比特币技术调研报告

目 录

[1. 概述 3](#_Toc519264669)

[2. 信息安全技术 3](#_Toc519264670)

[2.1. ECC非对称加密技术 3](#_Toc519264671)

[2.2. 数字签名技术 5](#_Toc519264672)

[3. 数据模型 5](#_Toc519264673)

[3.1. 账单（Transaction） 5](#_Toc519264674)

[3.2. 区块（Block） 7](#_Toc519264675)

[4. 写入流程 8](#_Toc519264676)

[4.1. 生成账单 8](#_Toc519264677)

[4.2. 接收账单 9](#_Toc519264678)

[4.3. 生成候选区块 10](#_Toc519264679)

[4.4. 挖矿 10](#_Toc519264680)

[4.5. 广播区块 11](#_Toc519264681)

[4.6. 追加新区块 12](#_Toc519264682)

[5. 冲突解决 13](#_Toc519264683)

[5.1. 理想情况 13](#_Toc519264684)

[5.2. 现实情况 13](#_Toc519264685)

[5.3. 双重支付 14](#_Toc519264686)

[6. 区块链的局限 15](#_Toc519264687)

[6.1. 系统局限 15](#_Toc519264688)

[6.2. 增加块大小？ 15](#_Toc519264689)

[6.3. 一些解决方案 16](#_Toc519264690)

[**隔离证明** 16](#_Toc519264691)

[**闪电网络** 16](#_Toc519264692)

# 概述

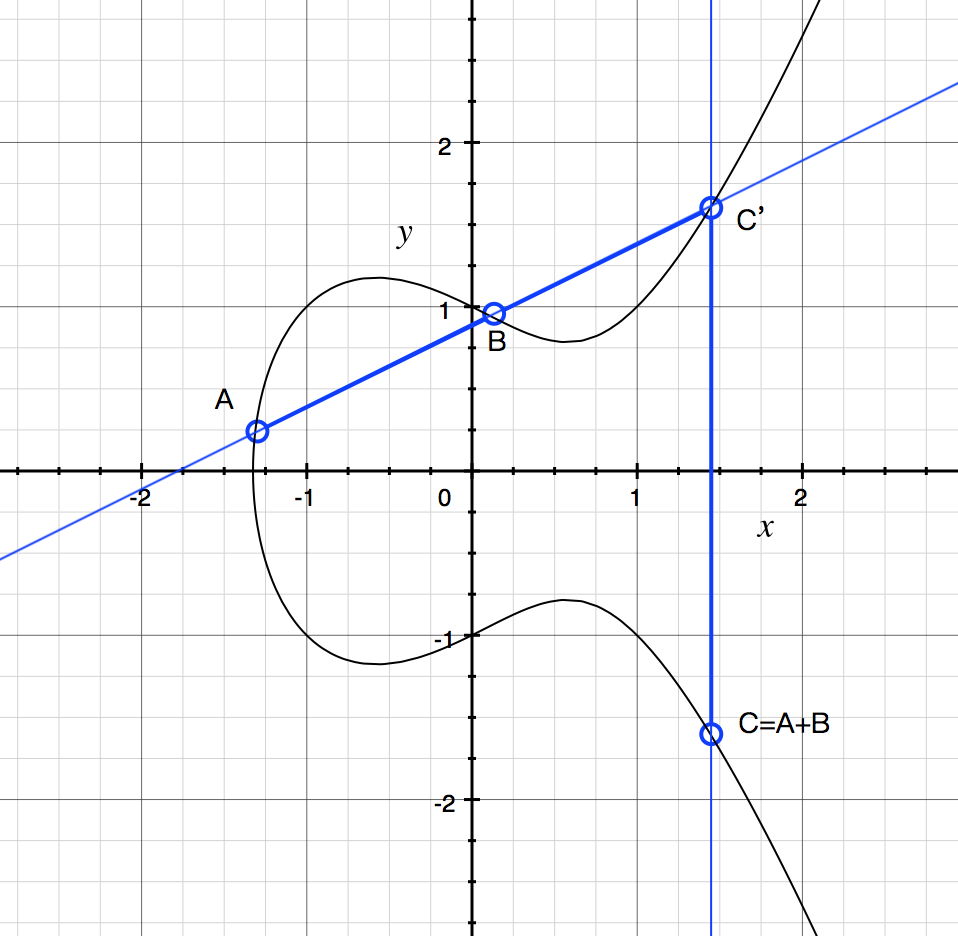
本文档主要介绍了比特币的产生背景，以及比特币系统的设计思想与原理。面向的读者为有一定计算机、数据库基础知识的技术人员。

