

古典密码→现代密码

- 古典密码
 - 对字符或单词进行处理（加解密）
 - 明文一般是有意义的自然语言文本
- 现代密码
 - 面向计算机和芯片（对比特、字节或字进行处理）
 - 明文可以是任意比特串

密码体制安全性的两种定义（无条件安全性，计算安全性）

① 无条件安全性（信息论角度）

即使攻击者拥有无限的计算资源，也无法被破译

② 计算安全性（计算复杂性角度）

攻击者用(现在或将来)可得到的资源无法破译

讨论加密体制的安全性时，需要考虑两点：

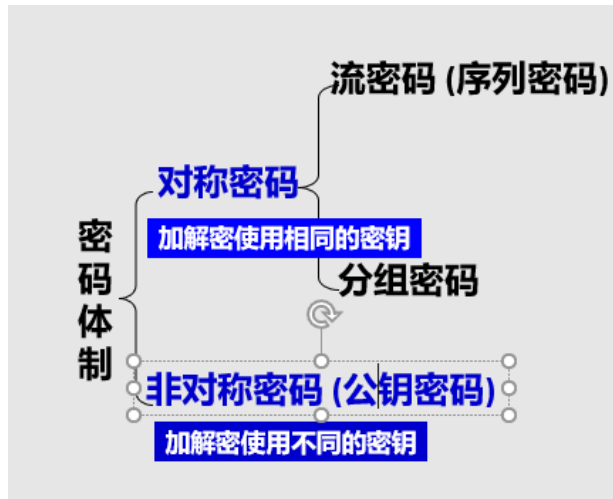
① 能抵抗攻击者什么类型的攻击

唯密文攻击、已知明文攻击……

② 能阻止攻击者实现什么样的目标

找到密钥、破译特定的密文……

攻击者的攻击能力越强，要实现的目标越低，越难对付



计算机本身只能产生伪随机序列

随机序列的类型

1. (一般) 伪随机序列

随机序列的类型

(一般)

随机序列

性质1：伪随机序列应满足的性质：

看上去是随机的

这表明它通过了所谓的随机性统计测试：

① 1和0的数目应该大约相等
 ② 一些0和1组成的模式出现的不要“太频繁”，也不要“太不频繁”？

结论：伪随机序列 应该在统计上是随机的

2. 密码学意义上安全的伪随机序列

性质2：并不是任何伪随机序列都能用于密码学，密码学意义上安全的伪随机序列还必须具备另一个性质：**不可预测性**。

不可预测性即使知道了产生序列的算法、以前产生的所有比特，也不可能通过计算来预测下一个比特是什么（准确预测成功的概率很低）目的是防止攻击者在知道若干比特后，成功猜测后续比特

3. 真随机序列

不能被可靠地重复产生

用完全同样的输入操作两次，得到的是两个不相关的序列


Q: 如何得到真随机序列？

使用一种专门的设备(真随机数发生器)，输入是各种无法预测的信号：周围空气状况、电流的变化率...

- 输入是不断变化的，输出也就不可重复。
- 有人能预测产生的下一个数吗？ 他必须重构输入信号，这没人能做到

一次一密

- 加解密很简单
 - 加密：明文流与密钥流对应比特异或
 - 解密：密文流与密钥流对应比特异或



1001010101101... ← 随机密钥流

1111010110101... ← 明文流

0110000011000... ← 密文流

一次一密

特点：**密钥流是真随机序列，且不重复使用(故得名 一次一密)**

安全性：因为密钥流是真随机的，所以没人能预测下一个比特

具有无条件安全性（完善保密性）：

即使攻击者具有无限的计算资源，也无法破译一次一密（理论上不可破译，无条件安全的）

原因：密钥是随机的，而且明文与密钥是统计上相互独立的，使得密文也是随机的，故而密钥的随机性很好的隐藏了明文的统计特性

一次一密 无条件安全（完善保密性）

无条件安全性的概率表示

对于任意密文 y 和明文 x ，都有 $\Pr[x | y] = \Pr[x]$
即，不能通过分析密文获得明文的任何信息

如果重复使用同一个密钥加密不同的明文，很容易被破译

已知明文攻击

若攻击者获得了一个密文 $c=(c_1\ c_2\ \dots)$ 和对应明文 $m=(m_1\ m_2\ \dots)$ 时，就很容易得出密钥 $k=(k_1\ k_2\ \dots)$ 。

