比特币白皮书

中本聪 (https://github.com/sylsaint译)

April 9, 2018

0.1 比特币白皮书

0.1.1 比特币：一个点对点的电子现金系统

摘要 在纯粹的点对点电子现金系统中，付款人可以不经过第三方金融机构而直接向收款人发起在线支付。电子签名解决了点对点支付的部分问题，但是如果必须要一个可信的第三方来判断是否存在双重支付 [1](double spending)，点对点支付的优势就荡然无存了。

基于此，我们提出了一种使用点对点网络来解决**双重支付**的方法。通过将交易哈希进不断增长的区块链上，网络会把所有的交易时间戳化，进而形成一个无法被修改的记录 (修改记录需要重做相应的工作量证明)。生成的最长链不仅能作为见证所有已发生事件序列的证明，也可以证明其来自于拥有最大 CPU 计算力的资源池 [2]。只要控制主要 CPU 算力的节点不会联合起来攻击比特币网络，他们就可以产生最长的链并且超过那些攻击者。比特币网络本身需要一个最小化的网络结构。消息在比特币网络上传播是基于尽力而为的策略的，节点可以自由的离开或重新加入网络，(在加入时) 接受在他们离开的时候产生的最长的区块链。

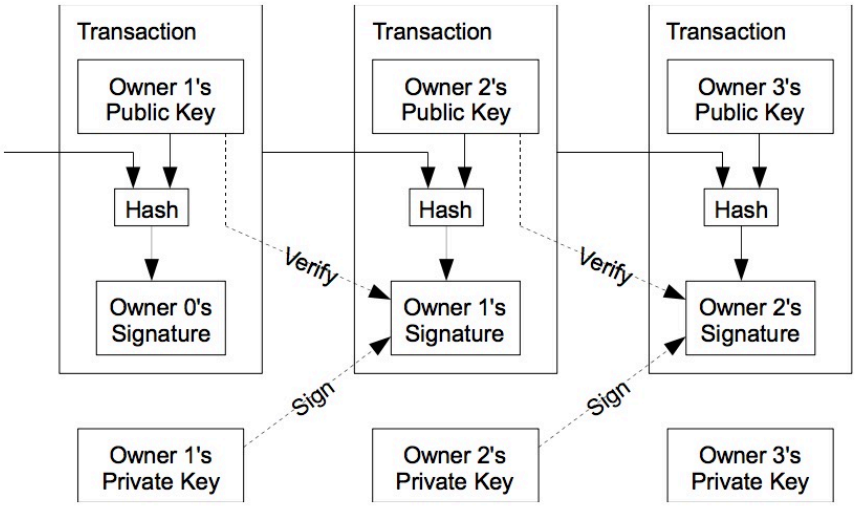
简介 互联网上的商业活动目前已经几乎都依赖于金融机构了，他们作为可信的第三方机构来处理所有的电子支付活动。虽然对于绝大部分交易，系统已经运行的足够好了，但是基于信任模型的系统具有与生俱来的弱点 [3]。完全不可逆的交易几乎是不可能的，因为金融机构无法避免调节纠纷。调解会导致交易费用的增加，从而限定了最小的交易额度以及切断了临时小额交易的可能。由于交易存在可逆性，对信任的需求更加广泛。商人必须更加小心他们的顾客，迫使顾客提供更多的个人信息，其中有些信息对于一次交易来说可能是不必要的。同时，一定比例的商业诈骗行为被认为是不可避免的。如果你在线下使用现金支付，额外的开销和交易的不确定性都是可以避免的。但是在没有可信第三方的情况下，目前尚无一个机制能够解决上述问题来确保线上交易的顺利完成。

因此，我们需要的就是一个不是基于信任而是基于密码学证明的电子支付系统，其允许有交易意愿的双方可以不经过可信第三方而直接交易。在计算上不可逆的交易可以保护卖家免于诈骗，而且可以很容易地实现例行的契约机制来保护消费者。