比特币是首个得到大规模部署的区块链技术应用。2008年，中本聪提出了比特币的白皮书《Bitcoin: A Peer-to-Peer Electronic Cash System》，2009年比特币系统上线运行，至今稳定运行了9年时间，从系统稳定性角度看，是一个非常成功的分布式系统了，无论在信息技术历史还是在金融学历史上都具有十分重要的意义。比特币项目在诞生和发展过程中，借鉴了来自数字货币、密码学、博弈论、分布式系统、控制论等多个领域的技术成果，本文的两位作者方向分别为分布式存储系统和密码学。

# 信息安全技术

## ECC非对称加密技术

椭圆曲线加密本质上是一个交换群，在椭圆曲线的定义中，单位元为无穷远点。给定一条椭圆曲线E和两点A和B，我们可以计算出A+B的值，如下图所示：



那么在ECC中是如何计算的呢，一般而言分为以下三步：

（1）

（2）若P=Q则



（3）若P不等于Q则



利用ECC进行加密首先需要给出p, a, b, G,n。其中p通常选取一个很大的素数以防止穷举，a和b是椭圆曲线的参数，G为给定的椭圆曲线上的点，n为G的阶。椭圆曲线加密正是利用了前面所说的给定自然数m计算mG很容易而给定mG的结果无法很快计算m这一个性质。下面给出一个ECC保密通信的算法：

1. Alice选定椭圆曲线，并在上面取一点作为基点G，同时计算G的阶。比如选取,则G的阶为37
2. Alice选取一个常数k作为私钥，并计算出作为公钥。比如选取k=25,则
3. Alice公开
4. Bob接收到信息之后首先将信息编码，并产生一个随机整数，假设r=6，要加密的信息为3，则，因此计算出y=28，则M(3,28)
5. Bob计算并发送给Alice
6. Alice计算即可以得到明文，因为：



比特币系统中采用的椭圆曲线为在secp256k1中，其参数如下：

p = 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F = 2^256 − 2^32 − 2^9 − 2^8 − 2^7 − 2^6 − 2^4 − 1

a = 0， b = 7

G=(0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798, 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8)

n = 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

h = 01

## 数字签名技术

签名的作用无非就是证明某个文件上的内容确实是我写的/我认同的，别人不能冒充我的签名（**不可伪造**），我也不能否认上面的签名是我的（**不可抵赖**）。通常而言，我们会利用公钥系统进行数字签名。签名主要分问以下几个步骤：

1. 用户A对外分发自己的公钥，并保管自己的私钥
2. 对需要签名的消息M计算摘要值（哈希）,得到H(M)
3. 用私钥对摘要值H(M)进行加密，得到D，并公布D和M
4. 其他用户可以计算H(M)并用A的公钥解密D，如果解密结果等于H(M)则说明是A本人签名的

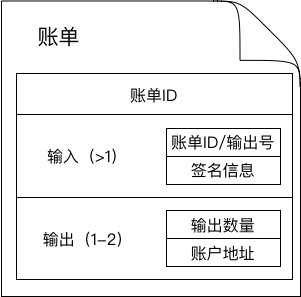
# 数据模型

## 账单（Transaction）

电子货币就是一个电子签名链，我们以一些比特币的拥有者(1)为例，对应上图中间的 Transaction：(1)想将这些比特币转移给下家(2)，需要做两件事：将上一笔账单的hash和下家(2)的公钥进行hash，并用自己的私钥将这个hash签名。同时将这笔账单加到链的末尾。 这笔账单包含了(1)的比特币的由来（上一笔账单），并且包含(1)的签名。其他人可以通过(1)的公钥验证这笔账单，(1)的公钥也包含在上一笔账单中。