若重复使用该密钥，一次一密就很不安全。

一次一密 一次一密的缺点

只使用异或运算，软硬件实现非常简单。但实用性不强

实用性不强的原因

1. 密钥是真随机的
2. 密钥长度至少等于明文长度；
3. 每个密钥只用一次。

分发/存储这样的密钥序列，并确保其安全是很困难的；如何获得真随机序列也是一个现实的问题

限制了它在商业上的应用，但在军事或政府领域仍有应用

一次一密的缺点（密钥是真随机，长度要等于明文长度，每个密钥只能用一次）

流密码

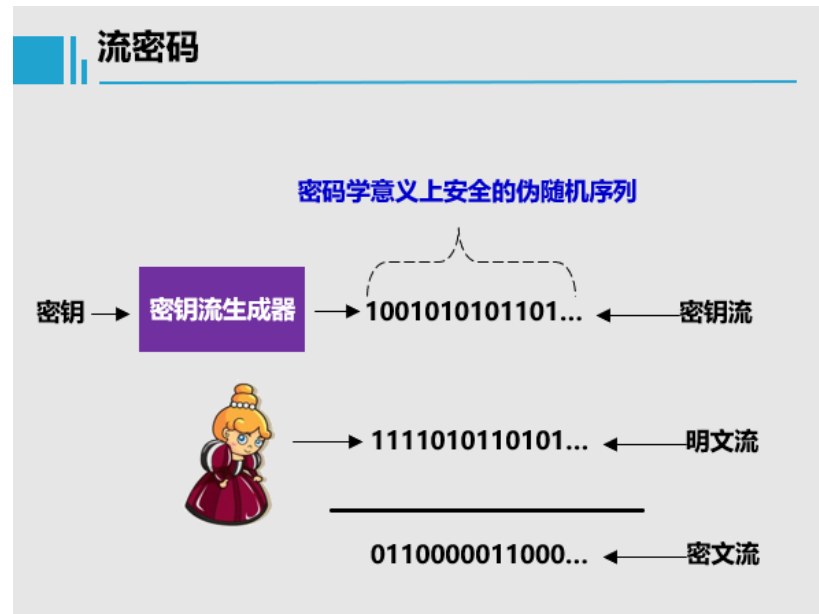
流密码的基本思想：

用一个较短的密钥生成密钥流，在攻击者(计算能力是有限的)看来是随机的(伪随机的)。
不是无条件安全的，仅是计算上安全的

产生密钥流的任务由密钥流生成器完成：

输入：“较短的密钥”