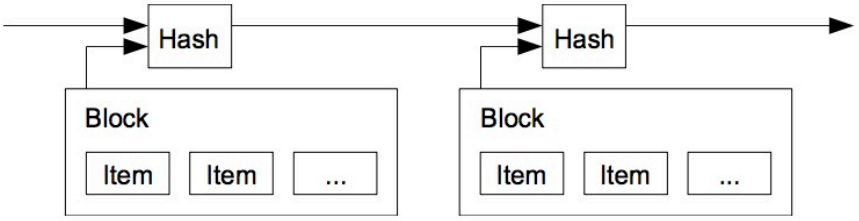
在本文中，我们提出了一种解决**双重支付**问题的方法，通过点对点的分布式时间戳服务器，生成交易时间顺序的计算证明。只要全体诚实节点控制的 CPU 算力大于任何一个攻击节点群组，系统就是安全可信的。

交易 我们定义了电子货币是一系列数字签名组成的链式结构。拥有者可以把货币转移给其他人，具体方法是，对上一笔交易的哈希值和下一个人的公钥用自己的私钥进行数字签名，并且将中文写信息 (哈希值，公钥，签名信息) 置于货币的末尾。收款人可以通过校验签名来确认链 (货币) 的所有权。

上述过程存在一个问题，收款人无法确认其中一个拥有者是否进行了**双重支付**。一个通用的方法是引入可信的第三方权威机构或者铸币厂来确认每笔交易是否存在**双重支付**的问题。在每笔交易之后，货币必须要返回到铸币厂，铸币厂销毁旧币并生成新币，只有从铸币厂直接发行的货币才被



交易



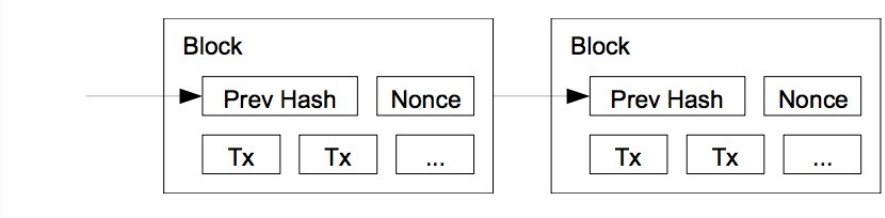
时间戳

认为是没有用于双重支付的。这种方案的弊端在于，整个金融系统的命运依赖于铸币公司，所有交易都要经过他们，就像银行那样。

那么，我们就需要一种手段能够确认货币之前的那些拥有者是否对更早的交易签名过。对我们来说，最早发生的交易才是有意义的，我们并不关心之后是否有人尝试**双重支付**。确认一个交易存在的唯一方法就是检查所有的交易。在铸币厂模型中，铸币厂检查所有交易并确定交易的顺序。为了能够在没有可信第三方的情况下实现这个目的，交易必须被公开声明出去，同时我们需要一个系统能够让所有参与者在交易的顺序上达成共识。系统需要向收款人证明在每一笔交易的时刻，大多数节点都同意它是第一个被接收的。

时间戳服务器 (miner) 我们的解决方案从时间戳 [4] 服务器开始，时间戳服务器的工作内容为接收需要时间戳化的区块项的哈希值然后将其广泛传播出去，例如新闻和 usenet post。显然，时间戳可以证明那时该数据确实存在过，因为哈希值是通过时间戳计算出来的。每个时间戳都会将之前的时间戳包含在自己的哈希值中，这会形成一个链式结构，此外，每一个新增的时间戳都会强化前面的时间戳。

工作量证明 为了能够以点对点为基础实现一个分布式的时间戳服务，我们需要使用类似 Adam Back 的 Hashcash 的工作量证明系统，而不是报纸或者 Usenet post。工作量证明涉及到使用诸如 HASH-256 方法生成哈希值，使得目标的哈希值能够以若干个零开始。平均期望的工作量与哈希值



