

# 05 比特币实现

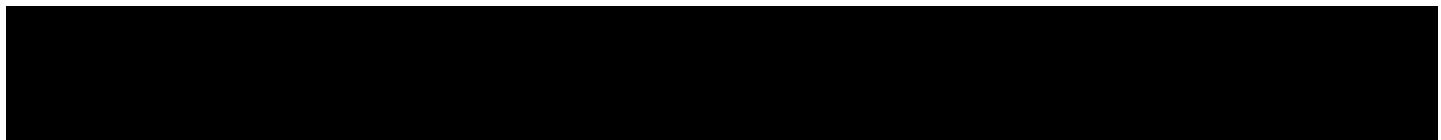
比特币是transaction based ledger（不显示每个账户具体的币数），以太坊是account based ledger（显示每个账户的具体币数）

全节点需要在内存中维护一个叫做UTXO的数据结构，unspent transaction output

在UTXO中，我们只保存没有被花掉的交易输出，也就是UTXO中只保存着没有被交易过的比特币信息。这样方便在验证新的交易的时候，可以快速检测double spending

我们需要一个交易的total inputs = total outputs，但是往往total inputs 比total outputs大，因为争取到记账权的节点需要抽取一些好处费。否则的话争取到记账权的节点完全可以只包含自己的交易记录，或者只包含一部分人的交易记录，而且这样做占用的内存小，在网络上传输占用的带宽也小。所以如果争取到记账权只能获得出块奖励的话，是不能够促进合法交易的写入的。所以比特币系统设计了第二个机制，就是transaction fee。

比特币系统平均每隔10分钟就会产生一个新的区块，那么每隔21万的区块（差不多是4年的时间），出块奖励就会减半。所以随着时间的推移，可能出块奖励会变得很少，那么大家争夺记账权主要就是为了交易费了，目前来说还是主要为了出块奖励。



## Block Example

### Block #529709



Summary		Hashes	
Number Of Transactions	686	Hash	00000000000000000000000000f1531dbfa069037188c5048b23f0cb979ce8728ed8c5
Output Total	4,220.46616378 BTC	Previous Block	00000000000000000000000000841c59a4679d6e707152f24f5b195b66b47397d65175e
Estimated Transaction Volume	651.93844862 BTC	Next Block(s)	
Transaction Fees	0.12458867 BTC	Merkle Root	6f73d36264105e0c55c4703fb5e85e628a29231e9673bcf3c02bfe76dc2b378
Height	<a href="#">529709 (Main Chain)</a>		
Timestamp	2018-06-29 06:17:26		
Received Time	2018-06-29 06:17:26		
Relayed By	<a href="#">BTC.com</a>		
Difficulty	5,077,499,034.879.02		
Bits	389508950		
Size	333.53 kB		
Weight	1160.618 kWU		
Version	0x20000000		
Nonce	3897564446		
Block Reward	12.5 BTC		

source: [blockchain.info](#)



Block Header中包含的一些基础信息，和一些哈希值。其中Difficulty指的是当前的挖矿难度，这个每经过2016个区块之后就需要从新调整从而保证产生一个新的区块的平均时间是10分钟。

...

## Block header

```
13  /** Nodes collect new transactions into a block, hash them into a hash tree,
14   * and scan through nonce values to make the block's hash satisfy proof-of-work
15   * requirements. When they solve the proof-of-work, they broadcast the block
16   * to everyone and the block is added to the block chain. The first transaction
17   * in the block is a special one that creates a new coin owned by the creator
18   * of the block.
19   */
20  class CBlockHeader
21  {
22  public:
23      // header
24      int32_t nVersion;
25      uint256 hashPrevBlock;
26      uint256 hashMerkleRoot;
27      uint32_t nTime;
28      uint32_t nBits;
29      uint32_t nNonce;
```

source: bitcoin/src/primitives/block.h



具体的c++代码

Block headers are serialized in the 80-byte format described below and then hashed as part of Bitcoin's proof-of-work algorithm, making the serialized header format part of the consensus rules.



