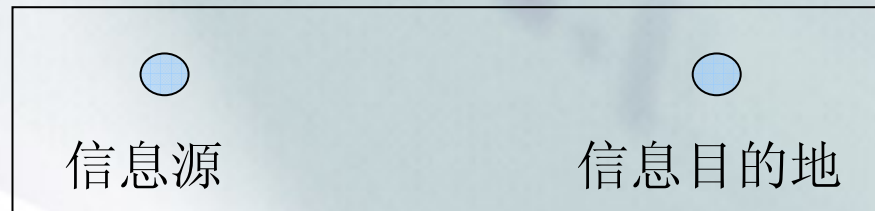


密码学与密码技术

为什么需要密码

- 信息的存储：在公开的地方
- 信息的交换：使用非隐秘介质
- 信息的传输：通过不安全信道





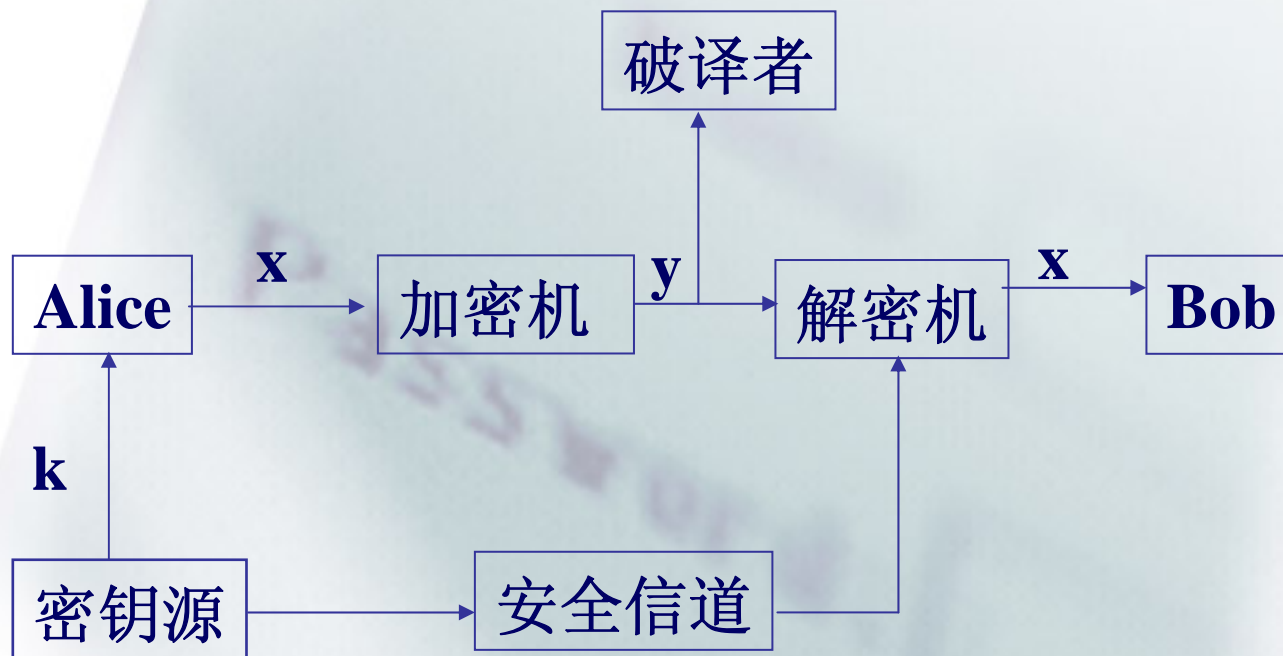
基本术语

- 消息被称为**明文(Plaintext)**。用某种方法伪装消息以隐藏它的内容的过程称为**加密(Encryption)**，被加密的消息称为**密文(Ciphertext)**，而把密文转变为明文的过程称为**解密(Decryption)**。
- **加密算法(Encryption Algorithm)**:对明文进行加密操作时所采用的一组规则。
- **解密算法(Decryption Algorithm)**:对密文解密所采用的一组规则。

凯撒密表

- 公元前**54**年，古罗马长官凯撒
- 明文字母 **abcdefghijklmnopqrstuvwxyz**
- 密文字母 **DEFGHIJKLMNOPQRSTUVWXYZABC**
- 有一个拉丁文句子
Omnia Gallia est divisa in Partes tres
(高卢全境分为三个部分)
RPQLD JDOOLD HVW GLYLVD LQ SDUWHV WUHV
- 把明文的拉丁字母逐个代之以相应的希腊字母

