同。

事实上，在一个秘密的分叉内，一个矿工可以试图通过提前控制随机种子的生成和赋予自身签名和挖币的权利。如果币卷是真正随机选择的，这种企图是很难成功，因为秘密的分叉不能够伪造某个币卷的所有权归属，而且必须要试图来造影（preimage）在种子上应用的哈希函数，进而实现签名和造币权利的分配。

事实上，在一个 $N = 2048$ 周期内，某个持有币卷中一小部分 f 的人将会获得平均值为 fN 的挖矿权，一旦接收到这个有效的部分，那么 f_0 的标准差将为：

$$\sqrt{\frac{1}{N}}\sqrt{\frac{1-f}{f}}$$

如果一个攻击者通过 W 个不同种子进行穷举式搜索，那么他的攻击的优势至多是

$$\left(\sqrt{\frac{2\log(W)}{N}}\sqrt{\frac{1-f}{f}}\right)fN$$

个区块。[3] 举个例子，一个控制了总量 $f = 10\%$ 币卷的攻击者，他可以预期在每个周期挖大约 205 个区块。假设他在试图控制种子的秘密分叉上有超过一万亿个哈希的算力，那么其可以分配给自己占总量区块 14.7% 的 302 个块。值得注意的是：

- 产生种子的哈希是通过一个复杂的密钥生成函数产生的，这让穷举式搜索变得不现实。

- 要想在挖矿中取得线性收益，攻击者将花费平方指数级增长的资源。

[3]这是在 W 正态分布变量的最大期望上的标准界限。

3.2.5 挖矿

随机种子被重复用来选择币卷。第一个被选中的卷将给予它的持有者一分钟后挖块的权利，这个第二个可以在两分钟以后，以此类推。

当一个持币者观察到种子并认识到他可以在下一个周期里挖掘一个高优先级的块的时候，他可以进行安全存币。

存在这样一个潜在问题，即没有持有人进行安全存币购买债券来挖一个特定的块。如果这种情况发生，那么在 16 分钟以后，这个块就可以在没有存款的情况下被挖出。

当购买债券的人不在挖矿时，购买债券的币将通过隐式的方式立即返还这些购买人。

3.2.6 区块签名

就这样我们构建了一个几乎完备可行的 POS 系统。但是一个问题是将一个链的权重定义为区块数量，这会带来自私挖矿的问题。

为此，我们引入了一个签名模式。当一个区块被挖出的时候，随机种子将随机分配 16 个签名权利给 16 个币卷。

持币者获得签名权利后一旦观察到正在被挖出的区块就会上传区块签名。这些签名会包括在下一个区块里，该工作由试图保证区块链中父子包含关系的矿工完成。

签名人获得的签名奖励与区块和其先祖区块之间的时间间隔成反比。

因此，签名人有很强的意愿尽快签名他们所认为在某一时刻产生的最优的区块。他们也有很强的动机达成签名哪个区块的共识，因为签名奖励只有在区块被写入区块链后才能发放。

如果优先级最高的块没有被挖出（可能是因为矿工没有在线），签名人会有因为存在矿工迟到的可能性而决定等待一些时间的动机。但是其它签名人可能会决定签名优先级最高的块，一个新块可能包含这些签名，让那些等待者一无所获，所以矿工不太可能采取等待的策略。

我们可以假设存在与之相反的情形，签名者过于担心竞争者抢先，而争相对所看到的第一个块进行签名。然而现实是这种情形其实对任何人没有什么好处。签名者没有理由认为会出现这种状况，更不用说去修改他们的程序代码来去这样做。虽然理论上存在一个恶意的持币者试图扰乱秩序的可能，但这样只会损害自己的利益，其他人也不会效仿。

3.2.7 链的权重

我们定义链的权重为签名数。

3.2.8 公开谴责机制

为了避免一个区块上的双挖以及双签问题, 一个矿工可以在他的区块中加入一个公开谴责机制。

该公开谴责机制采取双签名的形式。每一个造币的签名或者块的签名都签入区块高度，构成恶意行为的证据，这让不良行为很难隐藏。

尽管我们可以允许任何人来谴责不良行为，但没有人比矿工更适合来这样做。事实上，一个矿工可以简单地复制不良行为的证据并将其作为自己的发现转达给其他人[4]。

而一旦发现有双挖或者双签，那么其债券将被没收。

3.3 智能合约

3.3.1 合约类型

和比特币的 unspent 输出不同，Tezos 使用有状态的账户。当这些账户规定了可被执行的代码时，它们也就成了广泛意义上的合约。账本本身也是一种合约，只是没有可执行的代码。

每一个合约都有一个管理者，对于账户而言，这个管理者就是它的拥有者。如果这个合约标识为可以被花费的，那也意味着管理者可以花费与这个合约相关联的资金。此外，每一个合约都可能会规定一个公钥的哈希用来签署或者挖 POS 协议内的区块。私钥可由或不由管理者控制。

