区块链网课知识点:

第二节:

比特币被称为加密货币crypto-currency

区块链上内容都是公开的，包括区块的地址，转账的金额。

比特币主要用到了密码学中的两个功能:1.哈希2.签名

1.密码学中用到的哈希函数被称为cryptographic hash function: 它有两个重要的性质:

①collision(这里指哈希碰撞) resistance :例如x≠y H(x)=H(y) 两个不同的输入，输出却是相等的，这就称哈希碰撞。它是不可避免的，因为输入空间总大于输出空间。给出x，很难找到y，除非蛮力求解(brute-force)。

该性质的作用:对一个message求digest

比如message取m m的哈希值是H(m)=digest 如果有人想篡改m值而H(m)不变，则无法做到。

哈希碰撞无法人为制造，无法验证，是根据实践经验得来的。

②hiding 哈希函数的计算过程是单向的，不可逆的。(从H(x)无法推导出x) hiding性质前提是输入空间足够大，分布比较均匀。如果不是足够大，一般在x后面拼接一个随机数，如H(x||nonce)。

该性质的作用:和collision resistance 结合在一起，用来实现digital commitment(又称为digital equivalent of a sealed envelope)

把预测结果作为输入x，算出一个哈希值，讲哈希值公布，hiding让人们知道哈希值而不知道预测值，最后再将x公布，因为有collision resistance的性质，预测结果是不可篡改的。

除了密码学中要求的这两个性质外，比特币中用到的哈希函数还有第三个性质:

③puzzle friendly 指哈希值的预算事先是不可预测的。假如哈希值是00...0XX...X，一样事先无法知道哪个值更容易算出这个结果，还是要一个一个带入。

比特币挖矿的过程中实际就是找一个nonce，nonce跟区块的块头里的其他信息合一起作为输入，得出的哈希值要小于等于某个指定的目标预值。H(block header)≤target。block header 指块头，块头里有很多域，其中一个域是我们可以设置的随机数nonce，挖矿的过程是不停的试随机数，使得block header取哈希后落在指定的范围之内。

puzzle friendly是指挖矿过程中没有捷径，为了使输出值落在指定范围，只能一个一个去试。所以这个过程还可以作为工作量证明(proof of work)。

挖矿很难，验证很容易。(difficult to solve ,but easy to verify)

比特币中用的哈希函数叫作SHA-256(secure hash algorithm )以上三个性质它都是满足的。

2.

在比特币系统中开账户:

在本地创立一个公私钥匙对(public key ,private key)，这就是一个账户。公私钥匙对是来自于非对称的加密技术(asymmetric encryption algorithm)。

两人之间信息的交流可以利用密钥(encryption key)，A将信息加密后发给B，B收到后用密钥解密，因为加密和解密用的是同一个密钥，所以叫对称加密。前提是有渠道可以安全地把密钥分发给通讯的双方。因此对称加密的缺点就是密钥的分发不方便，因为在网络上很容易被窃听。非对称密钥是用一对密钥而不是一个，加密用公钥，解密用私钥，加密和解密用的都是接收方的公钥和私钥。公钥是不用保密的，私钥要保密但是私钥只要保存在本地就行，不用传给对方。公钥相当于银行账号，别人转账只要知道公钥就行，私钥相当于账户密码，知道私钥可以把账户上钱转走。公钥和私钥是用来签名。

假如A想向B转10个比特币，A把交易放在区块链上，别人怎么知道这笔交易是A发起的呢?这就需要A要用自己的私钥给交易签名，其他人收到这笔交易后，要用A的公钥去验证签名。签名用私钥，验证用公钥，用的仍然是同一个人的。创建账户产生相同公私钥的可能性微乎其微，所以大量创建账户来窃取其他人账户是不可行的。

我们假设产生公私钥时有一个好的随机源(a good source of randomness)，产生公私钥是随机的，如果随机源不好，就有可能产生相同的公私钥。比特币中用的签名算法，不仅是生成公私钥的时候要有好的随机源，之后每一次签名时也要有好的随机源。只要有一次签名用的随机源不好的话，就有可能泄露私钥。

第三节:比特币的数据结构

普通指针存储的是某个结构体在内存中的地址。假如P是指向一结构体的指针，那么P里面存放的就是该结构体在内存中的起始位置。而哈希指针除了要存地址之外，还要保存该结构体的哈希值H()。好处是:从哈希值这个哈希指针，不仅可以找到该结构体的位置，同时还能够检测出该结构体的内容有没有被篡改，因为我们保存了它的哈希值。

比特币中最基本的结构就是区块链，区块链就是一个一个区块组成的链表。区块链和普通的链表相比有什么区别:

①用哈希指针代替了普通指针(B block chain is a linked list using hash pointers)

区块链第一个区块叫作创世纪块(genesis block) 最后一个区块 是最近产生的区块(most recent block) 每一个区块都包含指向前一个区块的哈希指针

一个区块的哈希指针怎么算:是把前面整个区块的内容，包括里面的hash pointer ，合在一起取哈希值。通过这种结构，可以实现tamper-evident log。如果有人改变了一个区块的内容，后面一个区块的哈希指针就对不上，因为后一个区块哈希指针是根据前一个区块的内容算出来的，所以后一个哈希指针也得改，以此类推，我们保留的是最后一个哈希值也会变化。(见拍的图①)

②普通链表可以改变任意一个元素，对链表中其他元素是没有影响的。而区块链是牵一发而动全身，因为只需要保存最后一个哈希值，就可以判断区块链有没有改变，在哪里改变了。

因此比特币没有要保存所有区块的内容，可以只保留最近的几千个区块。如果要用到以前的区块，可以向系统中其他节点要这个区块。有些节点是有恶意的，怎么判断?这里要用到哈希值一个性质，如下:

其他节点给你一个区块，如何判断它是正确的?算出它的哈希值，与保留的区块的哈希值对比，即可。

比特币中的另外一个结构是:Merkle tree。(见拍的图②，其中最下面一层是数据块(data blocks)，上面三层内部节点都是哈希指针(hash pointers)，第一层是根节点，根节点的区块也可以取个哈希，叫根哈希(root hash))

另外一个概念:binary tree。

这种结构的好处:只要记住根哈希值，就能检测出对树中任何部位的修改。

它们的区别:①用哈希指针代替了普通指针。

比特币当中各区块之间用哈希指针连接在一起，每个区块所包含的交易组织成一个merkle tree的形式，最下面一行data blocks每个区块实际上是一个交易，每个区块分为两部分，分别是块头和块身(block header ,block body)。块头里面有根哈希值，每个区块所包含的所有交易组成的merkle tree的根哈希值存在于区块的块头里面，但是，块头里没有交易的具体内容，只有一个根哈希值，块身里面是有交易的列表的。

merkle tree 的作用:①提供merkle proof

比特币中的节点分为两类:全节点(保存整个区块的内容，即块头块身都有，有交易的具体信息)和轻节点(例如手机上的比特币钱包)(只有块头)

这时存在一个问题:如何向一个轻节点证明某个交易是写入区块链的?

这时需要用到merkle proof :找到交易所在的位置(最底行的其中一个区块)，这时该区块一直往上到根节点的路径就叫merkle proof。

拍的图三:最上面一行是小型的区块链，该图展现的是一个区块的merkle tree，最下面一行是包含的交易。假设某个轻节点想知道图中黄色的交易，是否包含在了merkle tree里面。该轻节点没有包含交易列表，没有这颗merkle tree的具体内容，只有一个根哈希值。这时轻节点向一个全节点发出请求，请求证明黄色的交易被包含在这颗merkle tree里面的merkle proof。全节点收到这个请求之后，只需要将图中标为红色的这三个哈希值发给轻节点即可。有了这些哈希值之后，轻节点可以在本地计算出图中标为绿色三个哈希值。首先算出黄色交易的哈希值，即它正上方的那个绿的哈希值，然后跟旁边红色的哈希值拼接起来，可以算出上层节点绿色的哈希值。然后再拼接，再算出上层绿色哈希值，再拼接，就可以算出整棵树的根哈希值。轻节点把这个根哈希值和block header里的根哈希值比较一下，就能知道黄色的交易是否在这颗merkle tree里。

全节点在merkle proof里提供的这几个哈希值，就是从黄色的交易所在的节点的位置到树根的路径上用到的这些哈希值。轻节点收到这样一个merkle proof之后，只要从下往上验证，沿途的哈希值都是正确的即可。(验证时只能验证该路径的哈希值，其他路径是验证不了的，即该图中红色的哈希值是验证不了的)

这样是否不安全呢?假如黄色交易被篡改，它的哈希值发生了变化，那能不能调整旁边红色的哈希值，使得它们拼接起来的哈希值是不变的呢?不行，根据collision resistance，这是不可行的。

merkle proof可以证明merkle tree里面包含了某个交易，所以这种证明又叫proof of membership或 proof of inclusion。

对于一个轻节点来说，验证一个merkle proof 复杂度是多少?假设最底层有n个交易，则merkle proof 复杂程度是θ(log(n))。

如何证明merkle tree里面没有包含某个交易?即proof of non-membership。可以把整棵树传给轻节点，轻节点收到后验证树的构造都是对的，每一层用到的哈希值都是正确的，说明树里只有这些叶节点，要找的交易不在里面，就证明了proof of non-membership。问题在于，它的复杂度是线性的θ(n)，是比较笨的方法。

如果对叶节点的排列顺序做一些要求，比如按照交易的哈希值排序。每一个叶节点都是一次交易，对交易的内容取一次哈希，按照哈希值从小到大排列。要查的交易先算出一个哈希值，看看如果它在里面该是哪个位置。比如说在第三个第四个之间，这时提供的proof是第三个第四个叶节点都要往上到根节点。如果其中哈希值都是正确的，最后根节点算出的哈希值也是没有被改过的，说明第三、四个节点在原来的merkle tree里面，确实是相邻的点。要找的交易如果存在的话，应该在这两个节点中间。但是它没有出现，所以就不存在。其复杂度也是log形式，代价是要排序。排好序的叫作sorted merkle tree。比特币中没有用到这种排好序的merkle tree，因为比特币中不需要做不存在证明。

这节讲了比特币中两种最基本的结构:区块链和merkle tree，都是用哈希指针来构造的。除了这两种之外，哈希指针还能用另一个方面。

只要一个数据结构是无环的(非循环链表)，都能用哈希指针代替普通指针。有环的话存在一个问题，他们的哈希值没法计算，没法确定一个哈希值固定的区块。

第四节:比特币的共识协议

数字货币和纸质货币区别是可以复制，叫作双花攻击 即double spending attack。

去中心化货币要解决两个问题:①数字货币的发行②怎么验证交易的有效性，防止double spending attack。

答案:①比特币的发行是由挖矿决定的

②依靠区块链的数据结构

比特币的发行者A拥有铸币权(createcoin

) 假如发行10个比特币 A(10)分别给B和C各五个 → B(5)C(5) 该交易需要有A的签名，证明经A同意。(designed by A)同时还要说明花掉的10个比特币从哪来的。

参考拍的图四 第二个方框中的钱是从第一个框内铸币交易中来的。

比特币系统中每个交易都包含输入和输出两部分。输入部分要说明币的来源，输出部分要给出收款人公钥的哈希。

有的交易部分比较复杂，如C的货币来源是第二第三个方框，要标识清楚。

图四就构成了一个小型的区块链，这里有两种哈希指针，一种哈希指针是连接在各个区块之间的，把它们串起来构成一个链表，前面学的就是这种哈希指针。而在该图中还有第二种哈希指针，是指向前面某个交易的指针，用来指明币的来源。为什么要说明币的来源:证明币不是凭空捏造的是有记录的，同时也是防范double spending。

现在来看第二个方框里A向B的转账，该交易需要A的签名和B的地址。比特币系统里收款的地址是通过公钥推算出来的。比如B的地址就是B的公钥取哈希然后经过一些转换得到的。

A如何知道B的地址?比特币系统中没有查询对方地址的功能，必须通过其他渠道。比如某个电商网站，接受比特币支付，就可以公开它的地址或公钥。

A需要知道B的地址，B需要知道A的什么信息吗?B其实也要知道A的公钥，这代表A的身份。不仅是B，所有节点都需要知道A的公钥。而签名是用私钥签名公钥验证(注意不要跟前面知识弄混了，加密是用接收人的公钥加密私钥解密)，所以区块链上每个节点都要独立验证。

那如何才能知道A的公钥?实际上交易里就包含了。输入时不仅要输入币的来源，还要输入公钥。那就存在了安全漏洞，假如B的同伙伪造了这次交易呢?其实第一个方框里铸币交易的输出就有A的公钥的哈希，所以第二个方框交易里A的公钥要跟前面哈希对的上。

在比特币系统当中，前面这些验证过程，是通过执行脚本来实现的。每个交易的输入提一段脚本，包括给出公钥的过程，公钥也是在输入的脚本里指定的。每个交易的输出也是一段脚本，验证其的合法性，就需要把当前交易的输入脚本跟前面交易(提供币来源的交易)的输出脚本拼在一起，然后看看能不能顺利执行，如果能执行说明是合法的。比特币脚本(BitCoin Script)。

该图对交易系统进行了简化，实际上每个区块(对应图中的每个方框)可以有很多交易，这些交易就组成merkle tree。每个区块分为块头和块身。

块头包含的是区块的宏观信息，比如:用的是比特币哪个版本(version)的协议，区块链当中指向前一个区块的指针(hash of previous block header)，整颗merkle tree 的根哈希值(merkle root hash)，还有两个域是跟挖矿相关的，一个是挖矿的难度目标预值(target)，另一个是随机数nonce。

这里的target，就是前面讲到的，整个块头的哈希要小于这个预值，即H(block header)≤target。block header里存的就是这个目标预值的编码(nBits)。这里需要注意，前一个区块的哈希只算的是前一个区块的块头，所以前面画的，一个区块引出一个剪头指向另一个区块中间，是不正确的，所以有的书剪头是指向一个区块的上面。取哈希时是把块头的所有部分都取哈希。

块身里面有交易列表(transaction list)。

前面还有一个内容讲的时候简化了:每个节点都需要验证所有的交易，实际上系统中的节点分全节点(full node)和轻节点(light node)，全节点是保存区块链所有的信息的，验证每一个交易，所以全节点又叫fully validating node。轻节点只保存block header的信息，一般来说轻节点没法独立验证交易的合法性。

比如一个交易是不是double spending，轻节点没有存以前的交易信息所以它没法验证。系统中大多数节点是轻节点，这节课内容主要针对全节点，因为轻节点没有参与区块链的构造和维护，只是利用了区块链的一些信息做一些查询。

区块链里的内容是如何写到区块链里面的呢:每个节点，每个账户都可以发布交易，交易是广播给所有节点的。有些交易是合法的，有些是非法的。谁来决定哪些交易应该被写入下一个区块中呢?按照什么顺序写呢?如果每个节点自己决定可以吗?如果每个人在本地维护一个区块链，那区块链的统一性得不到保证，而账本的内容是要取得分布式的共识(distributed consensus)。

下面的笔记跟比特币的应用关系不大，可以作为了解:

分布式的共识一个简单的例子就是分布式的哈希表(distributed hash table)，比如系统里有很多台机器，共同维护一个全局的哈希表。

这里需要取得共识的内容是什么？哈希表中包含了哪些键值对key valve pair。假如有人在自己电脑上插入一个键值对，'xiao'这个pair对应的是12345，即'xiao'→12345。那么别人在另一台读的时候也要能把这个读出来，这就叫一个全局的哈希表。

关于分布式系统有很多不可能结论(impossibility result)，其中最著名的是FLP。这三个字母是三个专家的名字缩写，他们的结论是:在一个异步的(asynchronous)系统里，(网络传输迟延没有上限就叫异步系统)，即使只有一个成员是有问题的(faulty)，也不可能取得共识。

还有一个著名结论:CAP Theorem。(CAP是指分布式系统的三个我们想要的性质，Consistency【系统状态的一致性】 Availability【别人都可以用】 Partition tolerance)。该理论内容是:任何一个分布式系统，比如分布式哈希表，这三个性质中，最多只能满足两个，假如想要前两个性质，那么就不会得到第三个性质。

分布式共识一个著名的协议是Paxos，该协议能够保证一致性，即第一个性质。如果该协议打成了共识，那么这个共识一定是一致的，即每个成员所认为的共识都是相同的。但是，某些情况下，该协议可能永远无法达成共识，这种可能性比较小但是客观存在的。

比特币中的共识协议(consensus in BitCoin):

比特币中共识要解决的一个问题是，有些节点可能是有恶意的。我们假设系统中大多数节点是好的，那么该如何取得共识协议?