加密通信的模型



密码学的目的： Alice和Bob两个人在不安全的信道上进行通信，而破译者不能理解他们通信的内容。



密码算法分类

- 按照保密的内容分:
 - 受限制的 (**restricted**)算法:算法的保密性基于保持算法的秘密。
 - 基于密钥 (**key-based**)的算法:算法的保密性基于对密钥的保密。



密码算法分类

- 基于密钥的算法，按照密钥的特点分类：
 - 对称密码算法 (**symmetric cipher**): 又称传统密码算法 (**conventional cipher**), 就是加密密钥和解密密钥相同, 或实质上等同, 即从一个易于推出另一个。又称秘密密钥算法或单密钥算法。
 - 非对称密钥算法 (**asymmetric cipher**): 加密密钥和解密密钥不相同, 从一个很难推出另一个。又称公开密钥算法 (**public-key cipher**)。其中的加密密钥可以公开, 又称公钥 (**public key**), 解密密钥必须保密, 又称私钥 (**private key**)。

密码算法分类

- 按照明文的处理方法：
 - 分组密码 (**block cipher**): 将明文分成固定长度的组, 用同一密钥和算法对每一块加密, 输出也是固定长度的密文。如多数网络加密应用的 **DES, IDEA, RC6, Rijndael**, 加密解密速度慢。
 - 流密码 (**stream cipher**): 又称序列密码。序列密码每次加密一位或一字节的明文, 也可以称为流密码。序列密码是手工和机械密码时代的主流, 如移动通信中的加密。



Kerckhoff原则

- **1883年Kerchoffs**第一次明确提出了编码的原则：加密算法应建立在**算法公开**不影响明文和密钥的安全的基础上。
- 即：**数据的安全基于密钥而不是算法的保密**
- 这一原则已得到普遍承认，成为判定密码强度的衡量标准，实际上也成为古典密码和现代密码的分界线。

密码分析

- 假设破译者是在已知密码体制的前提下来破译密钥。最常见的破解类型如下：

唯密文攻击：破译者具有密文串 y 。

已知明文攻击：破译者具有明文串 x 和相应的密文 y 。

选择明文攻击：破译者可获得对加密机的暂时访问，因此他能选择明文串 x 并构造出相应的密文串 y 。

选择密文攻击：破译者可暂时接近解密机，可选择密文串 y ，并构造出相应的明文 x 。

这一切的目的在于**破译出密钥或密文**



密码算法的安全性

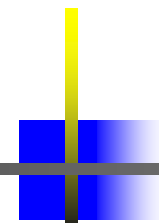
- 无条件安全 (**Unconditionally secure**)
无论破译者有多少密文,他也无法解出对应的明文,即使他解出了,他也无法验证结果的正确性.
- 计算上安全 (**Computationally secure**)
破译的代价超出信息本身的价值
破译的时间超出了信息的有效期.



密码学的起源和发展

古典密码 (classical cryptography)

- 密码学还不是科学, 而是艺术
- 出现一些密码算法和加密设备
- 密码算法的基本手段代替&置换
- 简单的密码分析手段出现



古典密码

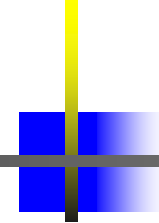
基于字符的密码

- 代替密码 (**substitution cipher**): 就是明文中的每一个字符被替换成密文中的另一个字符。接收者对密文做反向替换就可以恢复出明文。
(代替规则、密文所用字符、明文中被代替的基本单位)
- 置换密码(**permutation cipher**), 又称**换位密码** (**transposition cipher**): 明文的字母保持相同, 但顺序被打乱了。

一些准密码

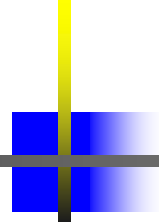
- 隐写术(**steganography**): 通过隐藏消息的**存在**来保护消息。(隐形墨水、字符格式的变化、图象图像)
- 暗号: 通过用物件的状态或行为来传达事先约定的信息。
- 把一些简单信息变换为常见的现象。
 - 窗台上的花瓶, 可代表“现在安全”
 - 姜太公用称为“阴符”的符契, 进行军事上的保密通信。

信息	符契	信息	符契
大胜克敌	长一寸	警众坚守	长六寸
破军擒将	长九寸	请粮益民	长五寸
降城得邑	长八寸	败军亡将	长四寸
却敌报远	长七寸	失利亡士	长三寸



代替(substitution)