Bytes	Name	Data Type	Description
4	version	int32_t	The block version number indicates which set of block validation rules to follow. See the list of block versions below.
32	previous block header hash	char[32]	A SHA256(SHA256()) hash in internal byte order of the previous block's header. This ensures no previous block can be changed without also changing this block's header.
32	merkle root hash	char[32]	A SHA256(SHA256()) hash in internal byte order. The merkle root is derived from the hashes of all transactions included in this block, ensuring that none of those transactions can be modified without modifying the header. See the merkle trees section below.
4	time	uint32_t	The block time is a Unix epoch time when the miner started hashing the header (according to the miner). Must be strictly greater than the median time of the previous 11 blocks. Full nodes will not accept blocks with headers more than two hours in the future according to their clock.
4	nBits	uint32_t	An encoded version of the target threshold this block's header hash must be less than or equal to. See the nBits format described below.
4	nonce	uint32_t	An arbitrary number miners change to modify the header hash in order to produce a hash less than or equal to the target threshold. If all 32-bit values are tested, the time can be updated or the coinbase transaction can be changed and the merkle root updated.

The hashes are in internal byte order; the other values are all in little-endian order.  
source: bitcoin.org



在现在，当前的挖矿难度导致单纯的调整nonce值已经不足以找到满足target的值了，所以除了nonce之外我们还可以调整这个区块包含的merkle tree的根哈希。因为每一个区块都会有一个coin based transaction，也就是凭空创造出来的币，这个地方可以修改CoinBase的内容，一旦CoinBase的内容修改之后，带来的改动会影响Merkle tree的根哈希发生改变。

## Transaction

View information about a bitcoin transaction

90ffa15c268a5ae6a0a0381ec2660654fa283fe7cdd7ced583c57738e8dadd3

No Inputs (Newly Generated Coins) → 1C1mCxRukix1KfegAY5zQQJV7samAciZpv - (Unspent)  
Unable to decode output address - (Unspent)

12.62458867 BTC  
0 BTC

1 Confirmations | 12.62458867 BTC

Summary	
Size	243 (bytes)
Weight	864
Received Time	2018-06-29 06:17:26
Reward From Block	529709
Scripts	<a href="#">Hide scripts &amp; coinbase</a>
Visualize	<a href="#">View Tree Chart</a>