第一种方案是投票，首先应该确定哪些区块有投票权，有些membership是有严格要求的，这种情况下基于投票的方案是可行的。但比特币系统创建账户是很容易的，甚至一个人产生了公私钥对别人都无法得知，只有转账时别人才知道。所以有些人可以不停的创建账户，当超过账户总数的一半时就有了控制权，这种称为女巫攻击(sybil attack)。因此投票方法不可取。

比特币账户巧妙的解决了这个问题，不是按照账户数目投票，而是按照计算力来投票。每个节点都可以在本地组装出一个候选区块，把它认为合法的交易放在里面，然后开始尝试各种nonce值(占4 byte)，看哪一个能满足不等式H(block header)≤target的要求。如果某个节点找到了符合要求的nonce，它就获得了记账权。

所谓的记账权，就是往比特币账本里写入下一个区块的权利。只有找到这个nonce，获得记账权的节点才有权利发布下一个区块。其他节点收到这个区块之后，要验证这个区块的合法性。

比如括号里block header的内容填的对不对；block header里面有一个域，叫nBits域，实际上它是目标预值的一个编码检查一下nBits域设置的是不是符合比特币协议中规定的难度要求；该不等式是否成立。假设都符合要求，然后检查block body 里面的交易列表，验证一下每个交易都是合法的:①要有合法的签名②以前没有被花过。如果有一项不符合要求，这个区块就是不能被接受的。如果所有条件都符合，也不一定接受。

假如生成了一个新区块，怎么知道新区块插在了哪里呢?根据生成区块的指针。有可能就存在一个问题，如图5(第四个视频第65分钟) ，这两个交易指A转账给B，以及A转账给自己。这种情况不是double spending，判断一个交易是不是double spending ，是看这个区块所在的分支上币又没有被花掉。如图，一直到第三个区块，币都没有花过，所以这个交易是合法的。虽然该交易是合法的，但是它不在最长合法链(longest valid chain)上。这种称为分叉攻击(forking attack)。所以接收的区块应该是扩展最长合法链。

区块链在正常情况下也可能出现分叉:两个节点同时获得记账权。每个节点在本地自己组装一个它认为合适的区块，然后去试各种nonce，如果两个节点在差不多同一个时间找到了符合要求的nonce，就都可以把区块发布，这时会出现两个等长的分叉。这两条都是最长合法链，那该接受那条呢?比特币协议当中，在缺省(默认的意思)情况下，每个节点是接受它最早收到的那个。所以不同节点根据在网络上的位置不同，有的节点先听到新生成的其中一个区块，那就接受这个区块；有些节点先听到另一个区块，那就接受另一个区块。

如何判断接收了一个区块?比特币协议中用到了implicit consign，如果沿着这个区块往下继续扩展，就算认可了这个发布的区块。比如在新生成的其中一个区块后面又拓展一个区块，表明就认可了这个新区块。

等长的临时性的分叉会维持一段时间，直到一个分叉胜出。也就是哪一个链抢先一步生成了新的区块，哪一条就是最长合法链。另一个作废的就叫orphan block。这两个新区块有可能会各自拉拢，两个区块链看谁的算力强，有时候也是看谁的运气好，就会胜出。

竞争记账权的好处:首先获得记账权的节点本身有一定的权力，可以决定哪些交易写到下一个区块里。但这些不应该被设定为竞争记账权的动力，所以巧妙地建立了一个机制:区块奖励(block reward)。

比特币协议中规定获得记账权的节点在发布的区块里可以有一个特殊的交易:铸币交易。在这个交易里可以发布一定数量的比特币。

这里要回到前面的问题①，谁来决定货币的发行?coinbase transaction币基交易是比特币系统中发行新的比特币的唯一方法，后面的交易都是比特币的转移。这个交易不用指出币的来源。

那么能造多少币呢?开始时比特币刚上线的时候，每一个发布的区块可以产生50BTC(BTC就是比特币的符号)。协议中规定，21万个区块以后，初块奖励就要减半，就变成了25BTC。再过21万个区块，又要减半。

因此当一个区块胜出后，另一个作废的区块得到的比特币是没有作用的，其他诚实的区块是不会承认的。

比特币系统中要取得什么共识?去中心化的账本要取得共识。谁又能决定账本的内容呢?只有获得记账权的节点才能写东西。怎么获得记账权呢?就是解pow(挖矿)。按照算力记票，算力可以用每秒能试多少nonce数值表示。那怎样防范女巫攻击呢?按算力记票，即使创建再多的账户，也无法使算力增强。

比特币争夺记账权的过程叫作挖矿(mining)，比特币被称为数字黄金(digital gold)，争夺记账权的节点被称为矿工(miner)。

第五节

比特币系统的实现

区块链是去中心化的账本，比特币使用的是基于交易的这种账本模式(transaction[交易]-based ledger[账本])。系统当中并不会显示每个账户有多少钱。

比特币系统的全节点要维护一个叫UTXO(unspent transaction output)(还没有被花出去的交易的输出)的数据结构。区块链上有很多交易，有些交易的输出可能已经被花掉，有些还没有被花掉。所有没有被花掉的输出的集合就叫做UTXO。

一个交易可能有多个输出。假如A给B5个比特币，B花掉了。A也给了C3个比特币，C没有花掉。这时5个比特币就不算UTXO，而3个比特币算。UTXO集合当中的每个元素要给出产生输出的交易的哈希值，以及它在这个交易里是第几个输出。这两个信息就可以定位到UTXO中的输出。

要UTXO集合有什么作用?

为了检测double spending。即检测新发布的交易是否合法。因此全节点要在内存中维护UTXO这样一个数据结构，以便快速检测double spending。

每个交易要消耗掉一部分输出，也会产生新的输出。还看上面的例子，B花掉的5个比特币虽然不在UTXO里面，但如果他转账给D，而D没有花掉，那么这5个比特币又要保存在UTXO里面。如果D始终不花，那么这个信息要永久保存在UTXO里面。有可能是不想花，也有可能是把密钥丢了。

每个交易可以有多个输入，也可以有多个输出，所有输入金额之和要等于输出金额之和。即total inputs=total outputs。因此一个交易可能来自多个地址，可能有多个签名。

有些交易total inputs略微大于total outputs。

假如输入1比特币，输出0.99比特币，另外0.01比特币作为交易费给获得记账权发布区块的节点。

区块奖励也不能完全作为挖矿的奖励，发布区块的节点为什么一定要把你的交易打包在区块呢?他们还要验证你的交易的合法性，如果交易较多占用的带宽会比较大，网络传播速度也会更慢。所以只有区块奖励是不够的。

因此比特币系统设计了第二个激励机制:交易费(transaction fee)。也就是你把我的交易打包在区块里，我给你一些小费。交易费一般很小，也有一些简单的交易没有交易费。

21万个区块大概要挖多长时间呢?大约是4年。比特币系统设计的平均出块时间是10分钟，就是整个系统平均10分钟会产生一个新的区块。

除了比特币这种基于交易的模式，与之对应的还有基于账户的模式(account-based ledger)，比如以太坊系统。在这种模式中，系统是要显示的记录每个账户上有多少币。

比特币基于交易的模式，隐私保护性较好。缺点是比特币当中的转账交易要说明币的来源，而基于账户的模式就不用。

如图⑥(第五节视频 16分钟处)

一个区块的例子

第一行表明:该区块包含了686个交易

第二行:总输出XXX个比特币

第四行:总交易费(686个交易的交易费之和)

最下面一行:区块奖励(矿工挖矿的主要动力)

第五行:区块的序号

第六行:区块的时间戳

第九行:挖矿的难度(每隔2016个区块要调整挖矿的难度，保持出块时间在10分钟左右)

倒数第二行:挖矿时尝试的随机数

右边:第一行:该区块块头的哈希值

第二行:前一个区块块头的哈希值

(注意:计算哈希值只算块头)

两个哈希值的共同点:前面都有一串0。是因为，设置的目标预值，表示成16进制，就是前面一长串的0。所以凡是符合难度要求的区块，块头的哈希值算出来都是要有一长串的0。

第四行:merkle root 是该区块中包含的那些交易构成的merkle tree的根哈希值。

如图⑥(见第五节视频 第20分钟)块头的数据结构

最后一行:是32位的无符号整数。nonce只有2的32次方个可能的取值。按照比特币现在的挖矿情况来说，很可能把2的32次方个取值都验了一遍也找不到合适的。那怎么办呢?block header 的数据结构里还有哪些域是可以调整的呢?

如图⑦ 块头里各个域的描述(见第五个视频 第21分钟)

第一行:比特币协议的版本号(无法更改的)

第二行:前一个区块的块头的哈希值(无法更改)

第三行:merkle tree的根哈希值(可以更改)

第四行:区块产生的时间(可以调整)比特币系统不要求特别精确的时间，可以在一定范围内调整。

第五行:目标预值(编码后的版本)(只能按协议中的要求定期调整)

第六行:随机数

挖矿时只改随机数不够，还可以更改根哈希值。

如图⑧(见第五节视频 第23分钟)

铸币交易没有输入，它有一个coinbase，可以写入任何的内容。也可以把digital commitment里的commit的哈希值写入里面。也可以把第一节讲到的预测股市的内容写入里面，coinbase的内容是没有人会检查的，甚至可以写你的心情。

那这个域对我们有什么用呢?

如图⑨(见第五节视频 第24分钟)

对应的是最后一个block header里的根哈希值对应的merkle tree，左下角的交易是coinbase，把它的域改了之后，其上的哈希值就发生了变化，然后沿着merkle tree的结构往上传递。最后导致block header里的根哈希值发生变化(merkle root是block header的一部分)。块头里4个字节的nonce不够用，还有其他字节可以用，比如coinbase域的前八个字节当做extra nonce来用，这样子搜索空间就增大到了2的96次方。

所以真正挖矿的时候只有两层循环，外层循环调整coinbase域的extra nonce。算出block header里的根哈希值之后，内层循环再调整header里的nonce。

如图⑩ 普通的转账交易的例子(见第五节视频 第26分钟)

该交易有两个输入和两个输出。

左上角:这里的output其实是输入，指的是之前交易的output。

右上角:这里的output都是unspent，都没有被花掉，会保存在UTXO里面。

右边表格第一行:输入的总金额。

依次往下:输出总金额、两者之间的差值。

两表格下面:可以看出输入和输出都是用脚本的形式来指定的。

比特币系统中验证交易的合法性，就是把input scripts和output script配对后执行来完成的。注意:不是把图中的input scripts

和output scripts配对，因为这两个脚本是一个交易中的脚本。不是把同一个交易里的输入脚本和输出脚本配对，而是把这里的输入脚本和前面提供币来源的交易的输出脚本配对。如果输入输出脚本拼接在一起，能顺利执行不出现错误，那么该交易就是合法的。

如图十一，是在求解puzzle的过程。

注意:求哈希时只用到了block header的内容，而交易的具体信息在block header里面是没有的。block header里面只有merkle tree 的根哈希值，这个就已经能保证交易是没有被篡改的。

挖矿过程每次尝试一个nonce可以看作是一个Bernoulli trial(伯努利实验)。每一个随机的伯努利实验就构成了一个伯努利过程。它的一个性质是:无记忆性。

每尝试一个nonce成功的概率是很小的，要进行大量的实验。这时可以用泊松过程来代替伯努利过程。我们真正关心的是系统出块时间，出块时间是服从指数分布。可以画出一个坐标轴，纵轴表示概率密度，横轴表示出块时间(整个系统的出块时间，并不是每个矿工的出块时间)。具体到每一个矿工，他能挖到下一个区块的时间取决于矿工的算力占系统算力的百分比。

假如一个人的算力占系统总算力的1%，那么系统出100个区块，就有一个区块是这个人挖的。

指数分布也是无记忆性的。因为概率分布曲线的特点是:随便从一个地方截断，剩下一部分曲线跟原来是一样的。比如:已经等十分钟了，还没有人找到合法的区块，那么还需要等多久呢?仍然参考概率密度函数分布 ，平均仍然要等十分钟。将来还要挖多长时间，跟过去已经挖了多长时间是没有关系的。这个过程也叫:progress free。

如果没有progress free ，会出现什么现象:算力强的矿工会有不成比例的优势。因为算力强的矿工过去做的工作是更多的，过去尝试了那么多不成功的nonce之后，后面nonce成功的概率就会增大。以此progress free 是挖矿公平性的保证。

出块奖励是系统中产生新的比特币的唯一途径。产生的比特币构成的一个几何序列。21万＊50+21万＊25+21万＊12.5+......=21万＊50＊(1+1/2+1/4+......)=2100万

比特币求解的puzzle，除了比拼算力之外，没有其他实际意义。比特币的稀缺性是人为造成的。

虽然挖矿求解puzzle本身没有实际意义，但是挖矿的过程对于维护比特币系统的安全性是至关重要的。挖矿提供一种凭借算力投票的有效手段，只要大部分算力是掌握在诚实的节点手里，系统的安全性就能够得到保证。

虽然挖矿奖励越来越小，难度越来越大，但这几年挖矿的竞争是越来越激烈的，因为比特币的价格是飙升的。最终区块奖励为0了，是不是就没有动力挖矿了呢?不是的，因为还有交易费激励机制。

假设大部分算力是掌握在诚实的矿工手里，我们能得到什么样的安全保证?能不能保证写入区块链的交易都是合法的。挖矿给出的只是概率上的保证，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到有恶意的节点手里。

比如好的矿工占90%的算力，坏的矿工占10%的算力。那么10%的概率下记账权会落在有恶意的矿工手里，这时候会出现什么情况?

