

第2章 密码技术

中国科学技术大学

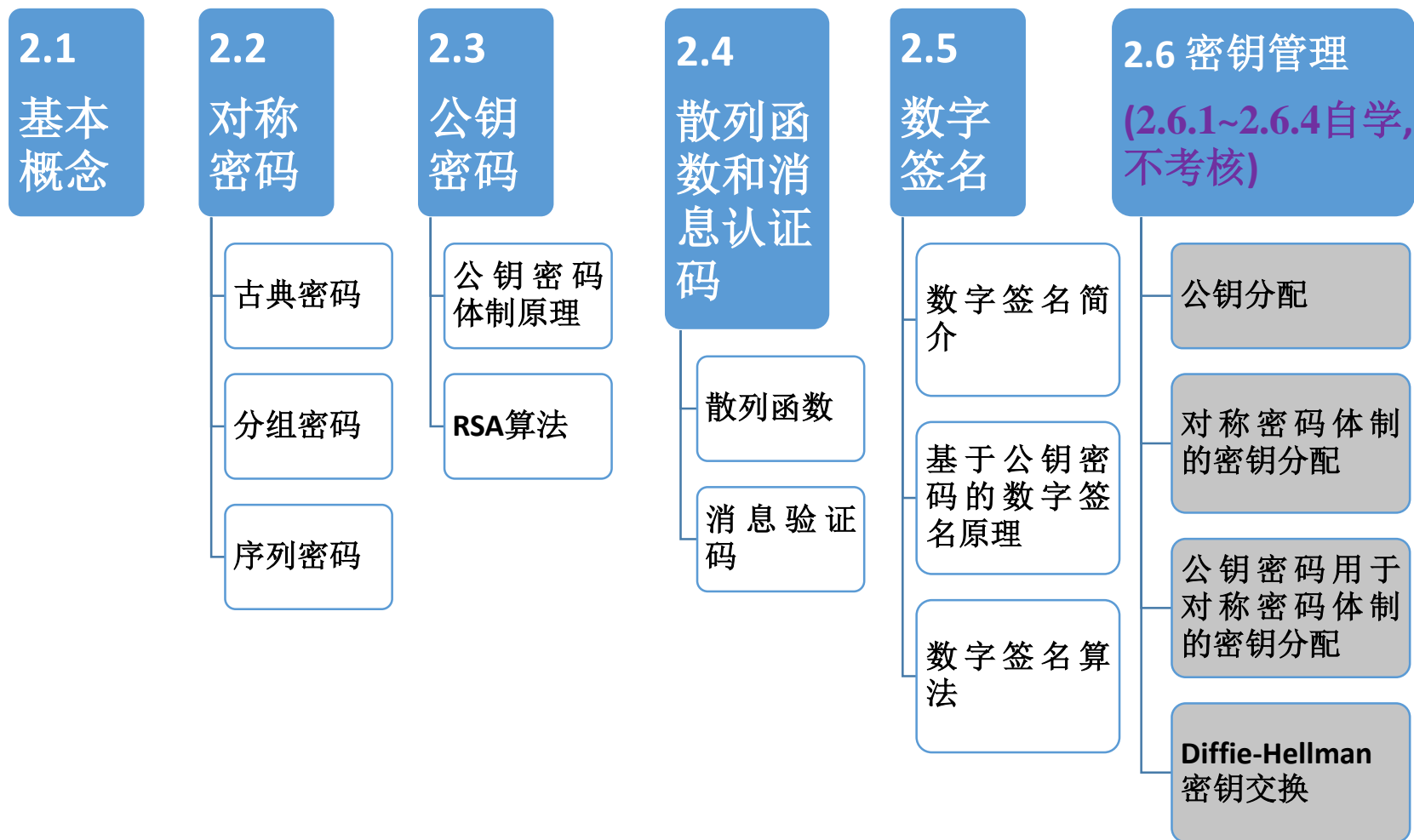
曾凡平

billzeng@ustc.edu.cn

课程回顾：第1章 绪论

1. 信息安全的概念
2. 信息安全发展历程
3. 信息安全技术体系
4. 信息安全模型
5. 信息安全保障技术框架

第2章 密码技术(核心基础安全技术)



2.1 基本概念

- 密码学=密码编码学+密码分析学
- **密码编码学**：研究密码变化的客观规律，设计各种加密方案，编制密码以保护信息安全的技术。
- **密码分析学**（或密码破译学）：在不知道任何加密细节的条件下，分析、破译经过加密的消息以获取信息的技术。
- **明文**：原始的消息(**p**laintext | **m**essage)
- **密文**：加密后的消息(**c**iphertext)

密码学模型

- 明文到密文的变换法则，即加密(**encrypt**)方案，称为**加密算法**；而密文到明文的变换法则称为**解密(decrypt)算法**。加 / 解密过程中使用的明文、密文以外的**其他参数**，称为**密钥(key)**。

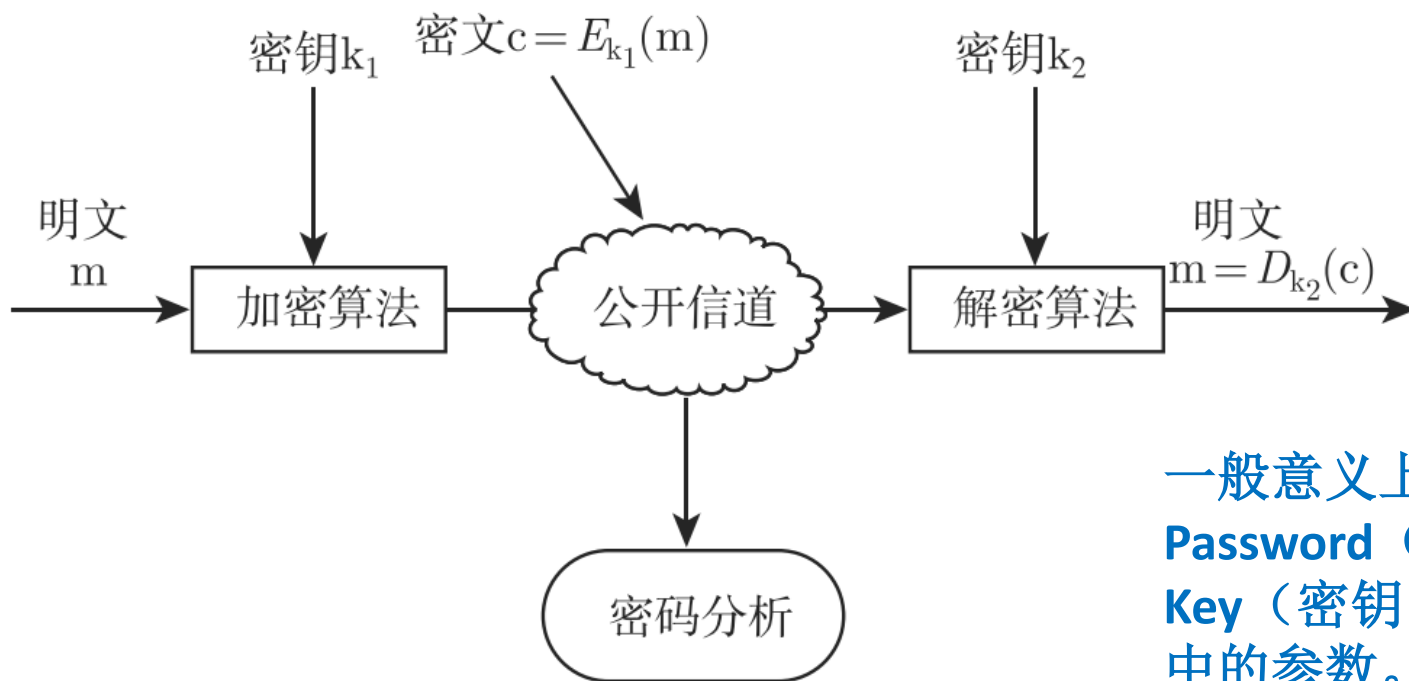


图2-1 密码学模型

一般意义上的**Password ≠ Key**，**Password**（口令）是一串字符，**Key**（密钥）是加密/解密过程中的参数。
在密码学中，不能混为一谈。

密码体制

- **密码体制**：一个用于加解密并能够解决网络安全中的机密性、完整性、可用性、可控性和真实性等问题中的一个或几个的系统。
- 密码体制可以定义为一个**五元组(P, C, K, E, D)**
 - P**称为明文空间，是所有可能的明文构成的集合；
 - C**称为密文空间，是所有可能的密文构成的集合；
 - K**称为密钥空间，是所有可能的密钥构成的集合；
 - E**和**D**分别表示加密算法和解密算法的集合，它们满足：
 - 对每一个 $k \in K$ ，必然存在一个加密算法 $e_k \in E$ 和一个解密算法 $d_k \in D$ ，使得对任意 $m \in P$ ，恒有
$$d_k(e_k(m)) = m$$

2种密码体制：对称密码和非对称密码

- **对称密码体制也叫单钥密码体制或秘密密钥密码体制：**在对称密码体制中，加密和解密使用完全相同的密钥，或者加密密钥和解密密钥彼此之间非常容易推导。
- **非对称密码体制也称为公钥（公开密钥）密码体制：**在公钥密码体制中，加密和解密使用不同的密钥，而且由其中一个推导另外一个是非常困难的。这两个不同的密钥，往往其中一个是公开的（**公钥，通常用于加密**），而另外一个保持秘密性（**私钥，通常用于解密**）。

密码体制的安全性(无条件安全和计算上安全)

- 对一个密码体制来说，如果无论攻击者获得多少可使用的密文，都不足以唯一地确定由该体制产生的密文所对应的明文，则该密码体制是**无条件安全**的。除了一次一密，其他所有的加密算法都不是无条件安全的。因此，实际应用中的加密算法应该尽量满足以下标准：
 - (1)破译密码的代价超出密文信息的价值。
 - (2)破译密码的时间超出密文信息的有效生命期。
- 满足了上述两条标准的加密体制是**计算上安全**的。
- 对一个计算上安全的密码体制，虽然理论上可以破译，但是由获得的密文以及某些明密文对来确定明文，却需要付出巨大的代价，因而不能在希望的时间内或实际可能的条件下求出准确答案。

攻击密码体制的两种方法

(1)密码分析攻击

- 攻击依赖于加密 / 解密算法的性质和明文的一般特征或某些明密文对。这种攻击**企图利用算法的特征来恢复出明文，或者推导出使用的密钥。**
- **①唯密文攻击：**攻击者在仅已知密文的情况下，企图对密文进行解密。这种攻击是最容易防范的，因为攻击者拥有的信息量最少。
- **②已知明文攻击：**攻击者获得了一些密文信息及其对应的明文，也可能知道某段明文信息的格式等。比如，特定领域的消息往往有标准化的文件头。
- **③选择明文攻击：**攻击者可以选择某些他认为对攻击有利的明文，并获取其相应的密文。如果分析者能够通过某种方式，让发送方在发送的信息中插入一段由他选择的信息，那么选择明文攻击就有可能实现

攻击密码体制的方法

- **④选择密文攻击**：密码攻击者事先搜集一定数量的密文，让这些密文透过被攻击的加密算法解密，从而获得解密后的明文。
- 以上几种攻击的强度依次增强。如果一个密码体制能够抵抗选择密文攻击，则它能抵抗其余三种攻击。

(2)穷举攻击（暴力破解）

- 攻击者对一条密文尝试所有可能的密钥，直到把它转化为可读的有意义的明文。
- 即，使用计算机来**对所有可能的密钥组合进行测试**，直到有一个合法的密钥能够把密文还原成明文。
- 平均来说，**要获得成功必须尝试所有可能密钥的一半**。

2.2 对称密码

- 在对称密码体制中，加密和解密使用完全相同的密钥，或者加密密钥和解密密钥彼此之间非常容易推导。
- 对称加密是20世纪70年代公钥密码产生之前唯一的加密类型。迄今为止，它仍是两种加密类型中使用更为广泛的加密类型。
- 对称密码包括
 - 古典密码：1949年之前的密码技术
 - 分组密码和序列密码：1949年之后发明的，公钥密码产生之前的主要密码技术，目前仍然在发展之中。