工作量证明

开头的零的个数成指数关系，而这个工作可以通过执行单个哈希进行验证。

对于我们的时间戳网络，可以通过增加区块中的随机数 (nonce) 来实现工作量证明，直到我们找一个符合预期的哈希值。一旦我们的 CPU 消耗的算力满足了工作量证明，在不重做相同工作的情况下是无法改变这个区块的。此外，因为接下来的区块都要从当前的区块链下去，更改当前的区块就意味着修改其之后所有的区块。

工作量证明同样解决了多数决策的表述问题。如果大多数节点采用一个 IP 一个选票的方式，那么可以分配大量 IP 的节点就可能颠覆共识。工作量证明本质上是一个 *CPU*[5] 一票。大多数的决策就可以用最长区块链表示，因其意味着最大的工作量证明。如果大多数的 CPU 算力由诚实节点掌控，那么诚实的区块链会增长最快并且超过竞争者。为了能够修改一个已经存在的区块，攻击者必须重做该区块及以后所有区块的工作量证明，此外还要追上并且超过诚实的区块链。之后我们会证明，一个较慢的攻击者能够追上的概率会随着后续区块的增加以指数级的形式递减。

为了弥补硬件性能的增长以及人们对于运行节点的兴趣变化，工作量证明的难度变化由一个移动平均指标——-平均每小时产生的区块个数决定。如果区块产生的速度过快，那么难度会相应增加。

网络　运行网络的步骤如下：

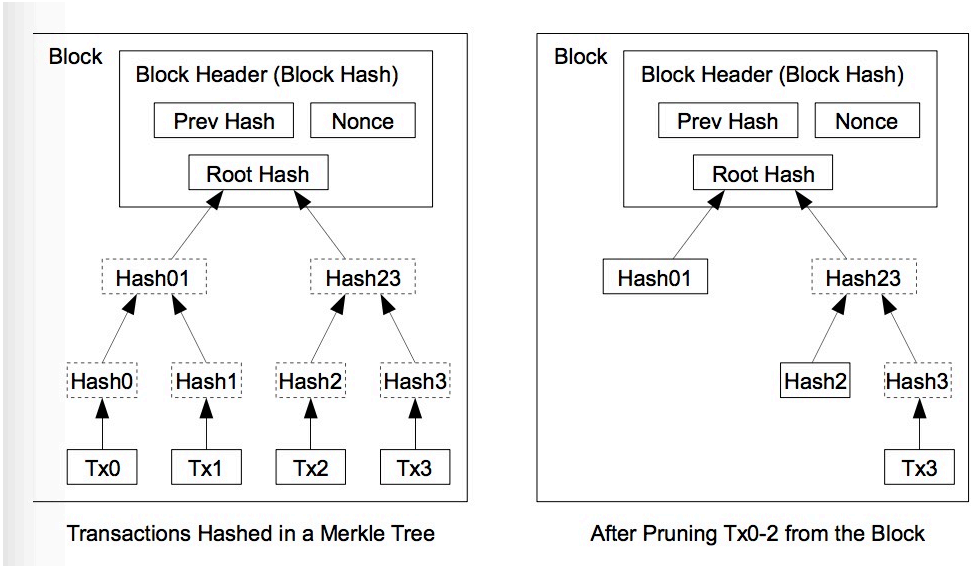
1. 新交易会被广播到所有节点
2. 每个节点都收集交易到一个区块
3. 每个节点都为自己的区块开始寻找工作量证明的困难工作
4. 当一个节点发现了工作量证明，其会将区块广播给所有的节点
5. 只有当区块内的交易都是有效的并且没有被消费掉时，节点才会接受它
6. 节点基于接受的区块继续挖矿，并使用已接受区块的哈希作为之前的哈希值

节点总是认为最长的区块链是正确的，并且不断地延长它。如果有两个节点同时广播他们发现的不同版本的下一个区块，不同的节点可能首先接受他们其中一个，然后基于接受的区块继续挖矿；同时，节点保留另一个链，以备另一个链更早地发现下一个区块。当工作量证明被发现而导致其中一个链更长的时候，平衡就被打破了；在其他链上挖矿的节点转向最长的链继续工作。

