尔格:一个具备强适应力的合约币平台

（Ergo: A Resilient Platform For Contractual Money）

尔格开发团队

2019.5.14
版本1.0

摘要

我们开发了一种全新灵活的区块链协议：尔格。尔格旨在开发分散式应用程序，其主要目的是提供一种有效，安全和简便的方法来实施金融合同。

为实现这一目标，尔格对现有区块链解决方案进行了各种技术和经济上的改进。尔格中的每个硬币都受到尔格脚本（ErgoScript）程序的保护，尔格脚本程序是一种基于Σ-协议的强大且协议友好的脚本语言。使用尔格脚本，我们可以编码使用硬币的条件：比如谁、什么时间、需要什么外部条件能够使用硬币、以及硬币可以给谁等等。

对轻型节点的扩展支持使得尔格对终端用户友好，因为它允许在不受信任的物品硬件上运行合同。为了能够长期使用，尔格遵循生存性方法 - 它使用不会在未来导致安全问题的广泛研究解决方案，同时还通过新的经济模型来防止性能随时间推移而降低。最后，尔格还有一个可自我修改的协议，这个协议允许它在未来吸收新想法并且进行自我改进。

1 导言

从十多年前的比特币开始[1]，到目前为止，区块链技术已被证明是一种保存公共交易账目的安全记账方式，且某种程度上能去除第三方信任中介，如传统金融机构。2017年，即使在比特币已实现超过3000亿美元的市值[2]，且潜在收益率很高的情况下，比特币网络也没有受到过严重攻击。加密货币具有高适应性，并带来了潜在的金融赋权和自我主权，这是通过现代加密算法和去中心化架构的组合得以实现。

然而，获得这种高适应性有所代价，而且在目前现有的区块链体系中，它的长远经济规模还不成立。在没有任何信任的情况下使用区块链，用户需通过下载和处理网络中的所有交易等网络资源来核对对方的信息。除了网络使用、

交易处理也占用大量计算资源，尤其是具有交易语言足够灵活体系。最后，区块链用户必须下载大量数据保存在本地存储和内存中，使得内存需求不断快速增长。因此，交易处理耗用了全世界数以万计的计算机资源，这些资源消耗由用户以交易手续费的形式承担[3]。尽管一些现有系统中存在区块奖励津贴，但手续费仍然非常高昂[4]。因此，尽管区块链技术问世已经超过10年之久，它的应用仍然仅限于金融领域，因为对于金融行业来说，区块链的安全性优势超过了其高交易成本劣势。

除了普通货币应用，区块链的另一个重要用途是搭建去中心化应用。这类应用是运用底层平台的功能，基于区块链特定的程序语言来建立逻辑，编写智能合约。根据区块链编写智能合约的能力来对区块链进行分类的一种方法是，判断它们是基于UTXO（例如比特币），还是基于账户（例如以太坊）。基于账户的加密货币，例如以太坊，引入了由代码控制的特殊协议账户，可能被入账交易调用。尽管这种方式允许执行任意的计算，但复杂的支出条件可能会导致一些问题，例如2017年，以太坊“简单”多重签名合约导致的1.5亿美元的损失[7]。在基于UTXO的加密货币中，每个代币都有一个与之相关的脚本，支出代币时必须要满足脚本所给定的条件。对于UTXO模型来说，运用这些保护性条件相对容易，但是做到随机图灵完备计算却是非常复杂的[8]。然而，大多数金融协议都不要求图灵完整性[9]。尔格正是基于UTXO模型的，并且提供了一种简便的方法，可以在涵盖了绝大多数公共区块链使用案例的金融应用中实行。

尽管合约组成部分对于建立去中心化应用很重要，但是，区块链的长期生存性也是至关重要的。以应用为中心的区块链平台仅仅存在了几年，整个领域都处于非常早期。一些平台已经遇到了性能随时间推移会退化的问题[10,11]，他们能否长期存活将是一大疑点。区块链的历史，十年太短，即使是现有基于UTXO的货币型区块链，也不能证明其具有长期的强适应力。为解决长期存续性问题，目前的解决方案包括尽量低存储要求的轻节点[12]，免除存储租赁费用以防止全节点膨胀，以及可以适应变化、在没有信任媒介时仍然能够迭代的自行进化协议等[13]。我们需要做的，就是将各种科技方案结合起来，在不做任何激进性变化的前提下形成改进的解决办法，而这正是尔格实现的目标。

2 尔格的愿景

尔格协议非常灵活，将来可被社区更改。 在本节中，我们定义了尔格应该遵循的主要原则，可以称之为“尔格社会契约”。 一旦有故意违反以下任何原则的，那么生成的协议就不应该被称为尔格。

- *去中心化优先。*尔格应尽可能做到去中心化：任何形式的，因其失职或恶意行为而会对网络安全造成威胁的组织形式（社会领袖，软件开发员，硬件制造商，矿工，基金会等）都应尽量避免。如果尔格存在进程中出现了任意这种形式的组织，社区应当设法削弱他的影响力。

- *为普通人创建。*尔格将是基于它搭建的应用的底层基础。 它适用于多种应用，但其发展重点是为金融合约提供一种有效、安全和简便的实施方法。

- *合约货币平台。*尔格是基础层，应用将建立在其上。它适用于多种应用，但其主要聚焦点是提供一种有效、安全和简便的方法来实施金融合约。

- *着眼长期发展。*尔格发展的方方面面都从长远的角度出发。在任何时候，尔格都应该具有在没有硬分叉、软硬件升级或其他一些不可预测条件下存活几个世纪的能力。由于尔格的设计定位是一个基础平台，因此搭建在尔格之上的应用程序也应当具备长期生存的能力。这一强适应力性的和长期生存能力也使得尔格成为一个很好的价值存储。

- *开放而且不设访问权限。*尔格协议对不论何种形式的运用不设任何限制。它应该允许任何人加入网络并参与协议而无需任何初始操作。与传统的金融体系不同，在尔格协议的核心层上，不应该有救助，黑名单或其他形式的歧视。另一方面，应用程序开发人员可以自由地实现他们想要的任何逻辑，并对其应用程序的道德性和合法性负责