输出：源源不断的密钥流



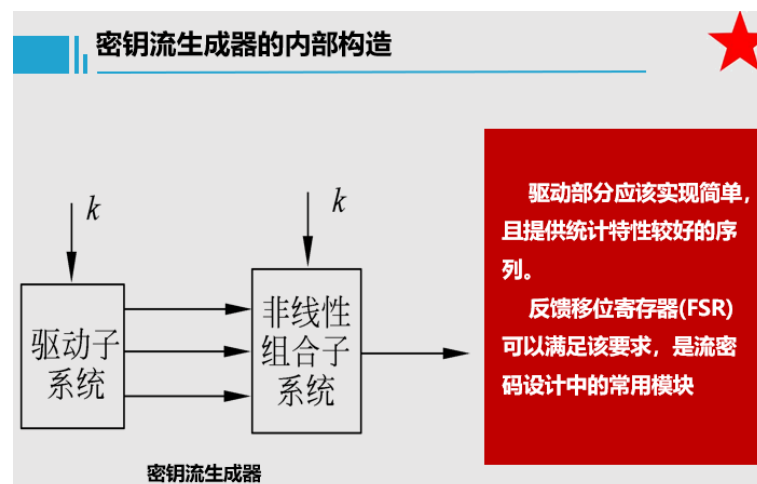
密钥流生成器的组成（驱动部分和非线性组合部分）

1.驱动部分

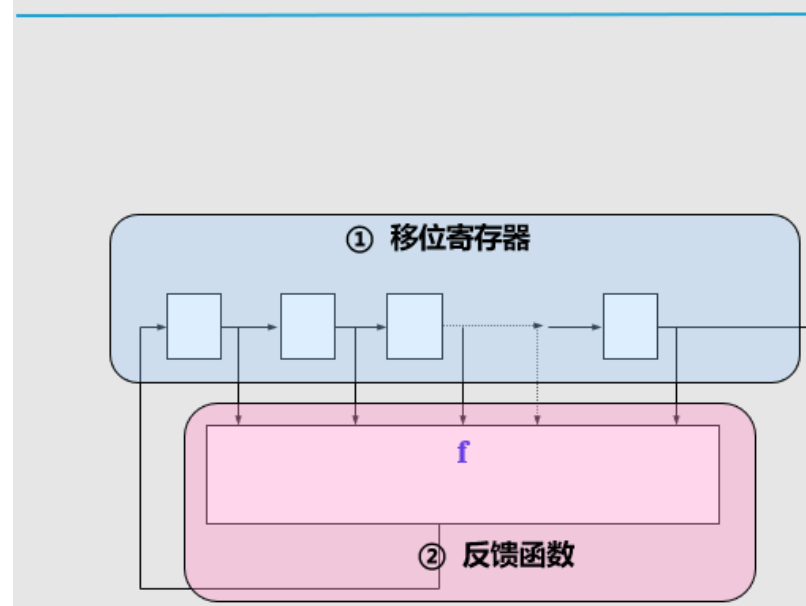
为非线性组合部分提供统计特性“好”的序列（一般伪随机序列）

2.非线性组合部分

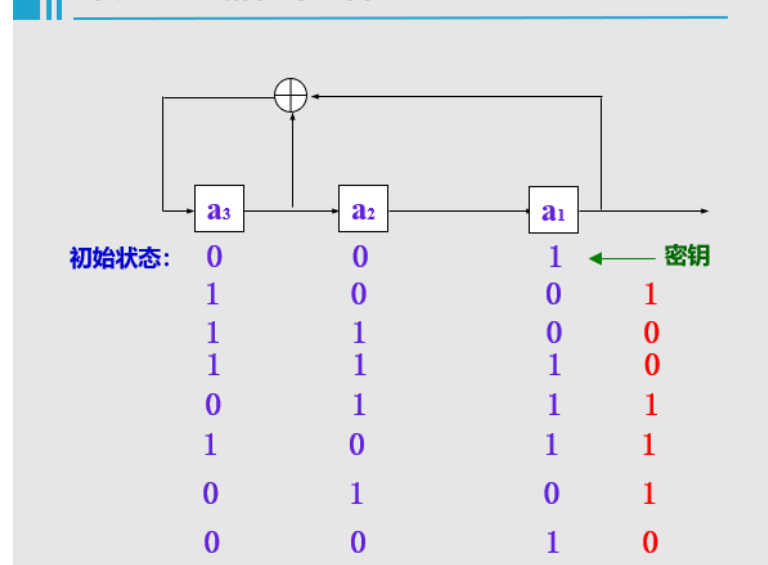
将提供的输入序列组合成密码学特性“好”的序列（密码学意义上安全的伪随机序列）



密钥流生成器的内部构造 反馈移位寄存器 (FSR)



密钥流生成器的内部构造 反馈移位寄存器 (FSR)



Q: 为什么不直接使用种子作随机数？

1. 速度问题：收集种子通常很耗时
2. 熵(不确定性)：种子的熵通常比较低，但不管其熵如何，伪随机序列发生器都能产生统计性好的序列。

密钥流生成器的用途 **伪随机序列发生器**

- 攻击者可能尝试重构生成密钥流的种子。
- 攻击原理：
 - 攻击者完全可以知道你收集种子的方法(柯克霍夫斯原则)
 - 如果你使用的种子不够“好”，就很容易被攻击者重构。
 - 必须使用“好”的种子，使之具有不可预测性

Q: 如何收集种子才是安全的?