## CoinBase

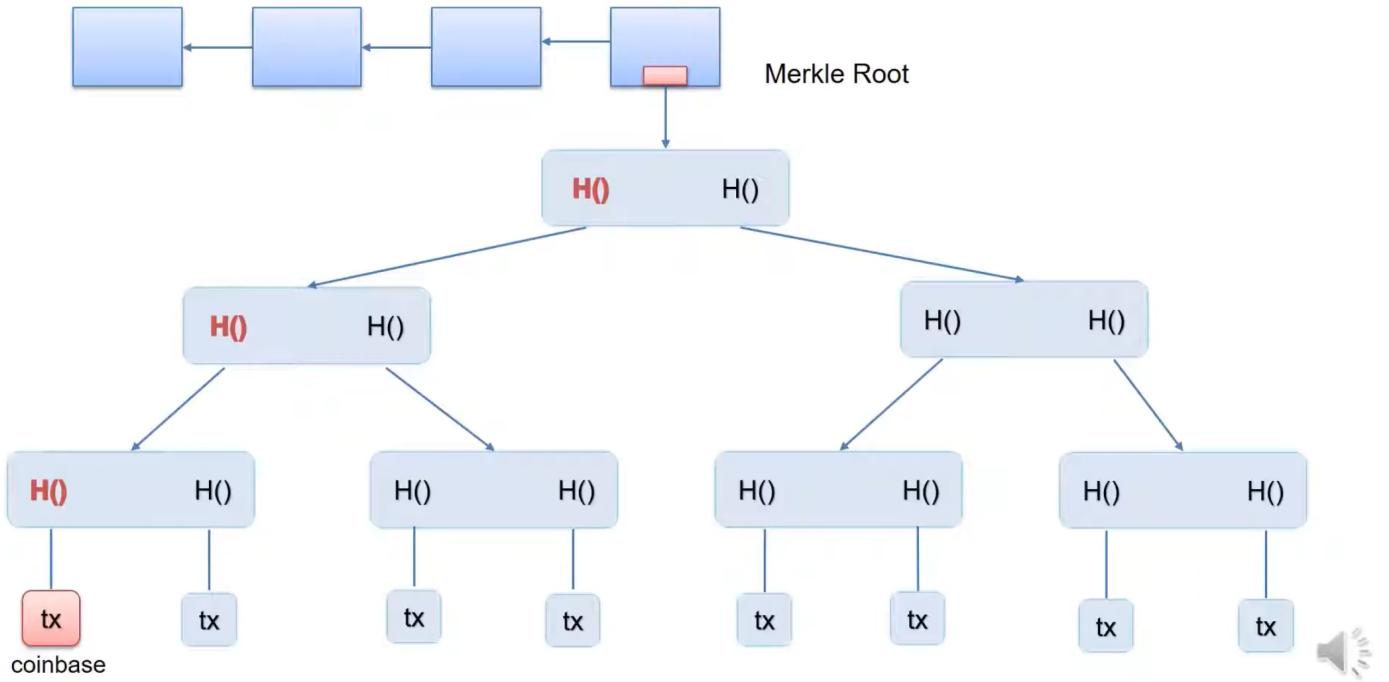
032d150804f6ce355b672f4254432e434f4d2ffabe6d6d1dbc17dbcbc4e98221c0b5f06459f705df3854da8e496d3175096abe84babe01000000000000008566bf194da9010000000000  
(decoded) ↴-→→→→5[gBTC.COM/→mm→→→]→→→!→→→dY→→→8T→lm1u j→→→→→f→→M→→

## Output Scripts

DUP HASH160 PUSHDATA(20)[78ce48f88c94df3762da89dc8498205373a8ce6f] EQUALVERIFY CHECKSIG  
RETURN PUSHDATA(36)[aa21a9ede6643c66fadd737364679ab3c22e03fd44264643ae6692287e54cdbe71b1a0ca]  
(decoded) ↴↑↑↑↑d

source: blockchain.info





## Transaction View information about a bitcoin transaction

3fd0d94dcf733a614f14a930a470241e0d99ea6966f9991fa6fb95396a6645f	1Kqj93Qiqbd1ahG9imm8comNDKpxVvLVF (Unspent)	0.0396 BTC
14BHIEP2sNRUJ5jSexgBjsza87psR2Uv (0.00408688 BTC - Output)	17Eu6pyMUJZhoaEKxj3u8k2D3izdw9QYRT (Unspent)	0.01019031 BTC
	1 Confirmations	0.04979031 BTC
<b>Summary</b>		<b>Inputs and Outputs</b>
Size 373 (bytes)		Total Input 0.05166751 BTC
Weight 1492		Total Output 0.04979031 BTC
Received Time 2018-06-29 06:13:26		Fees 0.0018772 BTC
Lock Time Block: 529708		Fee per byte 503.271 sat/B
Included In Blocks 529709 ( 2018-06-29 06:17:26 + 4 minutes )		Fee per weight unit 125.818 sat/WU
Confirmations 1 Confirmations		Estimated BTC Transacted 0.0396 BTC
Visualize <a href="#">View Tree Chart</a>		Scripts <a href="#">Hide scripts &amp; coinbase</a>