3 奥托吕科斯（Autolykos）共识协议

任何区块链系统的核心组成部分都是其共识协议，尔格运用的是自行开发的独特工作量证明（PoW）共识协议，称为奥托吕科斯（Autolykos），如下所述。尽管对可能的替代方案进行了广泛的研究，但由于其简单性、高安全性保证以及对轻量客户端的友好性，具有最长链规则的原始PoW协议仍然很收欢迎。然

而，通过十年的广泛测试，早期的“一个CPU一票”的概念也存在一些明显问题。

PoW系统的第一个已知的问题来自专用硬件（ASIC）开发，这使得一小部分配备ASIC的矿工能够比其他人快几个数量级、更有效地解决PoW难题。这个问题可以通过内存困难工作量证明方案来解决，减少ASIC和普通硬件之间的差异。最有希望的方法是使用非对称内存困难工作量证明方案，它验证某解决方案的内存需求比此解决方案的内存需求小的多[14,15]。

第二个对PoW网络去中心化的已知威胁是，即使是大型矿工也倾向于在矿池中联合起来，导致少数池运营方（在编写时，比特币中5个，以太坊中2个）控制了超过51%的算力。尽管该问题已在区块链社区里被讨论多次，但在尔格之前还没有出现过任何实际的解决方案。

尔格的PoW协议奥托吕科斯 [16]是第一个同时具有内存困难和池阻的共识协议。奥托吕科斯基于*单列k总和*问题（*one list k-sum problem*）：矿工必须从预先定义的R列中找到k= 32的元素，R列的大小是：$N = 2^{26}$（2Gb大小），这样$\sum_{j\in J} r_j - sk = d$就在{-b,...,0,...,b mod q}区间中。R列元素是从指数i、2个矿工公钥pk,w和区块头散列值m的单因素计算中得出的结果，其中，$r_i = H(i \| M \| pk \| m \| w)$，$H$是哈希函数回归到的$\mathbb{Z}/q\mathbb{Z}$值,$M$是用于让哈希计算放缓的静态大信息。另外，$J$指数元素集是通过伪随机函数*genIndexes*获得，可以阻止可能的解决方案搜索优化。

因此，我们假设矿工的唯一选择是使用以下“算法1”中给出的简单强力方法来创建一个有效区块。

算法 1 区块挖矿

Input: upcoming block header hash m, key pair $pk = g^{sk}$
Generate randomly a new key pair $w = g^x$
Calculate $r_{i\in[0,N)} = H(i||M||pk||m||w)$
while *true* **do**
    $nonce \leftarrow$ rand
    $J := genIndexes(m||nonce)$
    $d := \sum_{j\in J} r_j \cdot x - sk \mod q$
    if $d < b$ **then**
        return $(m, pk, w, nonce, d)$
    end if
end while

需要注意的是，虽然挖掘过程使用私钥，但解决方案本身仅包含公钥。解算验证由“算法2”完成。

算法 2 区块挖矿

1: **Input**: $m, pk, w, nonce, d$
2: require $d < b$
3: require $pk, w \in \mathbb{G}$ and $pk, w \neq e$
4: $J := genIndexes(m||nonce)$
5: $f := \sum_{j \in J} H(j||M||pk||m||w)$
6: require $w^f = g^d \cdot pk$

这种方法阻止矿池形成，因为挖矿需要密钥sk：一旦任何池矿工找到正确的解决方案，他就可以使用这个密钥来窃取块奖励。另一方面，揭示单个解决方案是安全的，因为它仅包含公钥并且揭示了2个秘密sk,w之间的单个线性关系。

造成内存困难是源于算法1需要保持整个R列用于主循环执行的事实。每个列表元素占用32个字节，因此N列所有元素需要在$N = 2^{26}$时占用$N \cdot 32 =$ 2Gb的内存。矿工可以尝试通过计算这些在运行中的元素而不需要保存在内存中来减少内存要求，但是这样他需要多次计算相同的哈希H（现代GPU大约10^4次），这就降低了效率和利润。

计算列表R也是一项非常繁重的计算任务：我们首次执行[17]是在Nvidia GTX 1070上，花了25秒来完成所有列表中的2^{26}个元素的填充。然而，如果矿工在内存中也存储了未完成的哈希值$u_{i \in [0,N)} = H(i \parallel M \parallel pk)$列表，则该部分就可以被优化，共消耗5个多Gb。在这种情况下，计算未完成的哈希值在挖矿初始化期间应该只进行一次，同时最终确定它们并填充新的头信息的R列，只消耗几十毫秒（在Nvidia GTX 1070上大约50毫秒）。

目标参数b置于难题本身，并通过难度调整算法[18]调整到当前网络哈希率，以保持块之间的时间间隔接近2分钟。该算法试图通过众所周知的*线性最小二乘法*得出的基于来自前8个周期的数据来预测即将到来的1024个区块长周期的哈希速率。这使得预测比通常的难度调整算法更好，并且还使得“跳币”攻击的利润更低。

4 尔格的状态

要检查新交易，加密货币客户端不会将账簿与此前发生的所有交易一起使用。相反，它只会用到历史交易中某一笔的状态快照。在比特币核心参考履行中，该快照可看作是活跃的一次性代硬币（即，UTXO），并且交易会在销毁一些代硬币同时也产生新的硬代币。在以太坊中，此快照存在于是长期帐户，

而交易会修改某些帐户的货币余额和内部存储。此外，与比特币不同，以太坊上的快照象征是固定写入协议中的，而快照的验证摘要被写入区块头。

尔格遵循比特币的UTXO设计，并使用一次性硬币来代表快照。与比特币的区别在于，除了具有货币价值和能保护脚本外，尔格一次性硬币，称为*币箱*（*box*），还包含了用户定义的数据。与以太坊类似，尔格块还在应用区块后后，存储全局状态的认证摘要，称为*stateRoot*（状态根）。

尔格币箱由寄存器组成（除了寄存器外别无他物）。此币箱可以有10个标记为$R_0, R_1, \ldots, R_9$的寄存器，其中前四个填充了强制值，其余的可以包含任意数据或为空。

- R_0（*货币值*）。在此币箱中锁定的尔格币数量。
- R_1（*保护脚本*）。保护此币箱的序列化脚本。
- R_2（*代币*）。一个币箱可以携带多个代币。寄存器包含了在此币箱中锁定的成对数组（代币标识符→金额）。
- R_3（*交易信息*）. 包含（1）声明了的创造高度（不应超过包含交易的实际区块的高度），（2） 创造此币箱的一个唯一交易标识符，（3）在交易输出币箱中的此币箱指数。
- R_4–R_9（*附加数据*）。包含任意用户定义的数据。

