# 导论

1. 除非拥有专业的团队进行研究和设计，否则不要自己设计加密算法
2. 不要使用未经证明的加密算法，包括对称密钥算法、非对称密钥算法、hash函数、密码协议（密码协议我们倒是经常自己设计和修订）
3. 大的密钥空间只是系统安全的必要非充分条件
4. 蛮力攻击的大概门槛

***使用蛮力攻击成功破解不同长度密钥的对称算法预计需要的时间***

|  |  |
| --- | --- |
| 密钥长度 | 安全评估性 |
| 56-64位 | 短期：只需要几个小时或几天破解 |
| 112-128位 | 长期：在量子计算机出现前，需要几十年破解 |
| 256位 | 长期：即使使用量子计算机，也需要几十年 |

需要强调的是：计算机能力在不断增强，按照摩尔定律，每15年计算机能力会增长1000倍，或者说每18个月，破解的成本降低一半。

2013年，我的pc机器每秒钟可完成100万次tea解密运算。假设我投入1000台同时对128位密钥暴力破解，也需要一个可以认为无限大的时间才能遍历完。

而如果是破解56位密钥的DES算法，假设我的计算机也能每秒完成100万次DES解密，1000台同时暴力破解，那只需要2.5年即可完成所有密钥空间的遍历。

可见密钥的位数非常关键。

1. 增加对称算法安全性的通常的方法是使用两次或者多次加密，每次加密使用不同的密钥，3DES和DES就是这样的关系，相当于扩大了密钥空间。 但是多次加密的方式并不一定总是有效的
2. 攻击类型：
   1. 选择明文攻击：攻击者可以任意指定（或者有相当的自由度）的[明文](http://zh.wikipedia.org/wiki/%E6%98%8E%E6%96%87)，并得到相应的[密文](http://zh.wikipedia.org/wiki/%E5%AF%86%E6%96%87)。
   2. 已知明文攻击，攻击者有一组明文和对应的密文，但他不能任意指定明文获得明文-密文
   3. 选择密文攻击，攻击者可以随意指定（或者有相当的自由度）密文，并获得对应的明文
   4. 唯密文攻击，攻击者仅知密文的情况下进行攻击。

# 序列密钥算法

1. 序列密钥算法，也叫流加密算法，使用不怎么广泛，比较成熟的算法有RC4和SEAL算法。这类算法的优点是加解密速度快。
2. 密码学中，随机性具有非常重要的作用，例如随机产生的密钥或者其他参数，必须足够的随机。C语言库中的rand系列函数是伪随机(PRNG)，一般不宜在密码学中使用。Linux环境下（IOS/android也是）可使用/dev/random设备读取真随机数(TRNG)。Windows下可用CryptGenRandom函数，这是一个密码学安全的伪随机数发生器(CSPRNG)

# 对称密钥算法

1. DES不再安全，因为密钥空间只有56位，容易被蛮力破解。实际应用一般用3DES，即不同的密钥加密三次，扩大了密钥空间，目前已知的实际攻击都不能破解3DES
2. 对称密钥算法的两个基本操作是
   1. 混淆：使得密钥和密文之间的关系尽可能的模糊
   2. 扩散：为了隐藏明文的统计属性而将一个明文符号的影响扩散到多个密文符号
3. 现代分组密码都具有很好的扩散性，修改明文的1位会导致平均1半的密文的位发生改变。
4. 除了DES外，还有很多分组密钥算法，例如AES、TEA。AES密钥长度有128位、192位和256位。
5. TEA密钥长度为128位。TEA比DES要简单，加解密速度也比DES快很多，它抗差分分析能力强，密钥空间大，安全性好

# 分组密钥算法的操作模式

1. 实际应用中，通常需要加密的明文不是单个的8字节或者16字节长的分组，而是任意长度的文件或者报文。如何对这些长的明文进行加密，可以有多种模式
2. 常用的模式有ECB（电子密码本模式）、CBC（密码分组链接模式）、CFB（密码反馈模式）、OFB（输出反馈模式）、CTR（计数器模式）等。
3. 各种模式下的加密方法，可以是AES或DES或3DES或TEA或其他的分组对称密钥算法
4. 通常，要求明文长度必须是分组的整数倍。可用的一种填充方法是：在明文后面附加单个1和多个0，直到明文刚好为整数倍的分组大小。如果明文刚好是分组大小的整数倍，那就用该方法再填充额外的一个分组
5. ECB模式下，每个分组都是单独加密的。即：