- 北宋时期，把军中之事归纳成**40**条，“请弓”、“请刀”、“未见贼”、“请固守”、“请添兵”等，“以旧诗四十字，不得令字重，每字依次配一条，与大将各收一本。见北宋《武经总要》，曾公亮著。
- 如双方以唐代王勃的《送杜少府之任蜀川》“城阙辅三秦，风烟望五津。与君离别意，同是宦游人。海内存知己，天涯若比邻。无为在歧路，儿女共沾巾。”作为密码本。如果军队在战斗在粮食将尽，将领从密码本中查出“请粮料”的编码，假如是第九，诗中第九字是“五”。于是就将“五”字写到一件普通公文书牒之中，并在字上加盖印章。指挥机关接到这件公文后，查出盖印章的“五”字，再对照密码本上的顺序，就得知了前方的情报。

- 
- 美国二战时候特别征募使用的印第安纳瓦约通信兵，使用约瓦纳语进行情报传递。纳瓦约语的语法、音调及词汇都极为独特，不为世人所知道，纳瓦霍语密码成为历史上从未被破译的密码。从电影《风语者》(**Windtalkers**)中能窥其一二。

置换 (permutation)

- 斯巴达人用于加解密的一种军事设备 (**500 B.C.**) , 发送者把一条羊皮螺旋形地缠在一个圆柱形棒上:



代替-换字表

- 明文字母 **abcdefghijklmnopqrstuvwxyz**
- 密文字母 **IGVRFHPJYBNKAXLTSMCWDUEOZQ**
- **hello**代替成什么？
- 答案是: **JFKKL**

多字母代替密码-Playfair

- **Playfair:**按成对字母加密
- 5×5 变换矩阵: I与J视为同一字符

C	I	P	H	E
R	A	B	D	F
G	K	L	M	N
O	Q	S	T	U
V	W	X	Y	Z

1. 取交叉: kt \rightarrow MQ OD \rightarrow TR
2. 同行取右边: he \rightarrow EC
3. 同列取下边: dm \rightarrow MT
4. 对于相同字母对, 加分隔符x, 例如 balloon
写为字母对 ba lx lo on

Playfair 举例

C	I	P	H	E
R	A	B	D	F
G	K	L	M	N
O	Q	S	T	U
V	W	X	Y	Z

- (1) balloon \Rightarrow ba lx lo on \Rightarrow db sp gs ug
(2) book \Rightarrow bo ok \Rightarrow rs qg
(3) fill \Rightarrow fi lx lx \Rightarrow ae sp sp

置换密码

- 换位密码把明文按行写入, 按列读出
- 密钥包含3方面信息: 行宽, 列高, 读出顺序

key: 3 4 2 1 5 7 6

**plaintext: a t t a c k p
 o s t p o n e
 d u n t i l t
 w o a m x y z**

ciphertext:

TTNAAPTMTSUOAODWCOIXPETZKNLY

转轮机 (rotor machine)





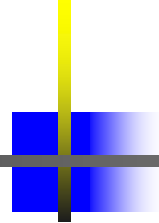
多轮替换

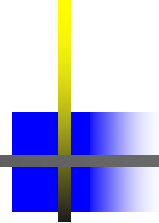
- 转轮机（转子机）是**DES**之前，多轮加密思想最重要的应用。
- 由多个独立转动的圆柱体组成，每个圆柱体定义了一个单字母替代。
- 多个圆柱体联合起来，前一个圆柱体的输出连接到下一个圆柱体的输入。
- **3个圆柱体使得有 $26*26*26=17576$ 种不同的替代字母可用，5个圆柱体的周期是11881376个字母。**

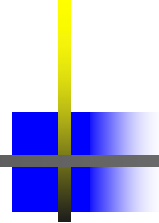


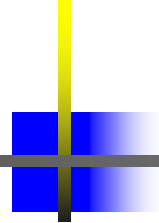
ENIGMA

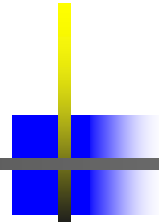
- **ENIGMA**密码最初是由一个叫胡戈·科赫的荷兰人发明的。起初主要提供给想保护自己生意秘密的公司使用，后来德国人将其改装为军用型，使之更为复杂可靠。于**1926**年开始使用。
- **1933**年，纳粹最高统帅部通信部决定将“**ENIGMA**”作为德国国防军新式闪击部队的通信装置。它结构紧凑、轻便，便于携带。德军曾断言：**ENIGMA**是绝对可靠的，只需调节一下转子和插头，密码机瞬间就可产生无数不同的密码。即使被敌方缴获，也无关紧要，因为除非对手了解变化无穷的调节程序，否则这机器毫无用处。
- **ENIGMA**,希腊语，意指“不可思议的东西”