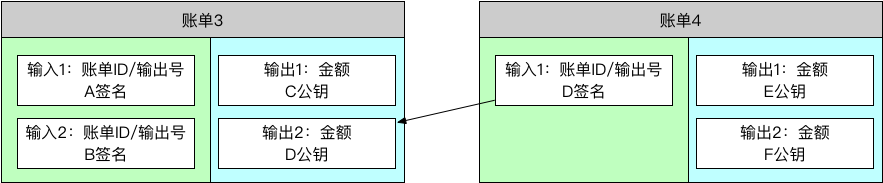
关于账单有一个限制：上一笔账单的输出，如果要用到下一笔账单中，就需要全部消费掉。每笔账单可以有多个输入和最多两个输出。输入可以是一个大金额的比特币，或者是很多少量的比特币。一个输出是收款方，另一个输出是找零。

比如我有3笔分别收入0.2个比特币，并且需要将 0.5 个比特币转给 A，这时这笔账单就有三个输入，分别是之前三笔账单的输出；这笔账单有 2 个输出，其中一个是将 0.5 个比特币输出给 A。多余的 0.1 个比特币找零给我。当一笔账单的输入总和大于输出总和时，差值会当做手续费付给将这个区块写入区块链的人。因此，一般不会将所有零钱返回给付款方，还会支付一些手续费。下图就是一笔账单：

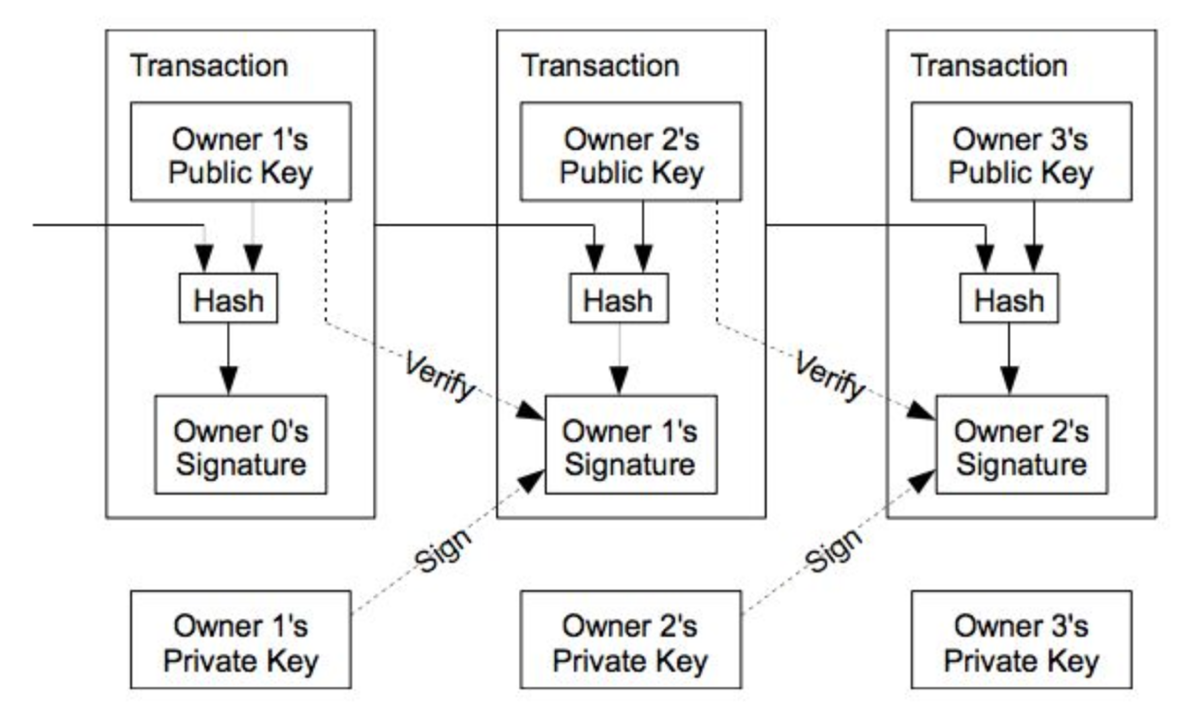


账单还会包含时间戳之类的属性信息，但是时间戳不是用来唯一确定一笔账单的，是用来给用户展示的业务逻辑。在数据库中很少有人用时间戳来标识一个数据，因为时间不准确，分布式系统不存在统一的全局时钟。唯一标识一笔账单的是账单ID。