加密：

解密：

其中 表示一个分组的明文和密文，不是指一个位

1. ECB模式下，优点是
   1. 可以并发的同时对多个分组进行加密和解密
   2. 不用等到完全传输完所有密文，就可以开始一个分组一个分组的解密
   3. 当由于噪音导致位错误时，仅仅影响对应的分组，对其他分组没有影响
2. ECB模式的缺点是
   1. 只要密钥不变，相同的明文总是产生相同的密文。明文的统计信息在密文中被保留。例如一张位图，经过ECB加密，密文还是隐隐约约的能看出图片的内容
   2. 攻击者如果替换或者重排密文分组，可能得到有效的明文，造成攻击。例如把转账报文里的关于目标账户的分组替换为历史上截获的恶意者自己的账户密文，实现给自己账户转账的目的
3. CBC模式主要是两个思想：
   1. 所以分组的加密都链接起来，使得密文 不只是依赖于， 还依赖于前面的所有明文分组，解决明文统计信息被保留的问题
   2. 加密过程使用初始向量进行随机化，解决相同明文加密得到相同密文的问题
4. CBC是这样的：

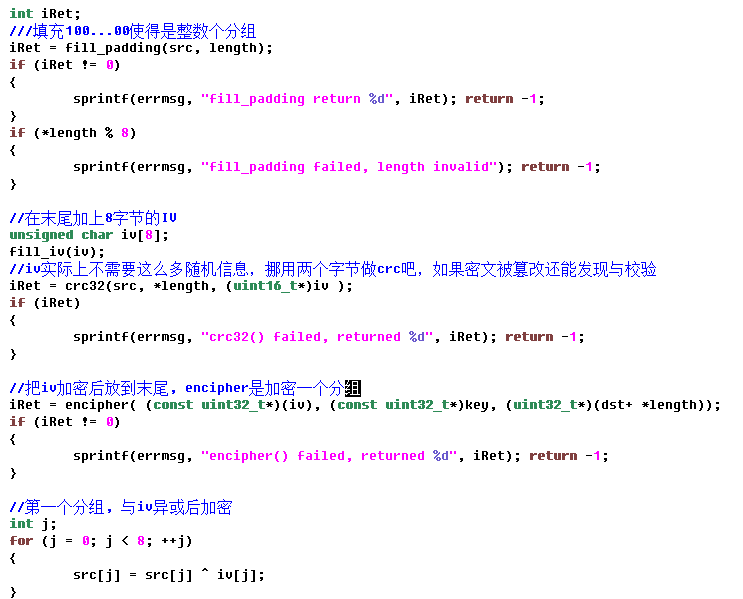
加密（第一个分组）：

加密（一般分组）：

解密（第一个分组）：

解密（一般分组）：

1. 实际应用中，IV一般是每次随机产生一个分组，附加到明文头部或者尾部。不需要通过额外的交互来约定和传递IV的值，见如下的代码片段，详细可以见tea.c文件



# 基于大整数分解的公钥算法

## 公钥算法简介

1. 对称密钥算法面临的主要问题有
   1. 密钥派发的问题：在建立安全的信道前，如何约定密钥
   2. 密钥个数增长的问题：当会话参与方增加时，密钥个数成平方速度增加
   3. 对欺骗没有防御：alice可以伪造bob的消息，bob也可以抵赖曾经发送过的消息
2. 公钥算法的基本原理是单向函数。其中一种单向函数是整数分解的问题。即给定两个大素数p 、q，计算他们的乘积n很容易；但是如果在获悉n的情况下，想要分解出p、q就很难。（***RSA算法中，n是公钥的一部分，是公开的，而p、q是保密的，否则根据p、q就可以很容易推导出私钥d***）。实际上，如果两个素数对应的十进制数字都超过150位，那即使使用上千台pc运行多年也不可能将乘积分解出这两个素数。
3. 公钥算法的缺点主要是对数据进行加密的计算量太大，相比对称密钥算法，要慢2-3个数量级。
4. 公钥的可靠性，一般通过证书机制来解决，后面介绍
5. 公钥算法，密钥的长度与安全级别不是直观的对应关系，如下表：

***不同安全级别所要求的公钥算法的密钥长度***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算法家族 | 密码算法 | 安全级别（位） | | | |
| 80 | 128 | 192 | 256 |
| 整数分解 | RSA | 1024位 | 3072位 | 7680位 | 15360位 |
| 离散对数 | DH DSA Elg | 1024位 | 3072位 | 7680位 | 15360位 |
| 椭圆曲线 | ECDH ECDSA | 160位 | 256位 | 384位 | 512位 |
| 对称密钥 | AES 3DES TEA | 80位 | 128位 | 192位 | 256位 |

## 整数分解公钥算法的数论基础

1. 欧几里德算法和扩展的欧几里德算法：用于计算最大公约数和逆元
2. 欧拉函数：（即集合{0,1,2…m}）内与m互素的整数的个数表示为

如果m有因式分解

那么，)

1. 费马小定理：假设a是一个整数，p为一个素数，且a<p，那么有
2. 根据费马小定理，a的模p的逆元就是，因为
3. 欧拉定理：假设a、m都是整数，且gcd(a,m)=1，则有：
4. 费马小定理是欧拉定理的特例，即当p为一个素数时， ，代入欧拉定理即可得

