## 一. 主要内容(比特币)

密码学基础（相关算法SHA256，DH，RSA等）

比特币的数据结构(哈希指针，Merkel树，UTXO，块头)

共识协议和系统实现

挖矿算法和难度调整

比特币脚本

软分叉和硬分叉

匿名和隐私保护

## 二、 BTC密码学原理

2.1 哈希

加密货币特性：crypto-currency 本身并不加密，转账地址是公开。主要使用了哈希函数，哈希函数特性：Cryptographic hash function ,collision resistance，抗碰撞性。

x=y 那么 Hash(x)=Hash(y)；而且 Hash(x)≠Hash(y)，那么x≠y

人为制造哈希collision没有好的方法，只能采用穷举法破解，brute-force。d在比特币系统中，采用SHA-256算法。输出空间为2^256。

哈希函数的作用：生成消息摘要digest。在实际生活中，以网盘上传文件为例，文件上传前计算文件的Hash值，下载保存文件后，再次计算文件的Hash值，如果2个Hash值不一致，文件就是被篡改了。如果2个Hash值一致，那么保存在云盘上的文件是没有被篡改过的。（云厂商为什么要篡改文件，云厂商本身的原因造成文件损坏，用户没有察觉这个问题，通过校验Hash值的方式可以发现）。

哈希函数作用二：信息隐藏。举例为，我要预测股票，将股票预测文本的摘要hash公布出去，但是股票预测这段文本M没有公布。当股票结果被证实了（经过一段时间），我再将股票预测文本M公布，所有人都可以用Hash算法验证M摘要信息hash2，如果hash2=hash，那么可以证明我之前公布文本M。

信息隐藏必须满足条件，输入的样本空间必须大。否则，恶意攻击者会穷举输入，找到产生摘要的原文。例如，股票的个数比较少，那么可以添加随机数，计算H(X || nonce)并公布，公布原文的时候要一并公布随机数nonce。可能有人会怀疑，我通过修改X和nonce，伪造了摘要信息，但是哈希函数的抗碰撞性，伪造出这个摘要值是非常困难的。

2.2 工作量证明

在比特币中要求，合法的区块必须满足：

H(block header)≤target

在区块头（block header）中有一个域nonce，穷举计算nonce，计算block header哈希值小于target，确定一个合法区块。

区块链的工作原理，记账节点（矿工）收集网络上广播的交易（tx）,维护一个交易集合，将若干笔交易打包，计算交易列表hash，放在block header中，穷举nonce域，使H(block header)≤target。

由于Hash函数的puzzle friendly特性，只能使用穷举算法。挖矿的主要工作就是通过穷举计算，找到符合条件的区块头。矿工找到符合条件块头，就获得了记账权，获得出块奖励和交易费，矿工要把这个区块信息发布到网络上，以便其他矿工能够收到这个区块信息。

2.3 账户信息

在比特币中，用户可以自己创建一个公私钥对，公钥就作为用户的账户信息。公私钥体系又称为非对称密码体系，一对密码分为（public key，private key），私钥保密，公钥公开。利用公私钥实现信息的保密通信和数字签名。

Alice想要给Bob发送信息，Alice在公开场景下获取了Bob的公钥，Bob的私钥自己保密。那么Alice用Bob的公钥加密信息发送给Bob，Bob接收到信息后，用Bob自己私钥解密。加密后的密文可以在不安全的信道上传输，没有Bob的私钥，无法解密密文。

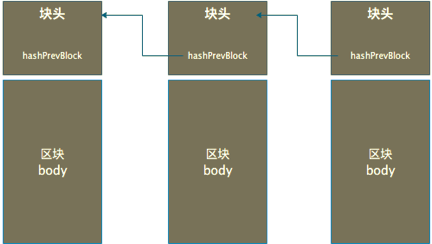
数字签名可以用来实现信息的真实性和抗抵赖性。Alice要向Bob提供一版word版报价单，Bob要求Alice提供报价单的真实性证明，Alice可以采用Hash算法计算文件的摘要，并用Alice自己的私钥对摘要加密，加密后的结果就是数字签名，Alice把原始word文档和数字签名一起发给Bob。Bob收到word文件时，可以计算word摘要，并用Alice的公钥解密数字签名，如果收到的word文档摘要和摘要数字签名解密后的结果一致，证明了报价单在传递过程中没有被篡改，也证明了确实是Alice认可的文档。

在BTC中，用户可以任意生成公私钥，将公钥公布作为自己的账号地址，私钥作为账户所属权证明。实现转账时，发起向某个公钥地址转账的交易（tx），那么这个公钥对应的账号就获得了这笔交易的金额。

在2.2节可以看到比特币的区块中记录的都是一笔一笔交易（tx）,而发起交易的节点并不是直接将交易写到区块链上，而是发布到网络上，让矿工节点接收。矿工节点通过挖矿获得记账权，发布打包交易的区块。

这里有一个问题，如果是用户自己产生公私钥，而私钥能够代表一个人身份，如果产生的公私钥重复，怎么办？比特币采用的公钥算法，公钥空间很大，这样即使用户自己产生公私钥，发生碰撞的概率也很小。同样是这个原因，通过穷举公私钥对比特币账户进行攻击，也是不能实现的。但是，用户在生成公私钥时，应采用优质的随机源。如果随机源用的不好，一次签名也可以泄露私钥。

三、 BTC中的数据结构

3.1 哈希指针

区块链是利用指针将区块连接起来，后一个指向前一个，并hash指针代替传统指针。采用Hash指针替代传统指针有2个特点：

（1）计算机中常用指针是内存中的地址，指向一块内存空间。比特币中区块链信息不可能全部保存在内存中，要在硬盘存储中实现指针，区块链利用用了(key，value)型存储数据库，区块头的hash值作为key，整个区块信息作为value进行存储。查找当前区块的内容，可以计算区块头的hash值，并在数据库中存储。区块头中保存了上一个区块哈希值，这个hash值就是hash指针，也是上一个区块的key值。

（2）通过hash指针实现区块链的防篡改，后面的区块保存了前一个区块Hash值（hashPreBlock），并用这个值计算自己的hash。如果想要篡改历史中交易信息，那么块头的hash值必然改变（如何产生影响，见3.2）。想要使修改的交易合法，就必须修改下一区块hashPreBlock，如此下一个区块又要修改下下一个区块的块头。这种修改区块链内容的方法，在实际过程中有很大的难度，修改一个区块，和挖矿获得记账的难度一样，从效率上是赶不上区块链中合法矿工的挖矿速度，因此很难让其他矿工接受。