一次性不可变对象（如比特币的UTXO模型）在以太坊的长期可变账户中具有一些优势。首先，它为回放或重排攻击提供了更简单，更安全的保护。其次，并行处理交易更加容易，因为它们不会修改它们访问对象的状态。此外，一项交易是指要么完全按预期修改了系统状态，要么根本没有改变状态（没有因燃气异常、可重入性问题等导致的副作用）。最后，使用一次性硬币构建完全无状态的客户似乎更容易[19]（尽管该领域的研究仍处于初始阶段）。

对一次性硬币的一个主要批评是该模型似乎不适合非平凡的去中心化应用。然而，尔格已经克服了这些问题，并通过演示构建在其上的许多非平凡的原型应用来证明这种说法是错误的（参见第7节）。

尔格协议以未被先前交易销毁的币箱形式来修复分类帐快照画像。详细地说，矿工应该维护在UTXO集之上构建的类似于梅克尔树状认证数据结构，并且必须在每个区块头中包含该结构的短摘要（仅33个字节）。必须在应用区块后才能计算此摘要。这个经过验证的数据结构构建在AVL+树之上[12]，它像常规哈希树一样，允许生成树中特定元素存在或不存在的证明。因此，维护完整树的用户能够生成他们币箱未用证据，并且一个小小的33字节摘要

就足以验证这些证据。但是，与常规哈希树不同，AVL+树还允许生成树修改证明，允许验证者计算新的树摘要。尔格矿工需要生成区块修改证明，并且此证明的哈希值与结果状态的摘要要一起包含在区块头中。因此，仅包含当前状态的短小摘要的轻节点就能够验证完整的区块 - 它们可以检查所有用过的币箱已从状态中移除，所有创建的币箱都被添加到状态中并且不再进行任何更改。

AVL+树允许构建有效的经过验证的字典，与先前的解决方案相比，可以减少证明大小并加快验证速度，速度是原来的1.4-2.5倍，使其更适合加密货币应用程序。例如，我们的证明比以太坊中用于相同目的的Merkle Patricia trie（MPT）的证明小3倍（见图1）。

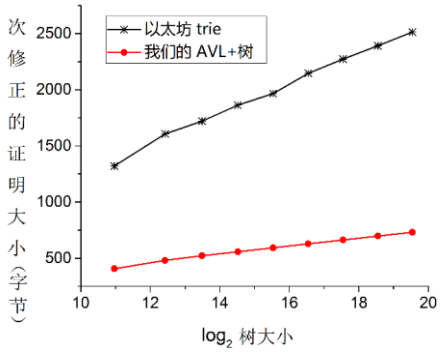

图 1: 尔格的AVL+ 树和 Merkle patriciatrie的证明大小比较

最后，将单个区块中的多项交易的证明打包压缩，将其总长度大概能减少2倍：

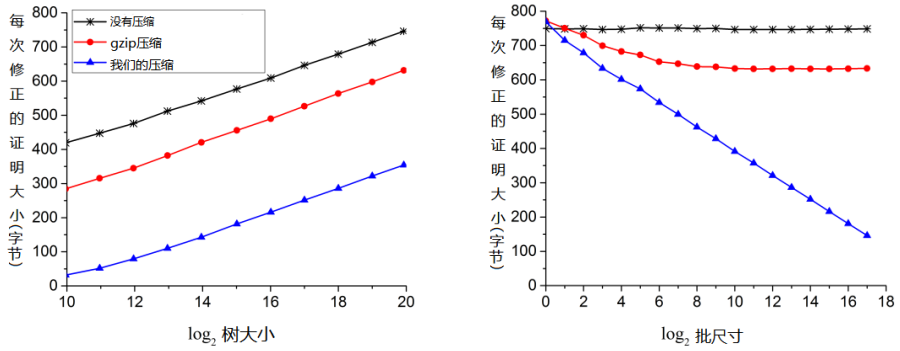

图2：左图：2000次交易每次修改的证明大小，作为起始树大小n的函数。右图：对于具有n= 1000000个键的树每次修改的证明大小作为批量大小B的函数。在左右这两种情况下，一半的修改是插入新的（键，值）对，另一半是改变现有钥的值。

因此，尔格状态提供了一种有效且安全的方式来证明其中的特定元素是否存在，以及证明树的修改。尔格智能合约语言支持这些树操作，从而提供了执行第7节中讨论的复杂合约的能力。

5 适应力强和生存性

作为一个合约币的平台，尔格还应支持长期的合约，至少是一般人的一生这么长。然而，现有的其他新兴的智能合约平台也会遇到性能下降和对外部条件的适应性问题。这就导致了一种情况，即加密货币依赖于一小组开发人员来提供固定硬分叉，否则加密货币将无法生存。例如，以太坊网络已经开始使用工作量证明共识算法，并承诺在未来切换到权益证明。然而，权益证明开发的延迟导致了很多次硬分叉修复[20]，同时社区仍然被迫依赖核心开发人员承诺实施下一个硬分叉。

第一个常见的生存性问题是，为了追求普及度，开发人员倾向于在没有适当初步研究和测试的情况下实施事先无准备的解决方案。这样的解决方案不可避免地会出现漏洞，然后出现了仓促的漏洞修复，接着再修复那些错误的漏洞修复...，这就使网络变得不可靠，甚至更不安全。一个值得注意的例子是IOTA加密货币，它实现了各种可扩展性解决方案，包括它的哈希函数和DAG结构，使其能够实现高普及度和高市场资本化。然而，当对这些解决方案进行分析[21,22]时，它揭示了多个严重的问题，包括能够偷盗货币的

实际性攻击，随后的硬分叉[23]通过切换到已知的SHA3哈希函数来解决问题，确认了这种创新的无用性。而尔格会使用经过良好测试的稳定解决方案，即使这些解决方案会导致短期创新变慢。尔格中使用的大多数解决方案都在同行评审会议[12,18,3,8,24,25]中提出的论文中正式化了，并且在社区中也得到了广泛讨论。