先考虑第一个问题:他能不能偷币?能不能把别人账上的钱转给自己?不能，因为他没有办法伪造别人的签名。

假设M是有恶意的，他想把A账上的钱转走，所以他发布一个A转给M的交易，但这个交易需要有A的签名，M虽然获得记账权，但他不知道A的私钥，所以伪造不了签名。

如果M把交易硬写在区块链上，诚实的节点不会接受这个区块，因为它包含有非法的交易。所以诚实的节点会继续沿前一个区块挖，生成新的区块代替非法的区块，其他诚实的区块会沿着这个合法的区块继续挖。比特币要求是扩展正常合法链，M生成的不是合法区块，所以该区块作废。这对他造成的代价是很大的，因为没有了区块奖励，又没有偷到钱。

第二个问题:他能不能把已经花了的币再花一遍(即double spending)?假如他把M→A的交易写在了一个区块里面，现在他获得了记账权，他又发布另一个交易，把这个钱转回给自己，即M→M'。同样，这很明显是double spending，只要是诚实的节点都不会接受这个区块。

他如果想发布这个区块，只能连在写了M→A交易区块的前一个区块。注意:区块插在哪个位置，在刚挖矿时就是要决定的，因为设置的block header里要填上前一个block header的哈希。所以他想插到那个区块的话，一开始就要认定，而不是等获得记账权以后再认定。

这样生成的两条区块链，都是合法的。参考图十二(第五节视频 第56分钟处)。要看其他节点沿着哪一个链往下扩展，最后一个胜出一个作废。

这种攻击的目的是什么?如果M→A的交易，产生了某种不可逆的外部效果，然后M→M'再把M→A的交易回滚了，那么M就可以从中不当获利。

比如:网上购物时，M购买一些商品，然后该网站接受比特币支付，M发起一个交易把账转给网站。网站监听到交易写入了区块链里，以为支付成功了，所以就把商品给了M。M拿到商品之后，又发起一个交易，把支出的钱转给自己，然后把下面的链拓展成最长合法链。这样的结果是:既得到了商品，又收回了花掉的钱，就达到了double spending的目的。

如何防范这种攻击呢?如果M→A的交易所在的区块不是最后一个区块，那么这种攻击的难度就会大大增加。要是想回滚M→A的交易，还是要插在它之前的一个区块，然后想办法成为最长合法链。这个难度是很大的。因为诚实的节点，不会沿着它生成的区块往下扩展，因为它不是最长合法链。因此防范这种攻击的方法就是多等几个区块，或者叫多等几个确认confirmation。

M→A交易刚刚写入区块里时，我们把它叫作one confirmation。这时后面加的区块，依次叫two confirmation、three confirmation...比特币协议当中，缺省(系统默认)的是要等六个confirmation。有了六个confirmation，才认定M→A的交易是不可篡改的。这需要等多长时间呢?平均出块时间是10分钟，因此要等一个小时。

区块链是不可篡改的账本，那是不是意味着凡是写入区块链中的内容就永远改不了呢?经上述分析可以看出，这种分析只是一种概率上的保证。刚刚写入区块链的内容，还是比较容易被改动的。经过一段等待时间之后，或者后面几个区块被确认之后，被篡改的概率就大幅度下降(指数级别的下降)。

其实还有一种，叫零确认(其具体位置可见第五节视频 第62分第26秒)。意思是说，这个转账交易发布出去了，但还没又被写入区块链里。即M→A的交易已经发布，但下面包含M→M'的区块还没有被挖出来。

这个概念相当于电商购物的例子中，在支付时你发布一个转账交易，告诉电商自己已经转过钱了。电商运行一个全节点或委托一个全节点监听区块链上的交易，他收到转账交易之后要验证该交易的合法性(有合法的签名，以前没有被花过)，甚至不用等到该交易写入区块链里。这种操作听起来风险很大，交易刚发布出去，都没往区块链里写呢。其实，零确认在实际当中，用的还是比较普遍的。为什么呢?

这其中有两个原因:①比特币协议缺省的设置是节点接收最先听到的那个交易。所以在零确认的位置，M→A的节点收到后，再发M→M'的交易，有比较大的概率诚实的节点是不会接受的。

②很多购物网站，从支付成功，到发货，是有一定的时间间隔的，即有一定的处理时间。

回到前面的问题:假设某个有恶意的节点获得记账权，它还能做什么坏事?能不能故意不把某些合法的交易写入区块链里?即发布的区块故意不包含某些交易。这是可以的。

比特币协议并没有规定获得记账权的节点一定要把那些交易发布到区块里。但出现这种情况问题也不大，因为这些合法的交易一定会被写入下一个区块里，总有诚实的节点愿意发布这些交易。

其实，区块链在正常工作下，也会出现合法的交易没有被包含进去的情况，可能就是这段时间交易的数目太多了。比特币协议中规定，每个区块的大小是有限制的，最多不能超过一兆字节。所以如果交易的数目太多了，那么有些交易可能就只能等到下一个区块再发布。

会不会出现这种情况?M→M'的交易所在的区块所在的链条虽然短，但是先偷偷的生成比上面更多的区块，然后等上面的链条公布后再公布，就能够胜过上面的几个区块了?这种方法叫作selfish mining。

正常情况下挖到一个区块马上就发布，原因是你不发布别人可能就发布了，那样就拿不到区块奖励了。而selfish mining是先藏着不急着发布，这是分叉工具的一种手段。

但这样成功的概率并不大，因为有恶意的节点本来算力占比就不高，还要生成更多的区块，就非常困难。

以上是selfish mining的其中一个目的，它还有另一个目的。假如A挖了两个区块都没有发布，而在B挖到一个区块公布后立马公布，这样B挖的区块就作废了。这样的好处就是减少竞争，因为A在挖第二个区块时，别人还在挖第一个区块(前提是A算力足够强)。

但这样也有不好的地方，假如A挖出一个区块，A以为他能赶在别人面前再挖一个区块，结果这时有人挖出了第一个区块，那这样的话A就要在别人发布之后立马发布，去争取区块奖励。

第六节

比特币网络

比特币工作在应用层(application layer:Bitcoin block chain)，它的底层是一个网络层(network layer:P2P overlay network)。

比特币的P2P网络是非常简单的，所有节点都是对等的。不像有的P2P网络有所谓的超级节点、纸节点。

要加入P2P网络首先得知道至少有一个种子节点，然后你要跟种子节点联系，它会告诉你它所知道的网络中的其他节点，节点之间是通过TCP通信的，这样有利于穿透防火墙。当你要离开时不需要做任何操作，不用通知其他节点，退出应用程序就行了。别的节点没有听到你的信息，过一段时间之后就会把你删掉。

比特币网络的设计原则是:简单、鲁棒，而不是高效。每个节点维护一个零度节点的集合，消息传播在网络中采取flooding的方式。节点第一次听到某个消息的时候，把它传播给去他所有的零度节点，同时记录一下这个消息我已经收到过了。下次再收到这个消息的时候，就不用转发给零度节点了。

零度节点的选取是随机的，没有考虑底层的拓扑结构。比如一个在加利福尼亚的节点，它选的零度节点可能是在阿根廷的。这样设计的好处是增强鲁棒性，它没有考虑底层的拓扑结构，但是牺牲的是效率，你向身边的人转账和向美国的人转账速度是差不多的。

比特币系统中，每个节点要维护一个等待上链的交易的集合。假如一个集合的交易都是等待写入区块链里的，那么第一次听到某个交易的时候，把这个交易加入这个集合，并且转发这个交易给节点，以后再收到这个交易就不用转发了，这样避免交易会在网络上无线的传播下去。转发的前提是该交易是合法的。

这里有冲突的情况，有可能你会有两个有冲突的交易，差不多同时被广播到网络上。比如说A→B和A→C，这两个如果同时广播在网络上，那么每个节点根据在网络中的位置的不同，收到两个交易的先后顺序不同。

比如一个人先收到第一个交易，就写入到集合里，再收到第二个交易的时候就不会写入集合，因为跟上一个交易有冲突，就认定是非法的。假设这两个交易花的是同一个币，那么写入集合的交易就会被删掉。

比如说节点听到一个新发布的区块，里面包含了A→B的交易，那么这个交易就可以删掉了，因为已经写入到了区块链里。如果节点又听到了A→C的交易，该怎么办?这时候也要把A→B删掉。因为A→C如果已经被写入到了区块里，那么A→B就变成了非法交易，就变成了double spending，这就是冲突的情况。可能某个先收到A→C的节点，抢先挖到了矿，发布了区块。

新发布的区块在网络上的传播有很多方式，跟新发布的交易是类似的。每个节点除了要检查区块的内容合法性之外，还要查它是不是在最长合法链里。越是大的区块，在网络上传播速度越慢。

比特币协议对区块的大小有1M字节的限制。比特币系统采用的传播方式是非常耗费带宽的，带宽是瓶颈。按1M的区块大小限制来算的话，一个新发布的区块有可能需要几十秒，才能传输到网络大部分境地，这已经是挺长时间了，所以这个限制值不算小。

还需要注意的一点:我们讲的比特币网络的传播属于best effort 。一个交易发布到比特币网络上，不一定所以的节点都能收到，而且不同的节点收到这个交易的顺序也不一定是一样的。网络传播存在延迟，而且这个延迟有的时候可能会很长，有的节点也不一定按照比特币协议的要求进行转发。

可能有的该转发的不转发，导致某些合法的交易收不到，也有的节点可能转发一些不该转发发的消息，比如说有些不合法的交易也被转发了。这就是我们面临的一个实际问题。

第七节

比特币的挖矿难度调整

目标预值越小，挖矿的难度越大。调整挖矿的难度就是调整目标空间在整个输出空间中所占的比例。

比特币用的哈希算法是SHA-256，这个产生的哈希值是256位。所以整个输出空间是2的256次方。调整这个比例，即目标空间占输出空间的比例，通俗的说，就是哈希值前面要有多少个0。比如说256位的哈希值，要是合法的区块，要求算出来的哈希，前面至少有70个0。当然这只是通俗的说法，因为这个目标预值，并不是说前面都是0，从某一个位置开始，后面都变成了1。

挖矿的难度跟目标预值是成反比的，公式是:difficulty=difficulty 1 target / target。上面是指挖矿难度等于1的时候所对应的目标预值，挖矿难度最小就是1，这个时候对应的目标预值是个非常大的数。

即target越大，挖矿是越容易的。所以公式里很大的一个数，除以当前的目标预值，得到的就是当前的挖矿难度。所以difficulty和target大小是成反比的。

为什么要调整挖矿难度呢?如果不调会有什么问题呢?系统里的总算力越来越强，挖矿难度保持不变的话，出块时间是越来越短的。

出块时间越来越短，会有什么问题吗?

比如说不到一秒就出一个区块，区块在网络上传播的时间可能需要几十秒，底层的比特币网络可能需要几十秒才能让其他节点都收到。别的节点没有收到这个区块之前还是继续沿着已有的区块链往下扩展。如果有两个节点同时都发布一个区块，这个时候就会出现分叉。

出块时间如果越来越短的话，这种分叉会成为常态，而且不仅会出现二分叉，可能会出现很多的分叉。比如10个区块同时被挖出来，系统可能会出现10分叉。

分叉如果过多，对于系统达成共识是没有好处的，而且危害了系统的安全性。比特币协议是假设大部分算力掌握在诚实的矿工手里。系统当中的总算力越强，安全性就越好，因为有恶意的节点想掌控51%的算力就越难。如果掌握了51%的算力，它就可以干很多坏事，比如分叉攻击。

如果后面分叉多的话，前面某个区块里的某个交易，很可能就遭受分叉攻击，恶意节点会试图回滚。因为后面分叉多，算力就会分散，恶意节点得逞的概率更大。这个时候恶意节点就不需要51%的算力了，可能10%的算力就够了，因此出块时间不是越短越好。

那10分钟的出块时间是不是最优的呢?不一定。改成其他值也可以，有间隔只是说应该有个常数范围。以太坊系统出块时间就降低到了15s，所以以太坊的出块速度是比特币的40倍。

出块时间大幅度下降之后，以太坊就要设计新的协议，叫ghost。在该协议中，这些分叉，产生的orphan block(即产生最长合法链后另一个要被丢弃的区块)就不能丢弃掉了，而是也要给它们一些奖励，这叫uncle reward。以太坊也要调整挖矿难度，使出块时间保持在15s。

讲完了为什么要调整挖矿难度，现在讲一下怎么调整挖矿难度。比特币协议中规定，每2016个区块后就要调整目标预值，这大概是每两个星期调整一次。

具体的调整公式:target =target×(actual time/expected time)。actual time指产生2016个区块实际花费的时间，expected time指产生2016个区块应用的时间，即2016×10min。

如果实际花费时间超过了两周，即平均出块时间超过了10min。那么这时候挖矿难度要调的低一点，应该让出块更容易。因此该公式算出来的target会变大，则难度会下降。

实际上，上调和下调都有四倍的限制。假如实际时间超过了8个星期，那么我们计算公式时也只能按4倍算，目标预值增大最多只能增大4倍。

那怎么才能让所有的矿工同时调整目标预值呢?计算target的方法写在比特币系统的代码里，每挖到2016个区块会自动进行调整。如果有有恶意的节点故意不调，会怎么样?

如果一个节点不调，将区块发布出去，诚实的节点是不会认的。nBits是target一个编码的版本，在block header里没有直接存储target的域，因为target的域是256位，直接存target的话要32个字节。nBits在header里只有四个字节，所以可以认为是它的一个压缩编码。

如果遇到有恶意的矿工，该调的时候不调，这时检查区块的合法性就通不过。因为每个节点要独立的验证发布的区块的合法性。检查的内容就包括:nBits，目标预值设的对不对。如果投机取巧设计一个过大的目标预值，使得你自己挖矿容易了，但这个区块是不会被接受的。

如图(第七节视频 第26分钟)显示的是比特币系统中总算力的变化情况。在比特币没有流行前，有很长一段时间，算力没有太明显的增长，前面这些年的hash rate几乎是0。其实这些年算力也是增长的，只是后面这些年算力增长的太快了，所以前面部分看上去像是一条直线。去年是涨得非常猛的一年，这也体现在了hash rate 的增长上，算力呈现出指数级的增长。即使在这段黄金时期，算力也不是单调递增的，中间也是有很多波动。

如图(第七节视频 第27分钟)是挖矿难度的变化情况，跟算力的增长基本上是同步的，这也符合难度调整的设计目标。通过调整挖矿难度，使得出块时间保持稳定。注意这个图显示的是挖矿难度，不是目标预值。

如图(第七节视频 第27分第27秒)是最近半年的难度调整曲线，可以看出很明显是一段一段的。每隔两个星期，难度上一个台阶，说明挖矿的人越来越多，用的设备越来越先进，反应出大家对比特币的热情越来越高。如果出现相反的情况，比如某个加密货币的挖矿难度越调越小，说明挖矿变得越来越容易了。但这不是好事，说明大家对币的热情是逐渐减小的。持续出现这种情况说明这个币将被淘汰。

如图(第七节视频 第28分第13秒)显示的是每天的出块时间。可以看出，总的来说出块时间稳定在10分钟上下振动。

如图(第七节视频 第28分第36秒)显示最近半年的出块时间，也是维持在10分钟左右。

挖矿难度的公式:下一个难度=前一个难度＊两周/挖前2016个区块用的时间(注意:前面的公式是目标预值的公式，不要混淆了)

第八节

比特币挖矿

全节点介绍(第八节视频 第31秒)

轻节点介绍(第八节视频 第3分)

比特币网络中大部分节点都是轻节点，如果只是转账而不是挖矿的话，没必要用全节点。挖矿过程中，如果你听到别人发布了一个区块，该区块是合法的，也是在延长最长合法链。这时候你该怎么办?

