# 加密与解密

## 流密码

同步流密码

自同步流密码

## 分组密码

分组密码多应用于网络加密，是对此密码体质中发展最为完善的密码体制。

分组密码对**固定长度**的一组明文进行加密，这一固定长度称为**分组长度**。

分组长度是分组密码的一个参数，它与分组算法的安全性成正比，其取值范围取决于实际引用的环境。

为保证分组算法的安全性，分组长度越长越好，分组长度越长，密码解析越困难；

为保证分组算法的实用性，分组长度越短越好，分组长度越短，越便于操作和运算；

分组长度的设定需要权衡分组算方法的安全性与实用性，一般设置为56位。但随着密码学的发展，分组长度只有56位的分组密码已经不能确保算方法的安全性。目前。分组密码多选择128位作为算方法的分组长度。

### 加密过程

对一个 长度为 n 的 明文分组进行加密操作，相应地产生一个n位的密文分组，由此可见，不同的n位明晚分组共有2的n次方个。每一个不同的n位明文分组都应该产生一个唯一的密文分组，加密过程对应的变换成为**可逆变换**或**非奇异变换**。

代表性的分组加密算法有DES、AES等。

### 设计原则

1. 不可破译原则

理论上不可破译

实际不可破译

扩散原则

混乱原则

密码体质过于复杂导致对密码分析者来说是无法利用的

如何衡量一个密码的安全性？

密码体制的破译所需要的时间和费用超出了现有的资源和能力

密码体制的破译所需要的时间超过了该体制所保护信息的有效时间

密码体制的破译所需要的费用超过了该体制所保护信息的价值

1. 针对实现的设计原则

软件实现的设计原则

硬件实现的设计原则

### 分组密码的工作模式

以DES算法为例，DES算法根据其加密算法所定义的明文分组的大小（56位），将数据分割成若干56位的加密区块，再以加密区块为单位，分别进行加密处理。最后剩下的不足一个区块的大小，称为短块，短块的吃力方法有填充法，流密码加密法，密码挪用技术。

1. 电子密码本模式（ECB）
2. 密文链接模式（CBC）
3. 密文反馈模式（CFB）
4. 输出反馈模式（OFC）
5. 计数器模式（CTR）

## 对称加密算法

加密与解密使用同一个密钥。

发送消息

密钥加密消息

密文

接收消息

密钥加密消息

密文

网络传输

密钥

## 非对称加密算法RSA

加密解密使用一对钥匙，称为公钥与私钥。而使用私钥加密的数据，只有对应的公约才能解密，而使用公钥加密的数据，只能使用对应私钥才能解密。

发送消息

接收消息

公钥 -> 加密算法

密文

密文

私钥 -> 解密算法

由于 非对称加密算法 效率较低，故可以通过 非对称加密算法 传送 对称加密算法 的密钥。

详细应用请看 网络协议

# 散列函数

又名哈希函数、消息摘要函数、单向函数、杂凑函数。

散列函数主要用于验证数据完整性，是 数据签名 的核心技术。散列函数常见算法有 MD（消息摘要算法）、SHA（安全散列算法）、Mac（消息认证码算法）

# 定位模拟法

'''

Created on 2018年8月25日

@author: Administrator

'''

'''

#https://zhuanlan.zhihu.com/p/38850888

title:

一副从1到n的牌，

每次从牌堆顶取一张放桌子上，

再取一张放牌堆底，

直到手机没牌，

最后桌子上的牌是从1到n有序，

设计程序，输入n，输出牌堆的顺序数组

'''

#弃牌 排序

# 1 2 3 4 5

# table | cards

#[] | [1,5,2,4,3]

#[1 ] | [5,2,4,3] -> [2,4,3,5]

#[1,2] | [4 , 3, 5] -> [3,5,4]

#[1,2,3] | [4,5] -> [5]

#[1,2,3,4] | [5]

#[1,2,3,4,5] | []

#排序 加牌

#[1,2,3,4,5] | []

#[1,2,3,4] | [] -> [5]

#[1,2,3] | [5] -> [4,5]

#[1,2] | [5,4 -> [3,5,4]

#[1] | [4,3,5] -> [2,4,3,5]

#[] | [5,2,4,3] -> [1,5,2,4,3]

# 结果数组 1 2 3 4 5 值

# 定位数组 1 3 5 4 2 位置

# 原数组 1 5 2 4 3

# 定位模拟法

# 将 等长的自然数数组 作为定位数组

# 按照变换规则将定位数组变换n次，得到结果数组进行n次变换后的位置

# 将结果数组 根据 定位数组 变换位置 得到原数组

n = 6

list\_result = list( range(1, n ) ) #声明一个结果数组

print( "结果数组:\t\t"+str( list\_result ) )

list\_location = list( range(1, n ) ) #初始化 定位数组

print( "初始定位数组:\t"+str(list\_location) )

#对 定位数组 按照规则 变换

count = 0

length = len(list\_location)

while count < length-1 : #根据变换规则进行变换

temp = list\_location[count+1]

list\_location.remove(temp)

list\_location.append(temp)

count=count+1

print( "变换后定位数组:\t"+str(list\_location ))

list\_ori = []

count = 1

while count <= length :

list\_ori.append( list\_result[ list\_location.index( count )] )

count = count + 1

print("原数组:\t\t" + str(list\_ori))

# 如果能令变换规则更加抽象，事情大概会更简单