## RSA算法

1. RSA的安全前提是：大数因式分解是困难的，且已知指数和幂，求根是困难的。后者称为RSA问题，可见<http://en.wikipedia.org/wiki/RSA_problem。目前最好的解决RSA>问题的方法是大数分解，求出私钥，解密获得明文（幂）。
2. RSA问题，我没能理解，觉得不是很难。我用GNU MP大数库，对一个512位的整数做了65537次方，然后又开65537次方，能够很快得到数据。
3. RSA加密和解密都是在整数环内完成的。假设使用RSA加密明文x，将x看做一个位数很多的整数，其整数值在集合内，所以x表示的二进制比小于n
4. RSA加密：给定公钥（n, e）=和明文x，加密函数为：

，其中x y属于

1. RSA解密：给定私钥d=及密文y，解密函数为：



1. 加解密过程中的n e d都是很大的整数，通常为1024位或者更长
2. RSA密钥生成过程，就是输出公钥（n, e）=和私钥d=的过程：
   1. 选择两个大素数（一般为512位以上）p、q （要随机、保密，具体如何找大素数，后面讲）
   2. 计算（真的就是大数计算，还是比较简单）
   3. 计算 (见欧拉函数)
   4. 选择满足以下条件的公开指数



（用欧几里德算法可以计算最大公约数）

* 1. 计算满足以下条件的私钥 d



（用扩展的欧几里德算法求e的逆元）

1. e的计算过程，从教科书看，是从3开始反复尝试和验证的过程。通常e可以比较小，65537是常用的值，d要足够的大，p和q不能太接近且足够的随机。
2. RSA密钥及其相关参数， p、q、d、都需要保密，n、e可以公开。因为大数相乘很容易，但将n分解为p、q非常难，所以d也推导不出来
3. RSA加解密需要进行指数运算。指数计算最直观的方法，就是反复乘。有个方法可以加快：混合乘和平方，例如计算，直观的方法是乘7次x。加快的方法是



算法上：将指数表示为二进制方式，然后从左到右依次扫描指数的二进制位，在每轮迭代中，计算当前结果的平方。当且仅当扫描到1时，将得到的结果乘x。为了保持中间结果始终最小，每次操作后都对n取模。

1. 如何寻找大素数：最直观的办法是随机生成大的奇数，并检查他们的素性，不满足尝试下一个整数。这里涉及两个问题：
   1. 要尝试多少次才能找到一个素数？
   2. 如何快速检测一个数是否为素数，即素性测试
2. 先回答上面的第一个问题：根据素数的普遍性（即一个奇数p为素数的概率为），对于512位的p、q，每测试177个随机数就可以找到一个素数。蛮快的。
3. 回答上面的第二个问题：用费马小定理测试，假设测试p是否为一个素数：
   1. 随机选择
   2. 如果，则p为合数，否则p**可能**为素数，重新步骤a），多次如果结果都说p可能为素数，那我们就信p是素数（太搞了）。
4. 前面介绍的都是教科书式的基础理论，在实际应用中不够安全。实际应用，需要加上填充。填充方案非常重要，实现不好，RSA实现就不安全。它在加密明文前将一个随机结构嵌入到明文中。有点类似对称密钥算法中的CBC等模式。
5. 1024位的RSA已经不安全了。2010年，实现了对768位的大数因式分解。也许几年后1024位就被分解了。详细见<http://en.wikipedia.org/wiki/RSA_Factoring_Challenge>

# 基于离散对数的公钥算法

## 数论基础：有限循环群

1. 群（不是QQ群）是指一个集合以及对集合中两个元素的操作。具有以下属性
   1. 操作的封闭性：即所有的, 有
   2. 操作的结合性
   3. 存在中性元或单位元，满足对所有的 有
   4. 对集合中每个元素存在逆元，满足
2. 如果群额外满足对所有，有，则该群称为阿贝尔群或可交换群
3. 对于正整数n，所有满足且的整数形成的集合，在此集合上应用乘法模操作形成一个阿贝尔群，且单位元，该群记作
4. 特别的，当n为一个素数，则。离散对数密码学中主要使用到该群
5. 有限群：如果一个群拥有有限个元素，则叫做有限群。集合中元素的个数叫做群的基或者阶，记作
6. 群的基（阶）等于
7. 元素的阶：群内元素的阶记作，指的是满足以下条件的最小正整数：
8. 循环群：如果群内存在一个元素，满足，那这个群是循环群，该元素是该群的本原元或者叫生成元。G中的每个元素都可以这个元素的幂值，产生了整个群
9. 对每一个素数p，
10. 定理：假设是一个有限群，则对每个 都有
    1. 可以整除
11. 定理：假设是一个有限循环群，则有
    1. G中的本原元个数为
    2. 如果是素数，则G中所有不等于单位元的元素都是本原元。所有元素的阶，要么是1，要么是
12. 子群：假设有群，集合且也是一个群，那么 是的子群
13. 循环子群定理：假设是一个循环群，则G里的每个元素，都是拥有s个元素的循环子群的本原元。其中。即循环群G内每个元素都可生成一个循环子群H
14. 拉格朗日定理：假设 是的子群，则可整除
15. 假设G为一个基为n的有限循环群，为对应的生成元，则对于整除n的每个整数k，G都存在一个唯一的基为k的循环子群H。子群H是由生成的。且G不存在其他子群。