### Input Scripts

```
ScriptSig: PUSHDATA(71)
[304402200e902feaaf8e49ea467bbc3d9034bd33fedfbfb662e75e8822372a3c0871e102204176d40c1781ca6f465d47db3628e2610f53d5a54ceb24e6118296ae71df5e201]
PUSHDATA(33)[03b339dc9b56131ad95753f6995808fcc2c688bad4f09ea0b9bc9e564315c1e515]

ScriptSig: PUSHDATA(72)
[3045022100ade6baa42d209d279033581a593340780181aa0dd8a7fd2d5b6162acd81ca4d022074bd1a62c6138505823efae9ec62d4dd16e57c454a245be25f961a6e8144930201]
PUSHDATA(33)[02cedda2c2a907c010dfa74dc34bc27a17dcab02a88993cdccc243c818279ed]
```

### Output Scripts

```
DUP HASH160 PUSHDATA(20)[cea9fb4638839ad9ebc9c8382dd03e2066aaeb65] EQUALVERIFY CHECKSIG
DUP HASH160 PUSHDATA(20)[4471a74ad7287cbfb6e6d1c092b22b55c886c1de] EQUALVERIFY CHECKSIG
```

source: [blockchain.info](#)



一个普通交易中会有inputs的币的来源，我们需要把来源的output脚本和当前交易的input脚本拼接到一起，如果能够顺利运行，则说明币的来源是正确的，否则就是非法交易。

我们可以用Poisson Process去近似Bernoulli Process。Bernoulli Process是memoryless的，或者称为progress free。就像掷硬币一样，即使前九次都是反面，第十次掷出来正面的概率也不会增加。所以比特币通过协议来设定平均十分钟的出块时间来服从exponential distribution（也是memoryless的，从图像上看，在任何一点切一刀，之后的图像和本身是一模一样的）。比如说我们在挖矿的时候已经挖了10分钟，但是还没有挖到，直觉上我们会认为应该快了，马上就挖到了，但事实并非如此。这样的性质是为了保证挖到矿的概率和算力是成正比例相关的，也就是一个矿工的算力如果是另外一个矿工的十倍，那么这个矿工挖到矿的概率也应该是另外一个矿工的十倍。

Bernoulli trial: a random experiment with binary outcome

Bernoulli process: a sequence of independent Bernoulli trials

memoryless

Poisson process

exponential distribution



比特币的总数是有限的  $21\text{万} * 50 + 21\text{万} * 25 + 21\text{万} * 12.5 + \dots = 21\text{万} * 50 * (1 + 1/2 + 1/4 + \dots) = 2100\text{万}$

有些人认为挖矿是在解决某些数学难题，而比特币越来越挖到是因为符合条件的数字越少。但事实上比特币求解的 puzzle 除了比拼算力之外是没有任何的实际意义的。比特币的稀缺性是人为造成的，通过出块奖励的衰减。

Bitcoin is secured by mining. 只要大部分算力是掌握在好的节点手中，那么整个系统的安全就会得到保证。

对于一个去中心化的系统来说，挖矿提供一种比拼算力的投票手段。所以挖矿从表面上看没有什么实际意义，但是这个机制的设置对于维护整个系统的安全性是十分有效的。

出块奖励的衰减会导致挖矿洞里减少么？事实证明相反，现在挖矿的竞争非常激烈。

挖矿得不到出块奖励之后，主要的挖矿动力就是交易费了。

假设大部分的算力都是好的节点，我们能得到什么样的保证呢？

从概率上讲，只是有很大的概率记账权会落到好的矿工手里，但是坏的矿工依然有几率拿到记账权。

如果坏的矿工拿到了记账权，我们考虑下面的几个问题：

## 1. 坏的矿工能不能通过记账权进行偷窃？能不能把别人账上的钱转给自己？

不能，因为没有别人的私钥，所以没有办法伪造别人的签名。

但如果把这个交易硬写到区块里呢？但是诚实的节点不会接受这个区块，所以诚实的节点会接着上一个节点往后挖。我们认定攻击成功的标准是诚实的节点会接受这个区块。这个攻击的区块的代价是非常大的，因为没有办法得到出块奖励了。