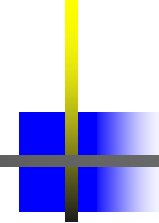
- 
- **1928**年的一天，华沙海关检查站接到德国驻波兰大使馆的紧急通知，要求立即交付德国外交部邮寄给它的一包邮件。波兰人感到十分怀疑和好奇，将这包邮件偷偷转交给了波兰情报部门。惊喜地发现邮件里装的竟是德国人吹嘘的“永远无法破译”的**Enigma**。在弄清其内部的连线关系和基本构造后，把邮件按原样封好，然后不动声色地交给德国大使馆。

- 
- 随后，他们很快从波兹南大学调来**3**名数学家，开始了对恩尼格玛密码的破译研究。**1934**年，波兰人研究出了破译**Enigma**的方法。但德国人在**1937**年又对**Enigma**作了大幅度改进。**1939**年，波兰情报部门邀请英国和法国的情报部门共商合作破译**Enigma**。两个多月后，波兰亡国。破译**Enigma**的重任全部落到了英国人身上。

- 
- 英国情报部门在伦敦以北约**80**公里的布莱奇利庄园汇集了英国当时的很多数学家。其中，最为著名的要数计算机领域的巨匠艾伦·图灵，当时他才**27**岁。图灵把波兰人用来破译**ENIGMA**的**BUM**机与他早先的数学研究结合起来，制造出了图灵“炸弹”
 - **1940年3月14日**，第一台图灵“炸弹”在布莱奇利庄园投入使用。最初的“炸弹”运行非常慢，有时要一个星期才能找到一个密钥，著名数学家戈登·韦尔什曼带领一些工程师对“炸弹”进行了大规模改进。改进后的“炸弹”可以在一个小时里找到一个密钥。

- 
- 为了保守机密，英国情报部门采取了一系列严密措施，甚至干脆放弃一些极有价值的情报。**1940年11月12日**，当破译得知德国空军将出动**500**多架轰炸机对英国的考文垂市进行大规模空袭时，丘吉尔首相果断下令不将此事通报考文垂市。结果，在德国空军按计划进行的空袭中，有**6**万多名市民丧命。
 - 英国用如此大的代价蒙蔽了德国人，使他们坚信恩尼格玛密码万无一失，于是便继续放心大胆地使用。

- 
- 在后来的阿拉曼战役、攻占西西里岛、诺曼底登陆等具有决定意义的军事行动中，英国和盟军都通过破译恩尼格玛密码及时掌握了战场主动权，使得胜负天平逐渐偏向盟军一方。
 - 为彻底埋葬这个秘密，战后，英国拆毁了千辛万苦研制出来的“炸弹”，销毁了设计图纸和各种文件资料。直到**20世纪70年代**，随着计算机加密技术的发展，恩尼格玛密码已经落后了，真相才逐渐大白于天下。

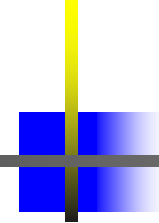


DES的产生

1973年5月15日, NBS（美国国家标准局, 即现在的国家标准与技术研究所**NIST**）开始公开征集标准加密算法,并公布了它的设计要求:

- (1)算法必须提供高度的安全性
- (2)算法必须有详细的说明,并易于理解
- (3)算法的安全性取决于密钥,不依赖于算法
- (4)算法适用于所有用户
- (5)算法适用于不同应用场合
- (6)算法必须高效、经济
- (7)算法必须能被证实有效
- (8)算法必须是可出口的

公众的回答表明了大家的兴趣, 但提交的算法都与要求相去甚远。



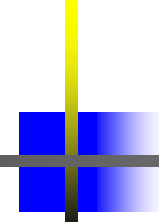
DES的产生

- **1974年8月27日, NBS开始第二次征集,IBM提交了算法 LUCIFER, 该算法由IBM的工程师在1971~1972年研制的。**
- **1976年11月23日, 采纳为联邦标准, 批准用于非军事场合的各种政府机构**
- **1977年1月15日,“数据加密标准” DES 的正式文本FIPS PUB 46发布,同年7月15日开始生效.**
- **1980年, 美国国家标准局 (ANSI) 赞同DES作为私人使用的标准,称之为DEA (ANSI X.392)**
- **1983年, 国际化标准组织ISO赞同DES作为国际标准, 称之为DEA-1**



分组密码算法

- 分组密码是将明文消息编码表示后的数字（简称明文）序列，划分成长度为 n 的组，每组分别在密钥的控制下变换成等长的输出数字（简称密文）序列。



简要介绍

- **DES**对**64**位的明文分组实施**16**轮加密，输出**64**位的密文。
- 加密和解密使用相同的密钥和相同的算法。
- 不过，加密和解密的子密钥（由加密密钥变换而得**16**个子密钥）顺序不同。若加密子密钥顺序为：**K1, K2, ..., K16**，那么解密的子密钥顺序为**K16, K15, ..., K1**。
- 密钥长度为**56**位（使用时要求输入的密钥为**64**位，但只用**56**位，另外**8**位做奇偶校验或随意设置）。