应该停止挖矿，然后重新在本地组装一个候选区块，再重新从头开始挖矿。因为如果沿着新发布的交易往下挖的话，那么本地组装的区块中包含的交易就会发生变化，有些交易可能已经被包含到新发布的区块里了。另外block header的内容也会发生变化，像block header里有交易所组成的merkle tree的根哈希值，还有指向前一个区块的指针，这些也都会发生变化。

这样做是不是有些可惜?挖矿的一个性质是无记忆性，无论是继续挖原来的区块，还是停下来挖一个新组装出来的区块，成功的概率是一样的。只要还没有挖到符合要求的nonce值，前面已经挖了多长时间都是无所谓的。

即使挖到了合法的区块，发布到区块链上，也不能保证胜利了。有可能你发布的区块最终没有成为最长合法链，可能存在一些冲突情况，别人可能同时发布一些合法的区块，或者存在一些你不知道的double spending，使得你的区块中的某些交易变成冲突的。

比特币是怎么保证安全的?两个方面:①密码学②共识机制

①别人没有你的私钥，就没有办法伪造你的签名，所以不能把你账上的钱转走。(前提是系统中拥有大多数算力的矿工是好的，是遵守协议的，不会接受那些没有合法签名的交易。如果没有这些，密码学上的保证也就没有用武之地。)

比如:你去银行取钱，按照规定取钱得出示合法的证件，银行工作人员才能把钱给你。合法的证件就相当于密码学上的签名，密码学的性质保证了别人没有办法伪造你的签名，也就没有办法伪造你的身份。产生私钥以及签名的时候，都要有好的随机源，产生的随机数要足够随机。但又这些也不是足够的，银行工作人员要足够自觉，不能把钱交给那些没有合法证件的人，只有这两条合在一起，才能保证别人不能把你账上的钱转走。

挖矿的设备:挖矿设备演化趋势是越来越趋于专业化，最早的时候用的是普通的CPU挖矿，像家里计算机、笔记本电脑。但如果买一台计算机专门用来挖矿是非常不划算的，计算机当中的大部分内存都是闲置的，挖矿只用到其中很小一部分内存，CPU当中的大部分部件也是闲置的，因为挖矿当中计算哈希值的操作只用到了通用CPU当中的很少一部分指令。硬盘和其他很多资源也都是闲置的，所以随着比特币挖矿难度的提高，用CPU挖矿，用通用计算机挖矿显得性价比太低。

所以挖矿转入第二代设备:GPU。GPU效率相比CPU提高了很多，主要用于大规模的并行计算。但GPU用来挖矿还是有点浪费了，GPU是用于通用并行计算而设计的，用来挖矿的话有很多部件仍然是出于闲置状态，比如说用于浮点数计算的部件。这些部件对于深度学习来说是很重要的，但比特币的操作只用到了整数挖矿。所以GPU虽然效率提高了很多但仍然有不小的浪费。这些年GPU价格涨得很快，有些人归因于深度学习的火热，其实有很多GPU是用来挖矿的。不过有一个好消息，随着比特币挖矿难度的提升，用GPU挖矿已经划不来了，已经超过了GPU的算力范围，所以GPU现在可以更多的用于深度学习、游戏应用的服务。

有一些新开发的加密货币有的还在用GPU挖矿，而现在更多用ASIC芯片挖矿，这是专门为了挖矿而设计的芯片，上面没有多余的电动逻辑，整个芯片就是为了比特币挖矿、计算哈希值的操作而设计的。它的性价比是最高的，这个芯片除了挖矿什么事都干不了，而且为某一种加密货币设计的ASIC芯片，只能挖这一种加密货币。除非这两个加密货币用同一个mining puzzle。

有些加密货币刚发行的时候，为了解决能启动问题，会故意用一个已有的加密货币的mining puzzle，比如说跟比特币一样的mining puzzle，这样可以吸引更多的人来挖矿，这种情况叫merge mining。除了这种情况，其他都是一个芯片只能为一个加密货币挖矿。ASIC芯片生产周期需要一年，但跟其他通用芯片相比，ASIC芯片研发速度已经是非常快的了。

在这么长的生产周期里面，如果比特币价格出现剧烈变化的话，前期投入的研发费用可能就打水漂了。从历史上看，比特币的价格变化是比较剧烈的。曾经发生好几次，比特币的价格在几个月之内，下跌了80%，然后又慢慢恢复。

如果比特币价格大幅度下降的话，挖矿可能是赔本的，可能还抵不上电费。即使在比特币发展的黄金时期，价格不断上涨，这时挖矿是有利可图的。但是竞争也是越来越激烈的，定制的ASIC芯片可能用不了几个月就过时了。一款ASIC矿机刚上市的时候大部分的利润是在它上市的前两个月获得的，因为这个时候它的算力在同类产品中是最强的。再往后随着更强的矿机出现，它就可能被淘汰掉。所以购买ASIC矿机的时机很重要，现在都是要提前预定的。有些不良厂商，ASIC矿机生产出来之后，不是立即提供给消费者，而是自己先用来挖矿一段时间，赚取比特币，等到最赚钱的黄金时间即这前两个月过去之后，再把矿机发给用户。当比特币系统中算力突然有一个很大的提升，就说明某个大公司生产出了新一款的ASIC矿机。所以在挖矿热潮中真正赚钱的不一定是挖矿的用户，而可能是卖矿机的大厂商。

挖矿机的变化趋势，是从通用变得越来越专用，CPU是通用计算，GPU是通用并行计算，ASIC是专用计算。ASIC一旦过时就作废了，不像CPU和GPU还能做其他工作。很多人觉得这是不好的，是跟去中心化的理念是不相符的，也违背了比特币设计的初衷。最民主的情况是，大家都用家里的CPU计算机挖矿。后来改为GPU噪音是很大的。而有些新的加密货币设计的是Alternative mining puzzle。而设计它的出发点是asic resistance(抗asic芯片化)，目的是让通用的计算机也能参与挖矿的过程。

挖矿的另一个趋势是大型矿池的出现，单个矿工即使用了ASIC芯片，挖矿从平均收益上看是有利可图的，但是收入是非常不稳定的。比特币系统中平均每10分钟出一个区块，这是说比特币系统中所有的矿工做一个整体来看平均10min会产生一个区块。但如果具体到某一个矿工来说，他可能要挖很长时间，如果他用一个矿机可能要挖一两年。这样子就好像是买彩票，挖到了就是中了一个大奖。单矿工还有其他问题，他除了挖矿之外还要承担全节点的其他责任(就是这节课最开始介绍的那些)。

所以要引入矿池，所谓的矿池，就是把这些矿工组织起来，作为一个整体，矿池的架构一般是一个全节点会驱动很多矿机，一个矿池有一个矿主，叫pool manager。下面连了很多矿工，这些矿工只负责计算哈希值，全节点的其他职责都由矿主来承担。他负责监听网上的交易，把这些交易组织打包成区块，同时要看一看有没有其他的节点抢先发布区块，如果有的话看怎样进行调整.....

ASIC芯片只能负责计算哈希值，它不能干全节点的其他功能。矿池的出现还为了解决另一个问题:收入不稳定。单个矿工的收入是不稳定的，所以大家一起干，有了收益再进行分配。

那么收益该如何分配?矿池一般有两种组织形式，一种是像大型数据中心那样，有的互联网公司，有成千上万个服务器，大的矿池里面也有成千上万的矿机，这些矿机如果是属于同一个机构的话，那么收入怎么分配就不重要了。

但也有矿机是来自不同机构的，即第二种组织方式:分布式的。矿工和矿主不在同一个地方，可能分散在世界各地，那么矿工要加入一个矿池，就是按照矿池规定的通讯协议跟矿主进行联系。矿主把计算哈希值的任务分配给他，矿工计算完之后，把结果反馈给矿主，将来获得出块奖励时一起分配。

如果矿工是来自五湖四海的，不是属于同一个机构的，那么利益该怎么分配?平均分配行不行?比如每个矿工挖到一个区块，得到了出块奖励，然后平分给其他矿工，这样行吗?不行，因为会有矿工偷懒。因此要按矿工的贡献大小进行分配，也就是这里同样需要工作量证明。那该怎么证明每个矿工做了多少工作呢？

为什么矿工的收入不稳定，因为挖矿太难了，如果把挖矿的难度降低之后，挖矿就会变得稳定了。怎么降低难度呢?以前的要求是，矿工要找到一个nonce，用nonce计算block header 的哈希值，前面至少有70个0才是合法的区块。降低挖矿难度之后，比如说前面只要有60个0就行了，这样挖到的叫作一个share，这个share叫做almost valid block。矿工挖到share或almost valid block之后，把它提交给矿主。矿主拿到这个区块有什么用呢？用来证明矿工所做的工作量，而没有其他用途。矿主无法得到区块奖励以及任何好处。所以矿主就统计每个矿工提交了多少这样的share，将来等到某个矿工真正挖到了合法的区块之后，再将出块奖励按照每个矿工所做的工作量，提交的share数目进行分配。

这样做为什么是可行的?每个矿工挖到矿的概率取决于他尝试的nonce数目，尝试的nonce越多，能找到的share就越多。

有没有可能一个矿工挖到一个合法的区块之后，不把它提交给矿主，而是自己偷偷摸摸发布出去，得到出块奖励?即平时挖到的share提交，但挖到了合法区块就不提交?不可能，因为每个矿工的任务是由矿主分配的，矿主负责组装好一个区块，然后交给矿工去尝试各种nonce，而且挖矿仅仅调nonce是不够的，还需要调整coinbase parameter。所以矿主会把不同的coinbase parameter所对应的nonce值的范围交给不同的矿工去尝试。那么这个区块里包含什么?coinbase transaction里面有收款人的地址，这个地址填的是矿主的地址，即pool manager的地址，所以矿工挖到区块之后，如果他不提交给矿主自己发不出去是没有用的。里面的收款地址是矿主的，他取不出钱来。所以只要是当初按矿主给分配的任务进行挖矿的，就不可能偷区块奖励。

如果他一开始就不管矿主的任务，自己组装一个区块，偷偷把收款地址改成自己地址，会怎样?那样他提交share给矿主的话，矿主是不认的，因为里面交易列表被改过了，coinbase transaction里面的内容发生了变化，算出的merkle tree 的根哈希值也是不一样的。这种情况下矿主是不会给他工作量证明的。那就相当于矿工一开始就单干，跟矿池是没关系的。

虽然不可能偷区块奖励，但会不会有人捣乱，比如平时挖到一个share，提交给矿主，作为工作量证明。等他挖到一个真正合法的区块之后，把它扔掉。这是有可能的，虽然没有经济好处，但有可能是别的矿池派来的卧底，不想让这个矿池得到区块奖励。这些矿工还是会分红，分的是别的矿工挖出来的区块奖励。

如图(第八节视频 第38分处 )是矿池在各个国家的分布比例，中国矿池占世界81%，远远超过其他国家，所以按矿池比例来看的话，中国的总算力是有绝对优势的。

如图(第八节视频 第38分第24秒)如果按照单个矿池来看，在2014年，曾经有叫GHash.IO的矿池，这个矿池的算力，占到了全球算力的一半以上。在当时曾引起一些恐慌，这一个矿石的算力就已经足以发动51%的攻击了。这个事情公布之后，该矿池主动把算力占比大幅度的减少，以免动摇大家对比特币的信心。

如图(第八节视频 第38分第56秒)是2018年的各矿池的算力分布，看上去没有那么集中了，GHash.IO矿池早已停止运营。当然，挖矿集中化的程度仍然是比较大的，几个大型矿池占了相当大的比重，但没有矿池占50%以上。这样看算比较安全了，但可能只是一个表面现象。假如一个机构有一半以上的算力，他不一定要把算力集中在一个矿池里，而可以把算力分散隐藏在很多矿池里，真正需要发动攻击的时候再集中起来发动攻击。

矿工转换矿池是很容易的，加入一个矿池就是按照这个矿池的协议跟这个矿主联系，矿池把组装好的区块信息发给矿工，矿工来尝试各种nonce值就可以了。

所以这就是矿池带来的危害，如果没有矿池，想要发动51%的攻击，攻击者要投入大量的成本来购买到足够的矿机，能够达到系统中半数以上的算力。有了矿池之后，他可能只占很小一部分比例的算力，只要能够吸引到足够多的矿工，足够多的不明真相的群众加入到他的矿池里来就行了。

一般来说，矿池的矿主要收取一定比例的出块奖励作为管理费。矿主也要按照比例收取管理费，有的是按照出块奖励的比例，也有的是抽取交易费。有的一些有恶意的矿池在发动攻击之前，可能故意把管理费降得特别低，甚至是赔本赚吆喝，吸引足够多的矿工加入之后就可以发动攻击了。这是大型矿池的一个弊端，使得51%的攻击更加容易了。

假如某个矿池占到了半数以上的算力，他具体能够发动哪些攻击呢?一个最常见的就是分叉攻击。假如一个区块链，其中一个区块包含了一个大笔的交易，又等了几个确认区块之后，自认为已经安全了。然后这时就可能有人在该交易前面的区块发动分叉攻击。

看上去好像追赶的道路是很漫长的，但如果拥有51%的算力，最终还是可以成功攻击。另外，不要把51%当成绝对的门槛，有可能不到51%就可以。算力都是估计的，而且算力还在不断变化。

攻击者还能做什么坏事?还可以做boycott(封锁境域)。比如说攻击者不喜欢某个账户，怀疑某个账户参与非法交易，想把这个账户封锁掉，所有跟这个账户相关的交易都不让上链。假如A把某个交易A→B发布到区块链上，攻击者就会马上进行分叉，产生一个不包含这个交易的区块，所有跟A有关的交易也都不包含进去。

这种攻击跟分叉攻击区别是什么?他没必要等后面几个确认区块。这时候如果攻击者等待确认区块，是为了让B放心，B以为后面有六个确认区块，已经没事了，然后攻击者再发动分叉攻击。而如果目的是为了boycott的话，就没有必要等后面区块生成。A→B交易一上链马上进行分叉，越早越好，因为攻击者是希望别人沿着他的链往下挖的。

