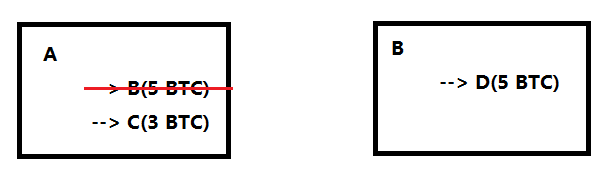
区块链是去中心化的账本，比特币采用的是基于交易的账本模式(transaction-based ledger)，只记录了转账交易和铸币交易，并没有直接记录每个账户上有多少钱。如果想知道某个比特币账户上有多少钱，要通过交易记录来推算。

1. 新型数据结构—UTXO

比特币中的**全结点**要维护一个叫UTXO(Unspent Transaction Output)的数据结构，即还没有被花出去的交易的输出。一个交易可能有多个输出，被花掉的就不在UTXO里了。如下图中A转账给B和C的交易，B的已经花出去了，也就不在UTXO里了，而C的还没花出去，就在UTXO里。



UTXO中的每一条内容要给出产生这个输出的交易的哈希值，以及它在这个交易中是第几个输出。用这两个信息就可以定位到一个确定的交易中确定的输出。

使用UTXO可以用来快速检测双花攻击，想知道新发布的交易是不是合法的，要查一下全结点存在内存中的UTXO。要花掉的币只有在这个UTXO这个集合里才是合法的，否则要么是不存在的，要么是已经花过了的。

随着交易的发布，每个交易会消耗掉一些输出，同时也会产生一些新的输出。例如上图例子中B将5个BTC给D，这消耗掉了前面A把5个BTC给B的输出，但产生了新的B把5个BTC给D的输出，这个输出会保存在UTXO里。

|  |
| --- |
| 如果一个账户从交易中获得比特币之后，不花出去，那么这些输出就要一直保存在UTXO里。有的人是一直不花，比如中本聪，有的是把密钥丢了没法花，但不论怎样这些没消耗掉的交易输出都要一直存在UTXO里，目前的大小是能保存在一个普通服务器内存中的。 |

**交易中的一个原则是total inputs=total outputs**

每个交易可以有若干输入和若干输出，所有输入的金额之和要等于所有输出的金额之和。但这不是绝对的，有些交易的总输入可能略微大于总输出，如可能总输出是1个BTC，总输出是0.99个BTC，这之中的差额就作为记账费给了获得记账权的那个结点。这是比特币设计的**第二个激励机制：交易费(transaction fee)**