- 毫秒计的时间
- 用户的输入
- 鼠标点击的位置, ...

它们混合在一起便有了不可预测性，可以抵抗种子猜测攻击



混乱

目的：

使明文和密文之间、密钥和密文之间的

相关统计特性极小化，从而使攻击者无法找到密钥

常用方法：代换

扩散

目的：

将明文及密钥的影响尽可能迅速地散布到较多个密文比特中。

常用方法：置换

什么叫 代换?

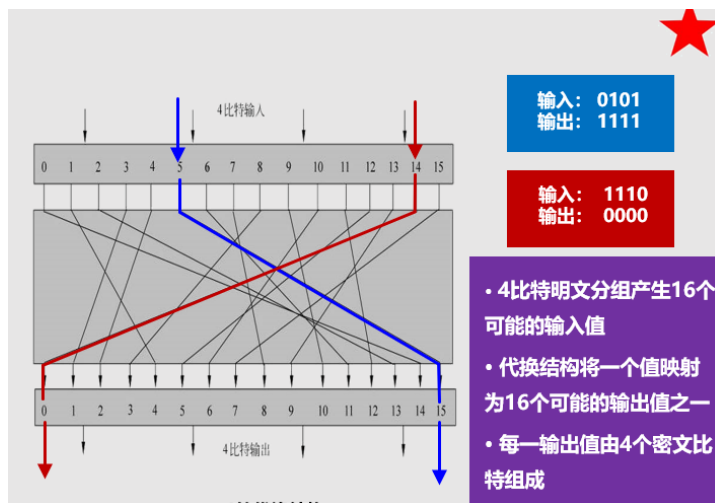
加密时, 明文的每个分组都应产生唯一的密文分组 (具有可逆性, 为了解密), 称这种明文分组到密文分组的可逆变换为代换

Q: 如果明文分组为 n 比特, 则明文分组有多少个可能的取值?

2^n

Q: 不同可逆变换有多少个?

$2^n!$



分组长度	明文分组、密文分组长度都是 64比特
密钥空间	密钥长度也是 64比特 其中有效密钥长度 56比特 (有8比特奇偶校验位)
算法	解密过程与加密过程完全相同 唯一不同的是, 子密钥的使用顺序完全相反 <ul style="list-style-type: none">• 加密: $k_1, k_2, k_3, \dots, k_{16}$• 解密: $k_{16}, \dots, k_3, k_2, k_1$

数据加密标准 (DES) 针对 DES 的攻击

• 差分密码分析

- 1990 年, 以色列密码学家 E.Biham 和 A.Shamir 提出, 可对 DES 进行选择明文攻击。
- 该成果主要是理论上的。因为它所要求的时间量和数据量超出每个人的承受能力。
- 因此如果正确实现 DES 算法, 对差分密码分析仍是安全的。
- 为什么 DES 能抵抗差分密码分析呢?
 - 设计者早在差分密码分析法公布于世之前就已经知道了这种攻击方法。

- 穷举密钥攻击
 - 由于 DES 的有效密钥长度只有 56 比特,因此考虑穷举密钥攻击还是很靠谱的。
 - 1999 年 1 月,电子前沿基金会(Electronic Frontier Foundation)仅用 22 小时 15 分钟,就宣告破解了一个 DES 的密钥。



流密码与分组密码的比较

流密码的优势	分组密码的优势
<ul style="list-style-type: none"> • 速度快 • 代码量少 (RC4的代码只有30行) 	<ul style="list-style-type: none"> • 密钥可以重用 • 有标准化算法(DES、AES)
总结	
<ul style="list-style-type: none"> • 没有哪种密码更好,具体选用哪个要看你干什么 • 需要重复使用密钥,或保证互用性时,选用分组密码,否则就用流密码 	

应用	所用密码	评价
数据包	AES	与其他软件的互用性不是问题
E-mail	AES	也可用流密码，但考虑到AES更安全更通用。使用AES可以使你与E-mail服务器保持互用性。
SSL	RC4(流密码)	速度是非常重要的，每次通信需要使用不同的密钥，而且事实上，所有浏览器和服务器都有RC4
文件加密	分组密码	可以用相同的密钥保护不同的文件，密钥管理很方便