前面讲过，有些有恶意的节点故意不把某些交易写入区块里，是可以的。但没有关系，后面的区块还是会包含的。但是如果这个坏人拥有51%的算力的话，他可能仗着自己算力强，公开抵制他想抵制的交易。这样别的矿工也不敢随便把交易打包进去了。

那么攻击者有没有可能掌握51%的算力后，把别人账上的钱转走。这是不可能的。因为他没有别人账户的私钥，没有办法伪造签名。如果他仗着算力强，强行把一个没有合法签名的交易发布到区块链上，会有什么样的结果?会造成分叉。因为诚实的矿工会沿着另外一个分叉去挖，不会沿着他发布的区块往下挖。所以盗币是不可能的。

总结:矿池的出现减轻了矿工的负担，矿工只需要挖矿，计算哈希值就行了，别的事情都由矿主来完成。矿工的收入分配也更加稳定。但矿池的出现也有危害，发动51%的攻击变得容易了。他不一定自己有这么强的算力，只要动员召集这些算力就可以了。

这有点类似于云计算中的on demand computing。平时不需要维护很大的计算机群，需要用的时候可以随时召回来。而矿池的情况，是on demand mining。

第九节

比特币使用的脚本与原理

如图(第15秒)是比特币的一个交易实例。该交易有一个输入两个输出。左上角写着output，其实是这个交易的输入。右边两个输出，上面unspent即没有花出，下面spent表示已花出。该交易已经收到了23个确认，所以回滚的可能性很小了。

下面是这个交易的输入输出脚本，输入脚本包含两个操作，分别把两个很长的数压入栈里。比特币使用的脚本语言是非常简单的，唯一能访问的内存空间就是一个堆栈。不像通用的编程语言，像C语言C++那样有全局变量、局部变量、动态分配的内存空间，它这里就是一个栈，所以叫做基于栈的语言。这里输出脚本有两行，分别对应上面的两个输出。每个输出有自己单独的一段脚本。

如图(第1分第40秒)是交易的具体内容。首先看交易的一些宏观信息。第一行:transaction ID，第二行hash，该交易的哈希值。第三行:使用的比特币协议的版本。第四行:该交易的大小。第五行:用来设定交易的生效时间。此处的0表示立即生效。绝大多数情况下，locktime都是0。如果是非零值，那么该交易要过一段时间才能生效。比如要等10个区块以后才能被写入区块链里。第六行第七行的vin、vout是输入输出部分，后面会详细讲解。第八行是这个交易所在区块的哈希值。第九行:该交易已经有多少个确认信息。第十行是交易产生的时间，第十一行是这个区块产生的时间。(time 和block time都是指很早的一个时间到现在过了多少秒)

如图(第3分第32秒)是交易的输入结构。一个交易可以有多个输入，在这个例子中只有一个输入。每个输入都要说明该输入花的币是来自之前哪个交易的输出，所以前两行给出输出币的来源。第一行:之前交易的哈希值。vout表示这个交易里的第几个输出。所以这里表示花的币来自于哈希值为c0cb...c57b的交易中第0个输出。接下来是输入脚本，输入脚本最简单的形式就是给出signification就行了，证明你有权利花这个钱。(后面的PPT中scriptsig就写成input script输入脚本)。如果一个交易有多个输入，每个输入都要说明币的来源，并且要给出签名，也就是说比特币中的一个交易可能需要多个签名。

如图(第5分)是交易的输出，也是一个数组结构。该例子中有两个输出，value是输出的金额，就是给对方转多少钱，单位是比特币，即0.22684个比特币。还有的单位是satoshi(一聪)，是比特币中最小的单位。1比特币=10的8次方聪。n是序号，表示这是这个交易里的第几个输出。

scriptpubkey是输出脚本，后面都写成output script。输出脚本最简单的形式就是给出一个pubkey。下面asm是输出脚本的内容，里面包含一系列的操作，在后面会详细解释。require sigs表示这个输出需要多少个签名才能兑现，这两个例子中都是只需要一个签名。type是输出的类型，这两个例子类型都是pubkeyhash，是公钥的哈希。addresses是输出的地址。

如图(第6分 第36秒)是展示输入和输出脚本是怎样执行的。在区块链第二个区块里有A→B的转账交易，B收到转来的钱后，又隔了两个区块，把币又转给了C。所以B→C交易的txid、vout是指向A→B交易的输出。而要验证交易的合法性，是要把B→C的输入脚本，跟A→B交易的输出脚本拼接在一起执行。

如图(第7分 第40秒)这里有个交叉，前面交易的输出脚本放在后面，后面交易的输入脚本放在前面。在早期的比特币实践中，这两个脚本是拼接在一起，从头到尾执行一遍。后来出于安全因素的考虑，这两个脚本改为分别执行。首先执行输入脚本，如果没有出错就再执行输出脚本。如果能顺利执行，最后栈顶的结果为非零值，也就是true，那么验证通过，这个交易就是合法的。如果执行过程中出现任何错误，这个交易就是非法的。如果一个交易有多个输入的话，那么每个输入脚本都要和所对应的交易的输出脚本匹配之后来进行验证。全都验证通过了，这个交易才是合法的。

如图(第8分第45秒)是输入、输出脚本的几种形式。一种最简单的形式就是P2PK(pay to public key)。输出脚本里直接给出收款人的公钥，下面一行checksig，是检查签名的操作。在输入脚本里，直接给出签名就行了。这个签名是用私钥对输入脚本所在的整个交易的签名。这种形式是最简单的，因为公钥是直接在输出脚本里给出的。

如图(第9分第18秒)是脚本的实际执行情况。这三行是把输入脚本和输出脚本拼接起来之后的结果。第一行来自输入脚本，后两行来自输出脚本。注意，实际代码中出于安全考虑，这两个脚本实际上是分别执行的。第一行:把输入脚本提供的签名压入栈，第二条把输出里提供的公钥压入栈，第三条checksig是把栈顶的这两个元素弹出来。用公钥检查一下这个签名是否正确。如果正确，返回true，说明验证通过。否则，执行出错，这个交易就是非法的。

如图(第10分第24秒)是P2PK的一个实例。上面交易的输入脚本就是把签名压入栈，下面交易是上面交易输入的币的来源。它的输出有两行，第一行是把公钥压入栈，第二行就是checksig。这是第一种形式。

如图(第10分第52秒)是第二种形式P2PKH(pay to public key hash)，跟第一种区别是输出脚本里没有直接给出收款人的公钥，给出的是公钥的哈希。公钥是在输入脚本里给出的。输入脚本既要给出签名，也要给出公钥。输出脚本里还有一些其他操作，DUP、HASH160等等，这些操作都是为了验证签名的正确性。P2PKH是最常用的形式。

如图(第11分第37秒)是脚本的执行结果，这个是把上一页的输入脚本和输出脚本拼接之后得到的，前两条语句来自输入脚本，后面的语句来自输出脚本，还是从上往下执行。第一条语句先把签名压入栈，第二条语句把公钥压入栈。第三条语句是把栈顶的元素复制一遍，所以栈顶又多了一个公钥。HASH160是把栈顶元素弹出来，取哈希，然后把得到的哈希值再压入栈。所以栈顶变成了公钥的哈希值。

第五行是把输出脚本里提供的公钥的哈希值压入栈。这个时候栈顶有两个哈希值，上面的哈希值是输出脚本里面提供的，收款人公钥的哈希，即我发布交易时，转账的钱是转给谁的，在输出脚本里提供一个收款人的公钥的哈希。下面的哈希是指你要花这个钱时在输入脚本里给出的公钥，然后前面的操作HASH160是取哈希后得到的。倒数第二行操作的作用是弹出栈顶的两个元素，比较是否相等，即比较其哈希值是否相等。这样做的目的是防止有人莫名顶替，用自己的公钥冒充收款人的公钥。假设两个哈希是相等的，那么就从栈顶消失了。最后一条作用是用公钥检查弹出栈顶的元素是否正确。假设签名是正确的，整个脚本就顺利运行结束，栈顶留下的是true。如果执行过程任何一个环节发生错误，比如输入里给出的公钥跟输出里给出的哈希值对不上，或者是输入里给出的签名跟给出的公钥对不上，那么这个交易就是非法的。

P2PKH是最常用的脚本信息，该实例(第14分第20秒)用的就是这种脚本。输入脚本就是把签名压入栈，把公钥压入栈。下面的输出脚本复制栈顶元素，然后取哈希值，hash160。然后把公钥的哈希压入栈，最后比较栈顶的两个哈希值，检查签名。

最后一种如图(第15分第25秒)，也是最复杂的一种脚本形式，是Pay to Script Hash。这种形式的输出脚本给出的不是收款人的公钥的哈希，而是收款人提供的一个脚本的哈希，这个脚本叫redeemscript，赎回脚本。将来花这个钱时输入脚本里要给出redeemscript(这个赎回脚本的具体内容)，同时还要给出让赎回脚本能够正确运行所需要的签名。

验证时分为两部(如图第15分第40秒)，第一步验证输入脚本里给出的赎回脚本是不是跟输出脚本里给出的哈希值匹配，如果不匹配说明给出的赎回脚本是不对的，就类似于刚才讲的pay to public key hash里面给出的公钥不对一样。匹配不上说明给出的赎回脚本是不对的，那么验证就失败了。如果输入里给出的赎回脚本是正确的，那么第二步还要把赎回脚本的内容当做操作指令来执行一遍，看看最后能不能顺利执行。如果两步验证都通过了，那么这个交易才是合法的。听上去有点抽象，那么下面看一个具体的例子。

(如图第16分第47秒)用pay to script hash实现pay to public key 的功能。这里的输入脚本就是给出签名，再给出序列化的赎回脚本，赎回脚本的内容就是给出公钥，然后用checksig检查签名。下面这个输出脚本是用来验证输入脚本里给出的赎回脚本是否正确。

如图(第17分第13秒)看一下pay to script hash的执行过程。开始也是把输入脚本和输出脚本拼接在一起，前两行来自输入脚本，后面三行来自输出脚本。首先把输入脚本的签名压入栈，然后把赎回脚本压入栈，然后是取哈希的操作，得到赎回脚本的哈希。这里RSH是指redeem script hash，赎回脚本的哈希值。接下来还要把输出脚本里给出的哈希值压入栈，这时栈里就有两个哈希值了。最后用equal比较这两个哈希值是否相等，如果不等就失败了。假设相等，那这两个哈希值就从栈顶消失了，到这里第一阶段的验证就算结束了，接下来还要进行第二个阶段的验证。

如图(第18分第28秒)第二个阶段首先要把输入脚本提供的序列化的赎回脚本进行反序列化，这个反序列化的操作在PPT上并没有展现出来，这是每个节点自己要完成的。然后执行赎回脚本，首先把public key压入栈，然后用checksig验证输入脚本里给出的签名的正确性。验证痛过之后，整个pay to script hash才算执行完成。

有人可能会问:干脆用pay to public key就行了，搞这么复杂干嘛?为什么非要把这些功能嵌入到赎回脚本里面?对于这个简单的例子来说确实是复杂了，但pay to script hash它的常见的应用场景是对多重签名的支持。

比特币系统中一个输出可能要求多个签名才能把钱取出来，比如某个公司的账户，可能要求五个合伙人中任意三个人签名才能把公司账户上的钱取走，这样为私钥的泄露提供了一些安全的保护。

比如说有某个合伙人私钥泄露出去了，那么问题也不大，因为还需要两个人的签名才能把钱取走。这同时也为私钥的丢失提供了一些冗余，即使有两个人把私钥忘掉了，省下的三个人依然可以把钱取出来，然后转到某一个安全的账户。

以上的功能是通过check multisig来实现的。

如图(第21分)，输出脚本里给出N个公钥，同时指定一个预值M。输入脚本只要提供接N个公钥对应的签名中任意M个合法的签名就能通过验证。

比如刚才举的例子中，N=5，M=3，五个合伙人中任意三个的签名都可以，输入脚本的第一行有一个红色的“×”，这是什么意思呢?

比特币中check multisig的实现，有一个bug，执行的时候会从堆栈上多弹出一个元素，这个就是它的代码实现的一个bug。这个bug现在已经没有办法改了，因为这是个去中心化的系统，要想通过软件升级的方法去修复这个bug代价是很大的，要改的话需要硬分叉。所以实际采用的解决方案，是在输入脚本里，往栈上多压进去一个没用的元素，第一行的“×”就是没用的多余的元素。另外需要注意给出的M个签名的相对顺序，要跟它们在N个公钥中的相对顺序是一致的才行。

如图(第22分第48秒)是check multisig的执行过程。这个例子假设三个签名中给出两个就行。图中可以看到这两个签名给出的相对顺序也是跟它们在公钥中的顺序是一样的。在公钥当中，第一个公钥排在第二个公钥前面。那么给出这两个签名的时候也是第一个签名排在第二个的前面。

第一行的false就是前面说的多余的元素。首先把多余的元素压入栈里，然后把两个签名依次压入栈，这个时候输入脚本就执行完了。接下来的输出脚本里把M的值，即预值M压入栈。然后把三个公钥压入栈，接着把N的值压入栈，最后执行check multisig，看看堆栈里是不是包含了这三个签名中的两个，如果是那么验证通过。

注意:这个过程中并没有用到pay to script hash。就是用比特币脚本中原生的check multisig来实现的。这么实现有什么问题吗？

早期的多重签名就是这样实现的，在实际的应用当中，有一些不是很方便的地方。

比如:网上购物。某个电商用multi签名，要求有五个合伙人中任意三个人的签名才能把钱取出来，要求网上购物的用户在支付的时候生成的转账交易里给出这五个合伙人的公钥，同时要给出N和M值。在这个例子中，N=5，M=3，这些都是用户在网上购物的时候生成转账交易时输出脚本里要给出的信息，给出这五个公钥，给出N和M值。

那么用户怎么知道这些信息呢?需要购物网站在网上公布出来，比如网上可以公布我们用了多重签名，我们用的五个签名中要给出三个，这是五个公钥，然后用户生成这个转账交易的时候，就把这些信息填进去。那么不同的电商采用的多重签名的规则是不一样的。有的电商可能是五个签名中要任意三个，有的可能要四个。这就给用户生成转账交易带来了一些不方便的地方，因为这些复杂性都暴露给用户了。

那么该如何解决?这里就要用到pay to script hash。

如图(第26分第39秒)是用pay to script hash实现的多重签名，它的本质是把复杂度从输出脚本转移到了输入脚本。现在这个输出脚本变得非常简单，只有这三行。原来的复杂度被转移到redeemscript赎回脚本里。输出脚本只要给出这个赎回脚本的哈希值就可以了。赎回脚本里要给出这N个公钥，还有N和M的值，这个赎回脚本是在输入脚本里提供的，也就是说是由收款人提供的。