## 2. 可以把已经花过的钱再花一次么？进行double spending么？

假如说M -> A，现在M又获得了记账权，尝试写入M-> M'转给自己。如果直接把这个区块加在最后肯定不行，因为明显是double spending，所有诚实的节点都不会接受这个区块。所以唯一的办法就是利用分叉攻击，沿着记录M->A的那个区块之前的区块进行扩展，这个时候就会出现两条最长合法链。

区块插入在哪个位置，是刚开始挖矿的时候就要决定的，因为需要写入前一个区块的哈希。

其他的节点沿着哪条链进行扩展，哪条链就会胜出，另外一个一条链就会作废。这样攻击的目的在于，M可以通过这样的方式回滚一些交易，使得钱回到自己的账上。比如说M购买了一些商品，转账给了A，然后A间听到交易写入了区块链，就把商品给了M，然后M通过上述的方法进行回滚，又把钱恢复到了自己的账上。

为了防止这样的交易发生，我们可以等M->A的后面出现多个区块之后，再进行发货。这样的话攻击的成功几率很小，因为有恶意的节点只获得一次记账权是不够的，必须把非法的交易拓展成最长合法链才行。这个从概率上来说几乎是不可能的。

M->A的区块被称之为one confirmation，下一个区块是two confirmation，以此类推。经过six confirmation的时候，我们就认为M->A的交易是不可回滚的了。这个平均的等待时间是一个小时，所以还是比较长的。

区块链是irrevocable ledger（不可篡改的账本），但是这个只是概率上的保证。刚刚写入的区块还是比较好篡改的，但是经过几个区块之后，被篡改的概率是成指数下降的。

还有一种方法叫做zero confirmation，就是交易已经发布，但是还没有写到区块链当中。这个时候电商可以运行一个全节点或者委托一个全节点来验证交易的合法性，但不需要等到交易写入区块链当中。这种方法听起来风险很大，但是运用的最多的。但比特币的协议规定只接受最先监听到的交易，所以诚实的节点不会接受后来的回滚交易。第二就是电商交货的期间会有一些时间等待，如果电商发货前发现交易没有被写入到区块链当中，可以取消发货。比如说你买了一个笔记本电脑，在第二天发货前电商没有监听到这个交易被写入了区块链当中，那么就会取消交易。

## 3. 坏矿工能不能故意不把合法的交易写入到区块当中？

但是这个问题也不大，因为总会有诚实的节点愿意写入这些交易。即使好的矿工，也会有一些交易写不进去，因为可能有一个时间段，交易太多的话，内存不够写不进去，就需要等到下一个区块出现才能写。

## 4. 坏的矿工可以挖到很多个区块（比当前的最长合法链要长），但是先隐藏着不发布，等对方把商品给我之后，我再一下子直接全部发布，这样就可以覆盖掉当前的最长合法链，从而达到回滚交易的目的。这个攻击方式被称之为selfish mining。

但是这样的攻击成功概率依然不大，因为只有坏的矿工和他的同伙会按照回滚交易的链继续挖。除非坏的矿工占了一多半，51%以上，这个时候在概率上才有可能成功。

## 5. 如果单纯为了出块奖励，那么self mining有什么好处呢？

如果你藏着挖到了的区块个发布的话，那么其他的算力还会沿着之前的区块挖，但是你自己会沿着新的挖，这样可以减少竞争（让别人做一些无用的功）。但是这样做的话风险很大，除非你的算力很强，别人挖出一个时候，你已经挖出来了两个，这样的话你可以一下发布两个，把别人挖到的那一个给顶替掉了。但风险在于如果别人比你挖的快很多，你藏着的区块可能全都白挖了。或者当你藏了一个区块，还没挖出来下一个的时候，当别人发布了一个区块你可以也将藏的这一个区块进行发布然后进行比拼，看看谁的能最终形成最长合法链。