每个账单可以包含多个输入和至多两个输出，其中输入包含此次消费的比特币来源：账单号+输出。输出包含金额，以及转给的账户公钥。



以上图中 账单4 为例，D 是一个人，他拥有一对公钥和私钥，私钥可以用来签名，这里签名是对整个 账单4 的内容 hash 进行签名。在账单3的输出中，D 收到了2.5个比特币，在账单4中，他要把这2.5个比特币花掉，转2个给E，0.5个给F，于是他签名授权了账单4，账单4就和账单3连上了，这就是账单链，在账单链上记录了比特币从开始运行到现在的所有账单，每一笔账单都可溯源。

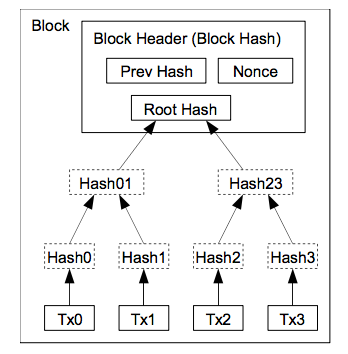


## 区块（Block）

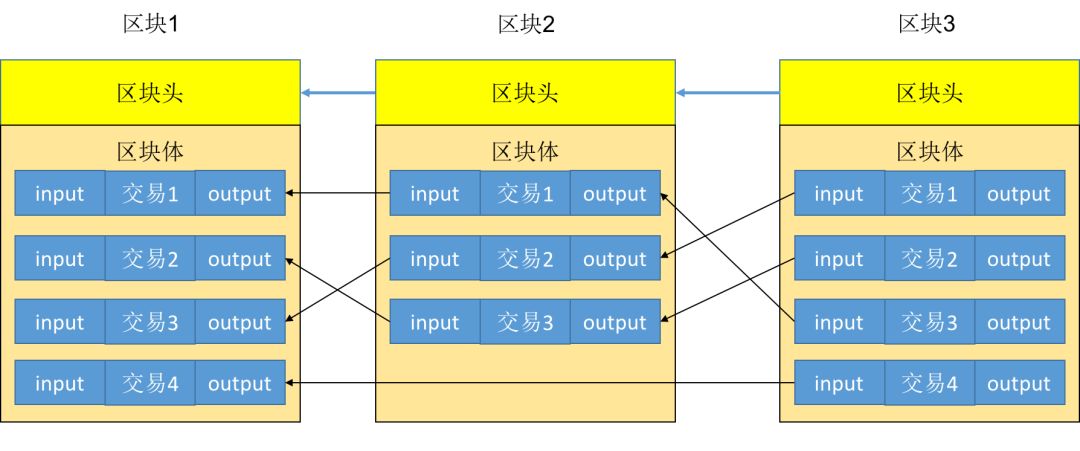
一笔账单就写一次区块链太浪费资源，数据库都喜欢批量写。比特币也不例外，多笔账单会打成一个包，叫做区块，区块链上的基本单位就是一个一个区块。每一个区块主要包括区块头和区块体。

区块体：包含一笔笔账单的数据，将每笔账单当做 Merkle Tree 的叶子节点，计算出根节点的 hash 记做 Root Hash，如果不成对就和自己做 hash。通常每笔账单的大小是 500 字节，而每个区块的大小在代码中写死了不超过1MB，基本每个区块可以装 2000 笔账单。