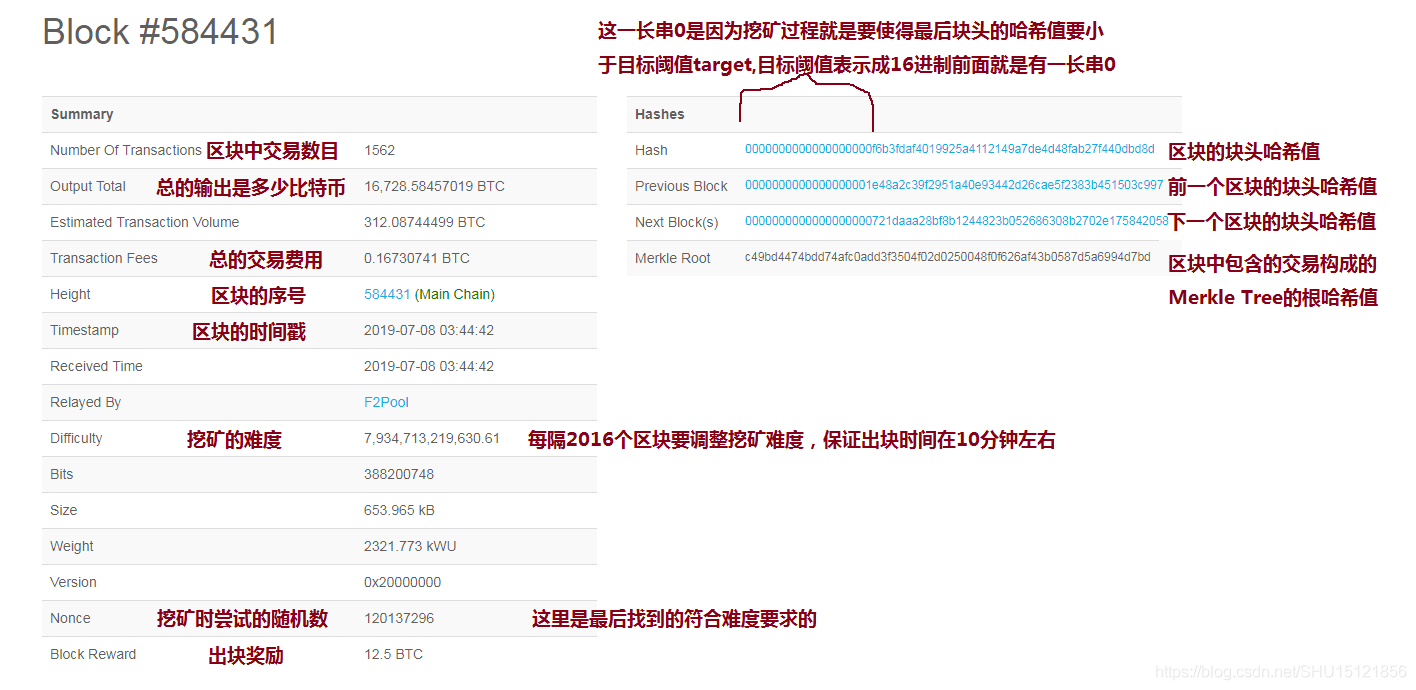
|  |
| --- |
| 这样设计是因为仅仅为获得记账权的结点给予出块奖励是不够的，获得记账权的结点为什么要把某些交易记下来？这样做对他有什么好处呢？一个结点完全可以只打包自己的交易，记录别人的交易不仅要去验证其合法性，而且一个区块装的交易多了，在网络上传输的带宽也会比较多，在网络上传输的速度也会慢。这里的差额作为记账费就解决了这个给别人记账的动机的问题。 |

目前来讲矿工挖矿，主要的目的还是为了第一个激励机制——得到出块奖励。因为出块奖励是要逐渐减少的，每隔21万个区块就要减半，比特币系统的平均出块时间是10分钟，大约每隔4年出块奖励就会减半。到很多年之后，出块奖励变得很小，这时候交易费就成为主要动机了。

除了比特币系统这样基于交易的账本模式(transaction-based ledger)，还有一些系统是基于账户的模式(account-based ledger)，比如后面要学的以太坊。在这种模式中系统要显式的记录每个账户中有多少个币。比特币系统的这种模式，隐私保护性比较好，但会带来一些代价，如转账交易要说明币的来源(币是从之前的哪个交易的哪个输出中来的)以防止双花攻击。