2.2.1 古典密码

- 在1949年之前的古典密码时期，密码学家凭借直觉进行密码分析和设计，以手工方式，最多是借助简单器具来完成加密和解密操作。
- **古典密码技术以字符为基本加密单元**，大都比较简单，经受不住现代密码分析手段的攻击，因此已很少使用。
- 但是，在漫长的发展演化过程中，古典密码学充分体现了**现代密码学的两大基本思想：置换和代换（Permutation and Substitution）**，还将**数学的方法引入到密码分析和研究中**。这为后来密码学成为系统的学科以及相关学科的发展奠定了坚实的基础。

1. 置换(Permutation)技术

- 保持明文中的字母本身不变，但将所有字母重新排列，即仅仅改变明文字母的位置，这样的密码技术称为置换。
- 栅栏技术是最简单的置换技术。栅栏密码把要加密的明文分成 N 个一组，然后把每组的第一个字符连起来，再加上第二个、第三个，以此类推。本质上，是把明文字母一列一列地(列高就是 N)组成一个矩阵，然后一行一行地读出。
- 如果令 $N=2$ ，则是最常见的2线栅栏。

2线栅栏的置换技术

- THE LONGEST DAY MUST HAVE AN END
- 去除空格后，两两组成一组，得
- TH EL ON GE ST DA YM
US TH AV EA NE ND
- 取每组的第一个字母，得
- TEOGSDYUTAENN
- 再都取第二个字母，得
- HLNETAMSHVAED
- 连在一起就是最终的密文：
- TEOGSDYUTAENNHLENETAMSHVAED
- 而解密的方式则是进行一次逆运算。先将密文分为两行：
- TEOGSDYUTAENN
- HLNETAMSHVAED
- 再按列读出，组合成一句话：
- THE LONGEST DAY MUST HAVE AN END
- 单纯的置换技术加密得到的密文中，有着与原始明文相同的字母频率特征，因而较容易被识破。而且，双字母音节和三字母音节分析办法更是破译这种密码的有力工具。

2. 代换(**Substitution**)技术

- 代换技术在现代密码学中也得到了广泛应用。所谓代换，是将明文字母用其他字母、数字或符号替换的一种方法。
- 为此，要建立一个或多个替换表，加密时将需要加密的明文字母依次通过查表，替换为相应的字符。
- 明文字符被逐个替换后，生成无意义的字符串，即密文。解密则查同样的表，执行反方向的替换。因此，这些替换表就是密钥。
- 比较著名的有Caesar密码、单表代换密码和多表代换密码。

1) Caesar密码

- 第一次有史料记载的密码是由Julius Caesar发明的Caesar密码，加密技术就是代换。Caesar密码的明文空间和密文空间都是26个英文字母的集合，加密算法非常简单，就是对每个字母用它之后的第3个字母来代换。
- 在Caesar密码中，密钥就是如下所示的替换表。
 - 明文：abcdefghijklmnopqrstuvwxyz
 - 密文：DEFGHIJKLMNOPQRSTUVWXYZABC
- 如果为每一个字母分配一个数值(a分配0, b分配1, 以此类推, z分配25)。令P代表明文，C代表密文，则Caesar算法能用如下的公式表示：

$$C=E(3, P)=(P+3) \bmod 26$$

- 一般的Caesar密码：

$$C=E(k, P)=(P+k) \bmod 26$$

$$P=D(k, C)=(C-k) \bmod 26$$

2)单表代换密码

- **单表代换密码的定义：**对明文的所有字母采用同一个代换表进行加密，每个明文字母映射到一个固定的密文字母，称为单表代换密码。
- **密钥短语密码：**选一个英文短语作为密钥字(keyword)或密钥短语(keyphrase)，如HAPPYNEWYEAR，去掉重复字母得HAPYNEWR。将它依次写在明文字母表之下，而后再将字母表中未在短语中出现过的字母依次写于此短语之后，就可构造出一个字母代换表，即明文字母表到密文字母表的映射规则，如表2-2所示。

表2-2 明文字母表到密文字母表的映射规则

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
H	A	P	Y	N	E	W	R	B	C	D	F	G	I	J	K	L	M	O	Q	S	T	U	V	X	Z

2)单表代换密码

- 若明文为

Caesar cipher is a shift substitution

- 则密文为

PHONHM PBKRNM BO H ORBEQ
OSAOQBQSQBJI

- 然而，因为明文都是英文字符，而**英语语言具有很强的统计规律**，如字母的使用频率分布、双字母组合频率分布等，对单表密码攻击也是比较容易的。
- 比如，首先把密文中字母使用的相对频率统计出来，然后与英文字母的使用频率分布进行比较。如果已知消息足够长的话，只用这种方法就已经足够了。

3)多表代换加密

- 为了进一步提高安全性，一种对策是对每个明文字母提供多种代换，即对明文消息采用多个不同的单表代换。这种方法一般称之为多表代换密码。
- Hill密码是一种著名的多表代换密码，运用了矩阵论中线性变换的原理，由Lester S. Hill在1929年发明。
- 每个字母指定为一个26进制数字：a=0，b=1，c=2，...，z=25。m个连续的明文字母被看做m维向量，与一个m*m的加密矩阵相乘，再将得出的结果模26，得到m个密文字母。即m个连续的明文字母作为一个单元，被转换成等长的密文单元。注意加密矩阵（即密匙）必须是可逆的，否则就不可能译码。

$$C = E(K, P) = KP \bmod 26$$

$$P = D(K, C) = K^{-1}C \bmod 26$$

m=4的多表代换加密的一个例子

$$\mathbf{K} = \begin{pmatrix} 1 & 3 & 5 & 7 \\ 10 & 4 & 6 & 8 \\ 2 & 3 & 6 & 9 \\ 11 & 12 & 8 & 5 \end{pmatrix}$$

对明文“cost”，用向量表示为 $[2 \ 14 \ 18 \ 19]^T$ (T代表矩阵转置)

则经过加密运算，得到密文

$$\mathbf{C} = \mathbf{K}[2 \ 14 \ 18 \ 19]^T = [3 \ 10 \ 9 \ 23]^T = dkjx$$

解密则需要用到矩阵 \mathbf{K} 的逆， \mathbf{K}^{-1} 由等式 $\mathbf{K}\mathbf{K}^{-1}=\mathbf{K}^{-1}\mathbf{K}=\mathbf{I}$ 定义，其中 \mathbf{I} 是单位矩阵。

$$\mathbf{P} = \mathbf{D}(\mathbf{K}, \mathbf{C}) = \mathbf{K}^{-1}\mathbf{C} \bmod 26$$

2.2.2 分组密码

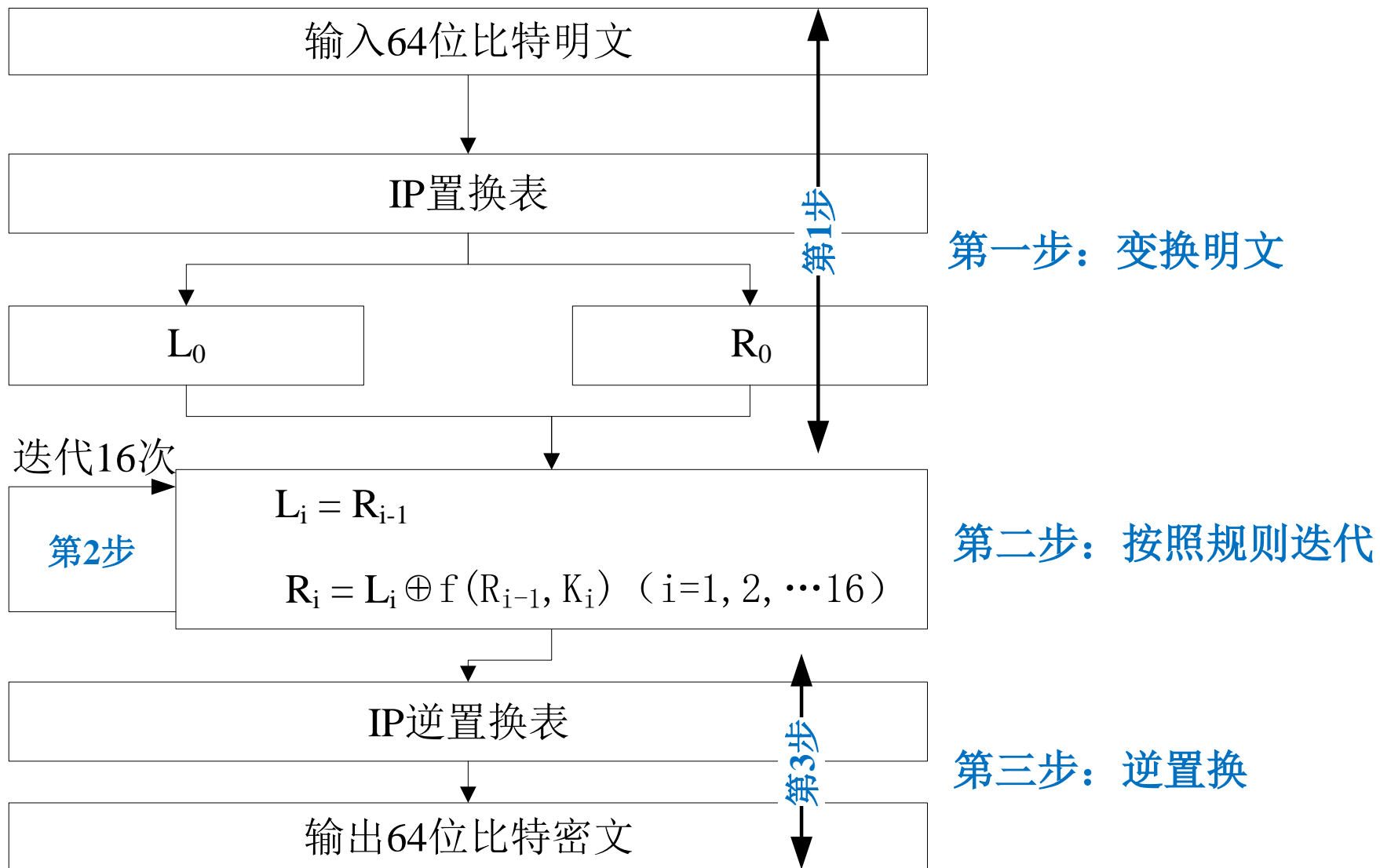
- 现代密码学阶段的标志事件：1949年，信息论的奠基人C. Shannon发表了一篇著名的文章《保密系统的通信理论》，为密码学的发展奠定了理论基础，使密码学成为一门真正的学科。
- 对称密码分为两大类：**流密码和分组密码**
 - ① **流密码(stream cipher)**又称作**序列密码**，加密和解密每次只处理数据流的一个符号（如一个字符或一个比特）。**古典密码都属于流密码。**
 - ② **分组密码(block cipher)**又称**块密码**，它将明文消息划分成若干长度为 $m(m>1)$ 的分组（或块），各组分别在长度为 r 的密钥 K 的控制下转换成长度为 n 的密文分组。

分组密码

- 如果 $m > n$ ，则称为**带数据压缩的分组密码**，可以增加密文解密的难度；如果 $m < n$ ，则称为**带数据扩展的分组密码**，其密文存储和传输的代价较大。
- 一般情况下，分组密码算法取 $m = n$ ，典型的大小是64位或者128位。当密钥长度为 r ，密钥空间大小是 2^r 。
- 由于加解密速度快，安全性能好，并得到许多密码芯片的支持，现代分组密码发展非常快。
- 一般来说，分组密码的应用范围比流密码要广泛。**绝大部分基于网络的对称密码应用使用的是分组密码。**
- 常见的分组算法有DES、DES3、IDEA、AES等。

数据加密标准(Data Encryption Standard, **DES**)

- DES是**曾经使用最广泛的密码系统**，属于分组密码体制。1973年美国国家标准局(现在的美国国家标准与技术研究所，NIST)征求国家密码标准方案，IBM将一个研究项目的成果提交，并于1977年被采纳为DES。
- DES采用了**S-P (Substitution - Permutation)** 网络结构，**分组长度为64位，密钥长度为56**。加密和解密使用同一算法、同一密钥、同一结构。区别是加密和解密过程中16个子密钥的应用顺序相反。
- DES加密运算的整体逻辑结构如图2-2所示。对于任意加密方案，共有两个输入：明文和密钥。DES的明文长为64位，密钥长为56位。实际中的明文分组未必为64位，此时要经过填充过程，使得所有分组对齐为64位，解密过程则需要去除填充信息。