像前面网上购物的例子，收款人是电商，他只要在网站上公布赎回脚本的哈希值，然后用户生成转账交易的时候把这个哈希值包含在输出脚本里就行了。至于这个电商用什么样的多重签名规则，对用户来说是不可见的，用户没必要知道。从用户的角度来看采用这种支付方式跟采用pay to public key hash没有多大区别，只不过把公钥的哈希值换成了赎回脚本的哈希值。当然，输出脚本的写法上也有一些区别，但不是本质性的。这个输入脚本是电商在花掉这笔输出的时候提供的，其中包含赎回脚本的序列化版本，同时还包含让这个赎回脚本验证通过所需的M个签名。将来如果这个电商改变了所采用的多重签名规则，比如由五个里选三个变成三个里选两个，那么只要改变输入脚本和赎回脚本的内容，然后把新的哈希值公布出去就行了。对用户来说，只不过是付款的时候，要包含的哈希值发生了变化，其他的变化没有必要知道。

如图(第29分第14秒)是具体的执行过程。这是把输入脚本和输出脚本拼接在一起后的情况，第一行的FALSE就是为了应付check multisig的bug而准备的一个没用的元素，执行的时候先把它压入栈，然后依次把两个签名压入栈，接下来是序列化的赎回脚本，目前只是把它作为数据压入栈，到这里输入脚本就执行完了。下面是输出脚本，取哈希，然后把输出脚本里提供的哈希值压入栈顶。最后判断两个哈希值是否相等，到这里第一阶段的验证就完成了。

如图(第30分第18秒)开始第二阶段的验证，把赎回脚本展开后执行。先把M压入栈，然后把三个公钥压入栈，把N压入栈，最后检查多重签名的正确性，三个里面有两个是正确的。第二阶段的验证过程跟前面直接使用check multisig的情况是类似的。

如图(第30分第52秒)是网上使用pay to script hash来做多重签名的一个实例。上面输入脚本的最后一个就是序列化的赎回脚本，反序列化之后得到的就是三个里面取两个的多重签名脚本。下面这个输出脚本的内容，跟前面讲的是一样的。现在的多重签名，一般都是采用这种pay to script hash的形式。

如图(第31分第25秒)这种脚本格式是比较特殊的，这种格式的输出脚本开头是return的操作，后面可以跟任意的内容。return操作的作用，是无条件的返回错误，所以包含这个操作的脚本永远不可能通过验证，执行到return语句，就会出错，然后执行就终止了，后面跟的内容根本没有机会执行。

为什么要设计这样的输出脚本呢？这样的输出岂不是永远花不出去吗？无论输入脚本写的是什么内容，执行到输出的return语句，它就会报错，那么这里的钱永远都花不出去。确实如此，这个脚本是销毁比特币的一种方法。

为什么要销毁比特币呢？这个一般有两种应用场景:

①有些小的币种要求销毁一定数量的比特币才能够得到这个币种，有时候把这种小币种称为AltCoin(Alternative coin)。除了比特币之外的其他小的加密货币都可以认为是Alternative Coin。比如有的小币种要求销毁一个比特币可以得到1000个小币，也就是说要用上述的方法证明已经付出了一定的代价才能够得到这个小币种。

②往区块链里写入一些内容。区块链是个不可篡改的账本，有人就利用这个特性往里面添加一些需要永久保存的内容，比如第一节课讲的digital commitment。要证明在某个时间，知道某些事情。比如涉及知识产权保护的，把某项知识产权的内容取哈希之后，把哈希值放到return语句的后面，其后面的内容反正是永远不会执行的，往里面写什么都没关系。而且放在这里的是一个哈希值，不会占太大的地方，而且也没有泄露出来你知识产权的具体内容。将来如果出现了纠纷，像知识产权的一些专利诉讼，再把具体的哈希值的输入内容公布出去，证明你在某个时间点已经知道某个知识了。

这个应用场景和coinbase域相似。coinbase transaction里面有个coinbase域，在这个域里写什么内容同样是没人管的，那这里为什么不用coinbase的方法呢？coinbase还不用销毁比特币，就可以直接往里写。

coinbase的方法只有获得记账权的那个节点才能用。如果是一个全节点，挖矿挖到了，然后发布一个区块，可以往coinbase transaction 里的coinbase域写入一些内容，这是可以的。

而我们说的上述方法，是所有节点都可以用的，甚至不一定是个节点，可能就是一个普通的比特币上的一个用户，任何人都可以用这种方法去写入一些内容。发布交易不需要有记账权，发布区块才需要有记账权。任何用户都可以用这种方法销毁很少的比特币，比如0.0000001个比特币，换取往区块链里面写入一些内容的机会。其实有些交易根本没有销毁比特币，只不过支付了交易费。

下面看两个实例

如图(第37分第44秒)是一个coinbase transaction。这个交易有两个输出，第一个输出的脚本是正常的pay to public key hash，输出的金额就是得到的block reward加上transaction fee。第二个输出的金额是0，输出脚本就是刚才提到的格式:开头是return，后面跟了一些乱七八糟的内容，第二个输出的目的就是为了往区块链里写一些东西。

如图(第38分第20秒)这是个普通的转账交易，输出脚本也是以return开头的。这个交易的输入是0.05个比特币，输出金额是0，说明输入金额全部用来支付交易费了。这个交易实际上并没有销毁任何比特币，只不过是把输入里的比特币作为交易费转给挖到矿的矿工了。

这种形式的脚本的一个好处是:矿工看到这种脚本的时候知道它里面的输出永远不可能兑现，所以就没必要把它保存在UTXO里面，这样对全节点是比较友好的。还有一点要说明:这个PPT当中涉及到比特币脚本的操作为了简单起见都没有加上OP前缀。比如CHECKSIG，实际上应该写成OP\_CHECKSIG，CHECKMULTISIG、DUP也是如此。

比特币系统中用到的这种脚本语言是非常简单的，甚至连专门的名字都没有，它就叫比特币脚本语言(bitcoin scripting language)。后面可以看到，以太坊当中用的智能合约的语言比这个要复杂的多。比如说比特币的脚本语言不支持循环，所以有很多功能这个语言是实现不了的，这样的设计是有其用意的，不支持循环就不会有死循环，就不用担心停机问题。以太坊当中智能合约的语言表达能力很强，所以就要靠汽油费的机制来防止程序陷入死循环。

另外一方面，这个语言虽然在某些方面功能是很有限的，但是在另外一些方面它的功能却很强大，比如跟密码学相关的功能。如checkmultisig，检查多重签名用一条语句就能够完成，这个比很多通用的编程语言要方便的多。所以比特币的脚本语言虽然看上去很简单，但其实针对比特币的应用场景做了很好的优化。

第十节

比特币分叉

区块链由一条链变为两条链就叫分叉。分叉可能是多种原因造成的，比如挖矿的时候，两个节点差不多同一个时候挖到了矿，就会出现一个临时性的分叉，我们把这个分叉叫作state fork，是由于对比特币区块链当前的状态有意见分歧而导致的分叉。

前面还讲过分叉攻击(forking attack)，它也属于state fork，也是属于对比特币这个区块链当前的状态产生的意见分歧，只不过这个意见分歧是故意造成的，人为造成的，所以我们又叫它deliberate fork。

除了这种state fork 之外，还有一种产生分叉的情况是，比特币的协议发生了改变，要修改比特币系统需要软件升级。在一个去中心化的系统里，升级软件的时候没有办法保证所有的节点同时都升级软件。

假设大部分节点升级了软件，少数节点因为种种原因可能没有升级，有可能是还没来得及升级，也可能是不同意对这个协议的修改。即假如你想把协议改成某个样子社区中可能是有人不支持的，这个时候也会出现分叉，这种分叉叫protocol fork(协议分叉)。因为对比特币协议产生了分歧，用不同版本的协议造成的分叉，我们称作protocol fork。

根据对协议修改的内容的不同，我们又可以进一步分成硬分叉和软分叉。出现硬分叉的情况:如果对比特币协议增加一些新的特性，扩展一些新的功能，这些时候那些没有升级软件的这些旧的节点，它是不认可这些新特性的，认为这些特性是非法的，这就属于对比特币协议内容产生了意见分歧，所以会导致分叉。

硬分叉的一个例子就是比特币中的区块大小限制(block size limit)。比特币系统规定每个区块最多是1M字节，有些人认为1M的限制太小了，也增加了交易的延迟。可以计算一下:1M=1百万 一个交易大概认为是250个字节 1百万/250=4000 一个区块大概是4000个交易 平均10分钟出现一个区块 4000/(60×10)=7 大概每秒钟产生7笔交易即7tx/sec 这个传输速度是非常低的。

有人发布一个软件更新，把block size limit从1M增加到4M。假设大多数节点更新这个软件，把block size limit更新到4M，少数节点没有更新。这里的大多数节点和少数节点不是按照账户数目来算的，而是按照算力，即系统中拥有大多数哈希算力的节点都更新了软件。新节点认为区块大小限制是4M，旧节点认为是1M。

如图(第11分第40秒)这时运行系统，会有什么结果?假如一个新节点挖出一个区块，这个区块比较大，但旧节点不认可，它忽略大区块的存在会继续沿着它的前一个小区块接着挖。而旧节点如果挖出了区块新节点是认可的，因为4M的限制指不能超过4M，比4M小是可以的。

那为什么会产生分叉呢?大区块挖出之后，因为大多数区块是更新了的，是认可新的大区块的，所以会沿着它继续挖。只有少数旧节点会接着下面链往下挖，这时新节点认为上下两条链都是合法的，但上面那条是最长合法链，所以会沿着上面一条挖。而且算力足够大会使上面那条链越来越长。而旧节点认为上面的链无论多长都是非法的，它们只会沿着下面的链挖。当然上面的链也可能出现小区块，因为新节点也可能挖出大小不到1M的区块，虽然这种是新旧节点都认可的，但这是没有用的，因为这条链上它们认为有非法的区块。所以这种分叉是永久性的，只要旧节点不更新软件，分叉就不会消失，所以才叫它硬结点。

比特币社区当中有些人是比较保守的，提高block size limit有些人就是不同意。而且区块的大小也不是越大越好，比特币底层系统是个P2P overlay network，它的传播主要采用flooding的方式，所以对带宽的消耗是很大的，带宽是瓶颈。

那么旧节点挖出的小的区块还有没有出块奖励呢？出现hard fork后出现了两条平行运行的链，平行运行链彼此之间有各自的加密货币。下面链的出块奖励在下面链里是认的。而分叉之前的币按道理应该是上下两条链都认可，所以会拆成两部分。

曾经出现过这样的问题:分叉前有A→B的交易，分叉后在上面链出现了B→C，下面链也出现了B→C，因为账户，私钥都是一样的。既然如此，就会有人利用这个特性，想收到上下两条链的转账。但如果没有人转账给他怎么办？

可以这样做:比如说B去购物，花一笔钱，给了C。后来B要退货，要取消这笔交易，C又把钱交给B。然后B又在下面一条链进行回放，就赚了一笔钱。那么在开始B转给C的交易在下面链会不会回放呢？所以这样做也是有风险的。为了解决这个问题，就让这两条链各带一个chain ID，所以现在以太坊的分叉已经没有问题了，就是两条独立运行的链了。

soft fork:

软分叉出现的情况是什么?如果对比特币协议加一些限制，加入限制之后原来合法的交易或区块在新的协议当中有可能变的不是合法了，这就引起软分叉。

假设有人发布一个软件更新，把这个区块大小变小了。调整区块大小不止是改变一个参数那么简单。一个去中心化的系统，改变一个参数，就可能导致分叉，而且取决于这个参数是怎么改的。有可能是硬分叉，有可能是软分叉。这里把区块大小变小只是为了解释软分叉这个概念，实际中是不会这么做的。

假设新节点把区块大小改为0.5M，旧节点依然以1M为准，这时候会出现什么情况？假如一个区块链开始分叉，新节点挖出小区块，这种区块旧节点也是认的。而旧节点挖出的大区块新节点是不认的。这样下去，旧节点看到上面链更长，而且是合法的之后，就会转去挖上面链。

所以为什么称这种分叉是软分叉?因为这种分叉是临时性的。所以旧节点如果不更新软件，它们挖的区块可能就白挖了。旧节点转向上面链挖的话，问题可能又会出现:它们可能又挖出了大区块。而新节点不认这个，新节点会继续沿着大区块前面一个小区块挖，如图(第29分第25秒)所示。

实际中可能出现软分叉的情况:给某些目前协议中没有规定的域增加一些新的含义，赋予它们一些新的规则，典型的例子就是coinbase域。前面讲过每一个发布的区块里可以有一个铸币交易(coinbase transaction)，coinbase transaction里有一个域叫coinbase域，这个域用来干什么是没人规定也没人检查的。

前面讲过coinbase域的一个用途:可以把它作为extra nonce。挖矿的时候要不断调整block header里的nonce，但block header里的nonce只有四个字节，最多只有2的32次方个可能性，所以实际中可以把coinbase前八个字节用来做extra nonce。两个合在一起就成了2的96次方，对于目前的挖矿难度，这个域已经是足够了。但coinbase域不止是八个字节，后面还有很多，剩下的字节有人就提议做UTXO集合的根哈希值。

目前这个集合只是每个全节点自己在内存中维护的，主要是为了快速查找、判断该交易是不是属于double spending，但这个集合的内容并没有写到区块链里，这跟前面讲到的merkle proof是不太一样的。

merkle proof能证明什么？证明某个交易是不是在给定的区块里。比如一个轻节点，没有维护整个区块的内容，只知道block header。轻节点问一个全节点:该交易是不是在这个区块里?全节点返回一个merkle proof作为证明，轻节点就可以验证是否属实。但如果是另外一种情况，想要证明某个账户上有多少钱，这个目前在比特币系统中是证不出来的。如果是全节点还可以算一下，方法如下:想要知道A账户有多少钱，就看一下A在UTXO里对应的输出总共收到多少个币，就是该账户上有多少钱。

对于全节点是可以算出来的，但如果是区块链钱包、有的手机上的APP，它不可能在手机上维护一个完整的区块链，它实际上是个轻节点，它想要知道账户的余额需要询问全节点。全节点返回一个结果，怎么知道这个结果是否属实呢？现在是证不出来的。如果你自己不维护一个UTXO集合，就没法用merkle proof 证出来。

有人提议把UTXO集合当中的内容也组织成一颗merkle tree，这个merkle tree有一个根哈希值，根哈希值写在coinbase域里面。因为block header没法再改了，改block header动静就太大了，coinbase域正好是没人用的，所以就写入UTXO的根哈希值。coinbase域当中的内容最终往上传递的时候会传递到block header里的根哈希值里。所以改coinbase域的内容，根哈希值会跟着改。因此这个提案就是说把UTXO集合的内容组织成merkle tree，算出一个根哈希值来，写入coinbase域里某个位置。coinbase域的内容本身也会算哈希，算到block header里的根哈希值，这样就可以用merkle proof证出来了。

假设有人发布一个软件更新，规定coinbase域要按照这个要求来填写，大多数节点都升级了软件，少数节点没有更新，这属于软分叉，因为新节点发布的区块旧节点认为是合法的，因为旧节点不管新节点写什么内容。但旧节点发布的区块新节点可能是不认的，因为如果coinbase域不按要求写它是不认的，所以属于软分叉。

比特币历史上比较著名的软分叉的例子是pay to script hash。P2SH这个功能在最初的比特币版本里是没有的，它是后来通过软分叉的功能给加进去的。这是什么意思呢?你支付的时候不是付给一个public key的哈希，而是付给一个赎回脚本的哈希。花钱的时候要把这个交易的输入脚本跟前面币的来源的交易的输出脚本拼接在一起执行。执行的时候验证分为两步，第一步是要验证输入脚本中给出的redeem script跟前面那个输出脚本给出的script的哈希值是对的上的，证明输入脚本里提供的script是正确的。第二步再执行redeem script，来验证输入脚本里给出的签名是合法的。

对于旧节点来说，它不知道P2SH的特性，只会做第一阶段的验证，即验证redeem script是否正确。新节点才会做第二阶段的验证，所以旧节点认为合法的交易新节点可能认为是非法的(如果第二阶段的验证通不过的话)。而新节点认为合法的交易旧节点肯定认为是合法的，因为旧节点只验证第一阶段。

总结:soft fork是什么?只要系统中拥有半数以上算力的节点更新了软件，那么系统就不会出现永久性的分叉，只可能有一些临时性的分叉。hard fork特点是什么？必须是所有的节点都要更新软件，系统才不会出现永久性的分叉，如果有小部分节点不愿意更新，那么系统就会分成两条链。

第十一节

课堂问答

一、转账交易时如果接收者不在线怎么办？

这个时候不需要接收者在线，转账交易只不过是在区块链上记录一下，把一个人账户上的比特币转移到他人的账户上，他人是否当时连接在比特币网络上是没有影响的。

二、假设某个全节点收到了一个转账交易，有没有可能转账交易中接收者的收款地址是这个节点以前从来没有听说过的?