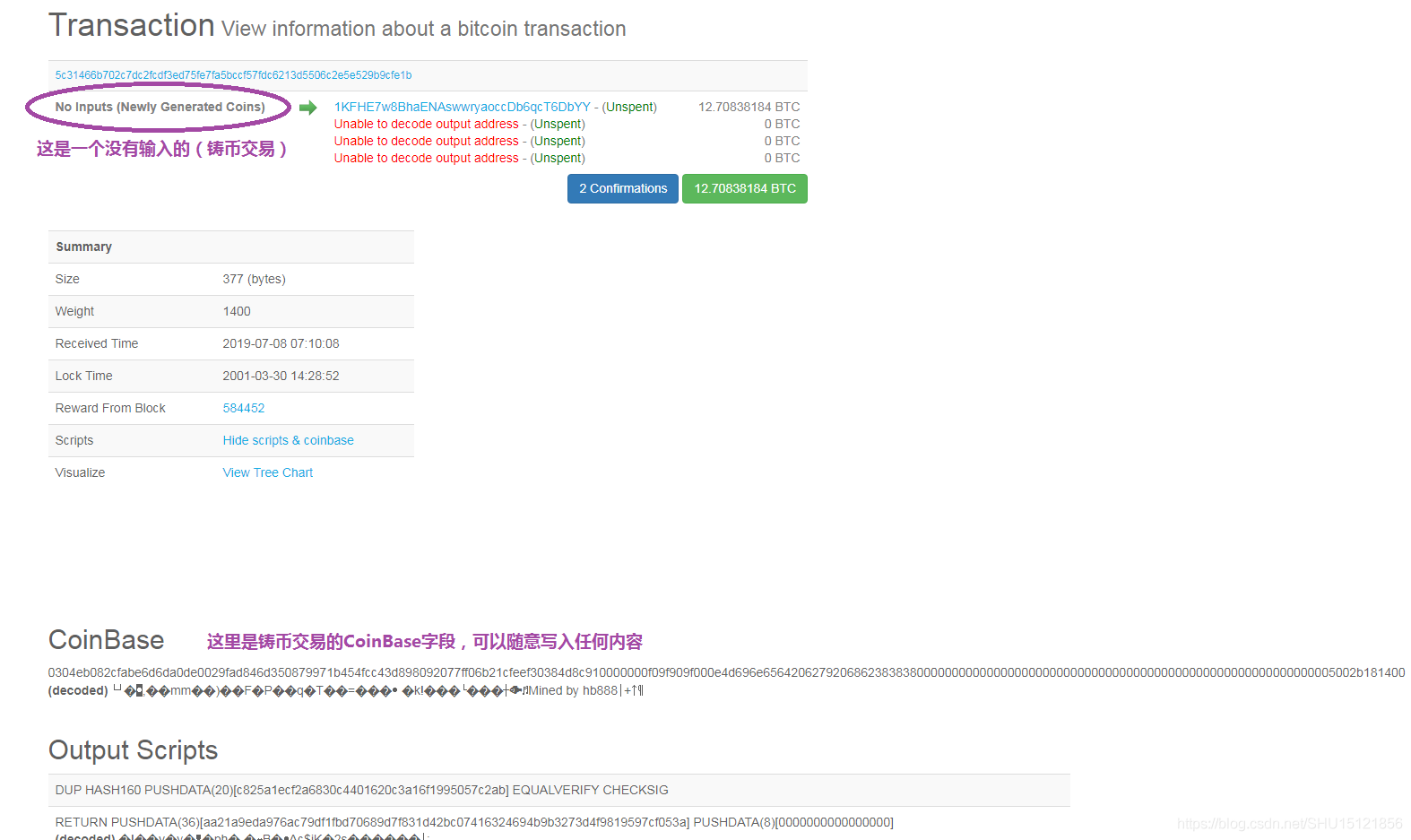
1. 比特币区块实例

在blockchain.info上截图下来的一个区块：



注意，区块中的nonce是4字节即32位整数，也就只种取值。因为比特币近些年太火爆了，挖矿的人数很多，所以挖矿的难度被调整的很高，单纯靠调整nonce是很可能得不到符合难度要求的解的(Hash函数是一对一输入输出的，输入的空间不够大输出自然无法达到要求值)。所以区块的块头中哪些域可以改？再来回顾一下块头中的域(括号中是占的字节数)：

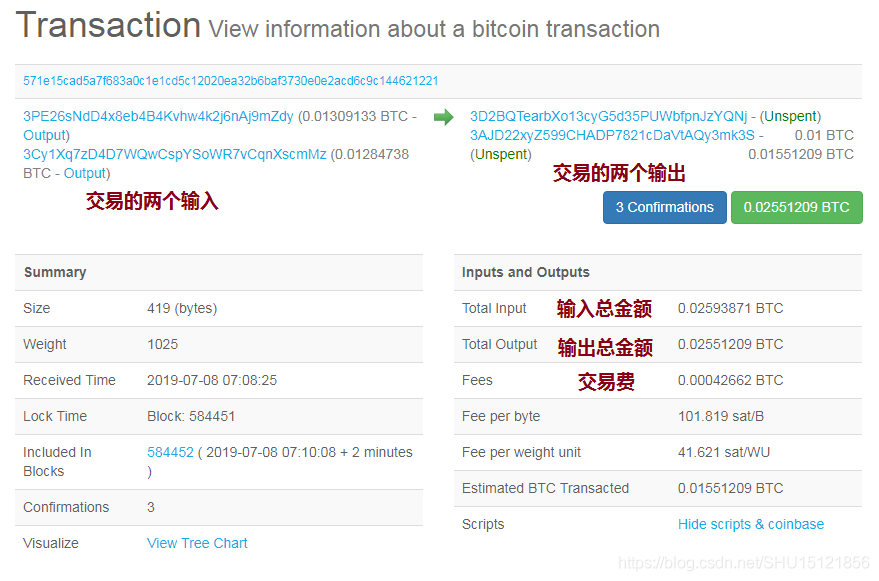
* 版本号(4)：不能改
* 前一个区块块头哈希值(32)：不能改
* Merkle Tree的根哈希值(32)：通过修改Merkle Tree中铸币交易的CoinBase域来调整其根哈希值
* 区块产生的时间(4)：有一定的调整余地，比特币系统并不要求非常精确的时间，这个域可以在一定范围内调整
* 挖矿目标阈值编码后的版本(4)：只能按照协议中的要求定期进行调整，不能随便改
* nonce(4)：可以改