去中心化（以及生存性）面临的第二个问题是缺乏安全的去信任轻量客户端。尔格解决区块链技术的这个问题，而且不会产生新的问题。由于尔格是PoW（工作量证明）区块链，因此它很容易从区块内容中提取小的区块头。只有该区块头就能验证在其上完成的工作，并且区块头链足以用于最佳链选择和与网络同步。区块头链虽然比完整的区块链小得多，但它仍然会随着时间推移线性增长。最近对轻量客户端的研究为轻量客户端提供了一种通过下载更少量数据来与网络同步的方法，从而解锁使用去信任的低端硬件（如手机）加入网络的能力[26,27] 。尔格使用经过验证的状态（参见第4节），对于区块中的交易，客户端可以下载其正确性的证明。因此，无论区块链大小如何，有手机的普通用户都可以加入网络并开始使用具有与完整节点相同的安全保证的尔格。

读者可能注意到了第三个潜在的问题，即虽然对轻客户的支持解决了尔格用户的问题，但它并没有解决尔格矿工的问题，矿工仍然需要保持整个状态以进行有效的交易验证。在现有的其他区块链系统中，用户可以将任意数据置于此状态中。这些数据永远存在，会产生大量尘埃，并且随着时间的推移它的大小会不断增加[28]。大的状态会导致严重的安全问题，因为当状态不适合随机存取存储器时，攻击者可以触发交易，他的验证由于需要随机访问矿工的存储而变得非常慢。这可能会导致分散式阻断服务（DoS）攻击，例如2016年以太坊上的攻击[29]。此外，社区对此类攻击的恐惧以及“状态膨胀”问题对矿工或持有状态的用户没有任何补偿阻止了扩展解决方案，否则可能已经执行（例如更大的区块大小）。为了阻止这种情况发生，尔格有一个存储租用特性如果一个币箱在该状态下保持4年而不被移动，矿工可以对该状态下保留的每个字节收取少量费用。

这种类似于常规云存储服务的想法最近才被提出用于加密货币[30]，并且产生了几个重大的影响。首先，它确保了尔格挖矿总是稳定的，而不像比特币和其他PoW（工作量证明）货币，挖矿可能会在完成发行后变得不稳定[31]。第二，状态大小的增长变得可控和可预测，从而帮助尔格矿工管理他们的硬件需求。第三，通过从过时的币箱收取存储费用，矿工可以将硬币返回循环中，从而防止由于丢失密钥而导致的循环供应稳定减少[32]。所有这

些影响都支持尔格在技术和经济上的长期生存性。

对生存性的第四个重要挑战是外部环境的变化和对协议的要求。协议应该适应不断变化的硬件基础架构和新思路，以提高随着时间推移出现的安全性或可扩展性，用例的演变等等。如果所有规则都是固定的，不能用去中心化的方式改变它们，即使是简单的持续变化也都导致激烈的辩论和社区分裂。例如，对比特币中区块大小限制的讨论导致其分成几个独立的硬币。相比之下，尔格协议是可自我修改的，能够适应不断变化的环境。在尔格中，区块大小等参数可以通过矿工投票进行恰当地更改。在每个1024区块投票周期的开始，矿工最多可以提出改变2个参数（例如区块大小的增加和存储费用因素的减少）。在该周期剩余的时间，矿工采取投票来决定更改的批准还是拒绝。如果该周期内的大多数投票支持更改，则新值将写入下一个周期的第一个区块的扩展部分，并且网络开始使用更新的值进行区块挖矿和验证。

为了采纳更基础性的变化，尔格采用了*软分叉方法*，允许在保持旧节点运行的同时显著地改变协议。在一个周期的开始，矿工也可以提出投票，针对是否对描述受影响验证规则做基础性改变（例如，向尔格脚本添加新指令）。对这些激进性变化的投票持续32,768次，并且要求有至少90％选“是”的投票。一旦投票被接受，1个包含32,768区块长的激活周期将开始为过时的节点提供更新其软件版本的时间。如果节点软件在激活期后仍未更新，则它会跳过指定的检查，仍继续验证所有已知的规则。先前的软分叉更改列表将记录到扩展中，以允许任何软件版本的轻节点加入网络并跟上当前的验证规则。软分叉与投票协议的组合允许改变网络的几乎所有参数，除了负责投票本身的PoW（工作量证明）规则。

6 尔格原生货币

尔格平台有其原生货币，称为尔格币，可分割为最多10^9个最小单位，nano尔格币（一个nano尔格币是尔格币的十亿分之一）。尔格币对于尔格平台的稳定性和安全性非常重要，原因如下所述。

在尔格生命的最初阶段，矿工将根据预定义和硬编码的货币发行计划来获得尔格币的奖励（更多细节见6.1）。这些硬币将激励矿工加入尔格网络，使网络免受基于哈希率的攻击，如已知的51％攻击[33]。

尔格币发行将在短短的八年内完成，之后矿工将仅从费用中收取尔格币。尽管后面还可以通过矿工链上的投票进行调整，但是在任何给定时间点，尔格区块大小和最大块计算成本都将受到限制，因此矿工被强制在高负载期间

仅选择来自内存池的一部分交易。费用将帮助矿工对交易进行分类，防止垃圾信息的攻击，同时允许矿工收录来自区块中诚实用户的交易。

除了网络和计算资源之外，交易还通过增加状态大小来使用存储。在现有的其他加密货币中，状态的一个元素，即基于UTXO区块链中的UTXO，在尔格中被称为一个币箱，一旦被创造，将获得永生，矿工和部分将此状态保存在高成本随机访问内存中的客户，他们都不收费。这导致了激励不一致和不断增加的状态规模。与此相反，尔格有一个存储租用特性，定期向用户对将每个字节保存在状态中收取尔格币费用。这种储存租金通过限制状态规模或确保对较大状态规模的适当补偿，将丢失的硬币退回到流通中并为矿工提供额外稳定和可预测的奖励，从而使系统更加稳定。

因此，作为合约币平台，尔格适合在其上构建应用和货币系统。然而，参与这样的系统将需要使用尔格原生尔格币，以便支付存储租金和交易费用，这将为矿工提供强有力的持续激励，以确保网络具有足够的哈希力。对于用户而言，如果他们发现尔格的应用程序具有高价值，那么他们将非常乐意购买，使用和保存尔格币。