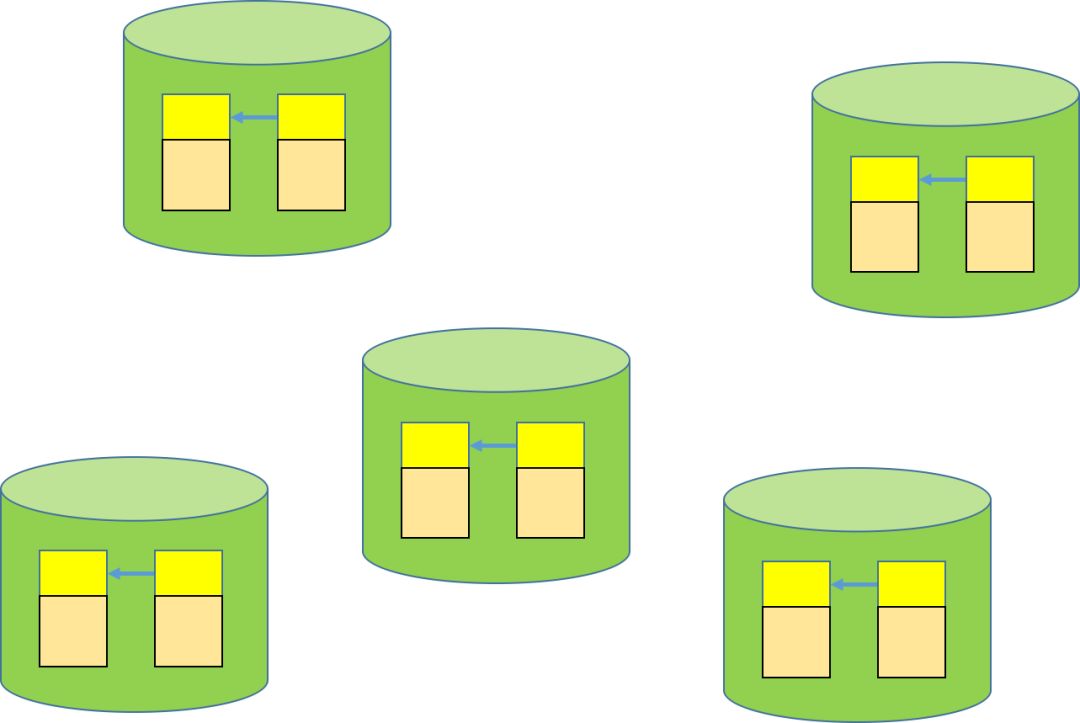
区块头：主要包括上一个区块的hash，上边提到的 Root Hash，时间戳，和一个可以随意填充的数：nonce。



因此，区块(Block) 就通过头部的 hash 值连起来了，注意区块链只能后边连前边的。完整的区块链示意图如下图。其中每个区块包含相同数量的账单，也可能不同。不仅区块连起来了，内部的账单也连起来了。



上图是比特币的基本结构，从比特币系统上线之日起，这个链就在不断增加。这个数据链包含了比特币世界的所有账单信息，这个链在比特币系统的每个节点都会存一份。完整的比特币系统如下图，各个节点之间都可以互联互通，且各个节点是平等的。



# 写入流程

## 生成账单

A 是一个真实的人，他想给 B 转账，A 需要使用比特币的客户端，选择转账的金额和 B 的钱包 ID，点击发送。比特币客户端可以获得 A 的地址，也就是 A 的钱包ID和电子签名。于是，一笔账单所需要的信息除了上一个账单的输出就都具备了。



## 接收账单

当一个比特币节点接收到一笔账单后，需要做如下操作：



广播账单的过程是异步的，不用等其他节点回复，其他节点接收到新的账单信息后也会做广播。

## 生成候选区块

假设系统的每个节点都有向区块链上写入数据的能力（其实某个节点往区块链里写数据就是往本地磁盘写）。每个节点将自己收到的有效账单信息放入一个区块里，存到本地，我们称为候选区块。

## 挖矿

每个区块头部有一个 nonce，挖矿的本质是使用计算机的计算能力找到一个合适的 nonce，使得这个区块的所有信息 hash 后的小数足够小，比如前70位都是0。因为 nonce 的改变会影响整个 hash 值，所以改变这个 nonce 值就可以改变 hash 后的值。由于这个阈值足够小，需要大量的计算，一般10分钟才会挖到这个 nonce，而挖到 nonce 的节点就有资格向本地区块链上写入这个候选区块了，即证明了工作量。这个挖矿的节点就可以叫做矿工。

为什么这些矿工不合作呢，每人分一段范围的数挖？这是不可行的，由于每个矿工都本地的候选区块都不一样，因此每个候选区块适用的 nonce 值也不一样，也就是说可能你分到了一个根本挖不到矿的范围，所以每个矿工只能自己挖。