填充

Q: 通常，明文长度是不固定的，按固定长度分组时，往往最后一个分组长度不足，如何解决？

方法：填充。

如何填充

① 填充一些字符补齐最后一个分组。

② 把最后一个分组的最后一个字节称作填充指示符，所表示的十进制数字就是填充了多少字节。

③ 明文尾部、填充的字符和填充指示符一起作为最后一组进行加密。

最后一个分组

明文尾部

填充字符

指示符

不论最后一个分组长度是否足够，都要进行填充

明文最后一个分组

长度不足时

明文尾部 填充字符 指示符

明文最后一个分组 补充一个分组

长度足够时

明文尾部 填充字符 指示符

Q: 为什么需要工作模式？

分组密码的输入是一个明文分组，是定长的。

要加密的明文是变长的，长度往往大于一个明文分组。

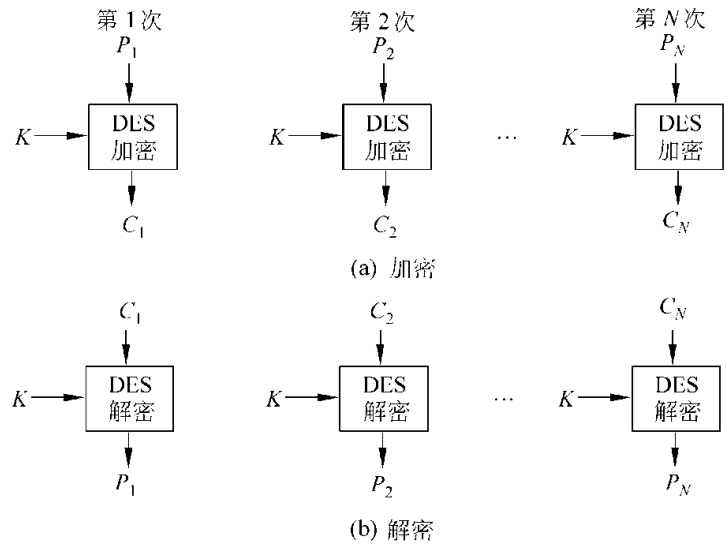
即使有了安全的分组密码体制，也需要采用适当的工作模式来隐蔽明文的统计特性，以提高整体的安全性。

五种工作模式

模式名称	缩写	英文全称
电子密码本	ECB	Electronic CodeBook
密码分组链	CBC	Cipher Block Chaining
密码反馈	CFB	Cipher FeedBack
输出反馈	OFB	Output FeedBack
计数器	CTR	Counter

电子密码本模式 ECB

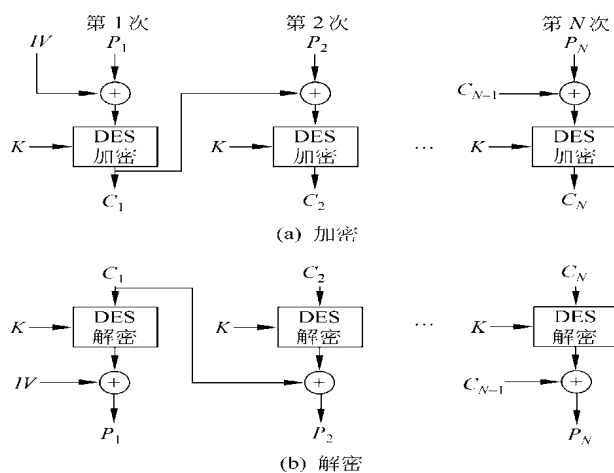
- 每个明文分组独立加/解密




ECB模式的特性	
优点	缺点：安全性差
<ul style="list-style-type: none">• 简单、高速• 无差错传播：单个密文分组出现错误只会影响该分组的解密，不会影响到其他分组。	<ul style="list-style-type: none">• 分组彼此独立，相同密钥下，相同明文分组得出相同密文分组。• 这会暴露明文数据的格式和统计特征。• 且易受重放、插入攻击，五种模式中安全性最弱
ECB适用于发送少量数据的场合，一般不推荐。	