6.1 发行

将在系统中循环的所有尔格币都以初始状态呈现，其中包含3个区块：

- *无预挖证明。*此币箱仅包含一个尔格币，并且受脚本保护，防止任何人使用它。因此，它是一个长寿命的币箱，将被留在系统中，直到存储租用特性销毁它为止。其主要目的是为了证明尔格挖矿并没有在宣布发起日期之前由任何人私下开启。为实现这一目标，此区块的其他寄存器包含了来自媒体（新华网、The Guardian、Vedomosti）的最新标题，以及来自比特币和以太坊的最新区块标识符。因此，尔格挖矿无法在现实世界中的某些事件和加密货币空间之前启动。

- *金库。*这个币箱包含4,330,791.5个尔格币，用于资助尔格开发。它的脚本保护[34]由两部分组成。

  首先，确保仅解锁区块值的预定义部分。在区块1-525,599（2年）期间，每个区块将释放7.5个尔格币。然后在区块525,600-590,399（3个月）期间，每个区块将释放4.5个尔格币。最后，在区块590,400-655,199（3个月）期间，每个区块将释放1.5个尔格币。此规则确保

尔格开发的资金将存在2.5年，并且在任何时刻，奖励不超过流通中尔格币总数的10％。

第二，它具有免受意外支出的定制保护。最初，它需要支出交易由初始团队成员控制的3个密钥中的至少2个签名。当他们支付这个币箱时，他们可以随意更改脚本部分，例如通过添加新成员来保护基金会资金。

在第一年，这些资金将用于支付预发行的EFYT代币。之后，它们将通过正在开发的社区投票系统以去中心化的方式发行。

- *矿工奖励*。这个币箱包含了 93,409,132个尔格币，将被区块矿工收集，以奖励他们的工作。它的保护脚本[35]要求支出交易具有以下属性的两个输出：

  - 第一个输出应该受到相同脚本的保护，其中的尔格币数应该等于其余矿工的奖励。在1 - 525,599（2年）的区块内，矿工将能够从这个币箱收集67.5 Erg。在区块525,600-590,399（3个月）期间，矿工将能够收集66个尔格币，之后，区块奖励将每64,800区块（3个月）减少3个尔格币，直到它在区块2,080,799处达到零。

  - 第二个输出应该包含剩余的硬币，并且应该受到以下条件的保护：它可以由解决区块的PoW（工作量证明）拼图难题的矿工花费，而不早于当前区块之后的720区块。

所有这些规则产生以下曲线，表示随时间流通的硬币数量：

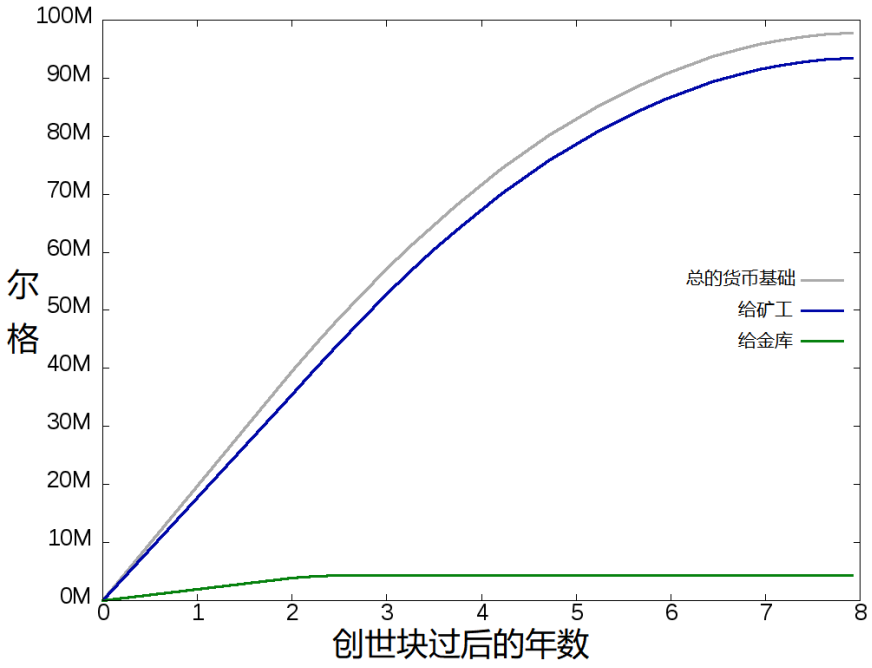

图 3: 尔格发行曲线

7 合约化货币

我们认为，绝大多数公链用例（即使是那些声称能够提供通用去中心化世界计算机的用例）都是用于金融应用，不需要图灵完备性。例如，如果预言机（oracle）将非金融数据写入区块链（例如温度），则此数据平时会在金融合约中使用。我们做的另一个细小观察是许多应用使用具有与原生币不同机制的数字代币。

对于应用开发人员，尔格平台提供自定义代币（属于一等公民）和用于编写币箱保护条件的特定领域的语言，应用于灵活，安全的金融应用。尔格应用程序的定义是保护内置于币箱中的脚本，其中也可能包含执行中涉及的数据。我们使用*合约化货币（contractual money）*这一术语来定义其使用受合约约束的尔格币（和二级代币）。这适用于现有平台上的所有币，因为任何带有其内容的币箱（尔格币，代币，数据）都受合约限制。

然而，我们可以区分两种类型的合约化尔格币。第一个被称为*自由尔格币（freeErgs）*，可以轻松地改变他们的合约，并且对支出交易的输出或其

他输入没有限制。第二种类型是*有界尔格币（bounded Ergs）*，其合同要求支出交易具有有特定属性的输入和输出币箱。

例如，如果币箱A仅受公钥保护（因此提供与交易对应的的签名就可以销毁币箱了），公钥所有者可以花费A并将尔格币转移到任意输出币箱。因此，A中的尔格币是自由的。相比之下，想象一个币箱B受公钥和条件的组合性保护，该条件要求支出交易创建一个具有与B中相同数量尔格币的输出币箱，并且其保护脚本具有哈希hashrBMUEMuPQUx3GzgFZSsHmLMBouLabNZ4cERm4N（用Base58编码）。在这种情况下，B中的尔格币是有界尔格币。