块头结构包含如下信息：

int32\_t nVersion; //版本号，4字节

uint256 hashPrevBlock; //前一个区块的区块头hash值，32字节

uint256 hashMerkleRoot; //包含进本区块的所有交易构造的Merkle树根，32字节

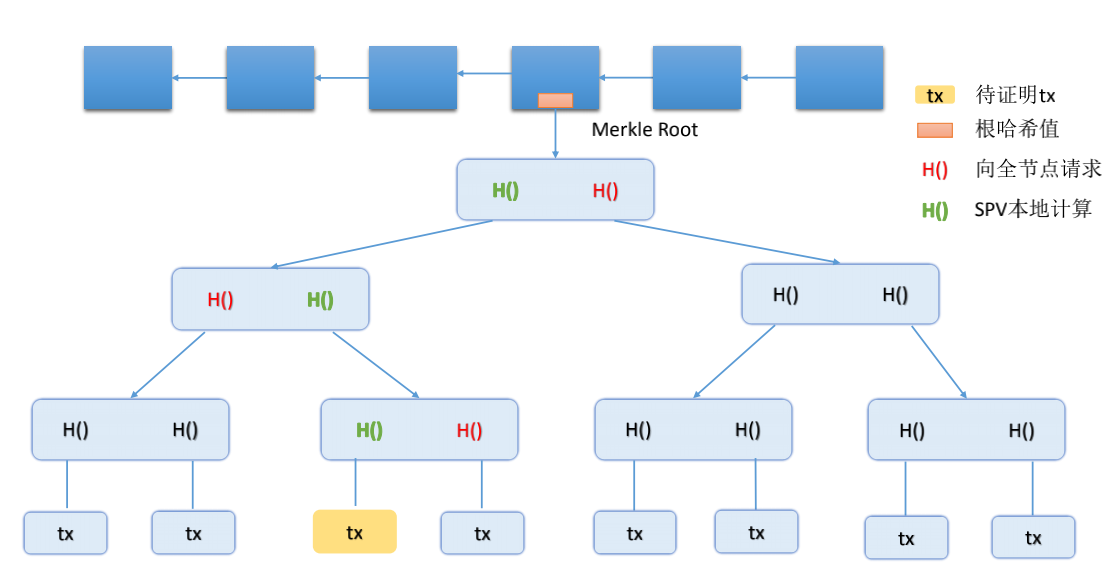
uint32\_t nTime; //Unix时间戳，4字节

uint32\_t nBits; //记录本区块难度，4字节

uint32\_t nNonce; //随机数，4字节



3.2 区块体交易结构

区块body中保存了交易列表，计算交易列表的hash，保存在区块header中。交易的组织采用了一个数据结构Merkle tree（默克尔树），结构如下图所示：

交易由下至上，组织成一棵二叉树，上层生成下层2个节点的hash值，最终生成一个哈希值Merkle root，保存在block header中。

如果恶意节点修改了交易（tx）中内容，那么Merkle tree就会改变，最终反映在Merkle root变化上，那么block header就会发生变化。采用这种方法保证了区块链中交易不可篡改性。

3.3 Merkle proof

区块链中包含2种节点，全节点和轻节点。全节点是保存完整区块链，参与挖矿记账的节点。在比特币网络中，大部分属于轻节点，轻节点不保存完整区块链，只保存最近一些节点的块头。常见的比特币手机钱包，就属于轻节点。

轻节点中没包含具体的交易信息，就给交易证明带来一些困难。在比特币系统中采用Merkle proof方法。轻节点想要证明图中tx（黄色框代表的某一笔）交易，轻节点保存了区块头信息，需要向全节点请求区中三个红色H()哈希信息，并在本地计算三个绿色H()哈希信息，并最终计算Merkle root信息，如果计算得到的Merkle root信息和轻节点保存在区块头中的Merkle root信息一致，那么证实了区块中确实记录了该笔交易。

Merkle proof在证明过程中，需要向周围全节点请求红色H()信息，似乎给恶意的节点篡改tx和H()使最终Merkle root合法机会，但是Hash函数的抗碰撞性保证构造出Merkle root是不可行的。恶意用户为证明非法的tx，伪造H()生成Merkle root是要花费计算资源暴力求解的。

这个暴力求解要比挖矿难度大得多，挖矿只要求hash结果小于target，而伪造Merkle root要算出指定的hash结果，这个值是唯一的。

想要证明一个区块中不包含一笔交易，在比特币系统是比较费力的，需要把区块的内容都发生给轻节点。可以采用Sorted merkle tree结构，首先将交易列表排序（采用交易Hash值排序），再按照Merkle tree构造Merkle root。当需要证明一笔交易不存在时，根据排序的交易列表，找到左右2个交易，并提供Merkle proof，如果证明了这2个交易位置，就可以证明了不存在特定的交易。

四、 BTC协议

4.1 双花与数字货币

央行可以发行一种和纸币对应的数字货币，数字货币中包含面额、编号和央行数字签名。如下图所示：



当Alice向Bob花费100元时，可以将手中的100元数字货币转交给Bob。但是，这种数字货币会有一个严重问题，Alice可以复制这个数字货币，并在下一次支付时候，再次使用。这个就是双花问题double spending。

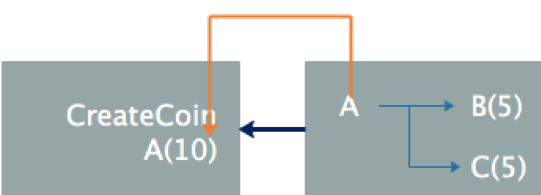
采用这样的数字货币，每笔交易都必须有央行参与，确定交易的数字货币是否属于支付方，并在支付成功后，将数字货币所属权修改为收款方。那么，这样的交易不需要货币支撑，就和我们日常的支付宝微信支付一样了。

4.2 去中心化数字货币

比特币是一种去中心化的货币，解决了2个问题：①谁来发行货币②解决双花问题。

货币的发行是根据比特币协议产生，由获得记账权的矿工取得的。当矿工挖矿的时候，交易列表中记录一笔“铸币”交易，由矿工获得当前区块的出块奖励。在时间线上，可能与我们的认知顺序不一样，你可能会认为矿工首先获得了记账权，再获得出块奖励，但实际情况是在出块之前，矿工就会将自己获得出块奖励的交易记录打包到区块中，再进行挖矿，当挖矿成功并得到确认之后自然得到出块奖励。

解决双花的方法是，每一笔交易都要指明花费来源于哪笔交易。下面是比特币系统中交易的例子，A通过挖矿获得10BTC，在下面A将这10BTC转给B 5BTC，转给C 5BTC。