DES算法的整体结构

DES算法的三个步骤

- **第一步：**变换明文

- 对给定的64位比特的明文 x ，首先通过一个置换IP表来重新排列 x ，从而构造出64位比特的 x_0 ， $x_0 = IP(x) = L_0 R_0$ ，其中 L_0 表示 x_0 的前32比特， R_0 表示 x_0 的后32位。

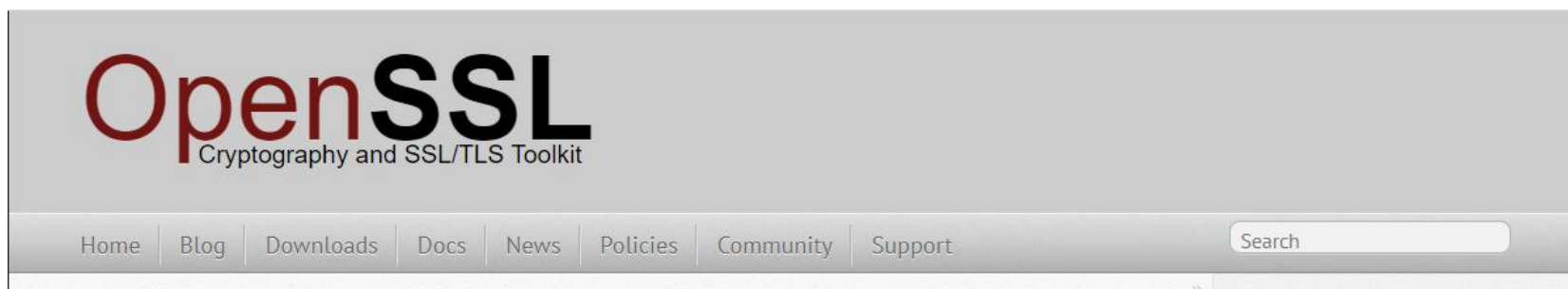
- **第二步：**按照规则迭代

- $L_i = R_{i-1}$
- $R_i = L_i \oplus f(R_{i-1}, K_i)$ ($i=1, 2, 3 \dots 16$)
- 经过第一步变换已经得到 L_0 和 R_0 的值，其中符号 \oplus 表示的数学运算是异或， f 表示一种置换，由S盒置换构成， K_i 是一些由密钥编排函数产生的比特块。
- f 和 K_i 见教材的相关说明

- **第三步：**对 $L_{16} R_{16}$ 利用 IP^{-1} 作逆置换，就得到了密文 y 。

DES算法及其实现的C源代码

- DES算法的详细内容请阅读教材相关内容，其具体实现的源代码请参考OpenSSL源代码：
- <https://www.openssl.org/> <https://github.com/openssl/openssl>



Note: The latest stable version(2019-Sep-10) is the **1.1.1** series. This is also our **Long Term Support (LTS)** version, supported until 11th September 2023. All other versions (including 1.1.0, 1.0.2, 1.0.0 and 0.9.8) are now out of support and should not be used. Users of these older versions are encourage to upgrade to 1.1.1 as soon as possible.

openssl-1.1.1j的源代码 (2021-Feb-16)

文件 | 主页 | 共享 | 查看

← → ↕ ↑ 此电脑 > S4TA (E:) > ido > infosec > openssl-1.1.1j > crypto > des

	名称	修改日期	类型
openssl-1.1.1j	asm	2021/2/16 23:24	文件夹
apps	build.info	2021/2/16 23:24	INFO 文件
Configurations	cbc_cksm	2021/2/16 23:24	C 源文件
crypto	cbc_enc	2021/2/16 23:24	C 源文件
aes	cfb_enc	2021/2/16 23:24	C 源文件
aria	cfb64ede	2021/2/16 23:24	C 源文件
asn1	cfb64enc	2021/2/16 23:24	C 源文件
async	des_enc	2021/2/16 23:24	C 源文件
bf	des_local.h	2021/2/16 23:24	C/C++ Header
bio	ecb_enc	2021/2/16 23:24	C 源文件
blake2	ecb3_enc	2021/2/16 23:24	C 源文件
bn	fcrypt	2021/2/16 23:24	C 源文件
buffer	fcrypt_b	2021/2/16 23:24	C 源文件
camellia	ncbc_enc	2021/2/16 23:24	C 源文件
cast	ofb_enc	2021/2/16 23:24	C 源文件
chacha	ofb64ede	2021/2/16 23:24	C 源文件
	ofb64enc	2021/2/16 23:24	C 源文件

DES的安全性——已经不安全

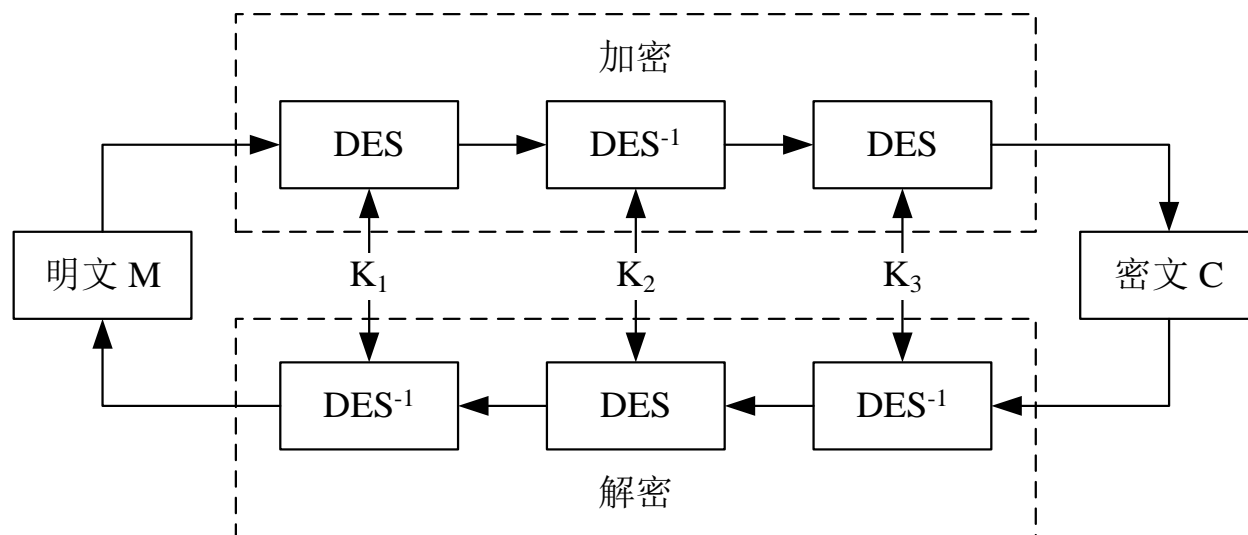
- 争论的焦点主要集中于**密钥的长度**和**算法本身的安全性**
- **DES受到的最大攻击是它的密钥长度仅有56比特。**
 - 56位的密钥共有 2^{56} 种可能，这个数字大约为 7.2×10^{16} 。
 - 在1977年，人们估计耗资两千万美元可以建成一个专门计算机用于DES的解密，需要12个小时的破解才能得到结果。所以，当时DES被认为是一种十分强壮的加密方法。
 - **1998年7月**，EFF(Electronic Frontier Foundation)宣布一台造价不到25万美元，为特殊目的设计的机器“DES破译机”**不到三天时间就成功破译了DES**，DES终于清楚地被证明是不安全的。EFF还公布了这台机器的细节，使其他人也能建造自己的破译机。
- **2000年1月**，在“第三届DES挑战赛”上，EFF研制的**DES解密机以22.5小时的战绩，成功地破解了DES加密算法**。随着硬件速度的提高和造价的下降，以及大规模网络并行计算技术的发展，破解DES的效率会越来越高。

DES的各种变种

- 由于DES的密钥长度仅为56比特，破解密文需要 2^{56} 次穷举搜索，在目前已难于保证密文的安全。

为了解决DES密钥长度过短的问题，可以采用组合密码技术，也就是将密码算法组合起来使用。**三重DES(简称为DES3或3DES)**是最常用的组合密码技术，**破解密文需要 2^{112} 次穷举搜索**，其算法如图所示。

- DES的其它变形算法还有DESX、CRYPT(3)、GDES、RDES、更换S盒的DES、使用相关密钥S盒的DES等。



三重DES

高级加密标准(AES)

- DES在出现之后的20多年间，在数据加密方面发挥了不可替代的作用。在进入20世纪90年代后，随着软硬件技术的发展，由于密钥长度偏短等缺陷，DES安全性受到严重挑战，并不断传出被破译的进展情况。
- 鉴于此，NIST决定于1998年12月后不再使用DES保护官方机密，只推荐为一般商业应用，并于**2001年11月发布了高级加密标准(AES)**，以替代DES。
- AES算法在提交的时候称为Rijndael。2001年2月28日，联邦公报发表了AES标准，从此开始了其标准化进程，并于2001年11月26日成为FIPS PUB 197标准。
- 目前，**建议所有的分组加密采用AES。**

2.2.3 序列密码（流密码）

- 序列密码的加密和解密每次只处理数据流的一个符号（如一个字符或一个比特）。
- 序列密码涉及大量的理论知识，提出了众多的设计原理，也得到了广泛的分析，但许多研究成果并没有完全公开，这也许是因为序列密码**目前主要应用于军事和外交等机密部门**的缘故。目前，公开的序列密码算法主要有RC4、SEAL等。
- RC4是Ron Rivest在RSA公司设计的一种可变密钥长度的、面向字节操作的流密码。RC4可能是应用最广泛的流密码。它被用于SSL/TLS（安全套接字协议 / 传输层安全协议）标准，以保护互联网的Web通信。它也作为IEEE802.11 无线局域网标准一部分的 WEP(Wired Equivalent Privacy)协议，保护无线连接的安全。

RC4流密码算法

- RC4算法非常简单，易于描述。它以一个足够大的表S为基础，对表进行非线性变换，产生密钥流。
- 一般S表取作256字节大小，用可变长度的**种子密钥K(1~256个字节)**初始化表S，S的元素记为S[0], S[1], ..., S[255]。
- 加密和解密的时候，密钥流中的一个字节由S中256个元素按一定方式选出一个元素而生成，同时S中的元素被重新置换一次。
- **这个被选出的元素与明文(或密文)异或，实现加密(或解密)**


1. 初始化S

- **(1) 对S进行线性填充。** S中元素的值被置为按升序从0到255，即 $S[0]=0$ ， $S[1]=1$ ，...， $S[255]=255$ 。
- **(2) 用种子密钥填充另一个256字节长的K表。** 如果种子密钥的长度为256字节，则将种子密钥赋给K；否则，若密钥长度为 $n(n<256)$ 字节，则将种子密钥的值赋给K的前 n 个元素，并循环重复用种子密钥的值赋给K剩下的元素，直到K的所有元素都被赋值。
- **(3) 用K产生S的初始置换。** 从 $S[0]$ 到 $S[255]$ ，对每个 $S[i]$ 根据由 $K[i]$ 确定的方案，将 $S[i]$ 置换为S中的另一字节。
 $j = 0$;
for $i = 0$ to 255 do
 $j = (j+S[i]+K[i]) \bmod 256$;
Swap($S[i], S[j]$);
- 因为对S的操作仅是交换，所以唯一的改变就是置换。S仍然包含所有值为0到255的元素。

2. 生成密钥流

- 表S一旦完成初始化，种子密钥就不再被使用。
- 为密钥流生成字节的时候，从S[0]到S[255]随机选取元素，并修改S以便下一次的选取。
- 对每个S[i]，根据当前S的值，将S[i]与S中的另一字节置换。
- 当S[255]完成置换后，操作继续重复，从S[0]开始。

```
i,j = 0;  
while (true)  
    i = (i+1) mod 256;  
    j = (j+S[i]) mod 256;  
    Swap(S[i],S[j]);  
    t = (S[i]+S[j]) mod  
    256;  
    k = S[t];
```