## DH密钥交换算法

誊抄完前面这些数学定义、定理，我已经要吐了。数学家真不是人当的。还好，我辈只需要了解基本原理。其实基本原理非常简单，是这样一个单向函数：

对于素数p和有限循环群里的本原元，随机一个整数，计算很容易，但是在知道情况下要反推x，非常困难。

DH密钥交换算法，用于解决密钥分配问题，即它允许通信双方通过不安全的信道进行交流，得到一个共同密钥。算法如下：

**Step1：**选择一个大素数p，和一个群

**Step2：**公开

**Step3：**通信的一方alice随机选择私钥

通信的另一方bob随机选择私钥

**Step4：**alice计算公钥，

bob计算公钥。两人交换信息A，B

**Step5：**alice计算得到共同密钥，

bob计算得到共同密钥

上面的希腊字母和英文字母注意区分开

为了安全，p应该大于1024位，选择私钥也需要用真随机数生成器。

DH密钥交换算法，对于参与方的真实性不能进行鉴别，所以存在中间人攻击的风险。DH密钥交换算法的一些变种避免了这样的攻击，例如Station To Station(STS)协议。

最后计算得到的共同密钥与p拥有相同的位长度，如果想把他用作AES等对称密钥，可以只选择其中的128位，或者md5后取其摘要。

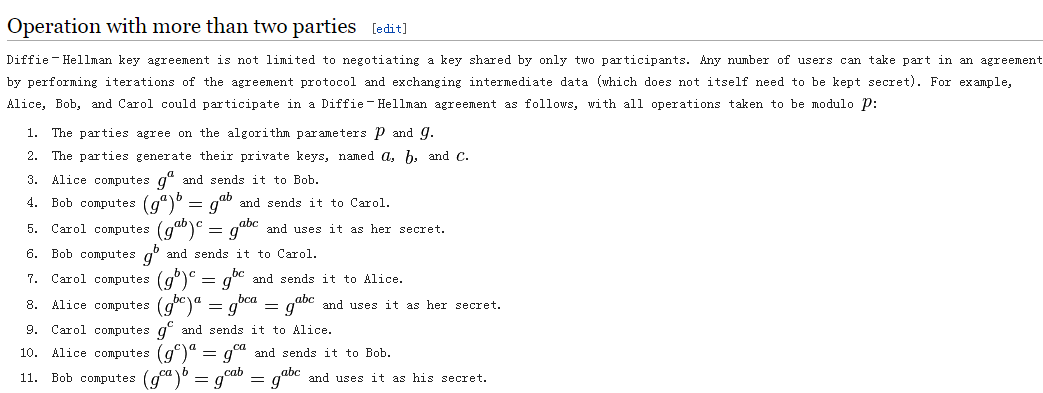
算法中，选择一个大素数，根据前面的费马小定理和素数的普遍性定理，可以容易获得；计算指数也可以用平方-乘方法快速计算。**唯独如何求本原元我始终不得其解**

**DH密钥交换算法也可以像RSA那样作为公钥体系用。Alice的公钥就是 。Bob为了向Alice发送消息，先选择随机数，然后一起发送和用加密的业务消息。只有alice才能解开这个加密的额消息。**

预先双方分享好的公钥能够避免中间人攻击，例如通常直接在客户端hardcode服务器的公钥，而不是每次下载，避免恶意者冒充server或者链路劫持的方式给客户端一个不合适的公钥。

DH密钥交换算法，不能用于生成数字签名-校验的场景，这种场景下要求“私钥加密、公钥解密”。可考虑用DSA

DH可以用于多方协商密钥：



## Elegamal加密算法

类似DH密钥交换算法， Elegamal算法也基于离散对数。他直接将加解密步骤揉合进来了。算法如下：

1. 握手阶段：
   1. Bob随机选择私钥，计算
   2. Bob把发送给alice
2. 加密阶段：
   1. Alice随机选择，
   2. Alice计算临时密钥, 计算掩码密钥
   3. Alice加密消息
   4. Alice将发送给Bob
3. 解密阶段：
   1. Bob计算掩码密钥
   2. Bob 计算逆元 （因为，可根据扩展的欧几里德算法计算逆元）
   3. Bob解密消息
4. 上面的加密和解密步骤循环进行。