一个合约可以正式表示为:

```
type contract = {
  counter: int; (* counter to prevent repeat attacks *)
  manager: id; (* hash of the contract's manager public key *)
  balance: Int64.t; (* balance held *)
  signer: id option; (* id of the signer *)
  code: opcode list; (* contract code as a list of opcodes *)
  storage: data list; (* storage of the contract *)
  spendable: bool; (* may the money be spent by the manager? *)
  delegatable: bool; (* may the manager change the signing key? *)
}
```

[4]零知识证明允许任何人从谴责不良行为获益，但获益多少并不十分明确。

一个合约的句柄是初始内容的哈希值。试图创建的合约的哈希不能与已经存在的相同，否则将不会被包含在一个有效区块内。

该数据表示为一个 union 类型。

```
type data =
  | STRING of string
  | INT of int
```

这里INT 是一个有符号的 64 位整数，string 是一个 1024 字节的数组。存储空间上限为16 384字节，将整数以八字节计数，strings 则以它们的长度计数。

3.3.2 起源

起源操作可以用来创建一个新的合约，主要是合约代码和合约存储的初始内容。如果该句柄已经是一个已存在的合约的句柄，那么起源操作将被拒绝。（除非是因为错误或恶意，否则基本不会发生。）

合约需要最低ꜩ 1余额来保证其正常运行。如果这个余额低于这个数字，该合约就会被销毁。

3.3.3 交易

交易是从一个合约发至另一个合约的消息，可表示为：

```
type transaction = {
  amount: amount; (* amount being sent *)
  parameters: data list; (* parameters passed to the script *)
  (* counter (invoice id) to avoid repeat attacks *)
  counter: int;
  destination: contract hash;
}
```

如果交易使用管理者密钥签名，或者以编程的方式通过合约中的程序代码执行，那么交易将从合约发出。当这个交易被接收，该交易金额就被加入目的合约的余额中，并且执行目的合约代码。该代码可利用接收到的参数，对合约存储进行读写，改变签名密钥，以及发送交易至其它合约。

计数器的作用是防止中继攻击。当合约的计数等于交易的计数的时，该交易才是有效的。一旦交易被确认，计数器就会增加 1，防止该交易被重用。

该交易也包含客户端认为有效的最近区块哈希值。如果攻击者成功地迫使一个分叉进行长重组，这些交易将不能被整合入区块链，使得分叉显得

明显伪造。这也是安全的最后一道防线，虽然 TAPOS 是有效防止这种长重组的伟大的系统，但是该系统对于防止短期双花并不是十分有效。

一个 (account_handle, counter) 对和比特币中未被花掉的输出几乎等价。

3.3.4 存储费用

存储是网络上最重要的成本之一，在存储上每增加一个字节大约最低的费用为 ꜩ 1。例如，如果交易执行后，一个整数被添加到存储中，并且在已经存在的存储的字符串上增加 10 个字节，那么 ꜩ 18 将从合约的余额中取出和销毁。

3.3.5 代码

该语言是基于栈的且具有高级的数据类型和严格的静态类型校验。设计的灵感来自 Forth, Scheme, ML 和 Cat。它有一个完全描述的指令集 [5]，详细描述了指令集，类型系统，以及词法和语义。这也意味着一个精准的完全参考指南，而不是一个简单的介绍。

3.3.6 交易费

到目前为止，这个系统和以太坊处理交易的方式很类似。然而，我们在处理交易费方面却很不同。在以太坊上用户只要支付随程序执行时间线性增长的费用，就可以运行任意长的程序。虽然这种方式提供了矿工验证交易的经济动机，但并不能给同样参与交易验证的其他矿工相同的动机。在实践中，大多数用智能合约编写的程序都是非常短的。因此，我们通过在程序的执行步骤上加上一个硬性的帽子来简化构建过程。

如果该帽子对一些程序而言被证明是紧上界，他们可以在多个执行步骤上中断并采取多个交易完全执行的方式。正是因为 Tezos 是自我进化的，这个帽子在未来可以改变，或者引入更多的高级原语。

如果账户同意，签名密钥可以通过发起签名信息请求修改。

4 总结

我们认为，我们已经构建一个较大吸引力的种子协议。当然，Tezos 的真正的潜力存在于：让持币者掌控最优质服务协议的选择权利。

参考文献

[1] Vitalik Buterin. Slasher: A punitive proof-of-stake algorithm. https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/, 2014.

[2] Ariel Gabizon Iddo Bentov and Alex Mizrahi. Cryptocurrencies without proof of work. http://www.cs.technion.ac.il/~idddo/CoA.pdf, 2014.

[3] Peter Suber. Nomic: A game of self-amendment. http://legacy.earlham.edu/~peters/writing/nomic.htm, 1982.

[4] Jérôme Vouillon. Lwt: a cooperative thread library. 2008.

[5] Tezos project. Formal specification of the tezos smart contract language. https://tezos.com/pages/tech.html, 2014.