可以看到，因为铸币交易是没有交易来源的，所以可以在其CoinBase域里随便写入内容，铸币交易的变化会使该交易的哈希发生变化，变化沿着Merkle Tree一路向上传递，最终使整棵Merkle Tree的根哈希值发生变化，间接地调整块头的哈希值。

所以可以把这个字段当做一个extra nonce，块头的nonce字段不够用，就再拿着这个域的一部分字节一起调整，就增大了搜索空间。例如，拿出这个域的前8个字节当做extra nonce，则搜索空间一下子就增大到了。实际挖矿时，一般也是为此设计了两层循环，外层循环调整铸币交易的CoinBase域的extra nonce，然后算出Merkle Tree的根哈希值；内层循环调整块头的nonce，计算整个块头的哈希值。

1. 比特币交易的实例



在这个转账交易中，左边是交易的两个输入(虽然旁边写的是Output，这是表示它们花掉的是之前哪个交易的Output)；右边是交易的两个输出(从绿色Unspent字样可以看到还没有花出去，所以会保存在UTXO里)。



可以看到比特币系统中交易的输入和输出都是用脚本来指定的，验证交易输入输出的过程就是把输入脚本和输出脚本配对执行(不是把同一个交易的输入输出脚本配对执行，而是把这个交易的输入脚本和提供币的来源的那个交易的输出脚本配对执行)。只要配对后都能成功执行，交易验证就是通过的。

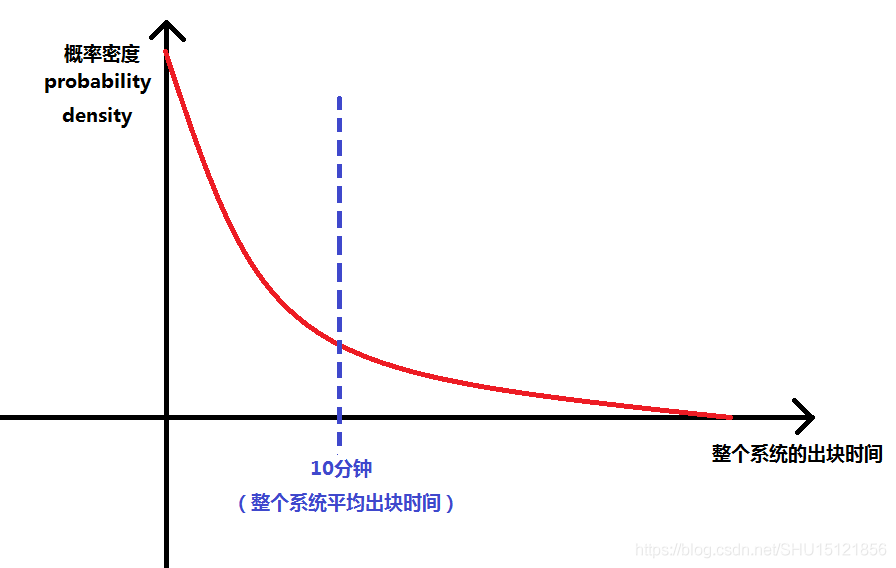
1. 挖矿过程的概率学分析—如何保证挖矿的公平性

挖矿的过程就是不断尝试nonce去求解puzzle，每次尝试可以看做一个伯努利试验(Bernoulli trial：a random experiment with binary outcome)。掷硬币就是一个最简单的伯努利试验，要么正面朝上要么反面朝上，这两个概率不必一样大，对于挖矿而言，成功和失败的概率相差非常悬殊，成功的概率很小。