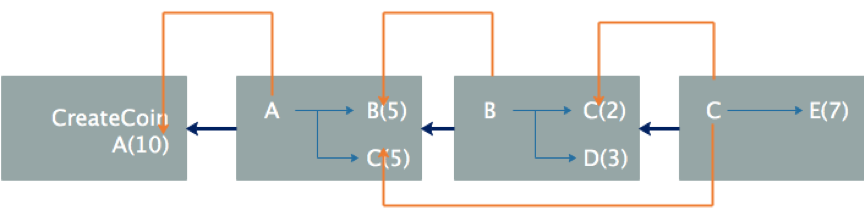


4.3 比特币中交易

一个典型的比特币交易如下图，连续4个区块，每个区块记录只记录了一笔交易。

在第一个区块中，A账户通过记账获得10BTC的记账奖励；第二个区块记录，A花掉了10BTC，转给B 5BTC，转给C 5BTC，A的10BTC来源是第一个区块；第三个区块记录B花掉5BTC，转给C 2BTC，转给D 3BTC，来源是第二个区块记录交易；第三个区块记录C转给E 7BTC，来源与第二个区块C获得的5 BTC和第三个区块中C获得的2 BTC。

以上例子中，每个区块只记录了一笔交易，而真实的比特币系统中会记录多笔（几千）笔交易。

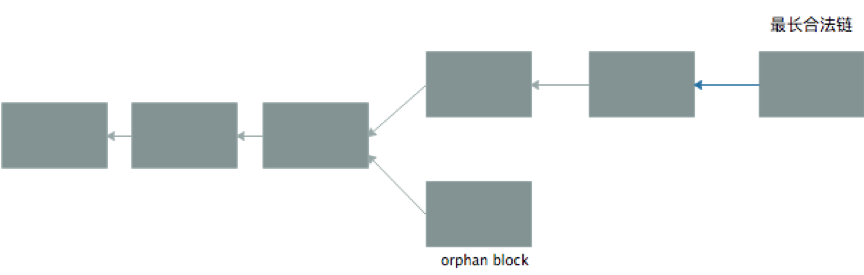


每笔交易，都需要转账者签名，收款方就是对应的公钥地址。A转给B 5BTC，指明币的来源是上一笔铸币交易，通常A要说明自己的公钥，并在这笔交易上附上自己的数字签名。

这笔交易在比特币网上扩散，矿工节点要验证交易的合法性，用A的公钥验证交易的数字签名，同时验证A的公钥和交易来源交易的收币用户是同一个用户。一笔转账交易的地址，通常用用户公钥的hash值，这样不会直接显示用户的公钥，有更好的匿名性。

4.4 比特币共识机制

在比特币网路中，每个账户都可以发布交易，每个节点独立打包交易，最终对账本内容取得分布式共识（distributed consensus）。在比特币中，分布式共识是通过投票机制达成共识，采用比拼算力的方法，实现投票权。投入的算力越多，获取记账权概率越大。联盟链hyperledger fabric，合法的用户才能投票。区块链中，可能出现临时性分叉。如图，两个矿工同时获得记账权，矿工会挑选一条链，继续往下挖，最终会产生一条最长链，其他的矿工都会沿着合法链往下继续挖矿，而分叉的链条被遗弃，没有任何作用。



4.5 出块奖励

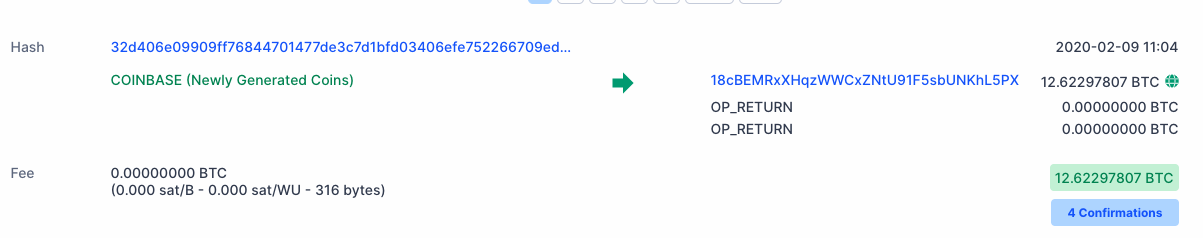
在比特币系统中，通过挖矿获得区块记账权的矿工获得比特币奖励。在初始阶段，每个区块获得50 BTC，约定每隔21万个区块，奖励数减半。

210000∗10 / 365∗24∗60≈4

每个区块的出块时间约为10分钟，21万个区块约占用4年时间。2009年，BTC初始时出块奖励是50BTC,2013年时变为25BTC，2017年是12.5BTC。离我们最近出块奖励变化时间是2021年，BTC出块奖励降为6.25BTC。2020年2月，比特币的价格约为9766美元，那么出块奖励折合成美元就是12.2万美元。

4.6 交易费

除了出块奖励外，获得记账权的矿工，还可以获得交易费。交易费是指，由支付方在每笔交易中除了应收金额外，多付给矿工的BTC，为了鼓励矿工将交易打包到区块中，那么交易就可以尽早确认。交易费不是必须的，交易中也可以不包含交易费。当前，比特币系统中，平均有10万条交易等待打包确认，而每个区块有1M大小限制，而矿工更愿意将包含交易费的区块。如下图的交易，COINBASE交易就是矿工的铸币交易，矿工获得的奖励除了12.622 BTC，当前的出块奖励是12.5 BTC，额外的0.122就是这个区块打包全部交易的交易费。



五、 BTC实现

5.1 UTXO

在比特币中，采用transcation-base ledger，即基于交易的记账本。通常大家更容易理解account-base ledger基于账户的记账本。由于采用基于交易的记账本，因此在比特币系统中很难确定一个账户下拥有多少比特币。比特币系统中，全节点负责维护UTXO数据结构，保存没有花费的交易。全节点将UTXO存储内容中，并用来验证收到的交易的正确性，当交易的输入是保存在UTXO中的未花费交易，那么交易来源的合法性是正确的。当转账交易发生之后，会在UTXO中增加一条记录。UTXO保存在内存中，一方面是方便快速查找，另一方面由于比特币是基于transcation-base ledger，可以通过重放交易记录重新生成UTXO，因此可以不用持久化存储。

问题：没找到资料UTXO的数据结构，如何快速验证交易是否存在在UTXO中。

有些交易长期保存在UTXO中，一些由于丢失私钥，那么这些比特币再也无法交易。还有一些，例如中本聪早期挖矿所得，他本人也不用来交易，也会长期存储在UTXO系统中。

5.2 Bernulli trail

提到挖矿是一个无记忆性的过程Memoryless，符合Bernulli trail。

挖矿的工程如果不是Prograss free，那么算力强的矿工会取得不成比例的优势。

这块的理解，以抛硬币为例也是一个bernulli trail，memory less的过程，我想要一个正面的结果，和我前面几次抛硬币获得反面的结果都没有关系。在比特币挖矿过程中，我计算一个nonce，再计算下一个nonce时是否能满足条件都是不可知的。