加密过程

- 对于一个输入的**64**位明文分组**m**，**DES**整个加密过程有三个阶段组成：**初始置换**，**迭代过程**（**乘积变换**）和**末置换**。
- 首先，**DES**对**m**进行初始置换，通过查表的方法，将**m**中的**64**位的顺序打乱，如将第**1**位放到第**58**位，而将第**58**位放到第**55**位等等。经过置换后得到**m'**。将**m'**分为左、右两半：**L0**、**R0**，每边各**32**位。

初始置换-末置换

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

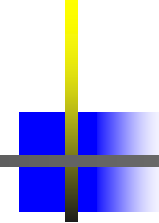
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

16轮迭代

- 然后，进行**16**轮完全相同的变换。每一轮迭代中使用的子密钥**K_i**都是由密钥**K**变换而得（变换过程较繁锁），子密钥长度为**48**位。
- **16**轮的操作可以表示为如下的赋值操作， \oplus 为模2加运算，**i**从**1**到**16**:

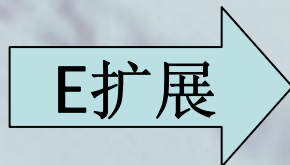
```
for ( i =1 to 16 )  
{  
    L[i]=R[i-1]  
    R[i]=L[i-1]  $\oplus$  f (R[i-1],K[i])  
}
```


$$f(x,y)$$

- 其中函数 $f(x,y)$ 较为复杂，首先将**32**位的 x 通过**E扩展**变为**48**位，然后将其和**48**位的 y 异或，得到的结果输入给**8**个**S盒子**，每个盒子处理**6**位数据，将其变换为**4**位数据。这样**8**个**S**盒子共输出**32**位数据，再将这**32**位结果进行一个**P置换**。至此，函数 f 的操作结束。
- 最后，对**R16**和**L16**应用初始置换的逆置换，便得到对 m 进行**DES**加密后的密文。

E 置换—32位扩展到48位

01	02	03	04
05	06	07	08
09	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28
29	30	31	32



32		01	02	03	04		05
04		05	06	07	08		09
08		09	10	11	12		13
12		13	14	15	16		17
16		17	18	19	20		21
20		21	22	23	24		25
24		25	26	27	28		29
28		29	30	31	32		01

S盒子1-4

S₁

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S₂

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S₃

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S₄

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S盒子5-8

S₅

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S₆

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S₇

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

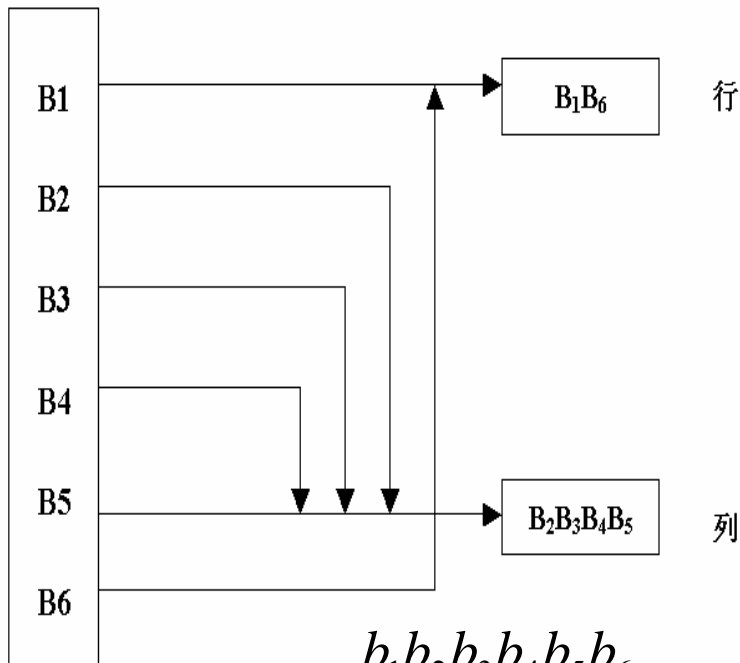
S₈

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

S盒子的使用

- 对每个盒，**6**比特输入中的第**1**和第**6**比特组成的二进制数确定的行，中间**4**位二进制数用来确定的列中相应行、列位置的十进制数的**4**位二进制数表示作为输出。