为什么会有人想当矿工呢？因为矿工挖到矿会收获比特币。这笔钱会以一个账单的形式记录在每个区块的开头。所以，其实各个矿工需要的就是CPU的计算能力和电，谁的计算能力越高，谁能挖到矿的概率越大，谁就越有钱，谁就能继续增加计算能力。

这样看，矿被挖到的时间会越来越短，因为计算机的计算能力是逐年升高的。这里，中本聪又设计了可以动态调整挖矿难度，也就是 hash 之后的值需要小于的那个阈值是可调整的。你挖的快了就调低一点，挖的慢了就调高一点。使整个区块链稳定在每10分钟增加一个区块这个水平。

因此，挖矿这个过程详细一点就是：挖属于自己候选包的矿，并将这个候选包写入本地区块链。



## 广播区块

当写入这个区块 B 后，矿工需要广播这个新写入的区块 B 给其他所有节点。其他接收到区块的节点的流程为：



其他节点会验证接收到的区块 B 是否满足要求，即是否 hash 小于阈值，且账单合法，账单合法的意思是没有重复消费比特币。如果不满足要求则不承认此区块。如果满足要求，就把该区块写到本地最长区块链的末尾。

## 追加新区块

5.5步只将这个区块写到了本地。如果产生新的区块就追加到 B 的末尾，继续广播新的区块。也就表明这个节点是相信 B 的有效性的，原意往 B 后边加数据。

一种理想的工作模式是：严格按照每10分钟有一个节点挖到矿，且一个区块在10分钟之内可以到达所有节点。

举个例子：比特币节点有几千个，1:00有一个节点挖到矿了，可以向区块链中写入区块，并在1:10分钟之前将这个区块广播给所有其他节点，而其他节点也都验证过了这个区块，并把这个区块加到本地链上。此时所有节点数据都一致了。1:10 时又有一个节点挖到矿了，继续这个循环。

这样整个分布式系统中的数据总是一致的，不一致的时间不超过10分钟，由于严格控制了写入速度，实现了系统没有冲突。

# 冲突解决

## 理想情况

我们之前介绍过一种理想的工作模式：严格按照每10分钟有一个节点挖到矿，且一个区块在10分钟之内可以到达所有节点。在这种理想的工作模式下，系统不会出现冲突，即两笔相同的账单被写入两次。现在有两笔账单 A 和 B，不分先后，都要消费同一笔比特币。

（1）第一种情况：A 已经写入了区块链，B 还未写入。首先在某个节点挖到矿准备写入 B 时，根据我们之前的假设，所有节点本地已经同步了最新的区块链，A 已经包含在这个节点的本地区块链中了。每个区块被写入时都需要检查其中的每一笔账单，看之前的链上是否有账单已经消费过这个来源了。因此，检查 B 时，发现 A 已经存在了，消费了 B 的输入来源，因此直接把 B 扔掉。

（2）第二种情况：A 和 B 都还未写入区块链。并且都被一个节点接收，这时这个节点直接在本地检查区块中的所有账单是否有双重支付情况，检查到 A 和 B冲突了，扔掉后来的一个。直接在本地就可以解决并发冲突。

（3）第三种情况：只有一个到达了某个节点，并被写入了区块链，这时会转移到第一种情况。

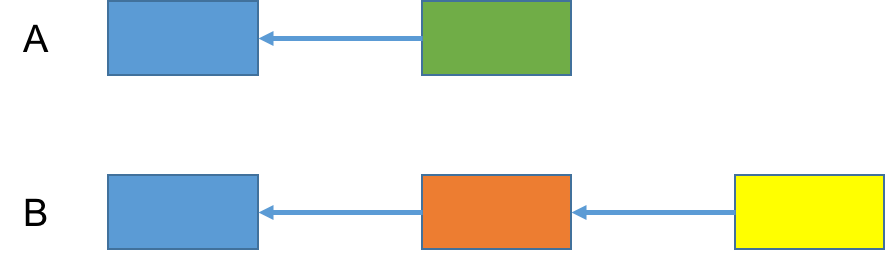
## 现实情况

现实情况是不一定严格10分钟有一个节点挖到矿，可能每隔任意时间就有节点挖到矿了，也可能两个节点同时挖到矿了。而且可能出现网络分区导致一个区块不确定多久会广播到所有节点。