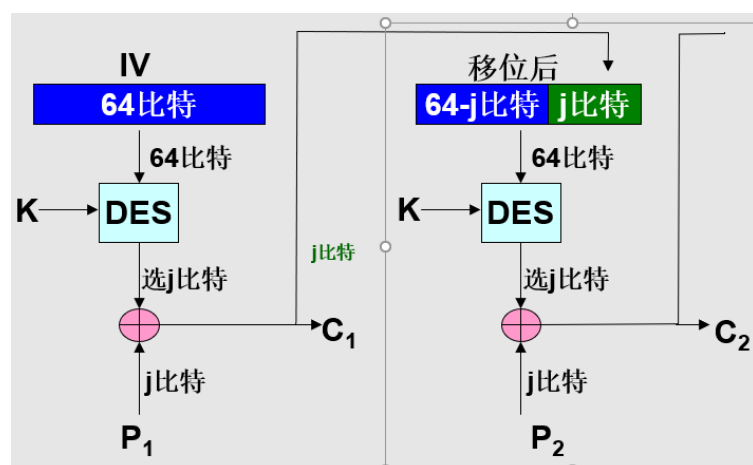
密码分组链模式 CBC

- 每个明文分组先与前一密文分组异或，再进行加密



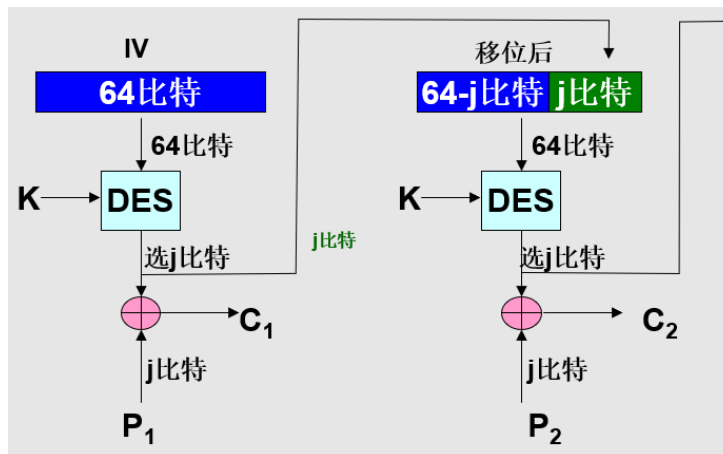
CBC模式的特性		
初始矢量 IV	<ul style="list-style-type: none">收发双方必须使用相同IVIV无需加密保护, 可以随密文一起发送给接收方最好使用不同IV(比如每次将IV加1)加密不同明文	CBC适用 于文件加密, 较ECB模式慢。
<ul style="list-style-type: none">加密第一个明文分组时,尚无反馈的密文, 为此需要预先置入一个, 称为 初始矢量 IV(Initial Vector)		
有限差错传播		
<ul style="list-style-type: none">单个密文分组出现错误会影响该分组和后面一个密文分组的解密。可自同步, 只要后面一个密文分组没错, 便不会影响后续密文分组的解密。		

密码反馈模式 CFB



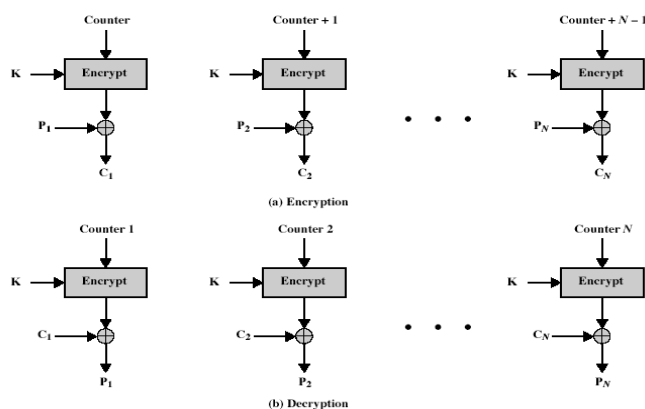
- CFB 适用于 明文按字符(如电传电报)或按比特处理的流密码中
- CFB 适用于 无延迟的加密和传播
- CFB 适用于 容忍以少量错误扩展换来恢复同步能力的场合