S盒子的使用



行\列	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

$$\begin{array}{l}
 b_1 b_2 b_3 b_4 b_5 b_6 \\
 \hline
 \boxed{110011}
 \end{array}
 \Rightarrow
 \begin{array}{l}
 \text{行: } b_1 b_6 = 11_2 = 3 \\
 \text{列: } b_2 b_3 b_4 b_5 = 1001_2 = 9
 \end{array}
 \Rightarrow
 \begin{array}{l}
 s_6 \text{ 盒子3行9列} \\
 \text{值: } 14 = \boxed{1110}
 \end{array}$$



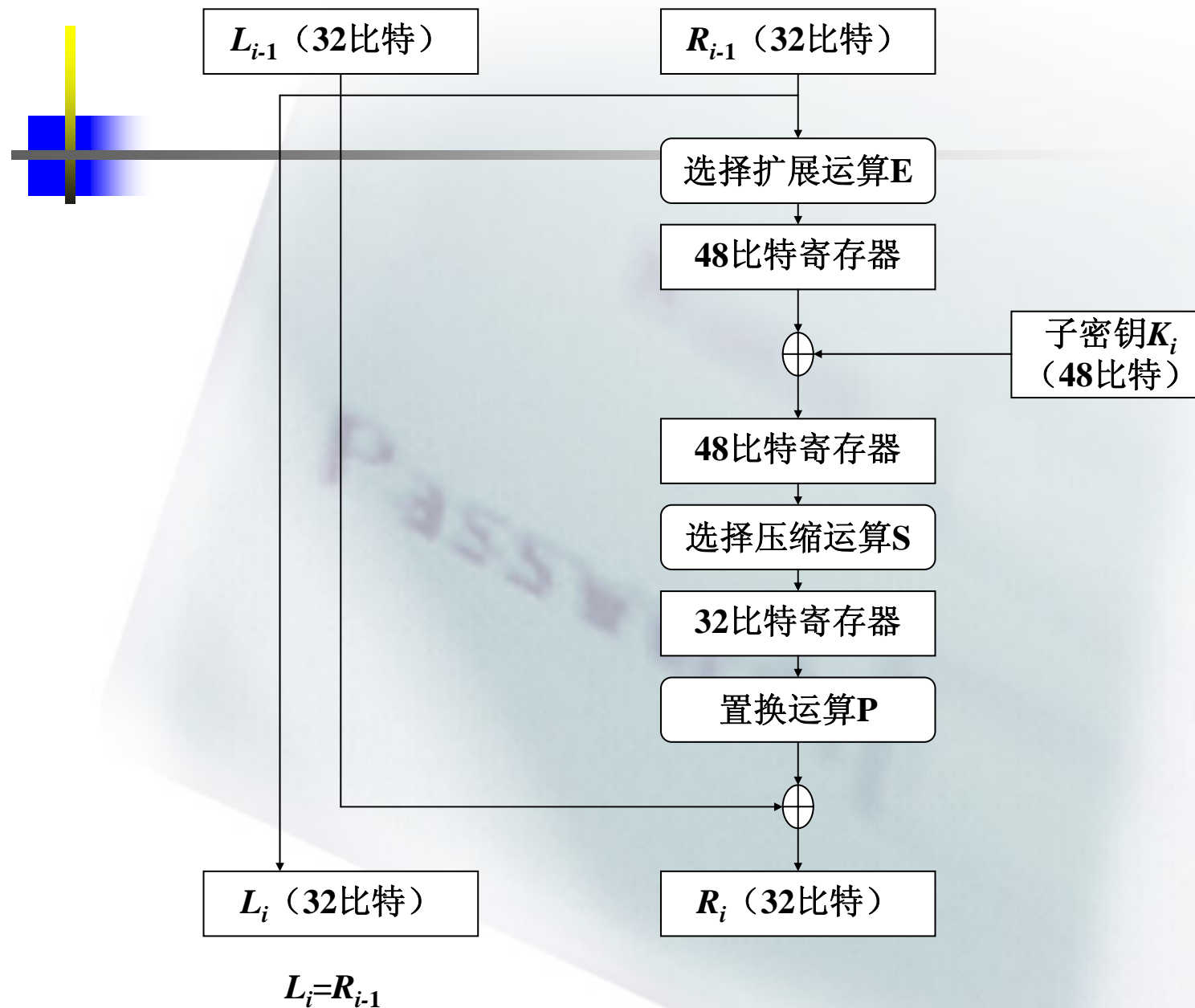
S盒中的不安全疑议

- 作为**DES**密码体制中的非线性组件，**S**盒的安全性是至关重要的。但是迄今仍未公布它的设计准则，因此，人们怀疑**S**盒的设计中可能会存在陷阱，使得**NSA**只要利用一个简单的方法就能解密密文！