当进行了大量的伯努利试验，这些伯努利试验就构成了伯努利过程(Bernoulli process：a sequence of independent Bernoulli trails)。伯努利过程的一个性质是无记忆性(memoryless)，即做大量的试验，前面的试验结果对后面没有影响，例如掷硬币很多次都是反面朝上，下一次掷硬币正面朝上的概率也不会增加。

当伯努利分布(也即二项分布)的n很大而p很小时(试验次数很多，每次试验成功概率很小)，可以近似为泊松分布。这里挖矿就是一个n很大p很小的伯努利过程，所以可以近似为泊松过程(Poisson process)。

比特币的概率学特性，尤其是无记忆特性对其公平性进行了保证。在比特币系统中，出块时间是服从指数分布(exponential distribution) 的，整个系统的出块时间按照比特币协议被调整在10分钟左右，而具体到某个矿工的出块时间，取决于其算力占整个系统中矿工的算力的百分比，这比较好理解，例如某个矿工的算力能占到整个系统总算力的1%，那么平均100个区块有1个是他挖到的，也即平均要等1000分钟能挖到1个区块。



这个出块时间服从的指数分布也是无记忆性的， 也就是说从任何一个位置将其截断，剩下的部分仍然是服从指数分布的。“将来还要挖多少时间”和“过去已经挖了多少时间”是没有关系的。体现在比特币系统的挖矿中也就是，不管大家已经挖了多长时间，接下来系统中要出块的平均时间仍然还是10分钟左右。

这也就是progress free——过去做了多少工作不会让后续成功的概率变化。这个性质看似无情但是是必要的。假设一个加密货币系统不满足progress free，即过去做的工作越多，后面成功的概率就越大，那么就会造成算力强的矿工会有不成比例的优势，而不能按照算力的比例计算优势。

1. 比特币总量的分析

出块奖励是系统中产生新的比特币的唯一途径，而出块奖励每隔21万个区块(大约每隔4年)要减半，所以新产生的比特币的总量就形成了一个几何序列(geometric series)。



这是一个收敛级数，计算可得：



这就是系统中所有比特币的总量。

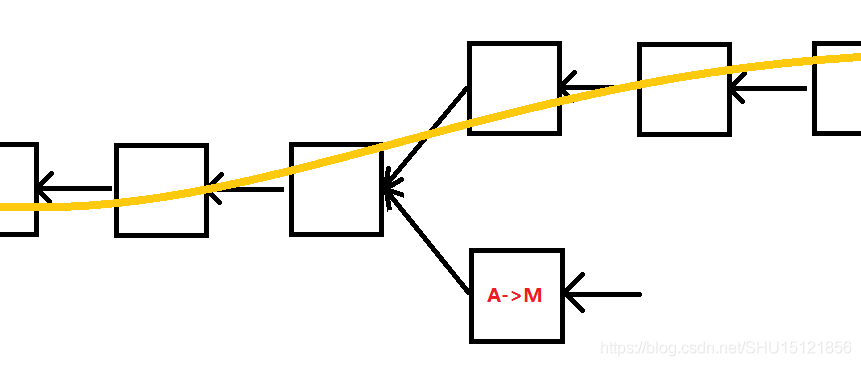
认为比特币挖矿是在解决某种数学难题是错误的观念，求解比特币挖矿的puzzle除了比拼算力之外，没有任何实际意义。比特币越来越难挖只是因为出块奖励人为地被减少了，并且加入的人越来越多而为保持系统中的平均出块时间而提高了挖矿的难度。但要注意，挖矿的过程虽然没有实际意义，但对维护比特币系统的安全性是至关重要的。Bitcoin is security by mining。挖矿提供了一种凭借算力投票的有效手段。

1. 比特币安全性的分析

大部分算力掌握在诚实的结点手里，只能说有比较大的概率下一个区块是由诚实的矿工发布的，但是**不能保证记账权不会落在有恶意的结点手里。**

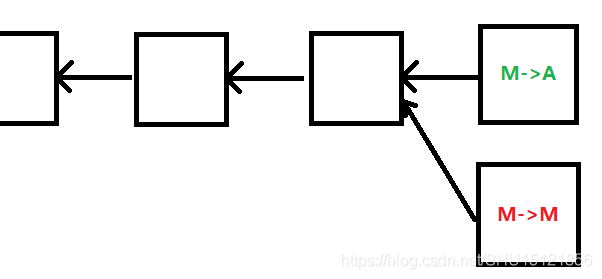
* 转走别人的BTC

假设一个有恶意的结点M获得了记账权，它想把结点A的钱转走，但因为没法伪造A的签名(没有A的私钥)，写个任何不正确的签名上去，都会导致诚实的结点不会接受这个候选区块，而是继续沿着上一个区块扩展。因为这个区块是不合法的，所以无论恶意节点及其同伙将此链扩展多长都不是最长合法链，这样的攻击是无效的。