假设发生多次消息传递，因为每次加密中使用的i是随机选择的，所以虽然使用了相同的公钥，即使明文相同，也会得到不同的密文。Elgamal是一种概率加密方案

# 基于椭圆曲线的公钥算法

## 数论基础

这个鬼东西比较费解，还是循环群的那些概念，大概是这样理解：

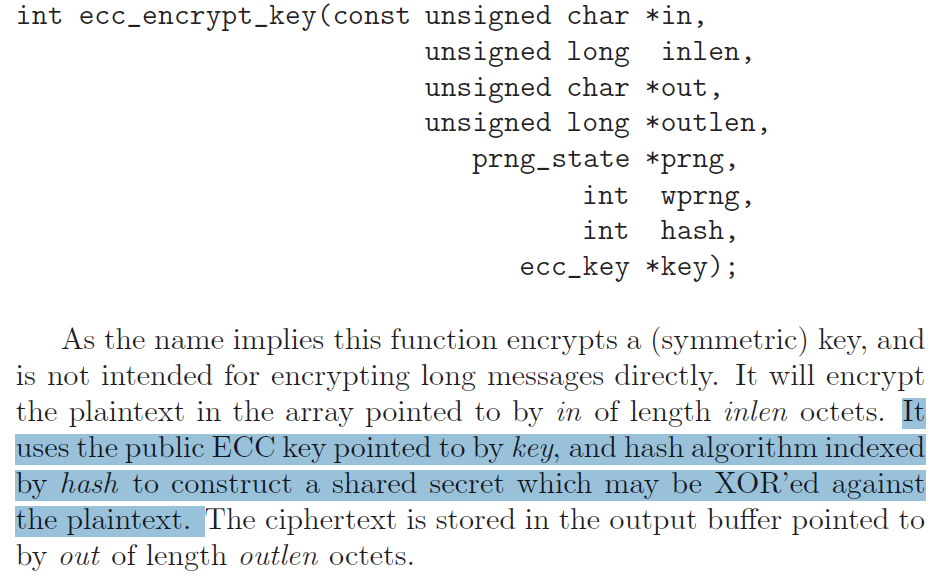
1. 有这样一条椭圆曲线：，其中:
   1. p是大素数，
   2. 且满足
2. 在椭圆曲线上定义曲线上两个点相加：

然后s又是一大堆x1 y1 x2 y2的公式，老子不抄写了。奇怪的是和b没有关系。

经过这样的一些定义，曲线上的有限个点形成了有限群，不再是无穷个点。

## ECDH

1. ECDH是这样的：
   1. 有一条椭圆曲线E和E上的本原点P（注意不是那个素数p），#E表示E上的点的个数
   2. Alice随机产生私钥，并计算(a个P相加)，将A发送给Bob
   3. 同样Bob随机产生私钥,并计算(b个P相加)，将B发送给Bob
   4. Alice计算 (a个B相加)
   5. Bob计算 (b个A相加)
2. ECC使用的单向性难题是：已知(a个P相加)，难以求出
3. 同样可以定义ECDSA
4. ECC使用较短的操作数，就可以获得与RSA或离散对数系统同等的安全级别（大概为160~256位比1024~3072位），在性能和带宽上更有优势。
5. 类似DH密钥交换算法的方式，ECDH也可以像RSA那样作为公钥体系用。**Alice的公钥就是 。Bob为了向Alice发送消息，先选择随机数，然后一起发送和用加密的业务消息。只有alice才能解开这个加密的额消息。**
6. 另外，和RSA一样，ECC是独立的签名验证方案，可以直接用ECC私钥进行签名、公钥验证。详细可以见libtomcrypt库的ecc\_sign\_hash() ecc\_verify\_hash()等函数
7. 好像ECC不能像RSA那样，真正意义上的用公钥加密，私钥解密？（待确认哈） 因为我看libtomcrypt虽然提供了ecc\_encrypt\_key() ecc\_decrypt\_key函数，但是内部实现其实是类似上面提到的ECDH的使用方式，当然，也许这只是一种具体的实现方式而已。





# 数字签名

1. 安全服务的四个核心问题
   1. 保密性
   2. 完整性：消息在传输过程中未被篡改
   3. 消息验证（数据源验证）：消息的发送者是可信的
   4. 不可否认性：消息发送者无法否认其发送了消息
2. 四个核心问题的解决办法
   1. 保密性由对称加密或公钥算法加密解决
   2. 完整性可通过数字签名和消息验证码解决
   3. 消息验证可通过数字签名解决
   4. 不可否认性可通过数字签名解决
3. RSA数字签名协议：
   1. Bob生成密钥对：私钥d和公钥(n, e)
   2. Bob通过一定的方式（例如CA证书）公布公钥(n, e)
   3. Bob对消息x进行摘要h(x)，计算签名
   4. Bob发送消息和签名(x, s)
   5. Alice验证，
   6. 如果
4. 存在性伪造攻击：如果上面不是对h(x)，而是直接对消息x进行签名，那么中间人oscar可以随意搞个s’’，然后用公钥计算出对应的x’’，将(x’’, s’’)发送给alice，alice也会验证通过。解决的办法是
   1. 对h(x)进行签名而不是对x进行签名
   2. 对消息进行格式化。因为oscar的攻击无法控制消息x’’的语义（即x’’是没有什么意义的二进制流）,所以对消息进行填充或者格式化，并验证可发现。 例如要求所有的消息x都是以消息体的crc结尾，或者以0001开头类似这样的。
5. 必须保证公钥是可信的，如果攻击者成功的向验证方提供了一个他自己的公钥，则攻击者可以对消息进行签名而欺骗验证者。可以使用证书防止这类攻击。
6. RSA数字签名的优势是：验证过程计算比较高效（因为e比较小），因为大多数情况下，签名是一次生成多次校验。
7. 另外还有Elgamal数字签名方案、DSA数字签名算法、ECDSA算法
8. 数字签名必须保证私钥是严格保密的。一般可以使用智能卡、USBKEY存储，签名的执行也是cpu在智能卡内完成。