同样，我们可以定义自由和有界代币。尔格合约可以有多个混合，例如有界尔格币和自由代币，或者在一个公钥下是有界的而在另一个公钥下是自由的。

7.1 尔格合约的初期阶段

比特币交易输出受到名为*Script*的基于堆栈语言中的程序保护，然而在尔格中，币箱受到逻辑公式的保护，该逻辑公式将上下文中的谓语和加密语句组合在一起，该加密语句通过使用AND, OR, 和k-out-of-n连接词的零知识协议变得可证明。该公式表示为类型有向无环图（typed direct acyclic graph），其序列化形式写在币箱中。为了销毁一个币箱，支出交易需要提供满足公式的参数（包括零知识证明）。

但是，在大多数情况下，开发人员不太可能根据图表来开发合约。相反，他会愿意用尔格脚本等高级语言来开发，由我们在参考客户端提供。

用尔格脚本编写脚本很容易。例如，对于一个one-out-of-two签名，保护脚本是$pk_1||pk_2$，意即“公钥pk_1对应的秘钥知识证明或公钥pk_2对应的秘钥知识证明”。我们有两个单独的文档可以帮助开发人员开发尔格脚本合约：“ErgoScript Tutorial”[36]和“Advanced ErgoScript Tutorial”[37]。所以，我们不深入展开使用尔格脚本开发合约的细节，而在以下部分中提供了几个激励案例。

尔格塑造合约可能性的另外两个特征是：

- *数据输入：*要在交易中使用，就不需要销毁币箱，而是可以只读取币箱。在后一种情况下，我们将该币箱称为交易数据输入的一部分。因此，交易获取两个币箱集作为其参数、输入和数据输入，并生成一个名为*输出*的币箱集。数据输入对于预言机应用和交互合约来说很有用。
- *自定义代币：*只要处理它们的预估复杂度不超限（一个由矿工投票

设置的参数），交易就可以承载很多代币。交易还可以发行具有唯一标识符的单个代币，该唯一标识符等于交易的第一（可花费）输入币箱的标识符。假设底层哈希函数抗冲击，标识符就是唯一的。发行的代币数量可以是[1, 9223372036854775807] 范围内的任何数字。对于代币，遵循弱保留规则，这要求交易输出中的任何代币的总量不应超过交易输入中该代币的总量（即，一定数量的代币可以被燃烧）。相反，尔格币遵循强保留规则，这要求输入和输出中的尔格币总量必须相同。

7.2 合约示例

在本节中，我们提供了一些示例，显示了与比特币相比，尔格合约的优越性。这些例子包括投注预言机提供的数据，非交互式混合，原子互换，补充货币以及在尔格区块链之上首次币发行。

7.2.1 一个预言机（oracle）的案例

配备自定义代币和数据输入，我们就可以开发一个简单的预言机案例，其中还显示了我们在使用尔格合约时发现的一些设计模式。假设Alice和Bob想要对明天的天气下赌注，则可将货币放入币箱中，如果明天的温度超过15度，则这些货币由Alice支配，否则由Bob支配。为了将温度传递到区块链中，就需要一个受信预言机。

以太坊及其长期存在的帐户通常可以提前知道受信信预言机的标识符，然而使用一次性币箱来传递数据就变得更为棘手。对于初始者来说，受到信预言机密钥保护的币箱是不可信任的，因为任何人都可以创建这样的币箱。可以将签名数据包含在一个币箱中并检查合约中的预言机签名（我们有这样一个例子），但这是非常复杂的。相反，自定义代币的解决方案却非常简单。

首先，应该发行一个识别预言机的代币。在最简单的情况下，此代币的数量可以是一个。我们将这样的代币称为*单一模式代币（singleton token）*。预言机创建一个包含该代币的币箱，包含数据（即温度）在寄存器R_4中，UNIX周期时间（UNIX epoch time）在寄存器R_5中。为了更新温度，预言机会销毁它的币箱并创建一个具有更新温度的新币箱。

假设Alice和Bob事先知道预言机的代币标识符。有了这些知识，他们就可以联合创建一个包含合约的币箱，该合约需要第一个（只读）数据输入来包含预言机的代币。合约从数据输入中提取温度和时间，并决定谁获得支付。代码很简单，如下所示：

算法 3 预言机合约案例

```
1: val dataInput = CONTEXT.dataInputs(0)
2: val inReg = dataInput.R4[Long].get
3: val inTime = dataInput.R5[Long].get
4: val inToken = dataInput.tokens(0)._1 == tokenId
5: val okContractLogic = (inTime > 1556089223) &&
6:           ((inReg  > 15L && pkA) || (inReg ≤ 15L && pkB))
7: inToken && okContractLogic
```

该合约显示了如何使用单一模式代币进行验证。作为一种可能的替代方案，预言机可以将时间和温度放在一个币箱中，并附上此数据的签名。然而，这需要签名验证，与单一模式代币方法相比，签名验证更复杂且更昂贵。此外，合约还显示了只读数据输入对需要访问存储在该状态下其他币箱中的数据的合约有什么用。如果没有数据输入，预言机则必须为每对Alice和Bob都发布一个可用的币箱。如果有数据输入，预言机则只需发出一个币箱。

7.2.2 一个混合的案例

隐私对于数字货币很重要，但实施隐私可能成本高昂或需要可靠的设置。因此，希望能找到一种更便宜的硬币混合方式。作为迈出的第一步，我们在两个用户Alice和Bob之间提供了一个非交互式混合协议，其工作方式如下：

1. Alice创建了一个币箱，要求支出交易满足特定条件。之后，Alice只听从区块链，而不需要与Bob进行任何交互。

2. Bob创建一个交易，消费Alice的币箱和他自己的一个币箱，并生成两个输出，它们具有相同的脚本但不同的数据。 Alice和Bob中的每一个人可能只花费两个输出中的一个，但是对于外部观察者来说，两个输出看起来无法区分，并且他不能确定哪个输出属于谁。

为简单起见，我们在案例中不考虑费用。混合的想法类似于非交互式Diffie-Hellman密钥交换。首先，Alice生成一个秘密值x（一个巨大的数字）并发布相应的公共值$gX = g^x$。她要求Bob生成一个秘密数字y，并在每个输出中包括两个值c_1，c_2,其中一个值等于g^y，另一个值等于g^{xy}。Bob使用随机硬币为{c_1，c_2}选择含义。如果无法获知秘密，外部观察者就无法以高于一半的概率猜对，c_1是等于g^y还是g^{xy}。这假设我们使用的密码学原语具有某种属性，即判定Decision Diffie-Hellman（DDH）问题很难。为了销毁输出币箱，就应该给出一个证据，即已知y使得$c_2 = g^y$，或者已知x使得$c_2 = c_1^x$。Alice币箱的合约检查c_1和c_2是否正确。Alice的硬币和混合交易的输出的代