- 在while循环中的每个k都是密钥流中的一个字节，用于加密或解密。加密中，将k的值与下一明文字节异或；解密中，将k的值与下一密文字节异或。

2.3 公钥密码

- 1976年*Diffie*和*Hellman*发表了《密码学的新方向》一文，提出了公开密钥密码体制（简称公钥密码体制）的思想，奠定了公钥密码学的基础。
- 在传统的对称密码体制中，加密和解密使用相同的密钥，每对用户之间都需要共享一个密钥，而且需要保持该密钥的机密性。当通信的用户数目比较多的时候，密钥的产生、存储和分发是一个很大的问题。
- 而公钥密码体制则将加密密钥、解密密钥甚至加密算法、解密算法分开，用户只需掌握解密密钥，而将加密密钥和加密函数公开。任何人都可以加密，但只有掌握解密密钥的用户才能解密。

公钥密码体制给密钥管理带来了诸多便利

- 公钥密码体制从根本上改变了密钥分发的方式，给密钥管理带来了诸多便利。
- **公钥密码体制**不仅用于加解密，而且可以**广泛用于消息鉴别、数字签名和身份认证等服务**，是密码学中一个开创性的成就。
- 公钥密码体制并不会取代对称密码体制，原因在于公钥密码体制算法相对复杂，加解密速度较慢。
- 实际应用中，公钥密码和对称密码经常结合起来使用，**对数据的加解密使用对称密码技术，而密钥管理使用公钥密码技术**。

2.3.1 公钥密码体制原理

- **传统密码体制是基于替换和置换**这些初等方法。公钥密码学与其之前的密码学完全不同。
 - ① **公钥算法建立在数学函数基础上**，而不是基于替换和置换。其**安全性基于数学上难解的问题**，如大整数因子分解问题、有限域的离散对数问题、平方剩余问题、椭圆曲线的离散对数问题等；
 - ② 与只使用一个密钥的传统密码技术不同，公钥密码是非对称的，加 / 解密分别使用两个独立的密钥：加密密钥可对外界公开，称为公开密钥或公钥；解密密钥只有所有者知道，称为秘密密钥或私钥。公钥和私钥之间具有紧密联系，用公钥加密的信息只能用相应的私钥解密，反之亦然。要想**由一个密钥推知另一个密钥，在计算上是不可能的**。
 - ③ 基于公钥密码体制，**通信双方无需预先商定密钥就可以进行秘密通信**，克服了对称密码体制中必须事先使用一个安全通道约定密钥的缺点。

公钥密码算法的重要特点

1. 加密 / 解密使用不同的密钥;
2. 发送方拥有加密密钥或解密密钥, 而接收方拥有另一个密钥;
3. 根据密码算法和加密密钥以及若干密文, 要恢复明文在计算上是不可行的;
4. 根据密码算法和加密密钥, 确定对应的解密密钥在计算上是不可行的。

Diffie和Hellman给出的 公钥密码体制应满足的5个基本条件

(1)产生一对密钥(公钥PU, 私钥PR)在计算上是容易的。

(2)已知接收方B的公钥 PU_B 和要加密的消息M, 消息发送方A产生相应的密文在计算上是容易的:

$$C=E(PU_B, M)$$

(3)消息接收方B使用其私钥对接收的密文解密以恢复明文在计算上是容易的:

$$M=D(PR_B, C)=D[PR_B, E(PU_B, M)]$$

(4)已知公钥 PU_B 时, 攻击者要确定对应的私钥 PR_B 在计算上是不可行的。

(5)已知公钥 PU_B 和密文C, 攻击者要恢复明文M在计算上是不可行的。

可用于数字签名的公钥密码必须满足的条件

- 如果加密和解密函数的顺序可以交换：

$$M = D[PU_B, E(PR_B, M)] = D[PR_B, E(PU_B, M)]$$

- 则该公钥密码算法可用于数字签名。
 - 著名的RSA密码满足上述条件，可用于数字签名。
-
- 公钥密码体制是建立在数学中的**单向陷门函数**的基础之上的。单向函数是满足下列性质的函数：
 - ✓ 每个函数值都存在唯一的逆；对定义域中的任意 x ，计算函数值 $f(x)$ 是非常容易的；但对 f 的值域中的所有 y ，计算 $f^{-1}(y)$ 在计算上是不可行的，即求逆是不可行的。

单向陷门函数

- 一个单向函数，如果给定某些辅助信息（称为陷门信息），就易于求逆，则称这样的单向函数为一个单向陷门函数。即单向陷门函数是满足下列条件的一类可逆函数 f_k ：
 - ① 若 k 和 x 已知，则容易计算 $y=f_k(x)$ ；
 - ② 若 k 和 y 已知，则容易计算 $x=f_k^{-1}(y)$ ；
 - ③ 若 y 已知但 k 未知，则计算出 $x=f_k^{-1}(y)$ 是不可行的。
- 公钥密码体制就是基于这一原理，将**辅助信息(陷门信息)作为私钥**而设计的。这类密码的安全强度取决于它所依据的问题的计算复杂度。由此可见，寻找合适的单向陷门函数是公钥密码体制应用的关键。目前比较流行的公钥密码体制主要有两类：一类是**基于大整数因子分解问题**的，最典型的代表是RSA；另一类是**基于离散对数问题**的，比如ECC。

2.3.2 RSA算法

- MIT的Ron Rivest, Adi Shemir和Len Adleman于1978在题为《获得数字签名和公开密钥密码系统的方法》的论文中提出了基于数论的非对称密码体制，称为RSA密码体制。
- RSA算法是最早提出的满足要求的公钥算法之一，也是被广泛接受且被实现的通用公钥加密方法。
- RSA是一种**分组密码体制**，其理论基础是数论中“**大整数的素因子分解是困难问题**”的结论，即求两个大素数的乘积在计算机上时是容易实现的，但要将一个大整数分解成两个大素数之积则是困难的。
- RSA公钥密码体制安全、易实现，是目前广泛应用的一种公钥密码体制，既可以用于加密，又可以用于数字签名。

RSA算法描述

- 密钥计算方法:

- ① 选择两个大素数 p 和 q (典型值为1024位)
- ② 计算 $n=p \times q$ 和 $z=(p-1) \times (q-1)$
- ③ 选择一个与 z 互质的数, 令其为 d
- ④ 找到一个 e 使满足 $e \times d = 1 \pmod{z}$
- ⑤ 公开密钥为 (e, n) , 私有密钥为 (d, n)

- 加密方法:

- ① 将明文看成比特串, 将明文划分成 k 位的块 P 即可, 这里 k 是满足 $2^k < n$ 的最大整数。
- ② 对每个数据块 P , 计算 $C = P^e \pmod{n}$, C 即为 P 的密文。

- 解密方法:

- 对每个密文块 C , 计算 $P = C^d \pmod{n}$, P 即为明文。

RSA算法实例

- 密钥计算:

- ① 取 $p=5$, $q=13$

- ② 则有 $n=p \times q=65$, $z=(p-1) \times (q-1)=(5-1) \times (13-1)=48$

- ③ 11和48没有公因子, 可取 $d=11$

- ④ 求满足 $11 \times e \equiv 1 \pmod{z} = 1 \pmod{48}$ 的 e , 得到 $e=35$

- ⑤ 公钥为 $(e,n)=(35, 65)$, 私钥为 $(d,n)=(11, 65)$

- 加密:

- 若明文 $P=63$, 则密文 $C = P^e \pmod{65} = 63^{35} \pmod{65} = 32$ 。

- 解密:

- 计算 $P = C^d \pmod{n} = 32^{11} \pmod{65} = 63$, 恢复出原文。

rsa_demo.py: RSA算法实例的python实现

```
def rsa_key():
```

```
    p = 5
```

```
    q = 13
```

```
    n = p * q
```

```
    z = (p - 1) * (q - 1)
```

```
    d = 11
```

```
    # find e: e*d = 1 (mod z)
```

```
    for e in range(1, 1000):
```

```
        ed = (e * d)
```

```
        edmodz = ed % z
```

```
        if edmodz == 1:
```

```
            print('Find e. e is ', e)
```

```
            print('Public key is (e,n)=(%d, %d)' % (e, n))
```

```
            print('Secrete key is (d,n)=(%d, %d)' % (d, n))
```

```
            break
```

```
    return [p, q, e, d, n]
```

```
def rsa_encdec(p, a, n):
```

```
    return (p ** a) % n
```

```
if __name__ == '__main__':
```

```
    rsa_params = rsa_key()
```

```
    print('[p,q,e,d,n]=', rsa_params)
```

```
    e = rsa_params[2]
```

```
    d = rsa_params[3]
```

```
    n = rsa_params[4]
```

```
    p = 63
```

```
    p_enc = rsa_encdec(p, e, n)
```

```
    p_dec = rsa_encdec(p_enc, d, n)
```

```
    print('P= %d, encP=enc(P)=%d, dec(encP)=%d' % (p, p_enc, p_dec))
```

RSA的缺点主要有以下两点

- (1)产生密钥很麻烦，受到素数产生技术的限制，因而难以做到一次一密。
- (2)分组长度太大，为保证安全性， n 至少也要600比特以上，使运算代价很高，尤其是速度较慢，**相比对称密码算法慢几个数量级**；且随着大数分解技术的发展，这个长度还在增加。
- 因此，一般来说RSA只用于少量数据加密，比如用于对称密码体制密钥的加密。

2.4 散列函数和消息认证码

2.4.1 散列函数

- 散列函数又叫做散列算法，是一种将任意长度的消息映射到某一固定长度消息摘要（散列值或哈希值）的函数。消息摘要相当于是消息的“指纹”，用来防止对消息的非法篡改。散列函数也常常被称为Hash函数(哈希函数)。
- 令H代表一个散列函数，M代表一个任意长度的消息，则M的散列值H表示为

$$H=H(M)$$

- 且H(M)长度固定。假设H安全，发送方将散列值H附于消息M后发送；接收方通过重新计算散列值H'，并比较H=H'是否成立，可以验证该消息的完整性。

散列函数的安全性

- 密码学中的散列函数必须满足一定的安全特征，主要包括三个方面：单向性、强抗碰撞性和弱抗碰撞性。
- **A. 单向性**是指对任意给定的散列码 H ，找到满足 $H(x)=H$ 的 x 在计算上是不可行的，即给定散列函数 H ，由消息 M 计算散列值 $H(M)$ 是容易的，但是由散列值 $H(M)$ 计算 M 是不可行的。
- **B. 强抗碰撞性**是指散列函数满足下列四个条件。
 - (1) 散列函数 H 的输入是任意长度的消息 M ;
 - (2) 散列函数 H 的输出是固定长度的数值;
 - (3) 给定 H 和 M ，计算 $H(M)$ 是容易的;

散列函数的安全性

(4) 给定散列函数 H ，寻找两个不同的消息 M_1 和 M_2 ，使得 $H(M_1)=H(M_2)$ ，在计算上是不可行的。
(如果有两个消息 M_1 和 M_2 ， $M_1 \neq M_2$ 但是 $H(M_1)=H(M_2)$ ，则称 M_1 和 M_2 是碰撞的。)

- **C. 弱抗碰撞性**的散列函数满足强抗碰撞散列函数的前三个条件，但具有一个不同的条件：给定 H 和一个随机选择的消息 M ，寻找消息 M' ，使得 $H(M)=H(M')$ 在计算上是不可行的，即不能找到与给定消息具有相同散列值的另一消息。

MD5和SHA

- 目前使用最多的两种散列函数是MD5和SHA序列函数。
- MD5的散列码长度为128比特。Ubuntu Linux发行版本内置了MD5算法: `md5sum filename`
- SHA序列函数是美国联邦政府的标准，如SHA-1散列码长度为160比特，SHA-2散列码长度为256、384和512位。
- SHA-1算法见教材第30页的内容。

MD5的碰撞问题

- MD5的散列码长度为128比特，已经被证明是不安全的。2004年，山东大学的王小云（现为科学院院士，清华大学和山东大学的教授）第一次发现MD5算法存在碰撞的可能，并给出了实例。

Sequence #1

d1	31	dd	02	c5	e6	ee	c4	69	3d	9a	06	98	af	f9	5c
2f	ca	b5	87	12	46	7e	ab	40	04	58	3e	b8	fb	7f	89
55	ad	34	06	09	f4	b3	02	83	e4	88	83	25	71	41	5a
08	51	25	e8	f7	cd	c9	9f	d9	1d	bd	f2	80	37	3c	5b
d8	82	3e	31	56	34	8f	5b	ae	6d	ac	d4	36	c9	19	c6
dd	53	e2	b4	87	da	03	fd	02	39	63	06	d2	48	cd	a0
e9	9f	33	42	0f	57	7e	e8	ce	54	b6	70	80	a8	0d	1e
c6	98	21	bc	b6	a8	83	93	96	f9	65	2b	6f	f7	2a	70