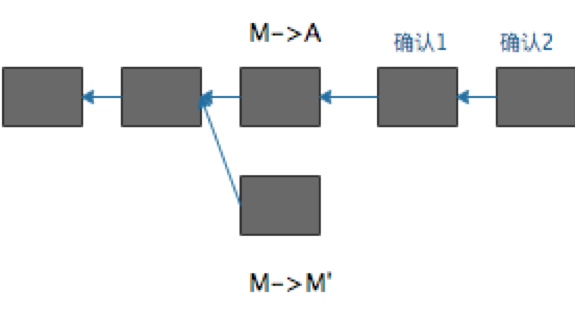
这个Bernulli process过程，和穷举密钥攻击的过程相识，但不一样。穷举密钥，在密钥空间中存在这样的密钥，计算一个结果就排除一个结果。但是挖矿过程，就好像抛一个2^256个面的骰子，只有2^target个面是符合条件的。

5.3 挖矿的意义

挖矿本身没有意义，但是挖矿是比特币安全的保障。Bitcoin is secured by mining。在分叉攻击的例子中，恶意节点发布一个包含回滚交易的区块，网络上的节点有可能按照恶意路径继续挖矿，造成交易信息失败。

防止分叉交易的方法，等待该链上多产生几个区块，又叫多等待几个确认。此处恶意节点想要回滚交易，就不是争夺一个区块的记账权，而是连续几个区块的记账权，难度很大。在攻击时，攻击者要沿着自己构造的区块往下生产，而网络上的其他矿工默认沿最长合法链挖矿挖矿，会保持正常的交易继续进行。

通过这个分叉攻击的例子，说明了挖矿是区块链安全的保障。在实际中，交易要等6个确认，才算被认为是生效。



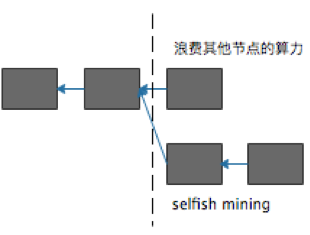
上图中，M首先转A若干BTC，同时发布一个区块，包含一个M转给M‘的区块，如果下面M转给M’的交易生效，根据UTXO的记录上面区块记录的M转给A的交易就不会生效。

5.4 Selfish mining

恶意节点挖到节点后，先不发布，并沿着自己挖到的区块继续挖矿，等有其他节点发布了，发布2个连续的区块。目的是浪费其他矿工的算力。

但是这种selfish mining是有风险的，不发布区块，就得不到区块奖励，连续领先挖出2个区块的概率是很低的。

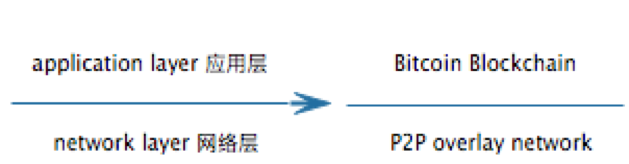
通常，正常的节点收到其他节点发布的区块，都会停下自己的挖矿工作，沿着发布区块往后继续挖矿。因为挖矿是pregress free，继续挖矿和沿着新区块挖矿意义一样。



六、 BTC网络

6.1 网络结构

节点之间采用flooding泛洪广播，网络层采用p2p网络，应用层采用区块链协议。

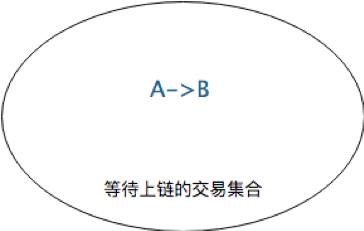


比特币网络的设计思想是简单鲁棒。节点退出网络，其他节点一段时间没有收到该节点的信息，删除保存的这个节点信息。比特币区块链网络忽略物理环境的拓扑结构，因此邻近节点可能是在真实环境中距离很远的节点，秉持越简单越鲁棒的特点。

比特币区块大小1M字节，大约花费几十秒广播到全部网络节点上。

6.2 广播

每个节点都会广播收到的交易信息和区块信息。



节点要维护一个接收交易的集合，这些交易都是经过广播，且被认为是合法的交易，但是由于去中心化的特性，还会遇到一些情况。

节点保留了一个A->B的交易，又收到一个A->C的交易，2笔交易指向同一笔未花费交易，此时节点会丢弃收到A->C 交易。

如果节点收到一个区块，包含A->C的交易，那么待上链A->B就是非法的，节点会丢弃这笔交易。

七、 挖矿难度

7.1 难度

比特币中设定合法的区块，区块头的哈希值必须小于等于target。采用的哈希算法是SHA-256，计算哈希的结果是256位，将结果看成一个整数，与target转化成256位整数比较。挖矿的过程就是找到nonce，计算block header的哈希满足上面的条件。公式描述了target和挖矿难度之间的关系，target越小难度越大。difficulty\_1\_target初始时比特币的难度。

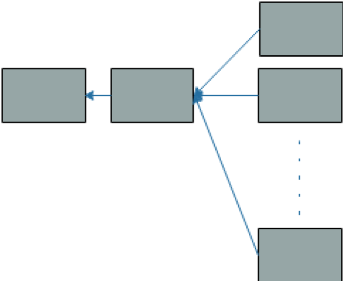


7.2 难度调整

为保证比特币的出块时间在10分钟左右，比特币系统会根据前一段时间的出块时间调整target值。每隔2016个区块，采用下面的公式对target调整。expect\_time就是10分钟，如果actual\_time大于10分钟，target变大，降低难度；actual\_time小于10分钟，target变小，难度增大。难度调整还有一个限制，调整幅度不能超过4倍和1/4。



比特币系统中，如果区块时间设置的更短，例如5分钟，那么比特币系统中会更容易出现分叉，如图所示。系统中出现多分叉时，相当于系统中合法节点算力被分散，那么系统更容易被分叉攻击。10分钟是比特币设计者中本聪设定的。



八、 BTC 挖矿

8.1 挖矿

比特币系统中有全节点和轻节点，全节点是挖矿，即争夺记账权、打包区块。挖矿的过程就是穷举nonce，使区块头哈希值小于等于target。

挖矿的过程是无记忆性的memoryless，又称progress free。之前的章节解释过这一个过程，比如抛硬币，我想要抛一个正面向上的硬币，和已经抛过的过程无关系。

每尝试一个nonce，都是在一定概率下满足挖矿条件，和之前尝试了多少次nonce没关系。这种memoryless的特性，使得挖矿成功的概率等于节点占总算力的比例。

8.2 挖矿方法

比特币的mining puzzle是哈希算法，流行的运算工具有CPU、GPU、ASIC等。

利用CPU挖矿，CPU中的大部分运算单元和内存都没有充分利用，挖矿速度也不高。利用GPU挖矿，可以并行计算。ASIC是专用芯片，由硬件实现算法，速度更快。

通常的矿机都是ASIC，高度并行化的专用芯片。但是ASIC制作成本高、周期长，设计一款挖矿通常需要一年时间，投入巨大。而矿机的淘汰速度也很快，过一段时间就有更好的矿机问世，比老款矿机速度更快，老款矿机就会因电费的原因被淘汰。