这是可能的。比特币账户在创建的时候是不需要通知其他人的，在本地产生一个公私钥对就可以了。只有在产生收款地址以后第一次收到钱时，其他节点才知道这个账户的存在。

三、如果账户的私钥丢失了，该怎么办?

私钥丢失之后是没有办法的。该账户上的钱就变成了死钱，永远取不出来了。在去中心化的系统里，是没有人可以给你重置密码的。

另外，有些加密货币交易时，一般来说交易所是中心化的机构。在交易所开办一个账户的时候，一般是要提供身份证明的。这种情况下把比特币保存在交易所里，私钥实际上是由交易所来保管的。登录这个交易所是按照登录银行差不多的程序，即一个账户名一个密码，一般来说需要二次验证，比如Google身份验证器产生一次性密码，通过二次验证然后登录。

这种情况下如果账户密码丢失了，是可以跟交易所联系的，通过身份验证之后重置密码。有一些在线钱包也提供保管私钥的功能，但比特币或加密货币的交易所处于一种缺乏监管的状态，这个跟股票交易所是很不一样的。历史上曾发生很多次加密货币的交易所被黑客攻击的情况。最著名的是日本的Mt.Gox的事件，它曾经是全世界最大的一个比特币交易所，交易量占到了全球比特币交易量的70%，后来被黑客攻击，丢失了大量的比特币。后来交易所破产了，其CEO被判刑。各种加密货币交易所出现问题的情况发生了很多次，也有内部监管不当，管理人员卷钱跑路也时常发生。相比之下，一些冷钱包和硬钱包是比较安全的。

四、如果私钥泄露了怎么办？

比如你发现自己账户上出现一些可疑的交易，这个时候该怎么办?这时应该尽快把自己账上的钱转到另外一个安全的账户上，这个也跟我们平时的生活体验不太一样，如果在银行账户上出现一些可疑的交易，我们首先想到的是通知银行，能否把密码重置，账户冻结，免得别人把钱取走，而这些在区块链的世界里都是做不到的。

比特币账户所谓的密码是什么？就是它的私钥。公私钥对生成之后是没有办法改的。可以生成一个新的账户，但是原来账户上的私钥是改不了的。同样，也无法阻止别人发布从这个账户上转账的交易，任何有私钥的人都可以发布一个转账交易，把账户上的钱转走，这个也是没有办法冻结的。所以我们能做的就是在第一时间抢在别人之前把自己账户上的钱转到一个安全的账户上。

五、如果转账的时候写错了地址怎么办？

这是没有办法的。如果写错了地址而转错了人，我们也没有办法取消已经发布了的交易，比特币当中转账交易一旦发布到区块链里，就没有办法取消了。当转错了地址，如果我们知道是转给了谁，可以跟对方进行联系。如果不知道转的是谁的地址，或者是不存在的地址，那就没有办法了。

什么叫不存在的地址？地址是公钥取哈希得到的。有些地址其实不是公钥的哈希得来的，比如第一节课曾讲的digital commitment的例子。你想把某项内容的哈希值发布到区块链上，证明你曾在某个时间知道某个事情。

在前面讲比特币脚本的时候，有个经典的说法，比如把哈希值放到return的后面，因为OP\_RETURN后写什么都是没有人管的。但是有人会用哈希值生成一个看上去像是比特币地址的东西。比如A→B，正常情况下B是某个比特币账户公钥取哈希之后得到的地址。在这里把他要保存的那个哈希值生成一个地址，作为收款人的地址。这个地址是没有对应的私钥的，它其实是个假的地址，比特币系统并不知道这个地址的真假，你这个哈希是怎么来的，别人也看不出来。所以这样转账的钱就变成了死钱。这个转账永远不可能被取出来。

这种做法一般牺牲一点比特币，比如转很少一点钱，换取往这个区块链里写入这个哈希值的机会。这个做法是不提倡的，因为这样的话转账交易的输出会永久的保存在UTXO里面。全节点收到这样一个转账交易，它其实并不知道你的地址的真假，它不知道你的钱其实是花不出去的，所以它必须把它永久的保存起来，这样对全节点是不友好的。

接着问一个问题:proof of burn 、OP\_RETURN这些实际当中是怎么操作的?

当一个全节点收到一个转账交易的时候，它首先要检查一下，这个交易的合法性，只有合法的交易才会被写入区块链里。而OP\_RETURN这个语句是无条件的返回错误，既然如此，它怎么可能通过验证，怎么可能被写到区块链里呢？

验证当前交易合法性的时候，不会执行这个语句。即当前交易的输出脚本在验证交易合法性的时候，是不会被执行的。只有有人想花这笔钱，后面再有一个交易，要花这个交易的输出的时候才会执行这个交易的输出脚本。

六、挖矿时会不会有的矿工偷答案?

不会。发布的区块里有coinbase transaction，里面有一个收款人地址，是挖到矿的矿工的地址。假如A挖到了矿，里面就是A的收款地址。如果要偷答案的话，就要把A的地址换成自己的地址，而地址如果一变化，coinbase transaction的内容就发生了改变。这样会导致什么？导致merkle tree的根哈希值变化，因为这个交易和区块中所包含的其他交易是合在一起构成了merkle tree。任何一个地方发生改变，根哈希值就会变。而nonce是在块头里面，根哈希值也是在块头里面，block header的内容发生了变化之后，原来找到的nonce就作废了。所以不可能偷答案，因为每个矿工挖到的nonce是和他自己的收款地址绑定在一起的。

七、怎么判断交易费该给哪个矿工？即事先怎么知道哪个矿工会挖到矿?

事先不需要知道哪个矿工会得到这个交易费。交易费是怎么算的?total inputs>total outputs，其差额就是交易费。发布的交易里面，一个交易可以有很多个输入，也可以有很多个输出，总输入减总输出就是交易费。给谁不需要事先知道，哪个矿工挖到矿了，就可以把这个区块里所包含的交易差额收集起来，作为他自己的交易费。

下面看一下比特币的一些统计数据:

如图(第23分第30秒)显示的是比特币区块链的大小的变化情况，可以看到区块链是越来越大的。这也不奇怪，区块链只能往里面添东西，所以区块链只会越来越长。目前的size对于硬盘的容量来说，还是完全没有问题的，区块链大部分内容还是可以保存在硬盘上的。

如图(第24分第13秒)是UTXO集合的大小变化。该集合总的趋势是不断变大的，有一些波动，主要原因就是比特币交易增多后UTXO的集合会跟着一起变大。当然还有另一些方面是历史原因造成的，有一些账户私钥丢失了，所以这些账户对应的输出在UTXO里就要永久的保存下去，时间长了也会累计增多。

如图(第25分)是比特币矿池挖矿的情况。可以看出，挖矿集中化的趋势也非常严重，几个大的矿池占了系统中很大一部分。

如图(第25分第24秒)是比特币的价格变化情况。如图(第25分第30秒)是比特币市值的变化情况，跟价格变化的图几乎是一样的。这里的市值是绝对的市值，不是说在加密货币整体中占的百分比。

如图(第25分第52秒)是比特币的交易量。可以看出最后两年交易量增加的很明显，而且波动非常大，这个交易量是按照美元价格算出的，所以这些波动当中有一些是比特币本身的价格波动造成的。

如图(第26分第24秒)是每天的交易数目。总的趋势也是在不断增长。如图(第26分第46秒)是每个区块的交易数量，趋势跟前面的图很接近，因为难度调整算法要把出块时间稳定在10分钟，这样的话每天能产生多少个区块就是差不多的。交易数目的变化主要是因为每个区块里所包含的交易数目发生了变化。每个区块最多包含交易的上限差不多是4000个，图中的情况是远远没有达到这个上限。很多人说1M太小了，但真实的区块链上很多区块是没有装满的。

第十二节

比特币的匿名性(bitcoin and anonymity)

什么叫匿名？一般来说，匿名是跟隐私保护联系在一起的。比特币中不要求用真名，可以用公钥产生的地址，所以比特币具有一定的匿名性。也就是你可以产生任意多的地址，然后用不同的地址干不同的事情。它用的是化名，但它不是完全没有名字，所以有人把它称为preudonymity。

比特币和银行存款哪个匿名性更好?

银行账户是实名制，你得提交身份信息，然后才能注册银行账户，而比特币不需要，从这点上看比特币匿名性要好。

其实，以前银行是可以用化名的，如果银行账户匿名的话隐私性和匿名性与比特币相比哪个好?

银行账户更好。比特币区块链的账本是公开的，所有人都能查到，每个人都可以上区块链把整个信息下载下来。而银行的账本是受控制的，银行的工作人员可以查到，一些司法手段也可以调取银行的信息，但普通百姓是查不到别人的账的。

比特币系统中什么情况下有可能破坏匿名性?

有的人推荐每次收款都用一个新的地址，这样的话可以有不同的地址，谁也不知道哪些是属于你的，看起来好像匿名性很强，但实际上这些地址是可以被关联在一起的。

比如网上购物，比特币交易允许有多个输入多个输出。而多个输入有可能是同一个人，因为这个人可能同时控制了这两个账户的私钥。为什么要有两个输入:因为你买的东西很可能不是你某个账户上全部的币。有多个输出，很可能有一个是找零钱的地址。这种交易一般都是比特币钱包软件生成的。很多交易软件每次交易的时候都会生成一个新的找零的地址，也是为了隐私保护。

有没有可能把输入地址和输出地址也关联起来呢？比特币生成交易的时候并没有规定找零钱的地址在outputs中出现的位置，所以想知道哪个是找零的地址也并不容易，但有些情况下可以分析出来。

比如第一个地址账户上有4个比特币，第二个有5个。产生的两个输出第一个输出转入6个比特币，第二个输出转入3个比特币。那很明显转入3个比特币的输出是找零的，因为如果它是商家的地址，就用不着两个inputs，任意一个输入都比3大。通过这种方法我们可以把输入地址和输出地址也关联起来。

如果想要更好的隐私保护，可以产生一些没必要的输出，为了迷惑别人。但是这些交易几乎都是用钱包软件生成的，现在很少有人手工生成这些比特币的转账交易，常用的比特币钱包就那么几种。所以把常用的比特币钱包生成交易的方式搞清楚，那么区块链上很大一部分转账交易都可以分析出来。常用的钱包到目前为止一般没有故意生成一些不必要的输出地址。

有可能破坏比特币匿名性的第一个方面:一个人可以生成很多个地址账户但这些地址账户是有可能被关联起来的。第二个方面:是这个地址账户跟现实世界中的身份也可能产生关联。什么时候会有关联？什么情况下别人有可能知道比特币账户对应的现实生活中的哪个人呢？

比特币系统一旦跟现实世界联系起来，就可能泄露你的真实身份，最明显的例子就是资金的转入和转出。怎么避免用比特币洗钱呢？盯住比特币的转入转出链是一个常用的手段。大笔的比特币和货币的交易想不引起司法部门的注意是很难的，转入转出也是比特币隐私容易被破坏的一个很重要的时机。

还有什么时候会泄露真实身份？用比特币做支付的时候。在实体世界中用比特币做支付，比如国外有的商家是接受比特币支付。但有一些麻烦之处:①延迟很长，等到交易确认要等六个区块生成，即一个小时。②交易费很贵。如果买咖啡，交易费可能都跟咖啡差不多贵了。这样的话，你支付的账户就跟你的真实身份建立联系了。这个账户可能跟个人的其他账户也是有联系的，所以这样很容易泄露个人隐私和身份。而且该交易不仅是该商家会知道，其他人也会知道。

比如:A想知道B的地址。A可以在B去买商品时，留意B支付的时间，然后去查找在这个时间点的交易。在B下一次购买商品时也留意支付时间，这样下去用不了几次就能知道哪个哈希值是B的。

这个例子告诉我们什么？信用卡记录不能公开，即使匿名，也不应将公钥取哈希后公开。但比特币系统是无法保密的，所以比特币的匿名性并不是绝对的，没有想象中那么好。

实际中用比特币的人匿名性保存的怎么样？中本聪发明比特币以来，没有进行过任何一笔交易，所以到现在也没人知道他的真实身份。曾经有一个叫丝路(silk road)的网站(eBay for illegal drugs)，有像eBay一样的网上交易平台，但卖的都是非法的违禁品。为了逃避司法制裁，其支付手段就是比特币，底下的网络层用的是洋葱路由(TOR)，在美国也有匿名邮寄的服务。最后运行了两三年，就被查封了。美国政府抓到其老板时没收了其十几万个比特币，在当时价值几千万美元。但他生活简朴，因为虽然有价值连城的比特币，但一旦消费就会暴露身份。

该网站被查封之后，有人又开了silk road2。也是运行没几年就被查封。还有一些类似的网上黑店，最后下场都不好。这些事件都说明了，比特币的匿名性没有我们想象中的那么好，尤其是想用它来做坏事。

所以回到前面的问题:比特币的匿名性有多好?匿名是跟隐私保护相关联的，但问题在于:你不想向谁暴露身份(hide your identity from whom)？如果你不想让身边的亲戚朋友知道，这是比较容易实现的。如果是非法组织，从事黑市活动，那保护起来就难多了。

一个比特币用户能采用什么样的方法尽量提高个人的匿名性?以前曾讲过，比特币系统是运行于应用层(application layer)的，底层是(network layer)。所以要提高匿名性可以从两个方面入手。

①网络层怎么提高匿名性?