由于节点挖到矿之后，并不需要请求其他节点同意就直接写入本地区块链了，因此很容易出现各个节点维护不同的链的情况。但是，毕竟是个分布式系统，不同的节点总会进行通信的，当他们通信时，就会检测出冲突，类似 git 的分支合并。git 的分支合并冲突是交给用户手动解决的，但是比特币的冲突可以靠系统解决。中本聪给出了一个简单的解决方法：

以最长链作为有效链，新的区块需要加到最长链后，短链会被抛弃。

类似排队上公交车，这时候有两个队，管理员说以最长队为准，这时候后来的人为了保证自己排到前面，都会去排最长的队。这时，短的链就会被丢弃。



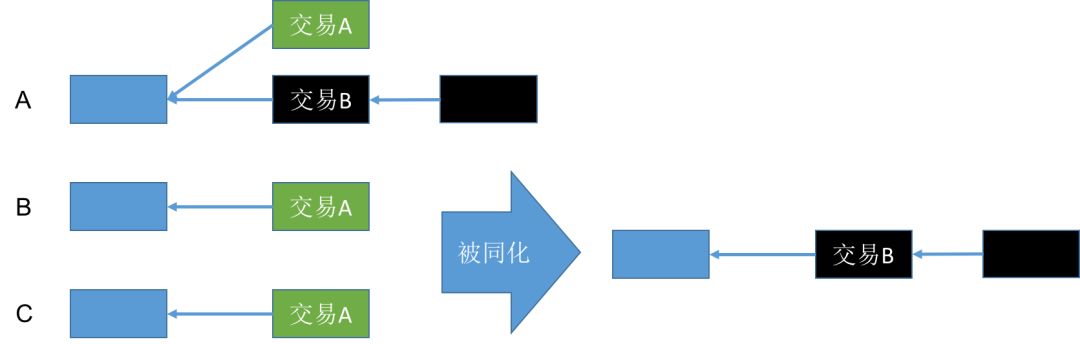
上边是节点 A 的链，下边是节点 B 的链，当节点 A 接收到 B 的链，A 会抛弃绿色块，把 B 的链同步过来，在黄色块继续写。

## 双重支付

比特币作为电子货币，且不利用第三方机构。首先需要解决的问题就是：如何防止双重支付（double-spending）。双重支付简单来说就是一份货币被使用了多次，1块钱当10块钱用。

这个约束在比特币结构的基础上就可以表示成：一个交易的输出最多接一个交易的输入，即最多被消费一次。

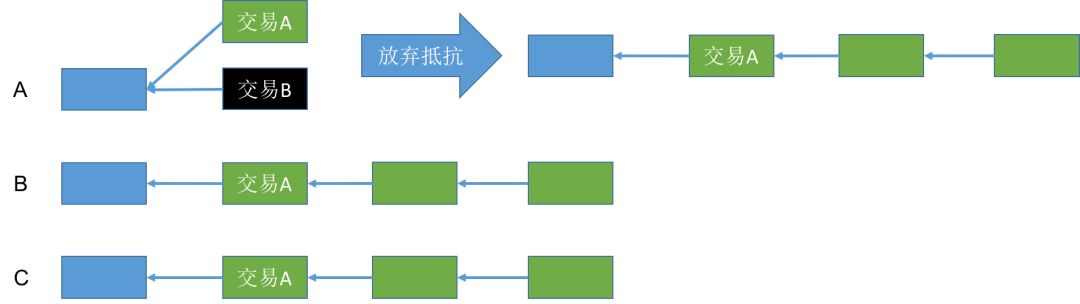
在这种机制下，有没有漏洞让我把一块钱花多遍呢？有的。黑客可以先把交易写入区块链，然后想办法把这笔交易废弃掉，也就是让这笔交易的链分支成为短的分支（再写一个长分支），这样这笔分支就不会被认可了。黑客就可以继续创建一笔新的交易来花这笔钱了。示例图如下：