新交易的广播并不需要传播到所有节点。只要到达了足够多的节点，他们就会很快进入到区块中。区块的广播也容忍消息丢失。如果一个节点没有收到区块，那么当它收到下一个区块的时候就会意识到自己缺失了一个区块。

激励　根据约定，区块中的第一笔交易是一个特殊的交易，该交易会产生新币并且赋给创造该区块的节点。这增加了节点对于网络的支持，并且提供了新发货币到流通市场的方法，因为比特币网络中没有一个中央权威机构发行货币。恒定数量的货币平稳增加，就好像淘金者花费资源将金子带入流通领域。在我们的案例中，消耗的是 CPU 时间和电力。

激励同样可以按照交易小费的方式给予。如果交易的输出值小于输入值，那么差值就作为这个区块内交易的激励值以小费的形式发给中介。一旦预置数量的货币全部发行完毕，激励就全部转化为交易小费。而且由于货币总量不再变化，通货膨胀的问题也不存在。



剪枝

激励也有助于节点行为诚实，如果一个贪婪的攻击者能够拥有比所有诚实节点更强的 CPU 算力，他就有了两种选择，第一种是通过欺骗他人偷回自己已支付的比特币，第二种是挖矿生成新币。攻击者会发现，如果按照游戏规则运行，他的收益更大。这些规则让他能够比其他所有节点都能够产生更多的新币，而不是破坏系统让自己的财富贬值。

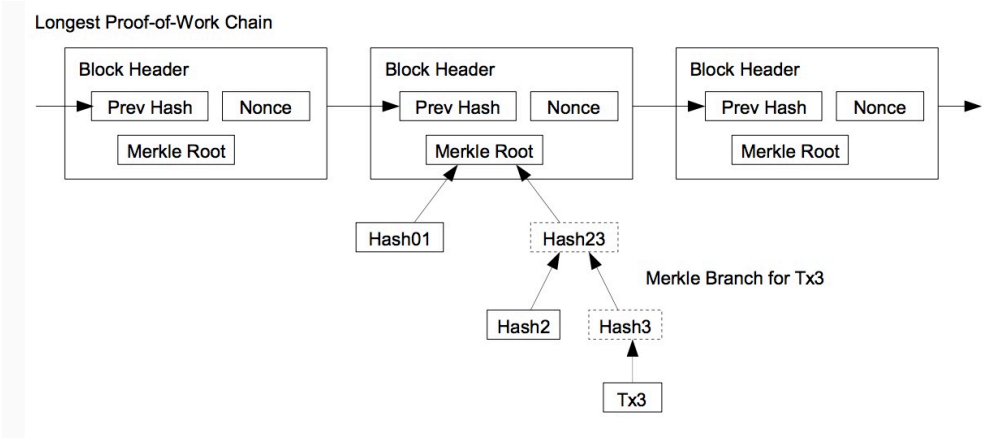
回收磁盘空间 为了节省磁盘空间，一旦货币的最近交易后面已经产生足够多的区块，我们可以丢弃该交易之前的那些已消费的交易。为了能够促进这个过程并且不破坏区块的哈希结构，交易以哈希值的方式存储在梅克尔树中，只有树根节点的哈希值存储在区块头中 (生成区块哈希值)。那么就可以通过对梅克尔树剪枝 [6] 的方法压缩旧的区块，内部的哈希值不需要被存储。

一个没有任何交易的区块头部大小是 80 字节左右，假设每 10 分钟产生一个 block，那么一年产生的容量为 `80bytes*6*24*365=4.0MB`。根据在 2008 年 2GB 内存的典型价格，以及每年 1.2GB 增速的摩尔定律，即使将所有的区块头存储在内存中，存储也不会成为瓶颈。