码片段分别由算法4和5显示。由于尔格脚本目前不支持证明某些x的知识，如对任意c_1说使得的$c_2 = c_1^x$的x，我们将支持证明稍长的语句，即证明x的知识，使得$gX = g^x$和$c_2 = c_1^x$。这叫做proveDHTuple。

算法 4 Alice的输入脚本

```
1: val c1 = OUTPUTS(0).R4[GroupElement].get
2: val c2 = OUTPUTS(0).R5[GroupElement].get
3:
4: OUTPUTS.size == 2 &&
5: OUTPUTS(0).value == SELF.value &&
6: OUTPUTS(1).value == SELF.value &&
7: blake2b256(OUTPUTS(0).propositionBytes) == fullMixScriptHash &&
8: blake2b256(OUTPUTS(1).propositionBytes) == fullMixScriptHash &&
9: OUTPUTS(1).R4[GroupElement].get == c2 &&
10: OUTPUTS(1).R5[GroupElement].get == c1 && {
11:       proveDHTuple(g, gX, c1, c2) ||
12:       proveDHTuple(g, gX, c2, c1)
13: }
```

算法 5 混合交易输出脚本

```
1: val c1 = SELF.R4[GroupElement].get
2: val c2 = SELF.R5[GroupElement].get
3: proveDlog(c2) // either c2 is g^y
4: proveDHTuple(g, c1, gX, c2) // or c2 is u^y = g^xy
```

我们向读者[37]以证明了输出的不可分辨性，以及为什么Alice和Bob只能花费他们各自的硬币。

7.2.3 更多案例

在本节中，我们将简要介绍一些案例以及提供有详细信息和代码的文档链接。

原子交换 尔格和任何支持SHA-256或Blake2b-256哈希预映像和时间锁的区块链之间的交叉链原子交换，可以用与比特币[38]类似的方式完成。 [36]中提供了尔格替代实现。由于尔格还具有自定义代币，因此单个尔格区块链（Erg-to-token或token-to-token）上的原子交换也是可以做的。对此的实现也可以在[36]中找到。

众筹 我们考虑最简单的众筹方案。在这个例子中，如果一个拥有已知公钥

的众筹项目可以在达到一定区块高度前收集总值不低于一定数量的未使用的输出，则该项目算是成功的。项目支持者创建一个受以下语句保护的输出币箱：如果支出交易的第一个输出币箱受项目密钥保护且金额不低于目标金额，则可以使用该币箱。然后项目可以（在单个交易中）收集最大的支持者输出币箱，总价值不低于目标金额（可以收集多达22,000个输出，这对于大型众筹活动来说已经足够了）。对于剩余的输出，可以构建后续交易。代码可以在[36]中找到。

本地交易所交易系统 以下我们将简要地介绍在尔格上的本地交易所交易系统（LETS）。在这样的系统中，社区成员可以通过个人债来发行社区货币。例如，如果Alice和Bob的余额都为零，Alice从Bob那里花了5个社区代币来买东西，在此次交易之后，Alice的余额为-5个代币，Bob的余额为5个代币。然后Bob可以使用他的5个代币购买东西，例如从Carol那儿买。通常，在这样的系统中，存在负余额限制（以避免搭便车问题）。

由于数字社区容易受到西比尔攻击[39]，因此需要一些机制来防止西比尔节点产生债务的此类攻击。 最简单的解决方案是使用一个可信赖的经理委员会来批准社区的新成员。 一个去信任但更复杂的解决方案是使用尔格币来做保证金。 为简单起见，我们在此考虑委员会的方法。

此示例包含两个交互合约。*管理合约*维护社区成员列表，如果满足某些管理条件（例如，提供阈值签名），则可以添加新成员。新成员与包含标识该成员标记的币箱相关联。此币箱包含*成员合约*，由一个特殊的交换脚本保护，该脚本要求支出交易进行公平交换。我们跳过相应的代码，这个代码可以在另一篇文章[40]中找到。

与前面的案例正相反，此合约显示的不是存储成员列表，而是关于币箱中含有经过身份验证的AVL+树的简短摘要。这样可以降低对状态的存储要求。执行查找或修改成员列表的交易应该为AVL+树查找或修改操作提供证据。因此，节省状态存储空间会导致更大的交易，但这种可扩展性问题更容易解决。

首次币发行（ICO） 我们接下来讨论首次币发行示例，该示例显示了如何在尔格中创建多阶段合同。像大多数ICO一样，我们的例子有三个阶段。在第一阶段，该项目在尔格币筹集资金。在第二阶段，项目发布一个新代币，其数量等于第一阶段中提出的nano尔格币数量。在第三阶段，投资者可以

撤回已发行的代币。

请注意，第一和第三阶段在区块链上有许多交易，而单个交易对第二阶段来说就够了。与之前的示例类似，ICO合同使用AVL+树来存储（投资者，金额）对的列表。完整的代码可在[41]获得。

更多案例 在[36,37]中，我们还有更多尔格应用的案例。这些案例包括时间控制发行，冷钱包合约，石头剪刀布游戏等等。

参考文献

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] Bitcoins price surpasses $18,000 level, market cap now higher than visas. [Online]. Available: https://cointelegraph.com/news/ bitcoins-price-surpasses-18000-level-market-cap-now-higher-than-visas

[3] A. Chepurnoy, V. Kharin, and D. Meshkov, "A systematic approach to cryptocurrency fees," *IACR Cryptology ePrint Archive*, vol. 2018, p. 78, 2018.

[4] Skyrocketing fees are fundamentally changing bitcoin. [Online]. Available: https://arstechnica.com/tech-policy/2017/12/bitcoin-fees-rising-high/

[5] N. Szabo, "Smart contracts," *Unpublished manuscript*, 1994.

[6] J. Zahnentferner, "Chimeric ledgers: Translating and unifying utxo-based and account-based cryptocurrencies." *IACR Cryptology ePrint Archive*, vol. 2018, p. 262, 2018.