# Hash函数

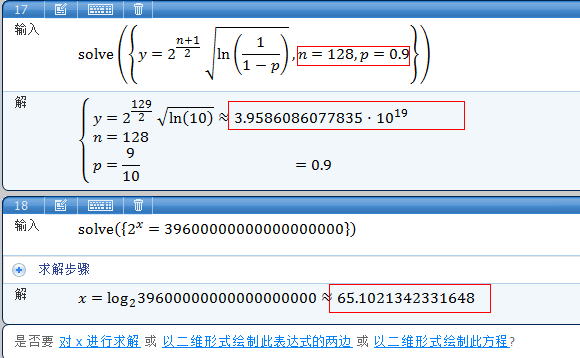
1. Hash函数的要求
   1. 任意消息大小都适用
   2. 输出固定的长度
   3. 计算相对简单
   4. 抗第一原相性（单向性）：给出一个输出z，找到满足h(x)=z的输入x是不可能的
   5. 抗第二原相性：给定x1和h(x1)，找到满足h(x1)=h(x2)的x2在计算上是不可能的
   6. 抗冲突性：找到满足h(x1)=h(x2)的一对在计算上是不可行的
2. 注意上面的e)和f)，都是在计算上不可能。实际情况，根据鸽笼原理或者说抽屉原理，这样的x1 x2肯定是存在的。
3. 上面的e)和f)，看似难度一样，其实不然。破解e)比f)要难得多。 试想一下：一种情况是bison是10月8号生日，要找一个和我同一天生日的人；还有一种情况是，找出两个人，是同一天生日，具体是哪天，无所谓。 前者比后者难很多
4. 针对抗第二原相性的攻击，恶意者可以随机的选择x2，计算h(x2)是否等于h(x1)，这与对称密钥算法中的穷尽密钥攻击类似。考虑到当前的计算机技术， n=80位的输出长度已经足够。针对抗冲突性的攻击，下面说的这种生日攻击就凑效得多，导致输出长度必须进一步加大。
5. 生日攻击，原型是至少多少人聚会，才能使至少两个人在一年中同一天生日（叫做冲突）的概率足够高？

引起至少一个冲突需要的人数，比我们想象的要少很多。只需要23个人，就能使得出现冲突的概率达到50%！

1. 搜索hash函数h()的冲突与生日冲突的问题完全相同。Hash函数中，每个元素对应的值的个数不是365个，而是个，n是h()输出的宽度。例如md5就是128位的。
2. 经过数学变换，对于hash函数冲突，我们有：

其中表示至少存在一个冲突的概率。

1. 以md5为例，n=128，假设我们希望，那么。用microsoft math工具计算的截图如下：



1. 在抗冲突性攻击方面，王小云的算法，相比生日攻击需要的步骤少很多。针对sha1，王小云的算法只需要步。针对md5，王小云的算法只需要
2. 对于md5，个步骤，假设用最直接的计算-存储-比较的方法：
   1. 时间复杂度：单机每秒假设完成100万次md5，那么需要400万秒，即50天
   2. 空间复杂度：假设每个条目为32字节（16B的文档编号，16B的md5），那么需要存储空间128G\*1K，一两千台机器就放到内存里了。
3. 生日攻击的典型场景是这样的：假设一份合同，消息为

x0=’编程12个月，每个月月薪1万，每月结清’，

攻击者另外有一个消息

y0=’编程120个月，每个月月薪10块，10年一结’

攻击者在不改变上面消息语义的情况下，通过对x0和y0插入空格/制表符来改变消息。

假设每个消息都有20个位置可以修改，则x0可以有个变种，记作；同样的y0也有个变种，记作。

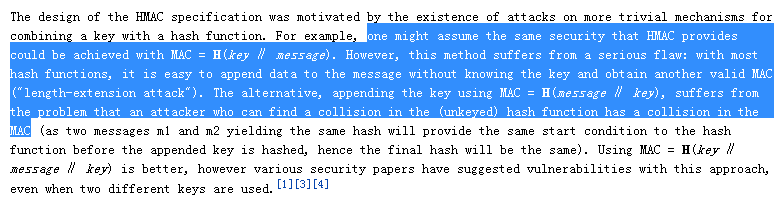
如果攻击者找出一对消息和，满足摘要计算结果相同，那么就可以用诱骗苦逼程序员签署合同