输出反馈模式 OFB



OFB 适用于 必须避免错误传播的高速同步系统

计数器模式 CTR



速度	其他	CTR适用于需要并行处理的应用领域
<ul style="list-style-type: none"> 实现简单（只要求加密算法） 效率高 <ul style="list-style-type: none"> 可预计算 可并行加密、吞吐量仅受可并行数量的限制 	<ul style="list-style-type: none"> 可随机访问密文数据块 可证明安全性 	

保护密钥是很重要的，因为所有的安全性都依赖于密钥的机密性
(柯克霍夫斯原则)

Q: 是否可以记住密钥，这样就不用存储，别人也偷看不了？

理论上可以，但不实际，因为密钥是随机的，很难记忆

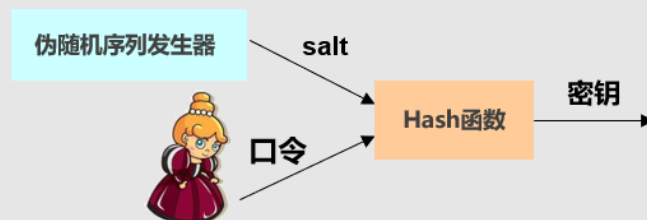
Q: 既然有地方安全保存密钥，为啥不直接把敏感信息放在那里？

保护短的密钥比保护数以兆计的信息更容易

对称密钥的管理策略

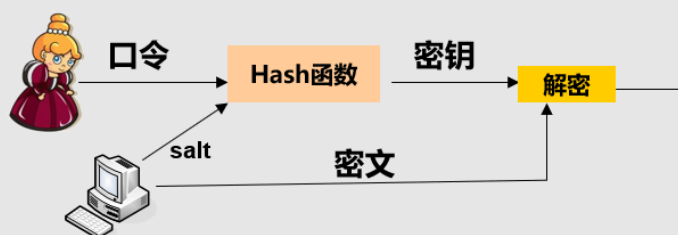
用 密钥 保护数以兆计的信息，用一些其他技术保护大约 16 字节(128bit)的密钥

基于口令的加密 工作原理如下(加密):



- ① 选择一个口令
- ② 伪随机序列发生器产生一个salt
- ③ 用一个算法混合口令和salt，通常使用Hash函数
- ④ 从Hash函数的输出中取出所需要长度的比特作为密钥
- ⑤ 密钥用完后丢弃。用脑子记住口令。
- ⑥ salt无需保密，可以和加密后的数据一起保存。

基于口令的加密 工作原理如下(解密):



- ① 输入口令
- ② 从保存点(磁盘)取出salt和密文
- ③ 用算法(Hash函数)混合口令和salt，产生密钥
- ④ 用该密钥解密密文，以恢复明文

Q: 为什么不直接用口令作密钥？

口令的熵很小，远达不到密钥所要求的随机程度

Q: salt (盐值) 是干什么用的？

目的：为防止字典攻击中的预计算。

如果密钥仅由口令生成，攻击者可创建一个常用口令表 (字典)，对里面所有口令预计算密钥。每次使用时就尝试这些密钥。字典可从黑客网站下载，也有很多发动字典攻击的软件

Q: 为什么 salt 和密文一起保存，保密 salt 不是更安全？

- 使用 salt 的唯一目的是防止字典攻击中的预计算，而不是增加安全性。
- 即使 salt 不保密，仍能达到目的。
- 此外，如果保密 salt 的话，还不如直接保密密钥。

保护密钥的密钥 	
通信时，通常使用两种密钥	使用方法
<ul style="list-style-type: none"> • 会话密钥 (临时密钥) • 密钥加密密钥 (KEK) 	<ul style="list-style-type: none"> • 每次通信使用不同的会话密钥保护明文 • KEK保护会话密钥，并将之传给对方
目的	
因为每次通信使用不同的会话密钥，攻击者无法获得同一密钥加密的大量密文，使得破译更加困难	