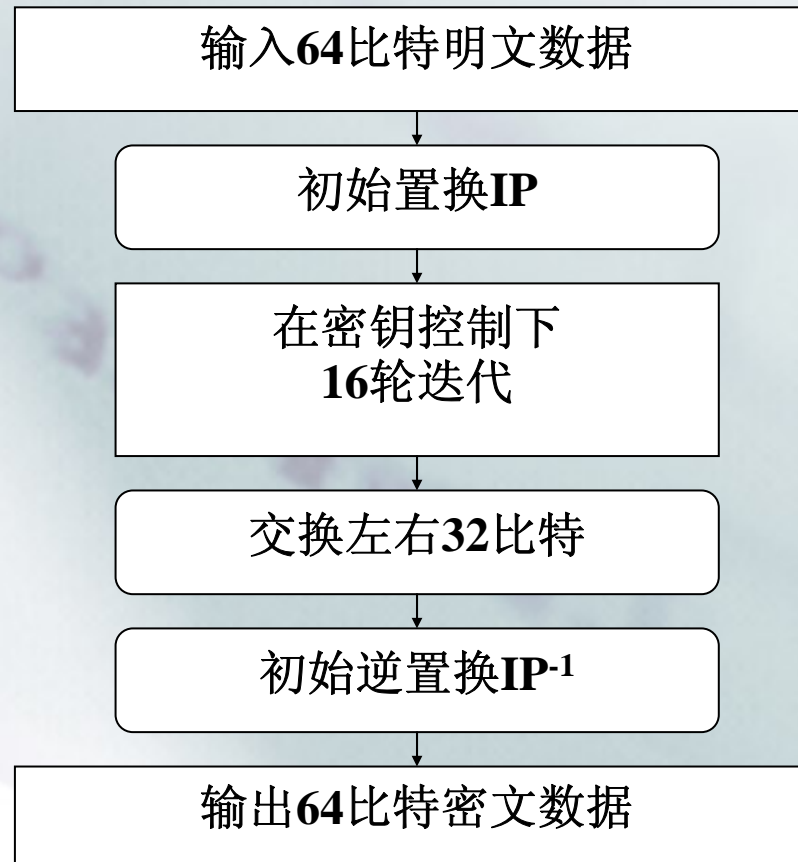
P置换

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25



DES的一轮迭代

DES加密总结



DES算法框图

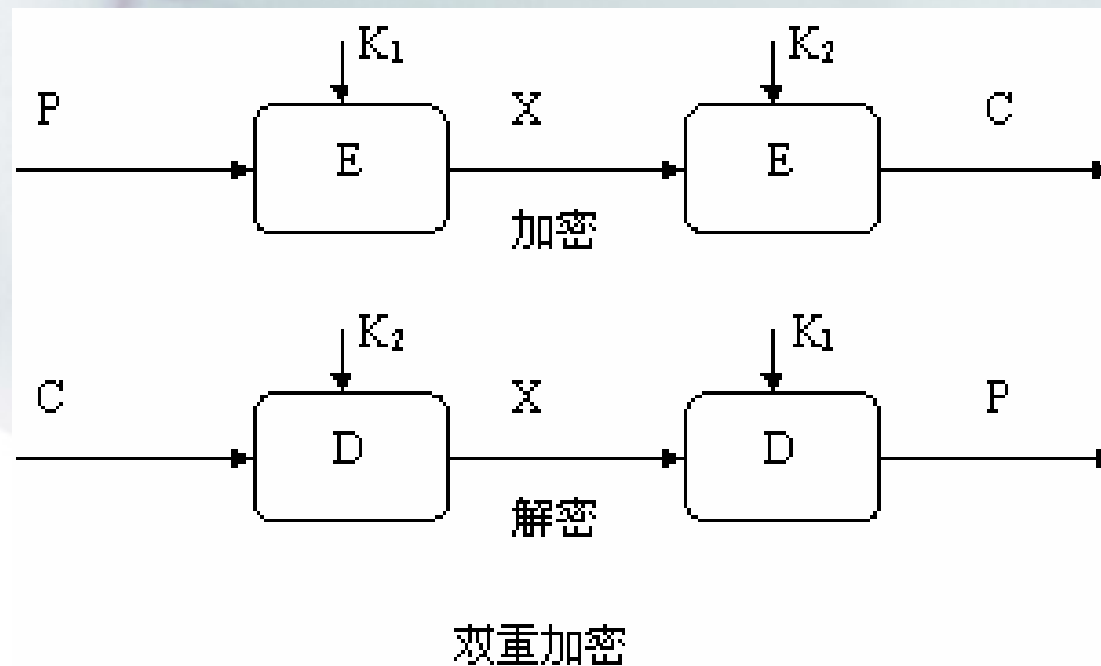


DES的解密

- **DES**算法的解密和加密使用相同的算法，二者唯一不同之处在于子密钥的使用顺序。解密时子密钥的使用顺序依次为：**k16, k15, ..., k1**。注意解密一开始时，得到的输入是密文，通过初始置换并将其分为两半：**L0**和**R0**。然后经过**16**轮运算后，得到**L16**和**R16**，再将其进行初始置换的逆置换，就可得到解密后的明文。