简单交易验证 (Simplified Payment Verification，简称 SPV) 在比特币网络中，即使不运行一个完整的网络节点也是可以去校验支付的。用户只需要保存一份最长链的区块头副本，然后向其他网络节点发送请求以确定自己持有的是最长的区块链，并且获取需要验证区块中的交易所关联的梅克尔树。一个 SPV 本身是无法校验交易的合法性的，但是通过将交易链到某个区块上，他知道至少有一个节点已经接受了这个交易，而在这个交易之后增加的区块能够让他更加确信网络已经接受了这笔交易。

如上所述，只要诚实节点控制着网络，校验就被认为是可信的。但是如果网络被攻击者控制的话，校验就变成很脆弱不可信了。尽管完整的网络节点可以独自校验交易，只要攻击者控制着整个网络，SPV 就可能被攻击者所编造的交易欺骗。一种解决策略是，当完整网络节点检测到无效交易的时候，SPV 可接收到来自他们的告警，并提示用户的软件下载完整的区块和有告警的交易，去确认不一致的行为。经常收到付款的商业组织仍然想要运行自己的节点来确保更加独立的安全性和快速验证。

合并与切分货币 尽管可以单独地处理货币交易，但是将每笔很小的交易都单独处理是不明智的。为了能够允许货币可以切分和合并，交易就可以包含多个输入和输出。一般来说，对于输入，要么



简单节点验证

是来自之前比较大交易，要么是来自多个小额货币的组合。相反，一般至多有两个输出，一个是收款方，一个是退回给付款人的找零。

注意到扇出 [7] 的情况，一个交易依赖于多个交易，而多个交易会依赖更多的交易。这并不是个问题，因为完全没有必要去获取一个交易完整的历史消息。

隐私 传统的银行模型通过访问权限的控制实现了一定的隐私保护，相应的信息只能由参与者和可信的第三方知晓。需要公开发布所有交易消息的固有特性使得比特币网络不会采用这种方法。但是我们可以通过打破信息流的方式在其他地方保证用户隐私：即确保公钥的匿名性。公众可以看到一个人向另外一个人支付了一定数额的比特币，但是没有信息能将这笔交易关联到任何人。这就像股票交易所里披露出来的交易信息，只有个体交易的时间和数额是公开的，交易的双方是匿名的。

作为额外的防御措施，每次交易都应该使用新的密钥对来防止这些交易被关联到同一个人。对于多输入的交易，一些关联是无法避免的，因为这些输入会泄露出它们属于同一个拥有者。如果某个密钥的拥有者信息发生泄露，关联关系可能存在泄露这个人的其他交易的风险。

计算 现在，我们来考虑攻击者试图生成一个比诚实链增长速度更快的链。即使这个攻击成功了，也不至于让系统变得非常脆弱，例如凭空造币或者取回不属于攻击者的比特币。节点不会接受一个无效的交易作为支付，而诚实节点也不会接受包含无效交易的区块。一个攻击者只能尝试改变他自己的交易并取回自己花费的比特币。

诚实链和攻击链之间的竞争可以描述为二项随机游走，成功事件是诚实链增加一个区块，领先程度 +1；失败事件是供给链增加一个区块，差距缩减-1。攻击者从一个指定的差距追上的概率可以类比于赌徒破产问题 [4]。假设拥有无限信用的赌徒开始是亏空的，可以玩了无数把试图达到收支平衡，我们可以计算他达到收支平衡的概率，或者说是攻击链追上诚实链的概率，如下所示：

1. p = 诚实节点找到下一个区块的概率
2. q = 攻击者找到下个区块的概率
3. q_z = 攻击者从落后 z 个区块的位置追上来的概率

那么可以得出

$$q_z = \begin{cases} 1 & \text{if } q \geq p \\ (\frac{q}{p})^z & \text{if } q < p \end{cases}$$

假设 $p > q$，那么攻击者能够追上的概率会随着区块的增加以指数级的速度递减。以攻击者的赔率，如果他最开始没能足够幸运地挖出很多区块，那么随着他越来越落后，追上的概率极为渺茫。