Q: 用 KEK 保护会话密钥，用什么保护 KEK？

•使用 基于口令的加密，步骤如下：

- ① 用口令和 salt 产生 KEK。
- ② 用 KEK 加密会话密钥，用会话密钥加密明文
- ③ 记住口令，保存 salt 和加密后的会话密钥。
- ④ 将密文和加密后的会话密钥传给对方。
- ⑤ 至于 KEK，丢掉就 OK 了。

Q: 假设攻击者闯入你的电脑，偷走 salt 和密文(或 加密后的会话密钥)，他如何计算密钥？

方法一

穷举攻击密钥

(无需 salt 也可以发动)

方法二

字典攻击 (猜测口令)

因为 salt 不保密，如果攻击者获得了 salt，他仍能发起字典攻击，只不过要多花一些时间。

构造或下载一本常用口令的字典，然后尝试每个口令和 salt 产生的密钥。

两种攻击方法的比较

相比之下，字典攻击速度更快。但若口令不在字典里，当然不会成功

一个聪明的攻击者一般会这样做：

首先尝试字典攻击，失败后换用改进的穷举密钥攻击。

如何抵挡字典攻击？

基本思想：想办法降低攻击者的计算速度

基本方法：Hash 函数混合 salt 和口令后，将输出再次用 Hash 函数混合，重复进行多次，假设 1000 次。

优点

- Hash函数计算速度比你想象的快得多。
- 事实上，计算1000次Hash函数比你从键盘输入口令的时间还要短。
- 但攻击者不得不对字典中每个口令计算1000次，使他总的计算时间会很长。

其他的实用方法

- 对于一些在线系统，如ATM机、网上银行、电子邮箱、论坛等，可以限制用户输入口令的次数
- 若在规定次数内没有输入正确的口令，便吞卡，或提示半小时后才能再次使用系统

基于硬件的密钥存储

把密钥保存在一个硬件设备上

- 令牌
- 密码加速器

使用令牌的好处

- 可以随身携带：钱包里、钥匙链上、当戒指戴
- 攻击者要获得密钥，必须先拿到令牌，这增加了攻击难度
- 令牌内部还有进一步保护措施
 - 需要输入一个正确的口令才能使令牌发挥作用
 - 试图用物理手段获得密钥，令牌还有自毁功能

令牌的唯一缺点

- 必须把密钥加载到内存里才行，有可能会被木马等病毒偷走
- 但密钥使用完毕就从内存中抹掉了，短短几分钟甚至几秒钟的停留还是问题不大的

密码加速器

- 有专门进行密码运算的芯片，比一般CPU处理密码更快，也比常规计算机更安全的存储数据。

- 安全特性

- 存储空间对外界不可见
- 一旦被撬，有自毁装置
- 不允许密钥离开。加密时把明文送入加速器，它返回密文。

- 通常与令牌一起使用

- 只有插入令牌，提供令牌正确的口令，加速器才能工作



生物统计学方法并不总是靠谱

- 生物统计学公司正在尝试制造相应的硬件，但至今未得到广泛应用，原因有二

二

- ① 设备的价格
 - ② 可靠性:是不是总能正确地提取出身体特征？
- 但令牌不存在这些问题，它总能 100%的工作

习题：

1. 流密码体制由 (D) 两部分组成
A. 驱动部分、反馈函数 B. FSR、反馈函数
C. FSR、非线性组合部分 D. 驱动部分、非线性组合部分
2. 设计分组密码的两种技术是 (C)
A. 置换和移位 B. 易位和置换
C. 混乱和扩散 D. 隐写和扩散
3. (D) 不是分组密码的工作模式
A. CBC B. OFB C. CTR D. PBE
4. 下列哪个不是分组密码体制 (D)
A. DES B. AES C. IDEA D. RC4
5. 下列哪种模式的安全性最差 (A)
A. ECB B. CBC C. OFB D. CFB
6. CBC 模式中，一个密文分组传输错误，会影响 (B) 个密文分组的解密
A. 1 B. 2 C. 3 D. 4