在现实中，如果一个人去网吧发了帖子，别人是有办法知道他是谁的。因为他的身份证代表了他的身份，这和他的IP地址是有很大关联性的。

而网络层的匿名性是比较好解决的。区块链是个新生事物，但网络层的匿名性学术界已经有了很好的方案:多路径转发。跟洋葱路由(TOR)是一样的原理。即消息不是由发出者直接发送给接收者，中间要经过很多次转发。中间的每一个节点，只知道它的上一个节点是谁，但并不知道最早发出消息的人是谁。当然中间一些节点可能是坏的，但路径上只要有一个节点是诚实的，就能够把最初发起人的身份隐藏起来。这也是洋葱路由的基本原理。

②应用层怎么提高匿名性?

把不同人的币混在一起(coin mixing)，即把你的身份跟别人的身份混在一起，让别人分不清楚谁是谁。不光是区块链，在其他各个需要匿名的领域都能用到。有一些专门做coin mixing的网站，提供一定的服务收取一定的服务费。所有想做coin mixing的人把币发给网站，网站内部进行一些重组，然后你再把币取回来，这时取出的币就不是发布到网站上的币了，它是随机抽取一些币给你。

coin mixing真正实施起来有一定的复杂性，如果设计不好的话，别人可以根据你当初存进去币的数额，推断出来哪些币是你存进去的。

而且，在当今的区块链的世界里，没有什么信誉度非常高的coin mixing的服务。很多coin mixing的服务它本身也是要保持匿名的，它匿名的后果是:有可能投进去的币被他卷款跑路了，投币者是一点办法都没有的。

实际上并不一定非要做coin mixing，有一些应用本身也带有coin mixing的性质，比如在线钱包。很多人会把钱存入在线钱包里，在线钱包就会把这些人的币混起来，再取回自己的币时可能就不是当初存进去的币了。但在线钱包并不保证要履行coin mixing的功能。

还可以通过加密货币的交易所，交易所一般有天然的coin mixing的性质。前提是交易所不会泄露提币、存币的记录，否则也是不行的。

为什么保护隐私性难度挺大？本质原因是区块链是公开的，而且是不可篡改的。不可篡改性对于隐私保护来说是灾难性的。

零知识证明:

概念如图(第44分第50秒)所示:零知识证明是指一方(证明者)向另一方(验证者)证明一个陈述是正确的，而无需透露除该陈述是正确的外的任何信息。

例如:要证明一个账户是我的，只需要我给出私钥就行。但私钥不能直接泄露，所以就给出由私钥产生的签名，假设对方是知道这个账户的公钥的，那么就可以验证签名的正确性。这是不是一个零知识证明其实是有争议的，因为我给出了私钥之外的其他信息，具体算不算要看应用场合。

同态隐藏:

零知识证明的数学基础是同态隐藏。

如图(第48分第25秒)是同态隐藏的性质。

第一个性质说明加密函数值E不会出现碰撞，这跟哈希函数有所不同，哈希函数是可能出现碰撞的。这个性质反过来说明如果E(x)和E(y)是相等的，那么x、y也是相等的。(该语句是上面语句的逆否命题)

第二个性质说明加密函数是不可逆的，知道加密后的值，没办法推出加密前的值。

第三个性质是最重要的，叫作同态运算。它说的是对加密之后的函数值进行某些代数运算，等价于对这些输入直接进行代数运算然后再加密。

同态加法:加密值的和等于和的加密。

同态乘法:加密值的乘积等于积的加密。

举一个例子:如图(第51分第53秒)所示Alice想要向Bob证明她知道一组数x和y使得x+y=7，

同时不让Bob知道x和y的具体数值。

简单的解答版本如图(第53分第27秒)

·Alice把E(x)和E(y)的数值发给Bob

·Bob通过收到的E(x)和E(y)计算出E(x+y)的值(利用了性质3)

·Bob同时计算E(7)的值，如果E(x+y）=E(7)，那么验证通过，否则验证失败。

Bob可以用蛮力算法，一个一个试而计算出x和y的值，因此Alice要对x和y的值做一些随机化处理，保证x和y加起来还是不变的。

不考虑去中心化的前提下，前面在讲double spending时讲过，要对每一个数字货币进行编号就能防止double spending。回到这节课讲的隐私保护问题，央行是什么都知道的，那么有没有什么办法让央行做中心化的记账检测double spending，又不让它知道呢?即虚拟货币的编号不能是央行产生的，改成自己产生的，又不会被篡改掉。

这里就要用到盲签方法。如图(第59分第39秒)。

·用户A提供SerialNum，银行在不知道SerialNum的情况下返回签名Token，减少A的存款

·用户A把SerialNum和Token交给B完成交易

·用户B拿SerialNum和Token给银行验证银行验证通过，增加B的存款

·银行无法把A和B联系起来

·中心化

解读如下:

·用户A提供序号，银行进行签名但此时看不到序号的内容，A要取钱所以银行要减少A的存款。

·A给B转账交易的时候把序号和签名给B，这个时候序号是明文，B是可以看到序号的具体内容的。

·B把序号和签名给银行验证，这个时候序号也是明文，这一步验证的目的是检测double spending。

这样设计的好处是:银行不知道B的币是从哪来的。

零币和零钞(它们也是加密货币，跟比特币是一类属性):

比特币在很大程度上提供了匿名性，但它不能完全消除关联性，那么我们能不能设计一种新的加密货币，这个货币从一开始的结构设计上就用了密码学的原理保证了匿名性，所以就有了零币和零钞。

如图(第62分第47秒)零币和零钞

·零币和零钞在协议层就融合了匿名化处理，其匿名属性来自密码学保证。

·零币(zerocoin )系统中存在基础币和零币，通过基础币和零币的来回转换，消除旧地址和新地址的关联性，其原理类似于混币服务。

·零钞(zerocash)系统使用zk-SNARKs协议，不依赖一种基础币，区块链中只记录交易的存在性和矿工用来验证系统正常运行所需要关键属性的证明。区块链上既不显示交易地址也不显示交易金额，所有交易通过零知识验证的方式进行。

这是专门为匿名性设计的加密货币。

零币中存在基础币(比如比特币)和零币。用的时候要证明本来是有一个基础币，让基础币变得不能花费(unspendable)，然后换取一个零币，零币在花的时候只需要用零知识证明你花掉的币是系统中存在的某一个合法的币就行了，但是不用透露你花的是系统中具体的哪一个币。这是跟比特币的一个本质区别，比特币是每一笔转账交易都要说明币的来源。这样才能证明花的币的真实性不是凭空捏造出来的。

但零币和零钞不是这样，零币和零钞是说证明的时候可以从数据上保证你花的币是以前区块链上某个合法存在的币，但不知道具体是哪个。这样的话就把关联性破坏掉了，就没法追溯了。

零钞没有基础币，是完全的零币。零钞和零币也不是100%匿名安全的，在影响匿名安全的因素中依然有一个因素无法解决，就是与实体发生交互的时候。比如有人想拿这些币干坏事，把很大的金额转换成这种加密货币的时候，或者是把这些加密货币转换成现金的时候，仍然要暴露身份。这些加密货币数学上设计的再好，只是说对已经在区块链当中的转账有匿名性，跟外界交互的匿名性仍然是一个弱点。所以它依然无法提供100%的匿名。

第十三节

比特币引发的思考

哈希指针:

指针保存的是本地内存的地址，那么只是在本地这台计算机上才有意义，发送到其他计算机上就没有意义了。那么在发布区块的时候哈希指针是怎么能够通过网络进行传输呢？

所谓的哈希指针只是一种形象的说法，实际系统中用的时候只有哈希，没有指针。回顾一下之前看到的block header的数据结构，如图(第1分第39秒)。第25行就是指向前一个区块的哈希，没有指针。block header里只有哈希值，没有指针。

那么怎么才能找到前一个区块的内容呢？全节点一般是把这些区块存储在一个(key，value)数据库里面。key是区块的哈希，value就是区块的内容。一个常用的key value数据库是level DB。所谓的区块链这种链表结构实际上是在level DB里面用哈希值算出来的。只要你掌握了最后一个区块的哈希值，那么你通过level DB的查找，哈希值key对应的value就可以把最后一个区块的内容取出来。然后这个区块块头里面，又有指向前一个区块的哈希值。那么再去查找key和value，可以找到前一个区块的内容，以此类推，一步一步往前找，最终能够把整个区块链都找出来。

所以说在实际系统当中，所谓的哈希指针，只有哈希，没有指针，或者也可以认为哈希值的本身就是指针。

有一些节点没有保存完整的区块链的信息，只保存了最近的几千个区块，如果需要用到前面的区块的信息可以问其他的全节点要。哈希指针的性质保证了整个区块链的内容是不可篡改的。

区块恋:

就是指，把一个私钥分成几份，有几个人各自保管，只有最终大家都拿出自己的部分私钥，才能合成完整的私钥。

这样存在的问题是:这些人中任何一个人把私钥丢了钱就取不出来了。还有更大一个问题:这种截断私钥的做法会降低账户的安全性。因为比特币系统中每个账户的安全性跟所用的私钥的长度是相关的。

为什么要用256位的私钥?因为这个长度的私钥用暴力破解的方法是不可行的。就算把全世界的计算机集中起来破解256位的私钥，也是不可能成功的。但是如果从中截断，一对情侣中一个人分手之后想把钱取出来，他已经知道了其中一半的私钥，只要把剩下的128位私钥猜出来就行了。私钥长度减少一半并不意味着难度降低一半，难度由2的256次方降到了2的128次方，前者远远大于后者，破解难度降了很多。如果是四个合伙人的例子，有三个人瞒着另一个人要把钱取出来，那么他们只需要尝试2的64次方就可以了。

因此对于多个人的共享账户，不要用截断私钥的方法，而最好采用多重签名，多重签名中用到的每一个私钥都是独立产生的。而且多重签名也提供一些别的灵活性，比如可以要求N个人当中任意给出M个签名就可以了。

在区块恋例子中，如果一对情侣分手了，那么他们的比特币将永久的保存在UTXO里面，这对矿工是不友好的。矿工是不知道这笔钱永远取不出来的，所以矿工要把这笔钱永久的保存在UTXO里面，造成这个集合的膨胀。

分布式共识:

前面已经讲过，从理论上实现分布式系统的共识是不可能的，但实际当中又怎么变的可能了呢?为什么比特币系统能够绕过分布式共识中的那些不可能结论?严格来说，比特币并没有取得真正意义上的共识，因为取得的共识随时有可能被推翻，比如出现了分叉攻击。你以为已经取得了一个共识，分叉攻击后系统会回滚到前一个状态，从理论上说甚至有可能回滚到创世纪块。

按照分布式系统理论的要求，共识一旦达成之后，就不应该再改了，所以从这方面来说比特币并没有绕过分布式系统那些不可能的结论，因为它并没有达到真正意义上的共识。这说明理论和实际往往是有区别的。很多理论上的不可能结论对于实际当中是并不适用的，因为这种不可能结论只是对某种特定的模型下是不可能的，实际当中把模型稍微改一改不可能结论就不成立了。

比特币的稀缺性:

矿工挖矿的原因是为了获得收益，挖矿的收益要大于开销才是有利可图的。要吸引别人来挖矿，要么增加挖矿的收益，要么降低挖矿开销。任何一个新发行的加密货币，都有一个能启动的问题。早期为了吸引矿工来挖矿，可以给矿工更多的收益。比特币的做法是:①早期难度设置的比较低。②早期的出块奖励比较高。

实际上，比特币这种总量恒定的性质是不适合用来做货币的。后面讲的以太坊就没有出块奖励定期减半的做法，一些新型的货币甚至要自带通胀的功能，每年要把货币的通行量提高一定的比例。因为稀缺的东西是不适合用来做货币的，通货膨胀会导致钱变得更不值钱了，但一个好的货币是要有通货膨胀的功能的。

量子计算:

随着量子计算的发展，量子计算机计算力变得越来越强大，加密货币会不会变得不安全了?

这种担心是没必要的:①量子计算技术离实用还有很长一段距离，在比特币的有生之年不一定能产生实质性的联系。如果量子计算在将来能强大到破坏加密体系的话，首先会冲击的是传统金融业。比如我们在网上进行的很多金融活动:网上银行、网上转账、网上支付，都会变得不安全了。所以与其担心量子计算对比特币的冲击，还不如担心量子计算对传统金融业的冲击，因为大多数的钱还是放在传统金融业里面的，加密货币的市值只占了现代金融体系当中的很小一部分。

②比特币当中没有把账户的公钥直接暴露出来，而是用公钥取哈希之后得到一个地址。比特币当中用的非对称加密体系，从私钥是可以推导出公钥的。所以只要把私钥保管好，公钥其实丢了也没有关系。从公钥显然是不能推出私钥的，否则就麻烦了。

假设将来量子计算技术发达了，能够从公钥中推出私钥，那怎么办呢？比特币在设计的时候又加了一层保护，没有用公钥本身，而是用公钥的哈希。所以如果有人想偷你账户上的钱的话，首先是要用地址推导出你的公钥，相当于把公钥的哈希值进行逆运算，而这一点即使是用量子计算机也是没有办法完成的。

加密和取哈希是两个不同性质的操作，加密的目的是为了将来能够解密，所以加密算法要保证信息的完整性，加密过程是不能丢失信息的，这样解密的时候才能够还原原来的输入。但是取哈希的过程一般是会造成信息的损失的，哈希函数一般都是不可逆的，因为有些信息在取哈希的过程中就已经丢失了。

比特币系统中用的哈希算法是SHA-256，算出的哈希值是256位，无论输入有多大，即使有几个T，算出来的哈希值也是256位。这样的运算过程显然是不可逆的。如果可逆那可就变成了一个超级压缩算法。

所以在比特币系统中，如果要收款就没必要把公钥暴露出来，只暴露公钥的哈希生成的地址就行了。将来要取钱的时候才需要公钥和私钥产生的签名。假如一个坏人在网上监听到了你取钱的交易，知道了你的公钥，他要偷你的钱，就必须实时的从公钥推导出私钥来，然后要产生一个跟你竞争的交易。你要把钱转你账户，他要把你钱转给他账户，即使这个坏人拥有量子计算机也很难几分钟内把你的私钥破解了，而且他发布的交易要抢在你交易的前面。

所以安全起见，一个地址用过之后就不要再用了，每次取钱最好把钱一次取走，即使取不完，也最好把钱转给另一个安全的账户。后面会讲到，以太坊中的地址也是从公钥当中推导出来的，但也不是公钥本身，也是公钥取哈希之后进行的转换。