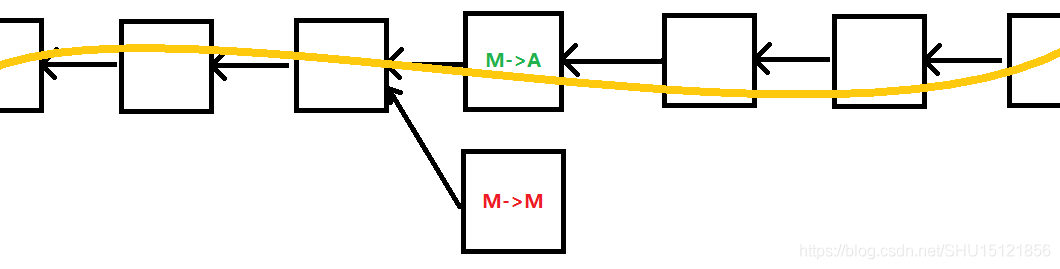
* 分叉攻击

M把BTC转给A，然后就紧接着挖矿挖到了一个区块，在这里填写了M把BTC转给自己的交易，以希望沿着这个区块成为最长合法链，这样就能将转给A的挤掉，从而将花出去的BTC回滚。这也是双花攻击的一种。

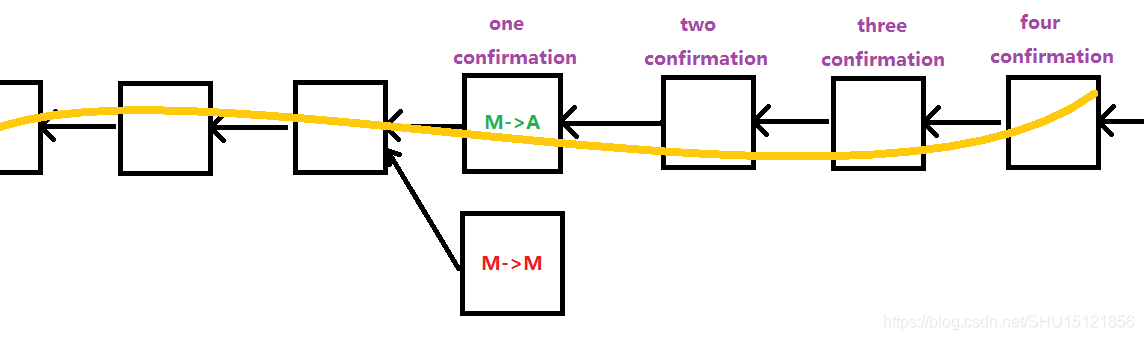


试想A是一个购物网站，允许BTC支付，在M->A这个交易刚写入区块链以后A就认为M支付成功，那确实会出现上述问题。

但如果M->A这个交易所在区块后面又跟上一些区块之后呢？这个攻击的难度就会大大增加。因为它最好的方式仍然是在M->A的前一个区块位置插入，但是想让它成为最长合法链却非常难，因为它已经不是最长合法链，诚实的结点只会去扩展最长合法链。