对于输出宽度是40位的一个摘要算法，上面的攻击是轻而易举的。因为对而言，时间复杂度和空间复杂度都很小。

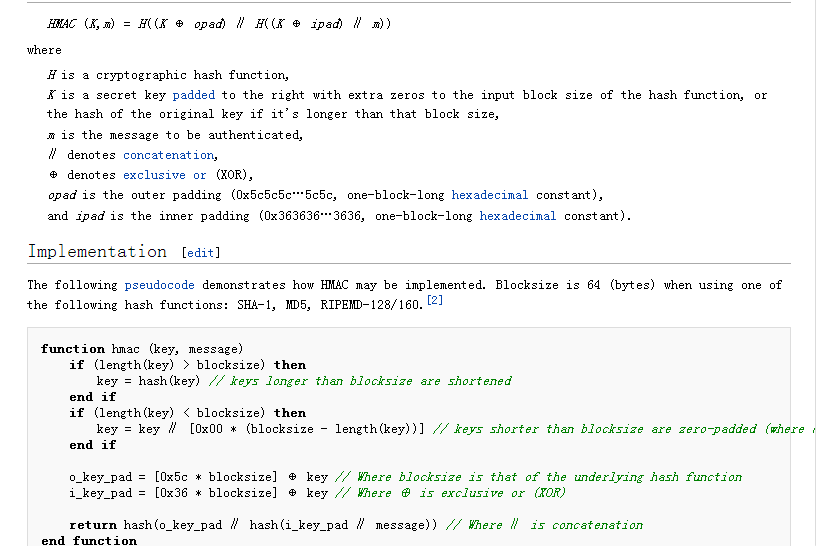
1. 需要强调的是：很多时候，hash函数的应用只需要抗第一原像性，即单向性，例如密码的存储。因此md5这种较短的摘要算法也是安全的。因为在这种情况下，冲突攻击没有造成威胁。单向性的主要威胁是彩虹表和针对弱密码的字典攻击。
2. 如果需要加强抗冲突性攻击，可以选择SHA2
3. 冲突攻击具体怎么实施，bison没有找到更巧妙的算法，似乎空间复杂度非常高哇，需要产生并存储大量的消息-摘要对，再找找资料看
4. 从对称的分组加密算法构建hash函数，基本思路是使用大小为b的分组密钥e对每个消息分组进行加密，加密完消息分组后，将各组的结果与原始消息分组进行异或。其中主流的几个算法有：
   1. Matyas-Meyer-Oseas算法
   2. Davies-Meyer算法
   3. Miyaguchi-Preneel算法
5. Bison对这里的疑问是：密钥e的泄漏不会导致hash函数不安全吗？ 至少对于小于一个分组长度的消息，应该是不安全的，不具备单向性呀

# 消息验证码

1. 和数字签名一样，消息验证码也提供消息完整性验证，但它不提供不可否认性。因为消息验证码是基于对称密钥算法或者hash函数的。消息验证码相比数字签名的优势在于：计算速度和验证速度要快很多。消息验证码（MAC）是对称密钥k和消息x的单向函数
2. Mac算法之一： CBC-MAC
   1. Step1：将消息x分割成分组，使用密钥k和初始值IV，我们就可以计算第一个迭代：
   2. Step2：其他迭代：
   3. Step3：mac就是最后一轮的输出：
   4. 可以看出，上述就是CBC加密的过程，不同的是只传输最后一个分组
   5. MAC验证的过程就是重复生成的过程，不是CBC解密的过程
3. 问题：我觉得下面两种方式也是可行的，但不知道如何验证其安全性，但wiki里列举了一些看似安全却有问题的设计方式，见第4点
   1. 算法一：将x分割成分组，k与每个分组异或后，对异或后的x进行hash，得到的摘要作为mac
   2. 算法二：对x进行hash，得到的摘要用k进行加密
4. 一些看似安全却是有问题的设计方式，说明我们不要随意设计mac，尽量用标准做法

  
（应该是一些hash例如md5抗冲突性不足导致的）

1. 注意区分消息验证码与消息校验码（例如crc\海明码\奇偶校验）：前者防止恶意篡改（恶意者不知道密钥k），后者不能防止恶意篡改，只能一定程度上防止差错，例如网络传输过程中的bug、信号错误等
2. Wiki说：md5抗冲突性有问题，但是hmac-md5又是ok的：“HMAC-MD5 does not suffer from the same weaknesses that have been found in MD5.”。我觉得主要是因为保密的密钥首当其冲的抵挡了冲突性攻击。
3. 以HMAC的实现为例，详细可以见<http://en.wikipedia.org/wiki/HMAC>，代码可以参考tomcrypt库里的hmac相关代码