Sequence #2

d1	31	dd	02	c5	e6	ee	c4	69	3d	9a	06	98	af	f9	5c
2f	ca	b5	07	12	46	7e	ab	40	04	58	3e	b8	fb	7f	89
55	ad	34	06	09	f4	b3	02	83	e4	88	83	25	f1	41	5a
08	51	25	e8	f7	cd	c9	9f	d9	1d	bd	72	80	37	3c	5b
d8	82	3e	31	56	34	8f	5b	ae	6d	ac	d4	36	c9	19	c6
dd	53	e2	34	87	da	03	fd	02	39	63	06	d2	48	cd	a0
e9	9f	33	42	0f	57	7e	e8	ce	54	b6	70	80	28	0d	1e
c6	98	21	bc	b6	a8	83	93	96	f9	65	ab	6f	f7	2a	70

Both produce MD5 digest 79054025255fb1a26e4bc422aef54eb4

SHA-1的碰撞问题

- On February 23, 2017, CWI Amsterdam and Google announced they had performed a collision attack against SHA-1. They had given 2 different PDF files with the same SHA-1 outputs.
- <https://shattered.io/>

Compared to other collision attacks



目前（2020年10月）：

- ① 对于安全要求较高的应用，不能使用**MD5**
- ② 找到**SHA-1**的碰撞是较困难的；**SHA-2**是安全的

2.4.2 消息鉴别码

- 完整性是安全的基本要求之一，是信息安全的三要素之一。恶意篡改、信道的偶发干扰和故障都将破坏消息的完整性。
- 另外，攻击者还可能实施假冒攻击，破坏信息的真实性。
- 保障消息完整性和真实性的重要手段是消息鉴别技术。

1. 消息鉴别的概念

- 消息鉴别也称为“报文鉴别”或“消息认证”，是一个对收到的消息进行完整性和真实性验证的过程。
- **过程：用鉴别函数产生一个鉴别符，根据收发端的鉴别符是否一致，对消息进行鉴别。**
- 根据鉴别符的生成方式，鉴别函数可以分为如下三类：
 - ① 基于消息加密：以整个消息的密文作为鉴别符。
 - ② 基于**消息鉴别码(MAC, Message identification code)**：利用公开函数和密钥产生一个较短的定长值作为鉴别符，并与消息一同发送给接收方，实现对消息的验证。
 - ③ 基于散列函数：利用公开的散列函数将任意长的消息映射为定长的散列值，并以该散列值作为鉴别符。

常用的鉴别方法：由Hash函数导出MAC

- 加密整个消息的鉴别方法是昂贵的，对于大量消息的鉴别是不合适的，特别是对于不需要保密的图像、声音、视频等媒体信息，对存储和传输都带来巨大的资源需求，如果加密整个消息，所需的计算资源是巨大的，在某些情况下甚至是不可行的。
- 实用的鉴别方法所需的计算和存储资源不应该太大。因此，**基于Hash函数导出MAC的方法成为了主流。**

2. 基于MAC的鉴别

1)消息鉴别码原理

- 消息鉴别码(message authentication code, MAC) 又称**密码校验和(cryptographic check-sum)**, 其实现鉴别的原理是: 利用公开函数和密钥生成一个固定大小的小数据块, 即MAC, 并将其附加在消息之后传输。接收方利用与发送方共享的密钥进行鉴别。
- 基于MAC提供消息完整性保护, MAC可以在不安全的信道中传输, 因为MAC的生成需要密钥。

MAC鉴别原理

关键：假定通信双方，比如A和B，共享密钥K

消息鉴别码 $MAC=C(K, M)$ ，其中：

M：输入消息；C：MAC函数；K：共享的密钥；

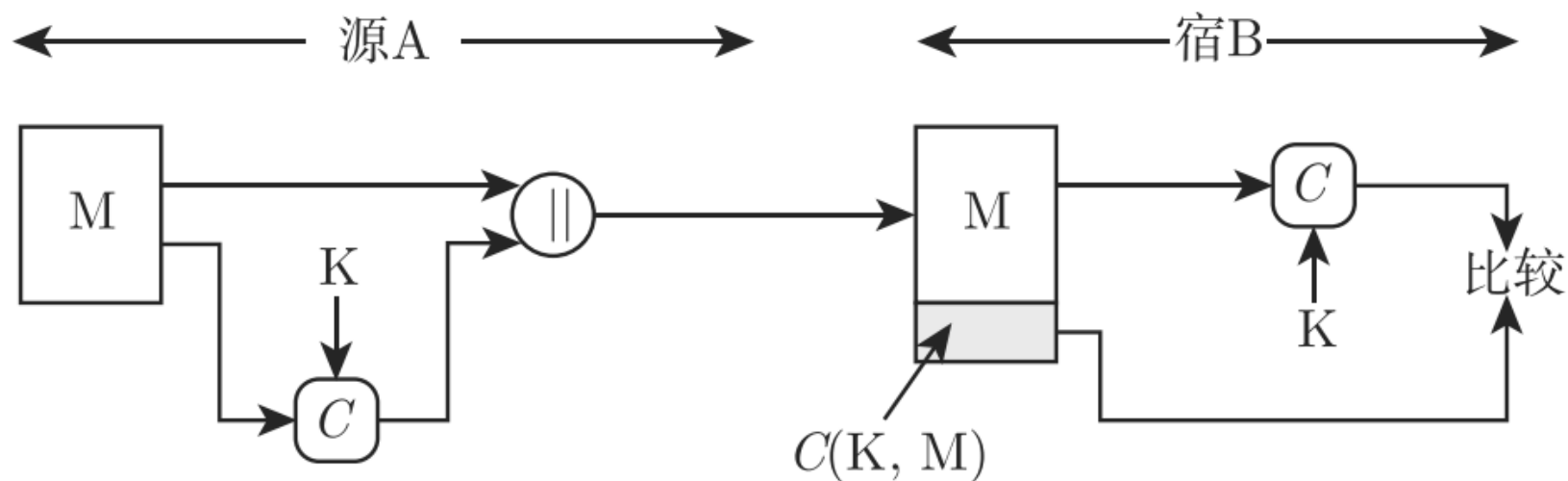


图2-7 MAC鉴别原理

MAC鉴别原理

- 图2-7所示的过程仅仅提供鉴别而不能提供保密性，因为消息是以明文形式传送的。若将MAC附加在明文消息后对整个信息块加密，则可以同时提供保密和鉴别。这需要两个独立的密钥，并且收发双方共享这两个密钥。
- MAC函数与加密类似，但加密算法必须是可逆的，而MAC算法则不要求可逆性，在数学上比加密算法被攻击的弱点要少。
- 与加密相比，MAC算法更不易被攻破。

2)基于DES的消息鉴别码

- 构造MAC的常用方法之一就是基于分组密码，并按CBC模式操作。
- 在CBC模式中，每个明文分组在用密钥加密之前，要先与前一个密文分组进行异或运算。用一个初始向量IV作为密文分组初始值。
- 数据鉴别算法，也称为**CBC-MAC(密文分组链接消息鉴别码)**，建立在DES之上，是使用最广泛的MAC算法之一，也是ANSI的一个标准。
- 数据鉴别算法采用DES运算的密文块链接(CBC)方式，参见图2-8。

采用DES运算的密文块链接(CBC) (略)

- 其初始向量IV为0，需要鉴别的数据分成连续的64位的分组 D_1, D_2, \dots, D_N ，若最后分组不足64位，则在其后填0直至成为64位的分组。
- 利用DES加密算法E和密钥K，计算数据鉴别码(DAC)的过程为：

$$O_0 = IV$$

$$O_1 = E_K(D_1 \oplus O_0)$$

$$O_2 = E_K(D_2 \oplus O_1)$$

$$O_3 = E_K(D_3 \oplus O_2)$$

\vdots

$$O_N = E_K(D_N \oplus O_{N-1})$$

基于DES的MAC (略)

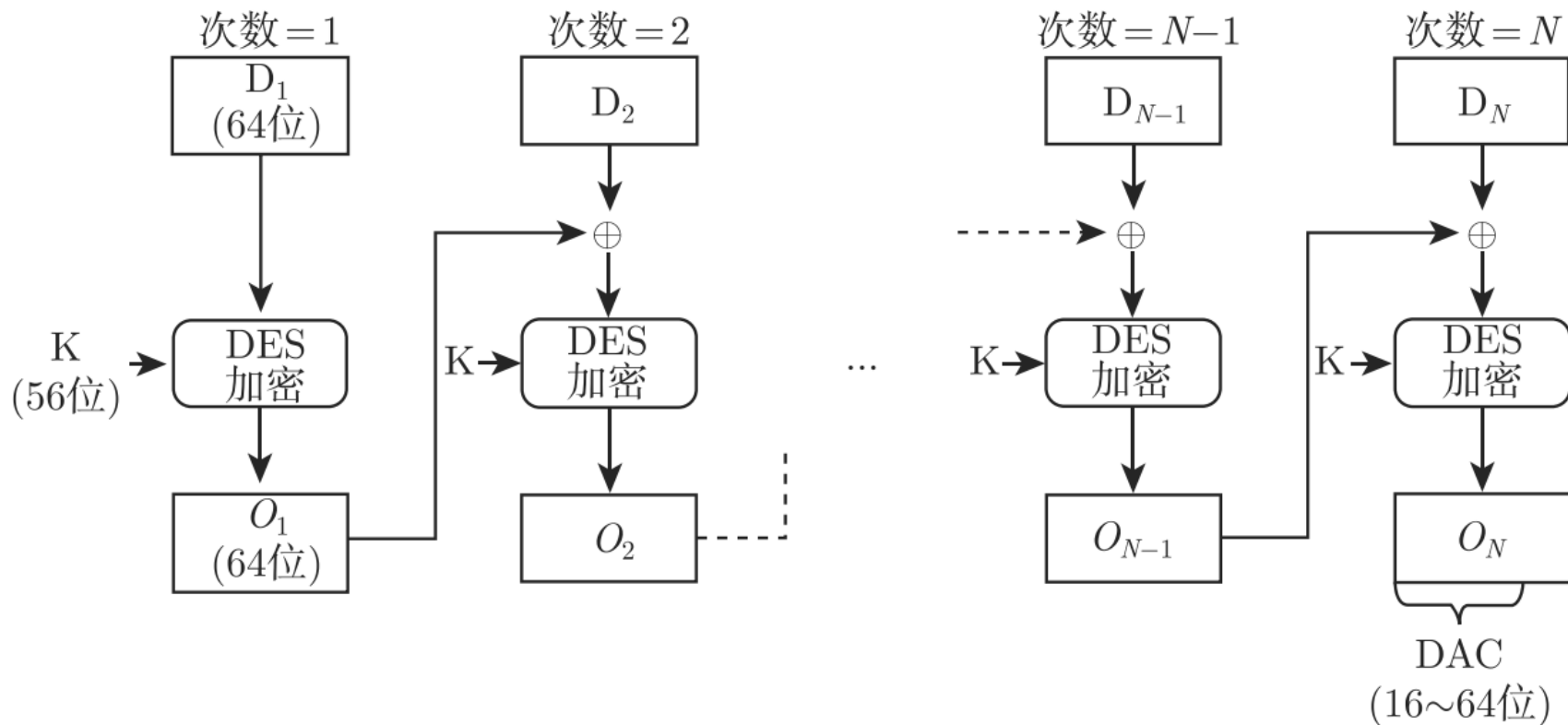


图2-8 基于DES的数据鉴别算法

- 其中，DAC可以取整个块 O_N ，也可以取其最左边的 M 位，其中 $16 \leq M \leq 64$ 。

3. 基于散列函数的鉴别

- 目前，已经提出了许多方案将密钥加到现有的散列函数中。HMAC是最受支持的方案，它是一种依赖于密钥的单向散列函数，同时提供对数据的完整性和真实性的验证。
- HMAC是IP安全里必须实现的MAC方案，并且其他Internet协议中(如SSL)也使用了HMAC。
- HMAC可描述如下：

$$\text{HMAC}(K, M) = H[(K^+ \oplus \text{opad}) \| H[(K^+ \oplus \text{ipad}) \| M]]$$