[7] I accidentally killed it: Parity wallet bug locks $150 million in ether. [Online]. Available: https://www.ccn.com/ i-accidentally-killed-it-parity-wallet-bug-locks-150-million-in-ether

[8] A. Chepurnoy, V. Kharin, and D. Meshkov, "Self-reproducing coins as uni- versal turing machine," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 57–64.

[9] M. Jansen, F. Hdhili, R. Gouiaa, and Z. Qasem, "Do smart contract languages need to be turing complete?" 2019. [Online]. Available: https://www.researchgate.net/publication/332072371_Do_Smart_Contract_Languages_Need_to_be_Turing_Complete/download

[10] Very slow syncing on hard drive. [Online]. Available: https://github.com/ethereum/go-ethereum/issues/14895

[11] Why is my node synchronization stuck/extremely slow at block 2,306,843? [Online]. Available: https://ethereum.stackexchange.com/questions/9883/why-is-my-node-synchronization-stuck-extremely-slow-at-block-2-306-843

[12] L. Reyzin, D. Meshkov, A. Chepurnoy, and S. Ivanov, "Improving authenticated dynamic dictionaries, with applications to cryptocurrencies," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 376–392.

[13] L. Goodman, "Tezos — a self-amending crypto-ledger white paper," *URL: https://www.tezos.com/static/papers/whitepaper.pdf*, 2014.

[14] A. Biryukov and D. Khovratovich, "Equihash: Asymmetric proof-of-work based on the generalized birthday problem," *Ledger*, vol. 2, pp. 1–30, 2017.

[15] Ethash. [Online]. Available: https://github.com/ethereum/wiki/wiki/Ethash/6e97c9cea49605264c6f4d1dc9e1939b1f89a5a3

[16] A. Chepurnoy, V. Kharin, and D. Meshkov, "Autolykos: The ergo platform pow puzzle," 2019. [Online]. Available: https://docs.ergoplatform.com/ErgoPow.pdf

[17] Autolykos gpu miner. [Online]. Available: https://github.com/ergoplatform/Autolykos-GPU-miner

[18] D. Meshkov, A. Chepurnoy, and M. Jansen, "Short paper: Revisiting diffi- culty control for blockchain systems," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 429–436.

[19] A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," Cryptology ePrint Archive, Report 2018/968, Tech. Rep., 2018.

[20] Ethereums blockchain is once again feeling the difficulty bomb effect. [Online]. Available: https://www.coindesk.com/ethereum-blockchain-feeling-the-difficulty-bomb-effect

[21] E. Heilman, N. Narula, G. Tanzer, J. Lovejoy, M. Colavita, M. Virza, and T. Dryja, "Cryptanalysis of curl-p and other attacks on the iota cryptocurrency."

[22] G. De Roode, I. Ullah, and P. J. Havinga, "How to break iota heart by replaying?" in *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018, pp. 1–7.

[23] Iota vulnerability report: Cryptanalysis of the curl hash function enabling practical signature forgery attacks on the iota cryptocurrency. [On- line]. Available: https://github.com/mit-dci/tangled-curl/blob/master/ vuln-iota.md

[24] A. Chepurnoy and M. Rathee, "Checking laws of the blockchain with property-based testing," in *Blockchain Oriented Software Engineering (IWBOSE), 2018 International Workshop on*. IEEE, 2018, pp. 40–47.

[25] T. Duong, A. Chepurnoy, and H.-S. Zhou, "Multi-mode cryptocurrency systems," in *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*. ACM, 2018, pp. 35–46.

[26] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof- of-work," Cryptology ePrint Archive, Report 2017/963, 2017. Accessed: 2017-10-03, Tech. Rep., 2017.

[27] L. Luu, B. Buenz, and M. Zamani, "Flyclient super light client for cryptocurrencies," *IACR Cryptology ePrint Archive*, 2019. [Online]. Available: https://eprint.iacr.org/2019/226

[28] C. Pérez-Solà, S. Delgado-Segura, G. Navarro-Arribas, and J. Herrera-Joancomart´ı, "Another coin bites the dust: an analysis of dust in utxo- based cryptocurrencies," *Royal Society open science*, vol. 6, no. 1, p. 180817, 2019.

[29] Ethereum network attackers ip address is trace- able. [Online]. Available: https://www.bokconsulting.com.au/blog/ ethereum-network-attackers-ip-address-is-traceable/

[30] A. Chepurnoy and D. Meshkov, "On space-scarce economy in blockchain systems." *IACR Cryptology ePrint Archive*, vol. 2017, p. 644, 2017.

[31] M. Carlsten, H. Kalodner, S. M. Weinberg, and A. Narayanan, "On the instability of bitcoin without the block reward," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 154–167.

[32] E. Krause, "A fifth of all bitcoin is missing. these crypto hunters can help," 2018.

[33] 51% attack. [Online]. Available: https://en.bitcoinwiki.org/wiki/51% 25 attack

[34] Script of the ergo treasury box. [Online]. Available: https://github.com/ScorexFoundation/sigmastate-interpreter/blob/1b7b5a69035fc384b47c18ccf42b87dd95bbcf12/src/main/scala/org/ergoplatform/ErgoScriptPredef.scala#L118

[35] Script of the ergo emission box. [Online]. Available: https://github.com/ScorexFoundation/sigmastate-interpreter/blob/1b7b5a69035fc384b47c18ccf42b87dd95bbcf12/src/main/scala/org/ergoplatform/ErgoScriptPredef.scala#L74

[36] Ergoscript, a cryptocurrency scripting language supporting noninteractive zero-knowledge proofs. [Online]. Available: https://docs.ergoplatform.com/ErgoScript.pdf

[37] Advanced ergoscript tutorial. [Online]. Available: https://docs.ergoplatform.com/sigmastate_protocols.pdf

[38] T. Nolan, "Alt chains and atomic transfers," 2013. [Online]. Available: https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949

[39] Sybil attack. [Online]. Available: https://en.wikipedia.org/wiki/Sybil_attack

[40] A local exchange trading system on top of ergo. [Online]. Available: https://github.com/ergoplatform/ergo/wiki/ A-Local-Exchange-Trading-System-On-Top-Of-Ergo

[41] An ico example on top of ergo. [Online]. Available: https://github.com/ergoplatform/ergo/wiki/An-ICO-Example-On-Top-Of-Ergo