现在我们来考虑一个交易的接收者经过多久可以认为发送方无法修改这笔交易记录了。我们假设付款人是攻击者，他想让收款人认为他已经付钱给他一段时间了，然后过段时间再把这笔钱支付给自己。当第二笔交易发生的时候收款人会收到告警，攻击者希望此刻想要追回这笔钱为时已晚。

收款人可以在签名不久前才生成密钥对，并把公钥发给攻击者。这可以阻止攻击者事先准备好一些区块，并基于此不停地工作直到他足够幸运地领先很多，然后再执行把钱发送给自己的交易。一旦交易发送，不诚实的节点就开始在一个包含他自己版本的交易的并行链上秘密地工作。

接收者可以等到交易加入到区块中，并且有 z 个区块链到后面之后再确认这笔交易。接收者并不知道攻击者的进展如何，但是假定诚实节点产生的区块按照期望速度产出，那么攻击者的进展会是一个泊松分布，期望值为：

$\lambda = z\frac{q}{p}$[8]

为了能够计算出来攻击者能追上的概率，我们将每个进度值下追上来的概率乘以对应的泊松密度函数并且累加起来：

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \begin{cases} 1 & \text{if } k > z \\ (\frac{q}{p})^{z-k} & \text{if } k \leq z \end{cases}$$

重新组织下累加避免出现无限项相加的情况：

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!}(1 - (\frac{q}{p})^{z-k})$$

转化成 C 代码：

```
#include<math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for(k=0; k<=z; k++)
    {
        double poisson = exp(-lambda);
        for(i=1; i<=k; i++)
        {
            poisson *= lambda / i;
        }
        sum -= poisson * (1 - pow(q/p, z-k));
    }
    return sum;
}
```

总结　我们提出了一种不依赖于信任模型的电子现金交易系统。最开始我们讨论了基于数字签名的电子货币框架，虽然能对所有权提供强有力的保护，但是其无法阻止**双重支付**。为了能够解决**双重支付**的问题，我们提出了一种点对点的网络，其基于工作量证明，并公开所有的交易历史。如果诚实节点控制着主要的 CPU 算力的话，攻击者在计算上去篡改交易历史是不可行的。比特币网络因

其非结构化非常鲁棒。所有节点可以同时工作而几乎不需要协调。节点间不需要表明自己的身份，因为消息并不是发送给特定的节点而只需要按照尽力而为的策略传播即可。节点可以自由加入或者离开网络，接受在他们离开期间的区块链作为历史事件的证明。所有节点基于计算力投票，如果认为区块有效那么就在此基础上继续挖矿，否则就拒绝接受。任何需要的规则和激励都可以通过这个共识机制增强。

注释

1. 双重支付：在比特币网络中存在三种类型的双重支付攻击：1. 连续而快速地释放两个相互冲突的交易，这种被叫做竞争攻击 (race attack)，解决办法是等待其中一个交易首先确认；2. 第二种是在交易进入区块前，将同样的比特币再花出去，这种被称作 Finney 攻击，解决办法是等待整个区块链 6 个确认；3. 51% 攻击，即某个攻击者拥有了大部分算力，理论上可以通过重做工作量证明将已经花费出去的币取回来 (reverse transaction)，这个攻击是无法解决的，但是它面临的问题是，该攻击会导致整个比特币网络的削弱，失去信用，从而导致比特币贬值，所以其在现实中是一般不会发生的。参考：https://bitcoin.stackexchange.com/questions/4974/what-is-a-double-spend

2. 这个资源池代表了整个比特币全网的计算能力，根据 2017.07 的统计，全网算力目前已经达到 10^{18} 级别的数量级。参考：http://www.8btc.com/bitcoin-hashrate-six-exahash-july-1

3. 弱点是什么？无法避免诈骗？需要一个第三方？个人隐私泄露？

4. 时间戳：如果时间戳大于之前 11 个区块的时间戳中位数并且小于网络调整时间 +2 小时，那么其被认为是有效的。参考：https://en.bitcoin.it/wiki/Block_timestamp, https://bitcoin.stackexchange.com/questions/48711/where-do-the-timestamps-on-blocks-come-from