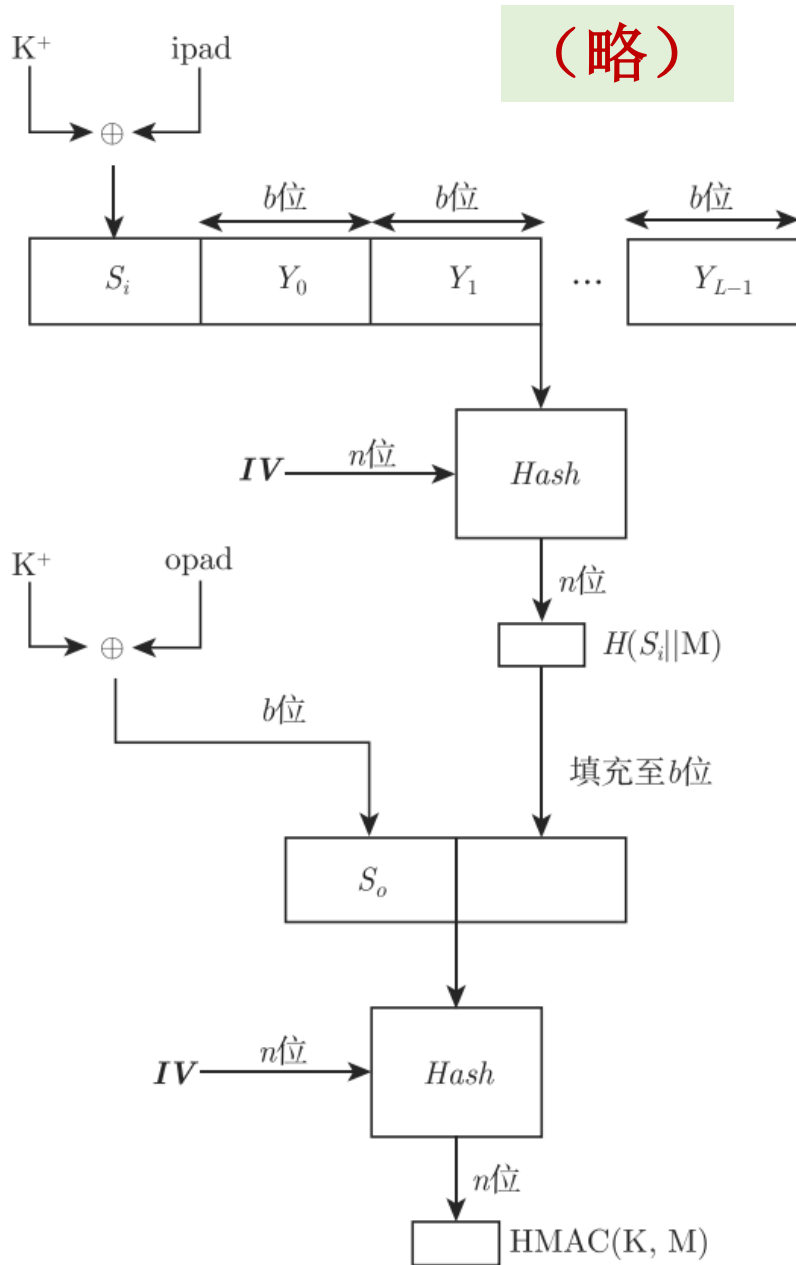


图2-9 HMAC的总体结构

H——嵌入的散列函数(如MD5, SHA-1, RIPEMD-160);

IV——作为散列函数输入的初始值;

M——HMAC的消息输入 (包括由嵌入散列函数定义的填充位);

L——M中的分组数;

Y_i ——M的第*i*个分组, $0 \leq i \leq L-1$;

b ——每一分组所含的位数;

n ——嵌入的散列函数所产生的散列码长;

K ——密钥, 建议密钥长度 $\geq n$ 。若密钥长度大于 b , 则将密钥作为散列函数的输入, 来产生一个 n 位的密钥;

K^+ ——为使 K 为 b 位长度而在 K 左边填充0后所得的结果;

$ipad$ ——内层填充, 00110110(十六进制数36)重复 $b/8$ 次的结果;

$opad$ ——外层填充, 01011100(十六进制数5C)重复 $b/8$ 次的结果。

HMAC的过程（略）

- (1)在K左边填充0，得到b位的 K^+ （例如，若K是160位， $b=512$ ，则在K中加入44个0字节）；
- (2) K^+ 与ipad执行异或运算（逐位异或）产生b位的分组 S_i ；
- (3)将M附于 S_i 后；
- (4)将H作用于步骤(3)所得出的结果；
- (5) K^+ 与opad执行异或运算（位异或）产生b位的分组 S_o ；
- (6)将步骤(4)中的散列码附于 S_o 后；
- (7)将H作用于步骤(6)所得出的结果，并输出该函数值。

2.5 数字签名

- 签名起到了鉴别、核准、负责等作用，表明签名者对文档内容的认可，并产生某种承诺或法律上的效力。
- 数字签名是手写签名的数字化形式，是公钥密码学发展过程中最重要的概念之一，也是现代密码学的一个最重要的组成部分之一。
- 数字签名已成为计算机网络不可缺少的一项安全技术，在商业、金融、军事等领域，得到了广泛的应用。各国对数字签名的使用颁布了相应的法案。
- 美国2000年通过的《电子签名全球与国内贸易法案》就规定数字签名与手写签名具有同等法律效力，我国的《电子签名法》也规定可靠的数字签名与手写签名或印章有同等法律效力。

2.5.1 数字签名简介

- 消息鉴别通过验证消息完整性和真实性，可以保护信息交换双方不受第三方的攻击，但是它不能处理通信双方内部的相互的攻击，这些攻击可以有多种形式。
- 数字签名是解决通信双方内部相互攻击的最好方法，它的作用相当于手写签名。用户A发送消息给B，B只要通过验证附在消息上的A的签名，就可以确认消息是否确实来自于A。同时，因为消息上有A的签名，A在事后也无法抵赖所发送过的消息。
- 因此，**数字签名的基本目的是认证、核准和负责，防止相互欺骗和抵赖**。数字签名在身份认证、数据完整性、不可否认性和匿名性等方面有着广泛的应用。

数字签名的概念及其特征

- 数字签名在ISO7498-2标准中定义为：
- “附加在数据单元上的一些数据，或是对数据单元所作的密码变换，这种数据和变换允许数据单元的接收者用以确认数据单元来源和数据单元的完整性，并保护数据，防止被人（例如接收者）进行伪造。”
- 数字签名体制也叫数字签名方案，一般包含两个主要组成部分，即签名算法和验证算法。
- 对消息M签名记为 $s = \text{Sig}(m)$ ，而对签名s的验证可记为 $\text{Ver}(s) \in \{0, 1\}$ 。

数字签名体制的形式化定义

- 定义3-1 一个数字签名体制是一个五元组 (M, A, K, S, V) ，其中：
 - ※ M 是所有可能的消息的集合，即消息空间。
 - ※ A 是所有可能的签名组成的一个有限集，称为签名空间。
 - ※ K 是所有密钥组成的集合，称为密钥空间。
 - ※ S 是签名算法的集合。
 - ※ V 是验证算法的集合。
- 满足：对任意 $k \in K$ ，有一个签名算法 Sig_k 和一个验证算法 Ver_k ，使得对任意消息 $m \in M$ ，每一签名 $a \in A$ ， $\text{Ver}_k(m, a)=1$ ，当且仅当 $a=\text{Sig}_k(m)$ 时。

数字签名必须具有的4个特征

- ① **可验证性**。信息接收方必须能够验证发送方的签名是否真实有效。
- ② **不可伪造性**。除了签名人之外，任何人不能伪造签名人的合法签名。
- ③ **不可否认性**。发送方在发送签名的消息后，无法抵赖发送的行为；接收方在收到消息后，也无法否认接收的行为。
- ④ **数据完整性**。数字签名使得发送方能够对消息的完整性进行校验。即数字签名具有消息鉴别的功能。

数字签名应满足的6个条件

- ① 签名必须是与消息相关的二进制位串。
- ② 签名必须使用发送方某些独有的信息，以防伪造和否认。
- ③ 产生数字签名比较容易。
- ④ 识别和验证签名比较容易。
- ⑤ 伪造数字签名在计算上是不可行的。无论是从给定的数字签名伪造消息，还是从给定的消息伪造数字签名，在计算上都是不可行的。
- ⑥ 保存数字签名的拷贝是可行的。

※基于公钥密码算法和对称密码算法都可以获得数字签名，目前主要是基于公钥密码算法的数字签名。

※在基于公钥密码的签名体制中，签名算法必须使用签名人的私钥，而验证算法则只使用签名人的公钥。因此，只有签名人才可能产生真实有效的签名，只要他的私钥是安全的。签名的有效性能被任何人验证，因为签名人的公钥是公开可访问的。