8.3 矿池挖矿

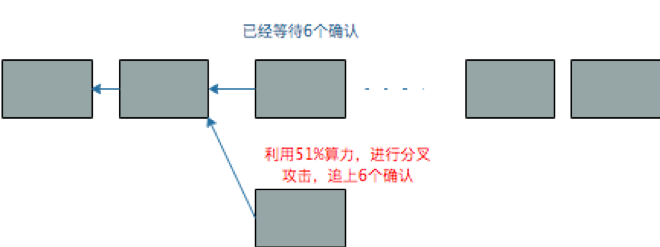
矿池可以将分散的算力集中起来，一个矿池可以管理很多个矿工，矿池负责打包区块，矿工负责接收矿池的任务进行hash运算。

矿池中的矿工按贡献率分配挖矿奖励，矿工的贡献也是采用工作量证明的方式。实际，矿池要求矿工提交一个较比特币puzzle难度低的结果，比如比特币puzzle要求包含70个0，而矿池要求puzzle是60个0。

矿工每提交一个这样的结果，被认为比较了一个share。这些share中可能包含了符合比特币要求的结果，矿池就可以获得收益。而绝大部分的share是没有意义的，仅仅作为矿工的工作量证明。

从概率上分析，加入矿池的矿工的收入期望并没有增加，获得的收益任然是矿工占全部节点算力的比例。但是，通过加入矿池，矿工收入的稳定性提高了。一些矿池为了吸引矿工加入，会为矿工提供额外的奖励，这也导致了超大型矿池的出现。

出现超大型矿池可能占全部节点算力的51%，这种情况在区块链历史上是出现过的。大型矿池可以利用51%算力，进行分叉攻击。而获取51%攻击想要盗取其他人账户中的比特币是办不到的。

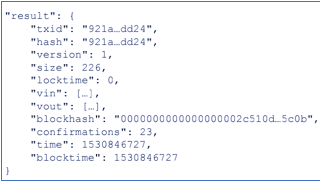


矿池中是否有恶意的矿工？当矿工挖到了符合比特币puzzle的结果，但是不将这个结果发送给矿池，矿池就不会得到出块奖励。矿池之间是竞争关系，可能将自己的节点加入对手的矿池，搞一些破坏，分散对手的算力。

九、 BTC脚本

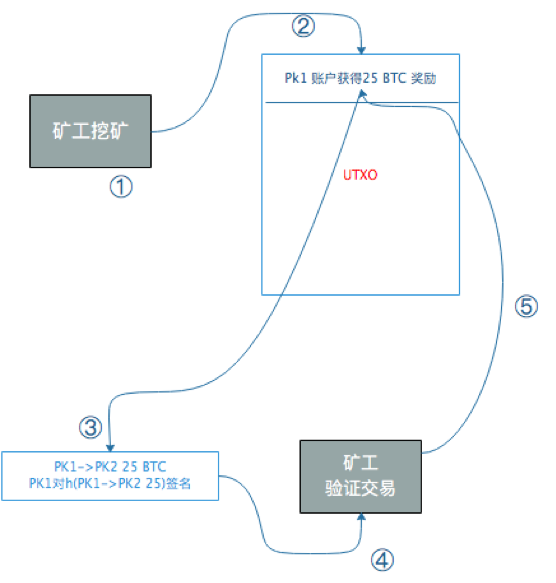
9.1 交易数据结构

比特币系统在网络中广播交易，数据结构如下图。



其中vin被称作输入脚本，vout被称为输出脚本。交易是否合法，首先要检查比特币的来源，输入脚本中要包含交易中比特币来源的证明，要素是签名。输出脚本中要包含接收方的地址。比特币系统中，不会校验输出脚本的合法性。

比特币被称作可编程的加密货币，交易过程中需要执行输入输出脚本。在比特币系统，全部币都来至挖矿，即铸币交易。在铸币交易中，没有输入脚本，输出脚本指明了矿工的地址。可以直接用公钥地址作目的地址[ws1] ，也可以用矿工公钥地址hash作为目的地址。



矿工挖矿成功，获得记账权，得到铸币奖励并写入区块链。

当其他节点收到广播的新区块，就会修改本地的UTXO，增加一条PK1账户转入25BTC的记录。

当PK1要花掉这个25BTC时，要构造一条交易，在交易中指明交易id（txid），并指明将钱转给PK2，并对交易用PK2对应的私钥签名。

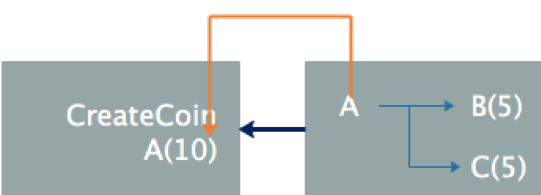
当其他节点收到这个交易的时候，会校验是否与待上链交易冲突。

全节点还要验证UTXO中存在txid交易，再用PK1公钥验证签名，验签通过说明是PK1发起的交易，最终证明交易合法。

9.2 输入脚本和输出脚本

比特币系统中，可以看做以交易tx驱动UTXO账本记录。通常一笔交易要在UTXO中花费掉记录，并生成新的记录。

UTXO中添加记录，被称为将比特币锁定在账本上，与锁定对应，要花掉比特币的行为称作解锁。一笔交易通常解锁、锁定过程。下图中，A转给B 5BTC和C 5BTC，A花费铸币交易称作解锁，在UTXO中产生B(5) C(5)称作锁定。



输入脚本的目的就是提供解锁证明，与之对应锁定脚本可以将比特币锁定在一下4种类型：

锁定在公钥中。即支付对象是接收者的公钥，pay to public key（P2PK）

锁定在公钥哈希中。即支付对象是接收者的公钥哈希，可以隐藏接收者的公钥信息，pay to public key hash（P2PKH）

锁定在锁定在多重账户中。支付对象以多个接受者为整体。

锁定在一段脚本中。Pay to Script Hash（P2SH）

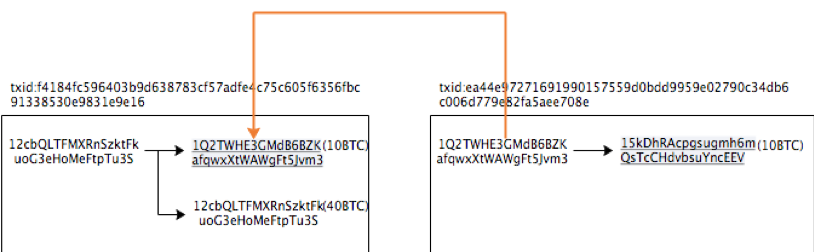
9.3 脚本实例

9.3.1 P2PK型交易

f418交易产生2个结果，ea44交易花掉了10BTC，交易信息如图(注f418和ea44是txid的前4位，交易txid也是交易的hash)。2笔交易的地址如下：

https://www.blockchain.com/en/btc/tx/f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16

https://www.blockchain.com/en/btc/tx/ea44e97271691990157559d0bdd9959e02790c34db6c006d779e82fa5aee708e