5. 更加精确的表述应该是一个单位算力一票，至于单位算力的的表述可以同现实世界的公民投票权类比

6. 剪枝算法：剪枝的条件，首先叶子节点的所有输出需要已经消费才可以被剪除；其次如果一个节点的左右孩子都被剪除了，它也可以被剪除。参考：https://bitcoin.stackexchange.com/questions/36100/pruning-the-branches-in-merkle-tree

7. 扇出：在比特币中的意思是一个交易中的输入包含之前多个交易的输出，而这些交易可能会依赖更多的交易，以此类推，像打开的扇子那样。参考：https://en.wikipedia.org/wiki/Fan-out_(software)

8. 因为是等待 z 个 block 生成，诚实节点生成 z 个 block 需要的时间是 $T_z = \frac{z}{p}$，那么攻击者产生区块数的期望就是 $Ex = qT_z = z\frac{p}{q}$

名词解释

1. best-effort delivery: https://en.wikipedia.org/wiki/Best-effort_delivery
2. best-effort basis: https://financial-dictionary.thefreedictionary.com/best-efforts+basis
3. 随机游走问题: https://en.wikipedia.org/wiki/Random_walk
4. 赌徒破产问题: http://www.columbia.edu/~ks20/FE-Notes/4700-07-Notes-GR.pdf

参考

1. W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.
2. H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In 20th Symposium on Information Theory in the Benelux, May 1999.
3. S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In Journal of Cryptology, vol 3, no 2, pages 99-111, 1991.
4. D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In Sequences II: Methods in Communication, Security and Computer Science, pages 329-334, 1993.
5. S. Haber, W.S. Stornetta, "Secure names for bit-strings," In Proceedings of the 4th ACM Conference on Computer and Communications Security, pages 28-35, April 1997.
6. A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.
7. R.C. Merkle, "Protocols for public key cryptosystems," In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122-133, April 1980.
8. W. Feller, "An introduction to probability theory and its applications," 1957.

In [14]:
```
import math

def attacker_success_probability(q, z):
    p = 1.0 - q
    lamb = z * (q / p)
    sum = 1.0
    i, k = 0, 0
    for k in range(z+1):
        poisson = math.exp(-lamb)
        for i in range(1, k+1):
            poisson = poisson * (lamb / i)
        sum = sum - poisson * (1 - math.pow(q/p, z-k))
    return sum

def calc_probability_threshold(P, q):
    r = 1.0
    z = 0
    while r > P:
        r = attacker_success_probability(q, z)
        z = z + 1
    return z - 1

print("q = 0.1")
q = 0.1
for i in range(11):
```

```
            print('z=%-8d P=%.7f' % (i, attacker_success_probability(q, i)))
        print()
        print("q = 0.3")
        q = 0.3
        for i in range(0, 51, 5):
            print('z=%-8d P=%.7f' % (i, attacker_success_probability(q, i)))

        print()
        print("Solving for P less than 0.1%...")
        for i in range(10, 46, 5):
            q = i / 100
            print('q=%-8d z=%-8d' % (i, calc_probability_threshold(0.001, q)))
```

```
q = 0.1
z=0        P=1.0000000
z=1        P=0.2045873
z=2        P=0.0509779
z=3        P=0.0131722
z=4        P=0.0034552
z=5        P=0.0009137
z=6        P=0.0002428
z=7        P=0.0000647
z=8        P=0.0000173
z=9        P=0.0000046
z=10       P=0.0000012

q = 0.3
z=0        P=1.0000000
z=5        P=0.1773523
z=10       P=0.0416605
z=15       P=0.0101008
z=20       P=0.0024804
z=25       P=0.0006132
z=30       P=0.0001522
z=35       P=0.0000379
z=40       P=0.0000095
z=45       P=0.0000024
z=50       P=0.0000006

Solving for P less than 0.1%...
q=10       z=5
q=15       z=8
q=20       z=11
q=25       z=15
q=30       z=24
q=35       z=41
q=40       z=89
q=45       z=340
```