2.5.2 基于公钥密码的数字签名原理

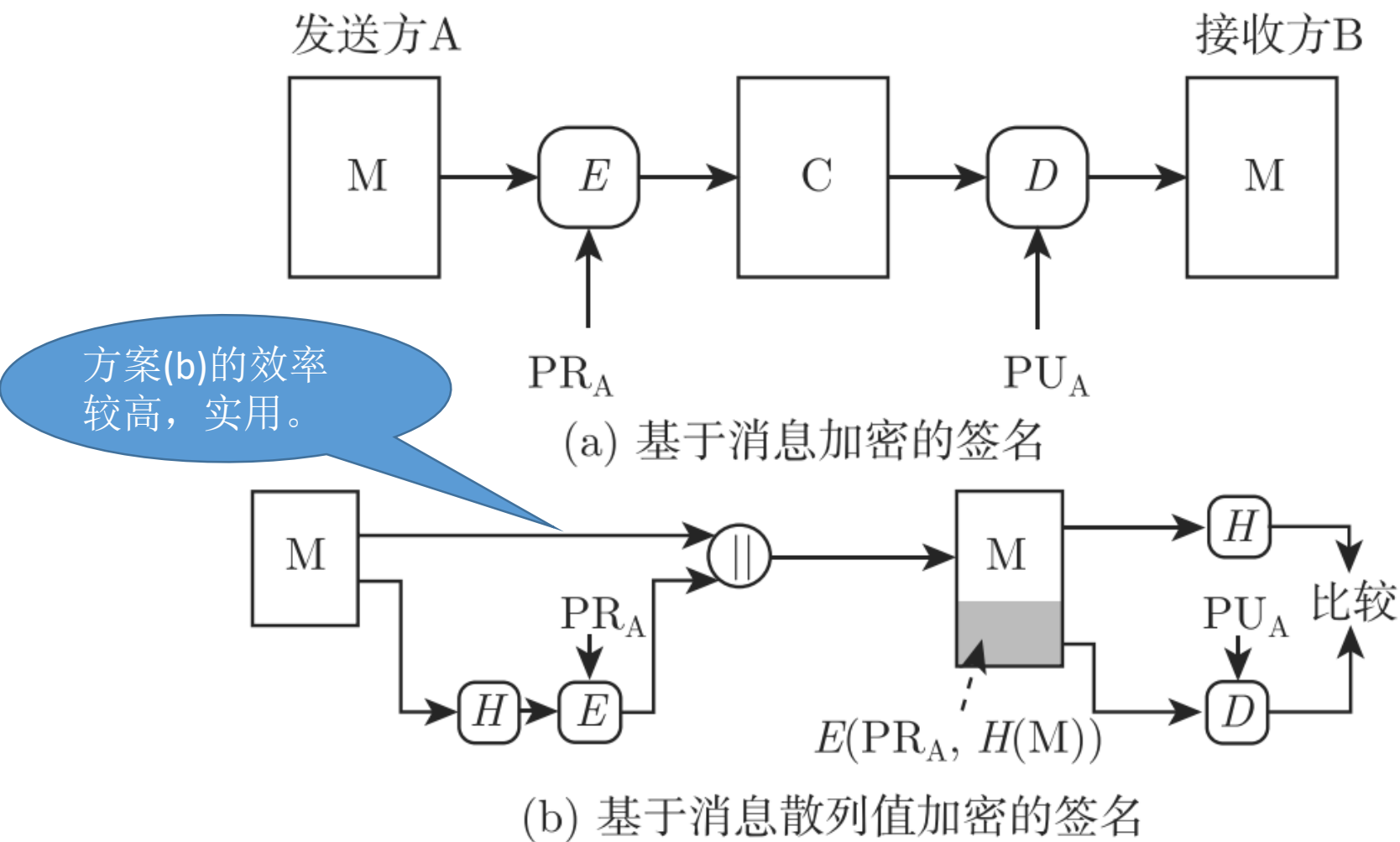


图2-10 基于公钥密码的数字签名原理

2.5.3 数字签名算法

1. 基于RSA的数字签名

- RSA数字签名方法要使用一个散列函数 H ，散列函数的输入是要签名的消息，输出是定长的散列码。

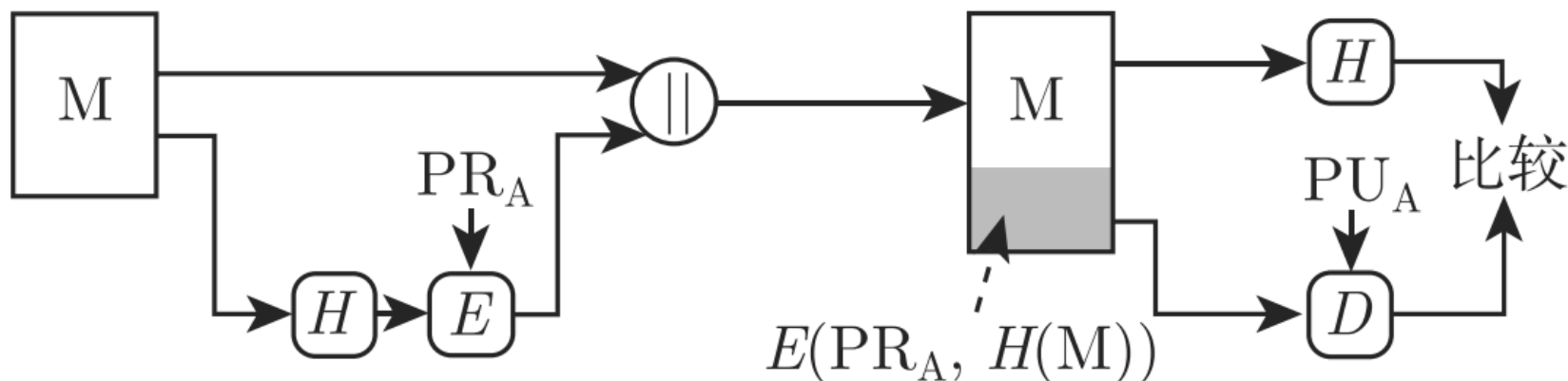


图2-11 RSA数字签名

- 发送方用其私钥和RSA算法对该散列码加密形成签名，然后发送消息及其签名。接收方收到消息后计算散列码，并用发送方的公钥对签名解密得到发送方计算的散列码，如果两个散列码相同，则认为签名是有效的。
- 因为只有发送方拥有私钥，因此只有发送方能够产生有效的签名。

2. 数字签名标准DSS

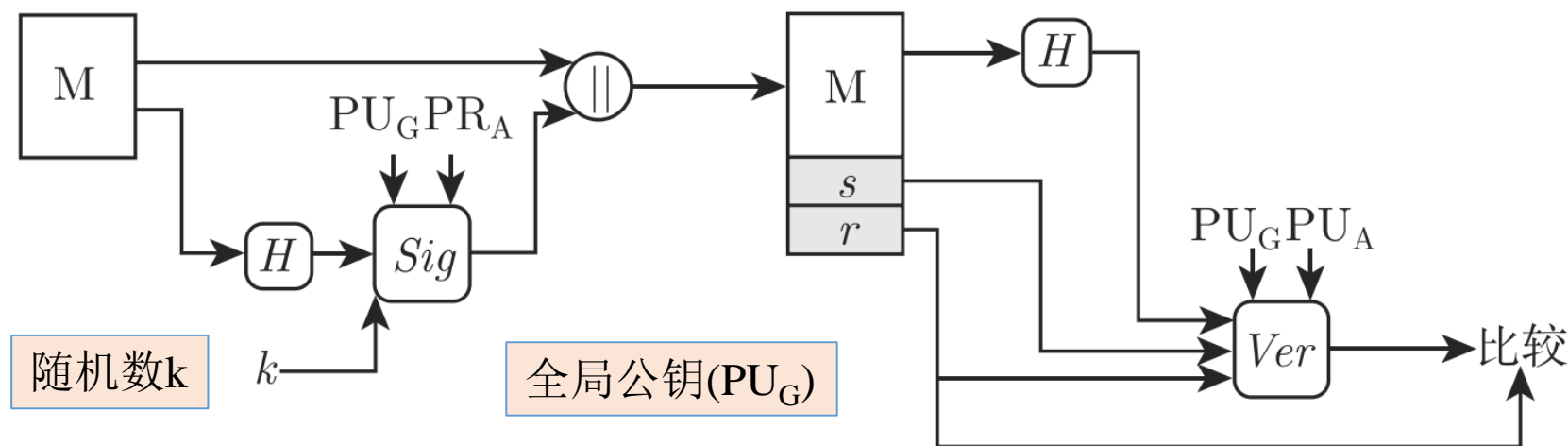


图2-12 DSS数字签名方法

- 美国国家标准与技术研究所(NIST)于1991年提出了一个联邦数字签名标准，称之为数字签名标准(DSS)。DSS使用安全散列算法(SHA)，给出了一种新的数字签名方法，即数字签名算法(DSA)。与RSA不同，DSS是一种公钥方法，但只提供数字签名功能，不能用于加密或密钥分配。
- DSS数字签名方法如图2-12所示。

DSS签名过程（略）

- DSS方法也使用散列函数，它产生的散列码和为此次签名而产生的随机数 k 作为签名函数的输入，签名函数依赖于发送方的私钥(PR_A)和一组参数，这些参数为一组通信伙伴所共有，我们可以认为这组参数构成全局公钥(PU_G)。
- 接收方对接收到的消息产生散列码，这个散列码和签名一起作为验证函数的输入，验证函数依赖于全局公钥和发送方公钥 PU_A ，若验证函数的输出等于签名中的 r 成分，则签名是有效的。签名函数保证只有拥有私钥的发送方才能产生有效签名。
- DSA 安全性基于计算离散对数的困难性，并起源于ElGamal和Schnorr提出的数字签名方法。

DSA算法（略）

全局公钥组成

p 为素数, 其中 $2^{L-1} < p < 2^L$, $512 \leq L \leq 1024$ 且 L 是64的倍数, 即 L 的位长为512~1024, 并且其增量为64位。
 q 为 $p-1$ 的素因子, 其中 $2^{159} < q < 2^{160}$, 即长为160位。
 $g = h^{(p-1)/q} \bmod p$, 其中 h 是满足 $1 < h < (p-1)$, 并且 $h^{(p-1)/q} \bmod p > 1$ 的任何整数

用户的私钥

x 为随机或伪随机整数且 $0 < x < q$

用户的公钥

$$y = g^x \bmod p$$

与用户每条消息相关的私密值

k 为随机或伪随机整数且 $0 < k < q$

签名

$$r = (g^x \bmod p) \bmod q$$

$$s = [k^{-1}(H(M) + xr)] \bmod q$$

$$\text{签名} = (r, s)$$

验证

$$w = (s')^{-1} \bmod q$$

$$u_1 = [(H(M')w)] \bmod q$$

$$u_2 = (r')w \bmod q$$

$$v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$$

$$\text{检验: } v = r'$$

M —— 要签名的消息

$H(M)$ —— 使用SHA-1求得 M 的三列码

M', r', s' —— 接收到的 M, r, s

图2-13 数字签名算法DSA

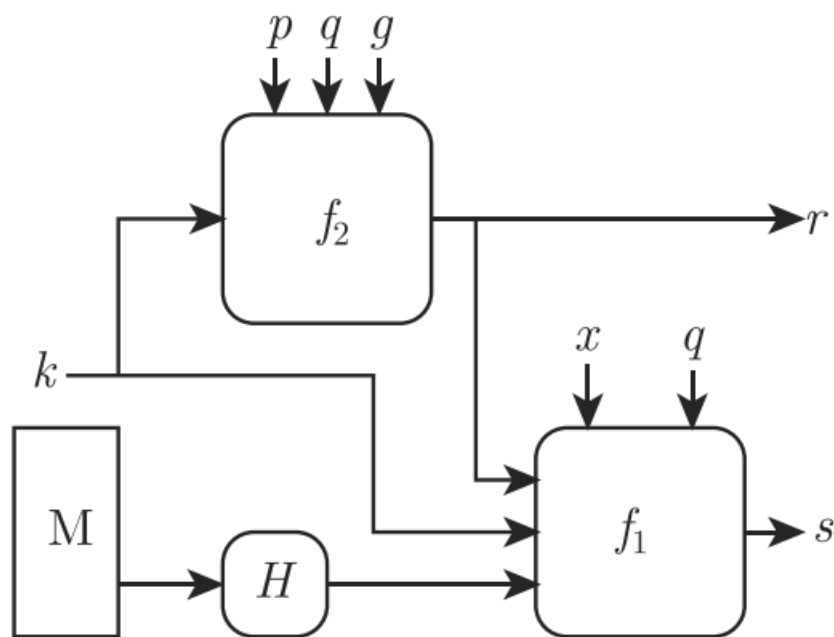
DSA算法的说明（略）

- 公钥由三个参数 p 、 q 、 g 组成，并为的一组用户所共有。首先选择一个160位的素数 q ；然后选择一个长度为512~1024的素数 p ，并且使得 q 是 $p-1$ 的素因子；最后选择形为 $h^{(p-1)/q} \bmod p$ 的 g ，其中 h 为1~ $p-1$ 的整数且 g 大于1。
- 选定这些参数后，每个用户选择私钥并产生公钥。私钥 x 必须是随机或伪随机选择的素数，取值区间是 $[1, q-1]$ 。公钥则根据公式 $y = g^x \bmod p$ 计算得到。由给定的 x 计算 y 比较简单，而由给定的 y 确定 x 则在计算上是不可行的，因为这就是求 y 的以 g 为底的模 p 的离散对数，而求离散对数是困难的。

DSA算法的说明（续）（略）

- 假设要对消息 M 进行签名。发送方需计算两个参数 r 和 s ，它是公钥 (p, q, g) 、用户私钥 (x) 、消息的散列码 $H(M)$ 和附加整数 k 的函数，其中 k 是随机或伪随机产生的， $0 < k < q$ ，且 k 对每次签名是唯一的。
- 为了对签名进行验证，接收方计算值 v ，它是公钥 (p, q, g) 、发送方公钥、接收到的消息的散列码的函数，若 v 与签名中的 r 相同，则签名是有效的。
- 图2-14描述了上述签名和验证函数。

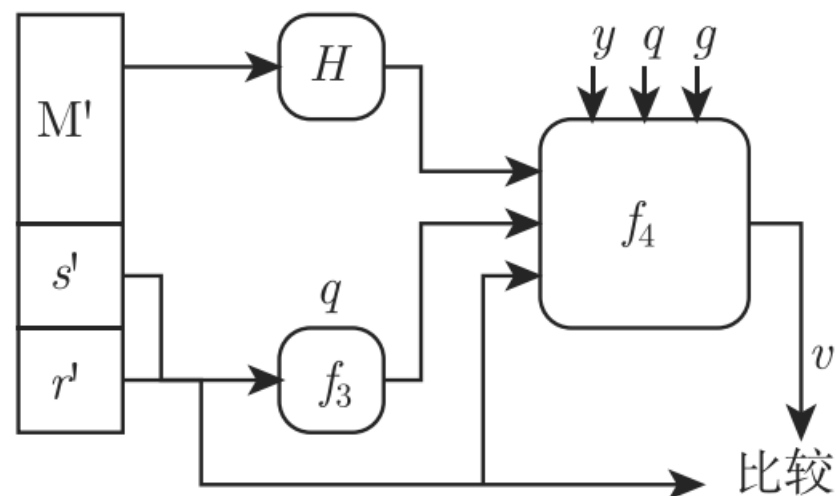
DSA算法的说明（续）（略）



$$s = f_1(H(M), k, x, r, q) = (k^{-1}(H(M) + xr)) \bmod q$$

$$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$$

(a) 签名



$$w = f_3(s', q) = (s')^{-1} \bmod q$$

$$v = f_4(y, q, g, H(M'), w, r') = ((g^{H(M')w} \bmod q) y^{r'w} \bmod q) \bmod p \bmod q$$