一、f418输出脚本:



查看交易信息，重点关注输出outputs，输出脚本Pkscript如下：

04ae1a62fe09c5f51b13905f07f06b99a2f7159b2225f374cd378d71302fa28414e7aab37397f554a7df5f142c21c1b7303b8a0626f1baded5c72a704f7e6cd84c

OP\_CHECKSIG

其中：

04：操作符 OP\_PUSHDATA

ae1a62fe09c5f51b13905f07f06b99a2f7159b2225f374cd378d71302fa28414e7aab37397f554a7df5f142c21c1b7303b8a0626f1baded5c72a704f7e6cd84c：公钥64位，公钥由2个32字节数组成，这里转换为16进制表示是64位。

OP\_CHECKSIG:操作符

点击HEX按钮，显示16进制信息。这段HEX中，开头4104、结尾ac都是操作符，这一段HEX字符就是输出脚本，就是锁定脚本。

4104ae1a62fe09c5f51b13905f07f06b99a2f7159b2225f374cd378d71302fa28414e7aab37397f554a7df5f142c21c1b7303b8a0626f1baded5c72a704f7e6cd84cac

二、ea44脚本输入：



通过1Q2T(UXTO)指明了来源，引用Pkscript信息。Sigscript是签名信息。

P2PK(Pay to Public Key)的执行脚本以堆栈方式执行，

input script:

PUSHDATA(Sig)

output script:

PUSHDATA(PubKey)

CHECKSIG

执行过程如下，最终结果为TRUE证明校验通过。



9.3.2 P2PKH型交易

P2PKH(pay to public key hash)是在交易中，指明收货地址的hash值。用Hash值代替公钥，可以隐藏公钥信息，而且Hash值的长度比公钥的长度短，数据量小。

9.3.3 P2SH型交易

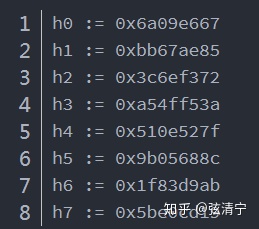
P2SH（pay to script hash）是指交易的地址是一段代码的hash值，与之与对应解锁交易必须给出这段交易的原文script，而且运行script必须得到正确的结果。

## 相关加密算法

## 1. SHA256算法

对于任意长度的消息，SHA256都会产生一个256bit长的哈希值，称作消息摘要。

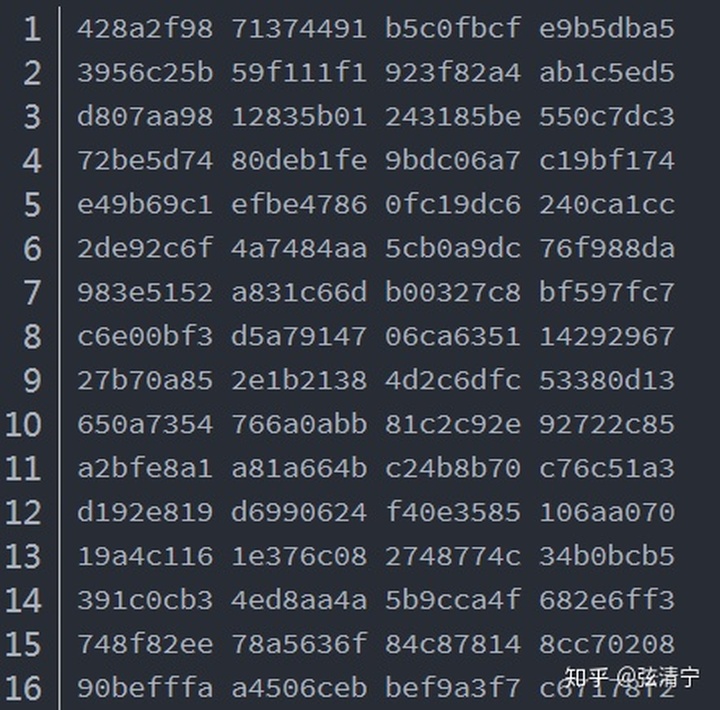
9.1.1常量初始化

SHA256算法中会用到8个哈希初值以及64个哈希常量。这八个初始值如下。

这些初值是对自然数中前8个质数（2,3,5,7,11,13,17,19）开平方根，对其小数部分取前32bit而来。如根号2约为0.414213562373095048



SHA256算法中，用到的64个常量如下：和8个哈希初值类似，这些常量是对自然数中前64个质数的立方根的小数部分取前32bit而来。



9.1.2信息预处理

SHA256算法中的预处理就是在想要Hash的消息后面补充一些特定信息，使整个消息满足指定的结构。信息的预处理分为两个步骤：附加填充比特和附加长度。

1）附加填充比特

在报文末尾进行填充，使报文长度在对512取模以后的余数是448。填充是这样进行的：先补第一个比特为1，然后都补0，直到长度满足对512取模后余数是448。需要注意的是，信息必须进行填充，也就是说，即使长度已经满足对512取模后余数是448，补位也必须要进行，这时要填充512个比特。因此，填充是至少补一位，最多补512位。

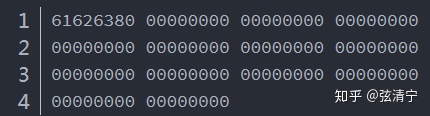
举个栗子：以信息“abc”为例显示补位的过程。

a,b,c对应的ASCII码分别是97,98,99

于是原始信息的二进制编码为：01100001 01100010 01100011

补位第一步，首先补一个“1” ： 0110000101100010 01100011 1

补位第二步,补423个“0”：01100001 01100010 01100011 10000000 00000000 … 00000000

补位完成后的数据如下（为了简介用16进制表示）：

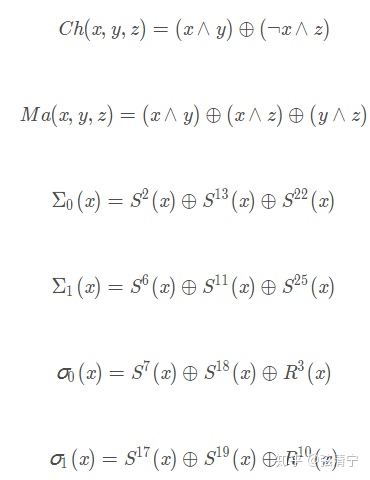
这时候会有个小问题，为什么是余448。这是因为在第一步的预处理后，第二步会再附加上一个64bit的数据，用来表示原始报文的长度信息。而448+64=512，正好拼成了一个完整的结构。

2）附加长度值

附加长度值就是将原始数据（第一步填充前的消息）的长度信息补到已经进行了填充操作的消息后面。SHA256用一个64位的数据来表示原始消息的长度。长度信息的编码方式为64-bit big-endian integer，就是所谓的大端存储，百度一下就好了。这里举个小栗子。比如存0x123456，正常存储就是0x123456，而大端存储就会存成0x563412这样子，两个数字占一个字节，所以56占一个字节，34一个，12一个，简单点说，就是按字节逆序存储。