# 密钥交换与密码协议

1. 在大多数安全系统中，需要使用仅在有限时间内有效的密钥。这样的密钥叫做会话密钥或者临时密钥
2. 限制密钥的有效时长的好处是
   1. 如果密钥泄漏了，引发的危害比较小
   2. 攻击者获得的该密钥的密文较少，时间窗比较短，密码学攻击变得困难
   3. 如果攻击者对更多明文感兴趣，他不得不破解多个密钥，攻击门槛进一步提高
3. 密钥分配问题：系统中有n个用户，两两间共享一对密钥进行通信，那么系统中总共将近个密钥，密钥管理非常困难（试想一下某个用户自己保存的密钥泄漏了，或者有新用户加入系统）
4. 密钥分配中心（KDC）是一个服务器，它与每个用户都共享一个密钥，且所有用户都信任它。KDC和每个用户间共享的密钥成为密钥加密密钥（KEK），它用于将会话密钥安全的传输
5. 使用KDC的系统中，只有n个长期密钥，每个用户只存储他自己的KEK，当新用户加入到系统时，只需要在KDC和用户间完成一次密钥分配，很好的解决了密钥分配问题
6. KDC方式的系统，最典型的是kerberos协议
   1. Alice和KDC共享KEK：, Bob和KDC共享KEK：
   2. Step1：alice想和bob通信，生成随机的一次性r，向KDC发送请求：
   3. Step2：KDC收到alice的请求后，生成随机的会话密钥，并为会话密钥产生有效截止时间T
   4. Step3：KDC应答alice:
   5. Step4：alice解密，验证r、、T
   6. Step5：alice生成消息x和消息的有效截止时间，发送请求y和给Bob
   7. Step6：bob收到请求后，验证、T、，处理消息x
7. 上面kerberos协议有以下安全考虑点：
   1. 在密文里包含了有效期T和，如果中间人一段时间后重放密文，会被丢弃
   2. 使用加密的随机数进行质询，如果返回的随机数匹配，可以确认是KDC发送的
   3. 通过在 中包含bob的身份ID，就可以确认是和bob在通信，同理对Bob也是
8. QQ的ctlogin登录协议也是类似kerberos的，可以认为alice是QQ用户，KDC是密码服务器，Bob是conn、音乐、游戏等业务服务器。Alice的KEK是用户密码，而bob的KEK是服务器间协商的密钥
9. Kerberos的主要问题是：
   1. KDC的单点故障
   2. 没有完美的向前保密，即KEK一旦泄漏，那么攻击者可能获得之前的会话密钥和会话内容（攻击者可能保存了之前的密文）
10. 非对称的会话密钥交换技术，例如DH密钥交换协议（DHKE），主要的问题是中间人攻击，攻击的基本思想是，恶意者将其他参与者发送出来的公钥替换成他自己的公钥。解决办法是用证书机制
11. 证书的大概原理如下：
    1. alice先到CA（有公信力的证书中心，例如CFCA）带上法律证件申请数字证书。
    2. 数字证书是一个经证书授权中心数字签名的包含公开密钥拥有者信息以及公开密钥的文件。一般有企业的公钥、企业的名字和电子邮件地址、公钥的有效期限、 发证机构的名称、数字证书的序列号、 发证机构的数字签名（重要）等字段。
    3. 申请到证书的同时，CA会给alice派发一个与证书中的公钥匹配的私钥（需要企业保密，用于数字签名）。
    4. Bob从alice服务器下载alice的证书后，通过发证机构的数字签名可以验证该证书是否是有效的。 Bob用发证机构的公钥解密发证机构的数字签名，发现能正常解开，且里面的摘要信息字段与整个证书能match上，那说明证书是权威发证机构派发且未经篡改过。CA的公钥一般是操作系统预埋的，且有多个CA。CA间可以相互认证。
    5. Bob从经过验证的证书中获得alice的公钥
12. 证书的本质，是信任转移。没有证书的DHKE协议下，alice和bob必须直接信任双方的公钥。随着证书的引入，他们只需要信任CA的公钥即可。如果CA对其他公钥进行了签名，则alice和bob都知道可以信任这些被签名的公钥
13. SRP协议（Secure Remote Password protocol）是基于DH的协议，他的原理是
    1. 选择一个大素数N，和一个群。是公开的。
    2. I表示用户ID，P表示用户明文密码，s表示用户的盐，H()是md5这样的单向函数
    3. 服务器端数据库保存有｛I，s，｝其中x=H(s, p)
    4. Step1: 客户端产生一个随机数a，并计算 (%N后面都不写了)，并将A发送给服务器
    5. Step2：服务器产生一个随机数b，并计算，并将B发送给客户端
    6. Step3：客户端计算，并生成，将发送给服务器
    7. Step4: 服务器计算，验证客户端发送过来的M1，如果相同，那么认为客户端合法
    8. Step5：服务器端生成，发送到客户端。客户端校验M2，如果相同，则认为服务端合法
    9. Step6：业务的会话，用K作为session key
    10. 改进：为了避免server被拖库而恶意者可伪造登录，协议会加上一个随机数，可查询<http://en.wikipedia.org/wiki/Secure_Remote_Password_Protocol>