(b) 验证

图2-14 DSS签名和验证函数

DSA算法的说明（续）（略）

- DSA算法有这样一个特点，接收端的验证依赖于 r ，但是 r 却根本不依赖于消息，它是 k 和全局公钥的函数。 $k(\bmod q)$ p 的乘法逆元传给函数的输入还包含消息的散列码和用户私钥，函数的这种结构使接收方可利用其收到的消息、签名、它的公钥以及全局公钥来恢复 r 。
- 由于求离散对数的困难性，攻击者从 r 恢复出 k 或从 s 恢复出 x 都是不可行的。

2.6 密钥管理

- 在现代密码学研究中，加密算法和解密算法一般是公开的，**密码系统的安全性就完全取决于密钥的保密程度**。因此，密钥管理成为一个重要的问题。如果密钥**得不到**强有力的保护，即使算法再复杂，密码系统也是脆弱的。
- 密钥管理包括密钥产生、密钥存储、密钥更新、密钥分发、密钥验证、密钥使用和销毁等过程。
- **密钥管理的核心问题是：确保密钥从产生到使用全过程的安全可靠。**

密钥的类型

- 根据应用场合的不同，密钥可以分成以下几类。

- (1) **工作密钥**，也叫**基本密钥或初始密钥**。由用户选定或系统分配，使用期限一般较长，达数月甚至一年等。
- (2) **会话密钥**，即通信双方交换数据时使用的密钥。会话密钥一般由通信双方协商决定，也可由密钥分配中心分配。会话密钥大多是临时的、动态的，可以降低密钥的分配和存储的数目。
- (3) **密钥加密密钥**，主要用于对要传送的会话密钥进行加密，也叫做二级密钥。
- (4) **主机主密钥**，对应于层次化密钥管理结构中的最顶层，主要用于对加密密钥进行加密保护，一般保存于主节点，受到严格保护。

密钥管理的4种方法（不考核）

2.6.1 公钥分配

2.6.2 对称密码体制的密钥分配

2.6.3 公钥密码用于对称密码体制的密钥分配

2.6.4 Diffie-Hellman密钥交换

不考核
感兴趣的同学自学

2.7 使用OpenSSL中的密码函数

- OpenSSL(<http://www.openssl.org/>) 是使用非常广泛的SSL的开源实现，是用C语言实现的。由于其中实现了为SSL所用的各种加密算法，因此OpenSSL也是被广泛使用的加密函数库。
- 有两种方式使用OpenSSL的加/解密功能：其一是在命令行下运行OpenSSL，以适当的参数运行openssl命令，就可以实现加密和解密功能；另一种是在自己的应用程序中使用加密函数，这需要利用openssl提供的C语言接口，以函数调用的方式使用加密函数库。

2.7.1 在命令行下使用OpenSSL

- Windows和Linux环境下的OpenSSL有相同的命令行程序名 openssl。在命令行窗口下运行“openssl help”，可以列出OpenSSL支持的命令。
- OpenSSL的命令分成三类：标准命令、数字摘要命令和加密命令。

在ubuntu linux系统运行openssl help

```
i@spring: ~/work
File Edit View Search Terminal Help
i@spring:~/work$
i@spring:~/work$
i@spring:~/work$ openssl help
Standard commands
asn1parse          ca                  ciphers             cms
crl                 crt2pkcs7           dgst                 dhparam
dsa                 dsaparam            ec                   ecparam
enc                 engine              errstr               gendsa
genpkey             genrsa               help                 list
nseq                ocsf                passwd              pkcs12
pkcs7               pkcs8                pkey                 pkeyparam
pkeyutil            prime                rand                 rehash
req                 rsa                  rsautl               s_client
s_server            s_time              sess_id              smime
speed               spkac                srp                  storeutl
ts                  verify               version              x509

Message Digest commands (see the `dgst' command for more details)
blake2b512          blake2s256          gost                 md4
md5                  rmd160               sha1                 sha224
sha256               sha3-224              sha3-256             sha3-384
sha3-512              sha384                sha512               sha512-224
sha512-256           shake128               shake256              sm3

Cipher commands (see the `enc' command for more details)
aes-128-cbc          aes-128-ecb          aes-192-cbc          aes-192-ecb
aes-256-cbc          aes-256-ecb          aria-128-cbc          aria-128-cfb
aria-128-cfb1         aria-128-cfb8         aria-128-ctr          aria-128-ecb
aria-128-ofb          aria-192-cbc          aria-192-cfb          aria-192-cfb1
aria-192-cfb8         aria-192-ctr          aria-192-ecb          aria-192-ofb
aria-256-cbc          aria-256-cfb          aria-256-cfb1         aria-256-cfb8
```

命令类别	子命令
标准命令 Standard commands	asn1parse, ca, ciphers, cms, crl, crl2pkcs7, dgst, dh, dhparam, dsa, dsaparam, ec, ecparam, enc, engine, errstr, gendh, gendsa, genpkey, genrsa, nseq, ocsf, passwd, pkcs12, pkcs7, pkcs8, pkey, pkeyparam, pkeyutl, prime, rand, req, rsa, rsautl, s_client, s_server, s_time, sess_id, smime, speed, spkac, srp, ts, verify, version, x509
数字摘要命令 Message Digest commands	md4, md5, mdc2, rmd160, sha, sha1
加密命令 Cipher commands	aes-128-cbc, aes-128-ecb, aes-192-cbc, aes-192-ecb, aes-256-cbc, aes-256-ecb, base64, bf, bf-cbc, bf-cfb, bf-ecb, bf-ofb, camellia-128-cbc, camellia-128-ecb, camellia-192-cbc, camellia-192-ecb, camellia-256-cbc, camellia-256-ecb, cast, cast-cbc, cast5-cbc, cast5-cfb, cast5-ecb, cast5-ofb, des, des-cbc, des-cfb, des-ecb, des-edc, des-edc-cbc, des-edc-cfb, des-edc-ofb, des-edc3, des-edc3-cbc, des-edc3-cfb, des-edc3-ofb, des-ofb, des3, desx, idea, idea-cbc, idea-cfb, idea-ecb, idea-ofb, rc2, rc2-40-cbc, rc2-64-cbc, rc2-cbc, rc2-cfb, rc2-ecb, rc2-ofb, rc4, rc4-40, seed, seed-cbc, seed-cfb, seed-ecb, seed-ofb

openssl 命令名称 -help

- OpenSSL 支持的命令是相当丰富的。如果对某个命令的用法不是很清楚，可以用“openssl 命令名称 -help”查看该命令的说明。
- 例如，如果不了解“openssl passwd”的用法，可以在命令行下输入“openssl passwd -help”，运行结果(ubuntu Linux)如下：

```
i@spring:~/work$ openssl passwd -help
```

```
Usage: passwd [options]
```

```
Valid options are:
```

```
-help          Display this summary
```

```
.....
```

```
-stdin         Read passwords from stdin
```

```
-6             SHA512-based password algorithm
```

```
-5             SHA256-based password algorithm
```

```
-apr1          MD5-based password algorithm, Apache variant
```

```
-1             MD5-based password algorithm
```

```
-aixmd5        AIX MD5-based password algorithm
```

```
-crypt         Standard Unix password algorithm (default)
```

```
-writerand outfile Write random data to the specified file
```

常用的OpenSSL的命令

功能	命令及说明
版本和编译参数	显示版本和编译参数: openssl version -a
支持的子命令、密码算法	查看支持的子命: openssl help SSL密码组合列表: openssl ciphers
测试密码算法速度	测试所有算法速度: openssl speed 测试RSA速度: openssl speed rsa 测试DES速度: openssl speed des
RSA密钥操作	产生RSA密钥对: openssl genrsa -out 1.key 1024 取出RSA公钥: openssl rsa -in 1.key -pubout -out 1.pubkey
加密文件	加密文件: openssl enc -e -rc4 -in 1.key -out 1.key.enc 解密文件: openssl enc -d -rc4 -in 1.key.enc -out 1.key.dec
计算Hash值	计算文件的MD5值: openssl md5 < 1.key 或 openssl md5 1.key 计算文件的SHA1值: openssl sha1 < 1.key

实例

- [实例1]
- 密钥在文件key.txt中，用des3算法对文件test.data加密和解密，并验证其正确性。
 - 加密为test.3des :
openssl enc -e -des3 -in test.data -out test.3des -kfile key.txt
 - 解密test.3des为test.dddd :
openssl enc -d -des3 -in test.3des -out test.dddd -kfile key.txt
 - 验证test.dddd和原始文件test.data相同：
openssl md5 test.dddd test.data

2.7.2 在Windows的C程序中使用OpenSSL

- OpenSSL提供了C语言接口所需的头文件、库文件和动态链接库。为了使用该接口，必须安装面向软件开发人员的软件包(安装文件较大)，并将openssl的lib和include目录添加到lib和环境变量中。对于Visual Studio C++开发平台，最简单的方法是将openssl的lib和include目录拷贝到VC目录(默认安装在C:\Program Files\Microsoft Visual Studio 9.0\VC)中，这样就不需要额外设置环境变量。
- 为了使用OpenSSL库函数，在C程序中必须包含相应的头文件，链接的时候必须加入相关的库。

2.7.3 在Linux的C程序中使用OpenSSL

- Linux系统的发行版一般预装了命令行OpenSSL程序，没有安装openssl库。为了在C程序中使用OpenSSL，需要安装openssl库。
- 在ubuntu Linux 系统中运行以下命令安装openssl库：
`sudo apt-get install libssl-dev`
- 在fedora Linux 系统中切换到root，再运行以下命令安装openssl库：
`yum install openssl-devel.x86_64 或 yum install openssl-devel.i686`

AES实例

- 例程: `cryptoDemo.cpp`
// 测试AES算法的例子。

演示

```
cryptodemo.cpp:10:10: fatal error: aes.h: No  
such file or directory
```

```
#include "aes.h"
```

```
^~~~~~
```

```
compilation terminated.
```

```
i@spring:~/work$ whereis openssl
```

```
openssl: /usr/bin/openssl /usr/include/openssl  
/usr/share/man/man1/openssl.1 ssl.gz
```

```
i@spring:~/work$ gcc -o cryptoDemo  
cryptodemo.cpp -lcrypto
```

```
i@spring:~/work$ ./cryptoDemo
```

```
test success
```

```
The original string is:
```

```
This is a sample. I am a programmer.
```

```
The encrypted string is:
```

```
"/96&$C H
```

```
The decrypted string is:
```

```
This is a sample. I am a programmer.
```

```
i@spring:~/work$
```

2.8 python语言的密码模块

- 在pipy.org有2个可用的python密码模块
旧的pycrypto:

- Latest version Released: Oct 18, 2013
- <https://pypi.org/project/pycrypto/>

新的pycryptodome :

- Latest version Released: Feb 9, 2021
- <https://pypi.org/project/pycryptodome/>

- Pycryptodome的安装和使用

安装: `pip install pycryptodome`

使用AES: `from Crypto.Cipher import AES`

用python密码库的AES算法，实现数据的加密和解密

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(data)

file_out = open("encrypted.bin", "wb")
[ file_out.write(x) for x in (cipher.nonce, tag,
ciphertext) ]
file_out.close()
```

```
from Crypto.Cipher import AES

file_in = open("encrypted.bin", "rb")
nonce, tag, ciphertext = [ file_in.read(x) for x in
(16, 16, -1) ]

# let's assume that the key is somehow available
again

cipher = AES.new(key, AES.MODE_EAX,
nonce)

data = cipher.decrypt_and_verify(ciphertext, tag)
```

<https://www.pycryptodome.org/en/latest/src/examples.html>

演示: aes_demo.py

关于密码学的其他方面

国密算法

- 国密算法，即国家商用密码算法。参考：
- <https://baike.baidu.com/item/国密>
- <https://baike.baidu.com/item/国密安全芯片>
- https://blog.csdn.net/weixin_42915431/article/details/106506899

轻量级密码

- 为适应计算和存储资源受限的设备而设计的密码算法。参考：
- <https://wenku.baidu.com/view/741ef68253d380eb6294dd88d0d233d4b04e3f54.html>

同态加密

- 为保护数据隐私而设计的密码算法。参考：
- <https://baike.baidu.com/item/同态加密>

差分隐私

- 为保护数据隐私而设计的密码算法。参考：
- <https://baike.baidu.com/item/差分隐私>

谢谢！