**9.1.3逻辑运算**

SHA256散列函数中涉及的操作全部是逻辑的位运算，包括如下的逻辑函数：

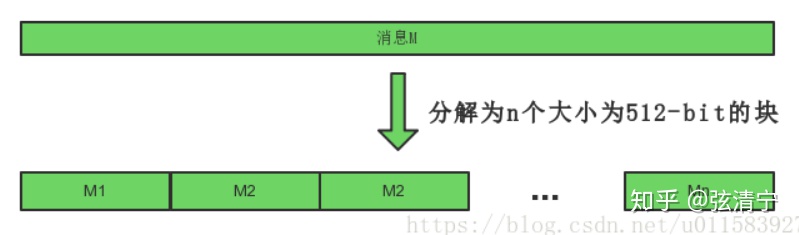




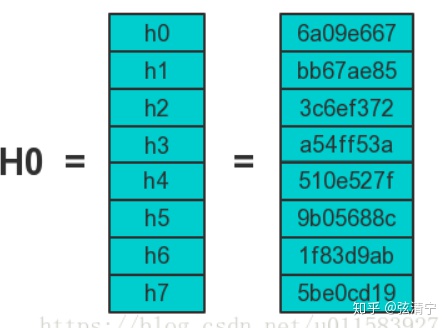
9.1.4计算消息摘要

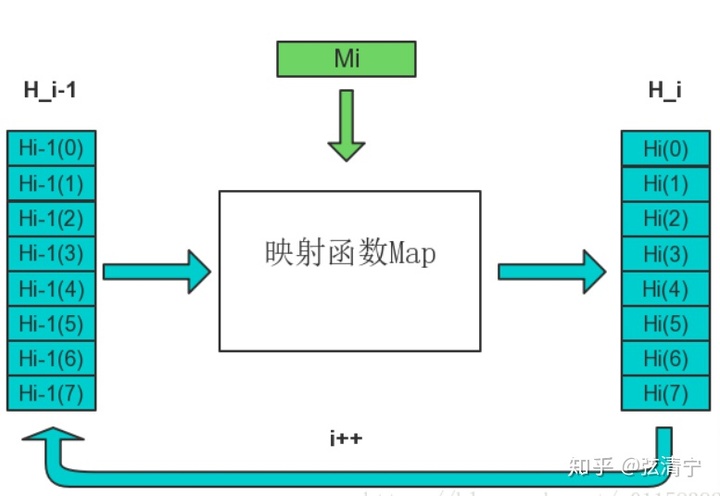
现在来介绍SHA256算法的主体部分，即消息摘要是如何计算的。

首先：将消息分解成512-bit大小的块。



假设消息M可以被分解为n个块，于是整个算法需要做的就是完成n次迭代，n次迭代的结果就是最终的哈希值，即256bit的数字摘要。





H0初值设置为前面介绍的8个哈希初值，经过第一个数据块进行运算，得到H1，即完成了第一次迭代H1经过第二个数据块得到H2，……，依次处理，最后得到Hn，Hn即为最终的256-bit消息摘要。图中256-bit的Hi被描述8个小块，这是因为SHA256算法中的最小运算单元称为“字”（Word），一个字是32位。

下面开始介绍每一次迭代的内容。

1）STEP1：构造64个字（word）

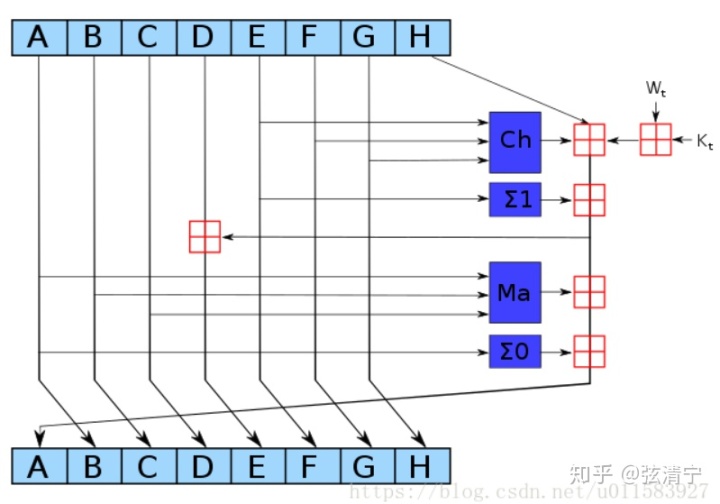
对于每一块Mi，将其分解为16个32-bit的big-endian的字，记为w[0], …, w[15]，也就是说，前16个字直接由消息的第i个块分解得到，而我们总共需要构造64个字，故其余的48个字由如下迭代公式得到：



于是经过这些我们就得到了W1，W2，...，W64。

2）STEP2：进行64次循环

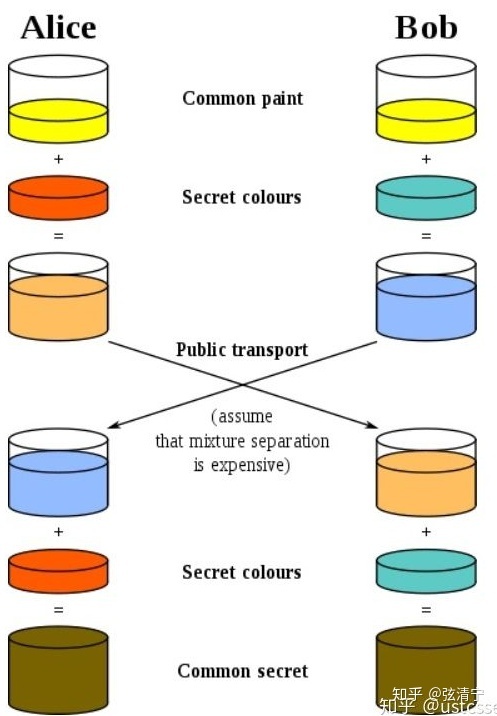
映射 Map包含了64次加密循环，每次加密循环可以由下图描述：



图中，ABCDEFGH这8个字（word）在按照一定的规则进行更新，其中深蓝色方块是事先定义好的非线性逻辑函数，详细定义看9.1.3里的图；红色方块是取模运算（mod2^32）；ABCDEFGH一开始的初始值分别为H{i-1}(0),H{i-1}(1),…,H{i-1}(7) ；Kt是第t个密钥，对应我们上文提到的64个常量；Wt就是刚刚得到的那些Wt。这个过程进行64次以后，就完成了这一块的迭代。

开始第二块迭代的时候，把第一块迭代结束的值当做第二块初始值灌入第二次迭代的开始，持续像第一次迭代那样。通过迭代n次对ABCDEFGH这八个字进行加密（其实整个算法过程进行了64n次），最后输出的8个字就是SHA256哈希后的值。