黑客控制着节点 A。交易A 是第一笔交易，并且已经被广播到其他节点了。为了让交易A 失效，黑客需要重新开一个黑链包含交易B。其实就是和其他节点赛跑，黑客需要先追上绿链，并且超过绿链一个块，这样黑客就可以通化其他的节点了。

而这就需要黑客连着挖到两个矿。这是个什么概念？从概率的期望来说，这就相当于黑客有比特币系统中一半以上的计算资源。黑客都这么强大了，还需要黑系统吗？黑客只需要努力挖矿就能得到很高的回报。

然而更多情况是下面这种：



黑客和别人的计算能力差不多，当黑客挖到一个矿写入一个黑块时，另外两个节点已经挖到两个矿了，他们互相同步数据根本不会理黑客，而这个差距还会越来越大，最终黑客只能选择放弃抵抗。

这里还有一个细节：块是一个一个广播出去的，不是整个链广播出去的，因此，当某个节点接收到一个和自己分支冲突的块时，不会马上丢掉，会先缓存在本地，当缓存的所有块构成的某个分支超过当前工作的分支，就会切换到最长分支上去。

# 区块链的局限

## 系统局限

在传统的关系型数据库的使用场景中，比如银行，为了保证数据不丢失，一般都是做主备的，数据一共有两份，每个节点存储一份，单机存储能力就是整个系统的最大存储数据量。

进入大数据时代，第一个问题就是数据量增多，一个节点的存储能力已经不能满足需求了，因此需要用多个节点来一起存储，每个节点存储一部分数据。如GFS、HDFS等分布式文件系统。在这些系统中为了保证数据不丢失，通常采用三副本机制，即每个数据存储 3 份，分别存放在不同节点。

回来看比特币系统，每个节点存储了全量数据，有几个节点就有几个副本，这个副本量可以达到几千。而他的存储能力也和传统的单机数据库一样，这就必然会导致一个问题，数据多了怎么办？

另一个问题是，我们之前讲过，区块链平均每10分钟产生一个区块，每个区块大概存储 2000 笔账单，那就是每秒 3.3 笔账单。对于一个全球电子货币系统，这个吞吐率是很低的。这么低的吞吐率正好保证了数据不会增长过快，就这样，系统维持在一个慢悠悠的水平。

吞吐率回答了数据多了的问题，因为数据就不会太多。但是，双十一支付宝的峰值是每秒 25.6 万笔。这个吞吐率极大的限制了比特币区块链的应用场景。于是，在比特币的社区，人们也分成了两个阵营表达了对这个问题的看法。

## 增加块大小？

正方：可以合理增加区块大小。

反方：保持区块大小不变，扩大网络规模。

（正方）可以增加每个区块的大小，原来是1M，增加为2M，4M，8M，并随需求逐渐增大。

（反方）增大区块的大小会增加数据量，要知道区块链的设计之初是允许任何人随时加入或离开网络的，每次加入网络需要下载全部数据。数据量的增加也会相应增大节点加入的成本，导致一些小节点无法支撑只能被关闭。这样，比特币系统可能被大公司掌控。

（正方）当前区块链总大小 145GB，每月增加 4G，将块大小翻倍也就是每月增加 8G，亚马逊云平台每月每 G 的费用是 2 美分，因此合理增加块大小不会导致节点退出。

（反方）这个推理太目光短浅了，只翻一倍只能满足一时，未来会不停翻倍，有可能会到达每块 1GB 的大小。到时候普通人就没法运行完整节点了。

## 一些解决方案

### **隔离证明**

这个机制已经被吸收进比特币系统了。基本功能是：在不改变块大小的基础上，使用新的块结构，与原系统无缝集成，达到增加块容纳的交易数的目的。

基本原理：原来每笔交易的输入部分会带一个解锁码（私钥签名+一个预留序列），现在将这些解锁码移到交易的末尾（隔离），并将这些解锁码压缩成原来的 1/4 来使得一个块能容纳的交易适当增多。

### **闪电网络**

在区块链的基础上构建的应用层闪电网络，目的是加速交易。区块链只记录少量验证信息，大量交易数据放在上层应用中