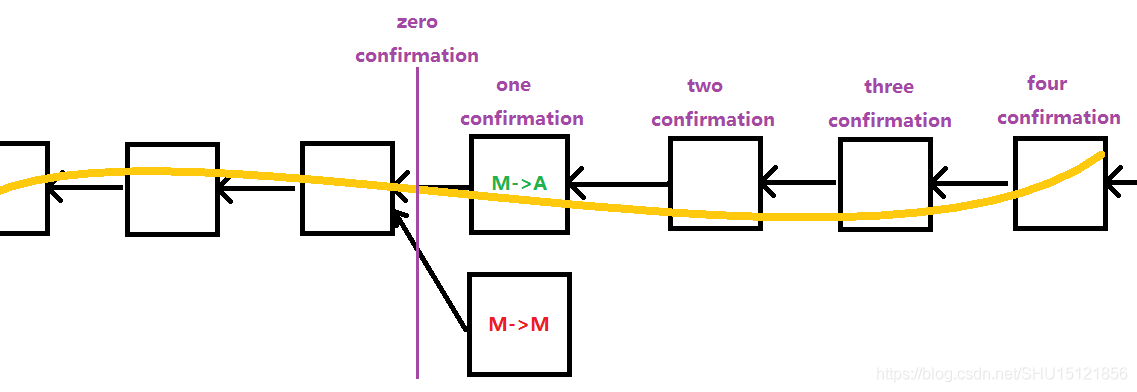
如果大部分结点掌握在诚实结点手里，这样攻击的难度非常大，有恶意的结点要连续获得好多次记账权才可能改变最长合法链。所以对于此种攻击一种最简单的防范方法就是多等几个区块，也叫多等几个确认(confirmation)。



比特币协议中缺省的是要等6个confirmation(大约一小时)，才认为one confirmation区块中的交易是不可篡改的。从前面的分析可以看出，比特币交易在区块链账本上的不可篡改性(irrevocable) 只是一种概率上的保证。

|  |
| --- |
| 除了上述这种缺省的6个confirmation的方式，BTC协议还有另外一种zero confirmation的方式。 |

zero confirmation是指交易刚发布出去，还没有写入区块链中的时候，就认为交易已经不可篡改了。



zero confirmation实际使用的比较广泛，有两个原因：

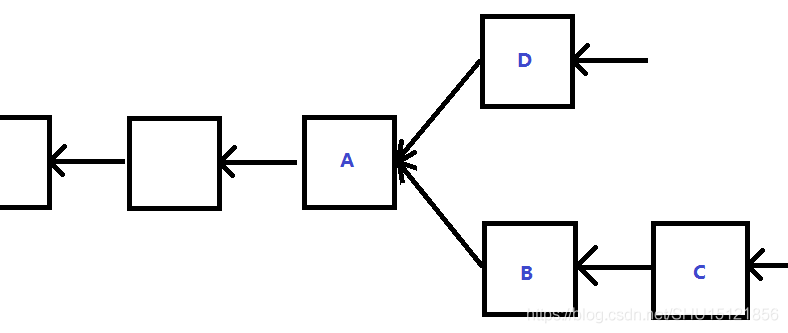
1. 两个交易有冲突，结点接收最先听到的交易。上面分叉攻击的例子中，网络中的节点们在收到M->A后，对于又收到的M->M，大多诚实结点会将其拒绝。
2. 购物网站委托全结点监听区块链，从支付成功到发货其实还有比较长的处理时间，如果发现这个交易最后没有写到最长合法链，购物网站还可以选择取消发货。(从这个方面看，zero confirmation主要是为了解决BTC协议默认确认方式即时性弱的缺点)

* selfish mining

正常情况下结点挖到一个区块就立即发布，这是为了得到出块奖励和收取交易费。selfish mining就是挖到的区块都留着，这样的动机是，比如在前面的分叉攻击中，一直等到6个confirmation过了，再一口气把算好的很长的分叉发布出去，替换掉最长合法链。

实际上这样做的难度还是很大，因为这个恶意结点的算力要超过那些诚实的算力才可能在一定时间后比它长。另外就是大多诚实的结点已经扩展那个M->A的交易所在的区块了，这个恶意结点的同伙结点也要很多才行。

有时候，selfish mining也不是为了进行恶意攻击，而只是为了赚取出块奖励和收取交易费。因为selfish mining也有好处——能够减少自己的竞争对手。比如下图中大家都在从A挖下一个区块，然后某个结点挖出了B先藏着，这时候别人还在从A挖下一个区块，然后这个结点紧接着挖出了C，将B和C一起发布出去，这样就少了一个结点C的竞争。或者是继续往下挖，当听到有人发布D时，将B和C一起发布出去，这样最长合法链是沿着ABC的，别人挖出的D就作废了。



但这样会带来不小的风险，假设在挖出C之前就有人挖出D并且发布了，这时候就只能赶紧把B发布出去，很可能连这个记账权都竞争不到了。在这个动机下，selfish mining的回报并非很高，只是让别人做了无用功，自己少了些竞争，但风险却是挺大的。