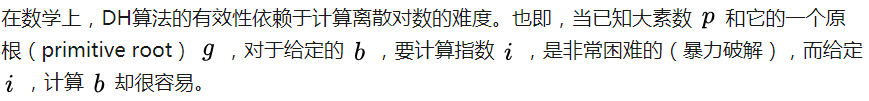
## 2.DH算法

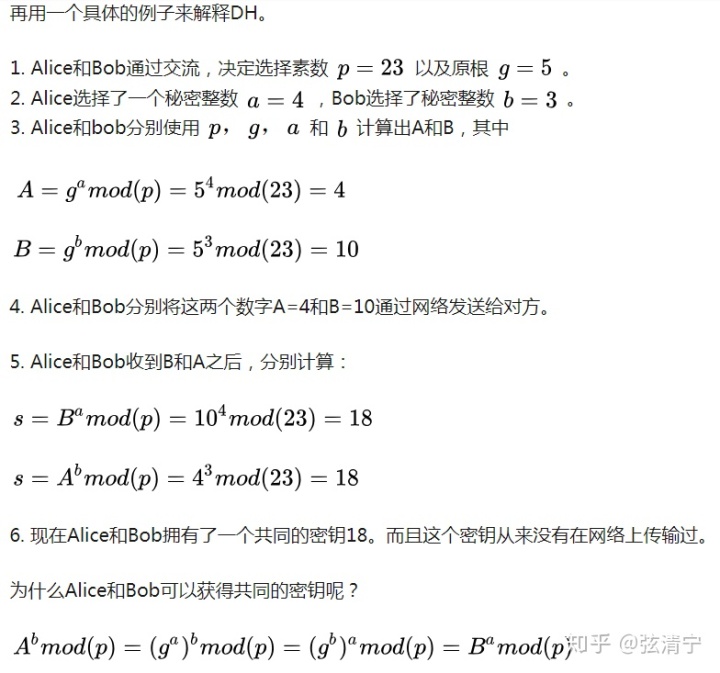
原理：DH协商过程，Alice和Bob首先挑选一个颜色（黄色），这个颜色是可以公开的（每次通信不同）；然后再各自挑选一个秘密的颜色（Alice橙色，Bob青色）。然后Alice和Bob各自将自己的秘密颜色和黄色进行混合，得到了另外的两个颜色（Alice橙褐色，Bob淡蓝色）。Alice和Bob分别将自己的颜色发给对方。Alice和Bob在收到对方发来的颜色后，再分别和自己的颜色相混合，此时，两人得到了一个相同的颜色（黄褐色）。

在这个过程中，攻击者Eve可以一直监听网络，并且获得Alice和Bob在网络上交换的所有信息。也即，可以获得黄色、橙褐色、淡蓝色这些信息。阻止Eve获得最终的黄褐色的是Alice和Bob分别挑选的秘密颜色(说白了就是因为有这个秘密颜色所以Eve得不到最终的秘密—黄褐色)，橙色和青色。也即，需要能够证明，即使Eve得到了黄色和橙褐色，Eve也不能推导出Alice的秘密颜色。

先定义一下原根。

1.对于一个数P,gi≡1(mod P)P,gi≡1(mod P)的最小正整数ii是φ(P)φ(P)，那么就称g是P的原根.  
2.假设一个数gg对于PP来说是原根，那么gimod Pgimod P的结果两两不同,且有 1<g<P,1<i<P1<g<P,1<i<P，那么g可以称为是P的一个原根





## 3.RSA算法

RSA算法是现今使用最广泛的非对称密码算法，也是号称地球上最安全的加密算法。

密文＝明文EmodN密文

公钥＝(E,N)公钥＝(E,N)

|  |  |
| --- | --- |
| 公钥 | （E，N） |
| 私钥 | （D，N） |
| 密钥对 | （E，D，N） |
| 加密 | 密文＝明文EmodN密文＝明文EmodN |
| 解密 | 明文＝密文DmodN明文＝密文DmodN |

求N

准备两个质数p，q。这两个数不能太小，太小则会容易破解，将p乘以q就是N

N=p∗qN=p∗q

求L

L 是 p－1 和 q－1的最小公倍数，可用如下表达式表示

L=lcm（p－1，q－1）L=lcm（p－1，q－1）

4.3 求E

E必须满足两个条件：E是一个比1大比L小的数，E和L的最大公约数为1   
用gcd(X,Y)来表示X，Y的最大公约数则E条件如下：

1 < E < L

gcd（E，L）=1

之所以需要E和L的最大公约数为1是为了保证一定存在解密时需要使用的数D。现在我们已经求出了E和N也就是说我们已经生成了密钥对中的公钥了。

4.4 求D

数D是由数E计算出来的。D、E和L之间必须满足以下关系：

**1 < D < L**

**E＊D mod L ＝ 1**

只要D满足上述2个条件，则通过E和N进行加密的密文就可以用D和N进行解密。   
简单地说条件2是为了保证密文解密后的数据就是明文。   
现在私钥自然也已经生成了，密钥对也就自然生成了。   
小结下：

|  |  |
| --- | --- |
| 求N | N＝ p ＊ q ；p，q为质数 |
| 求L | L＝lcm（p－1，q－1） ；L为p－1、q－1的最小公倍数 |
| 求E | 1 < E < L，gcd（E，L）=1；E，L最大公约数为1（E和L互质） |
| 求D | 1 < D < L，E＊D mod L ＝ 1 |

我们用具体的数字来实践下RSA的密钥对对生成，及其加解密对全过程。为方便我们使用较小数字来模拟。

**5.1 求N**

我们准备两个很小对质数，   
p ＝ 17   
q ＝ 19   
N ＝ p ＊ q ＝ 323

**5.2 求L**

L ＝ lcm（p－1， q－1）＝ lcm(16，18） ＝ 144   
144为16和18对最小公倍数

**5.3 求E**

求E必须要满足2个条件：1 < E < L ，gcd（E，L）=1   
即1 < E < 144，gcd（E，144） ＝ 1   
E和144互为质数，5显然满足上述2个条件   
故E ＝ 5

此时公钥=(E，N）＝ （5，323）

**5.4 求D**

求D也必须满足2个条件：1 < D < L，E＊D mod L ＝ 1   
即1 < D < 144，5 ＊ D mod 144 ＝ 1   
显然当D＝ 29 时满足上述两个条件   
1 < 29 < 144   
5＊29 mod 144 ＝ 145 mod 144 ＝ 1   
此时私钥＝（D，N）＝（29，323）

**5.5 加密**

准备的明文必须时小于N的数，因为加密或者解密都要mod N其结果必须小于N   
假设明文 ＝ 123   
则 密文＝明文EmodN＝1235mod323=225

5.6 解密

明文＝密文DmodN＝22529mod323=123  
解密后的明文为123。