双重DES (Double DES)

- $C = E_{K_2}(E_{K_1}(P)) \Leftrightarrow P = D_{K_1}(D_{K_2}(C))$





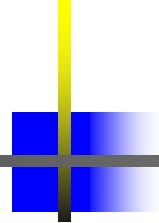
Triple-DES的四种模型

- **DES-EEE3**: 三个不同密钥, 顺序使用三次加密算法
- **DES-EDE3**: 三个不同密钥, 依次使用加密-解密-加密算法
- **DES-EEE2**: $K_1=K_3$, 同上
- **DES-EDE2**: $K_1=K_3$, 同上
- 密钥的有效长度 可达到**168**位
- 与**DES**的兼容性可以通过令 $K_3=K_2$ 或 $K_1=K_2$ 得到
- 许多基于**Internet**的应用里用到: **PGP**和**S/MIME**



DES的破解

- **DES**的实际密钥长度为**56-bit**，就目前计算机的计算机能力而言，**DES**不能抵抗对密钥的穷举搜索攻击。**56**比特密钥的穷举数量为**72,057,594,037,927,936**（也就是**72**亿亿）次。
- **1997**年**1**月**28**日，美国的 **RSA**数据安全公司在公司的安全年会上，公布了一个名为“秘密的密钥挑战”的竞赛项目，悬赏一万美元破解密钥长度为**56**比特的**DES**密码算法。明文以“**The unknown message is:** ”开头，其后的内容等待破解。

- 
- 使用**分组链接模式（CBC）**生成密文，向挑战者们公布密文和所用的**初始化向量**。
 - 挑战者如果解出能密钥，用 **E-mail**（互联网上的电子邮件）的方式迅速报告给 **RSA** 公司，第一个解出密钥的人即成为相应挑战赛的胜利者。

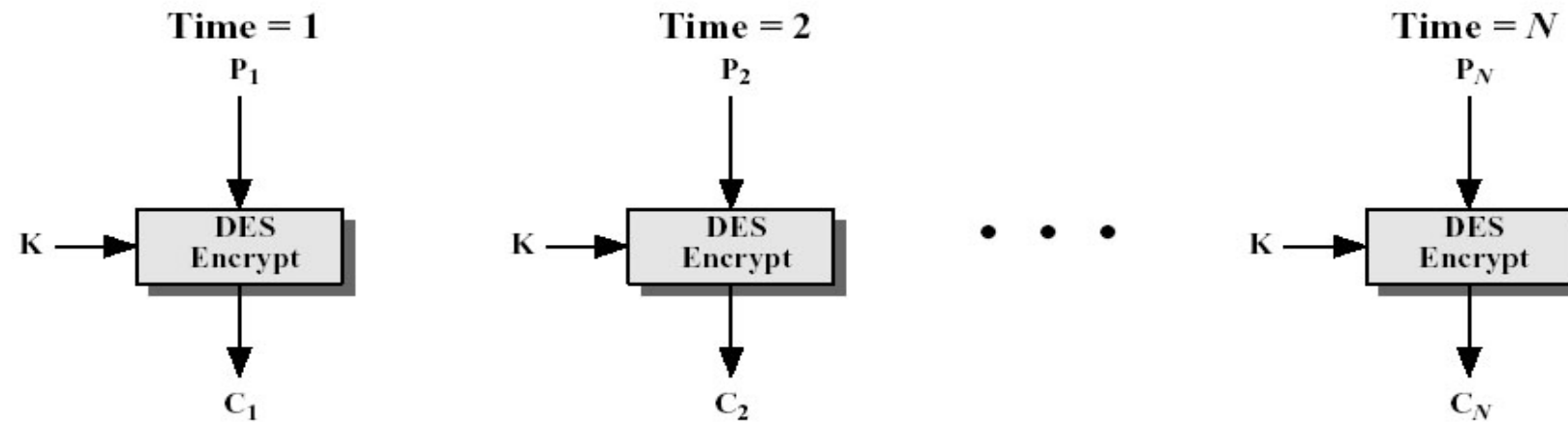


分组密码的操作模式

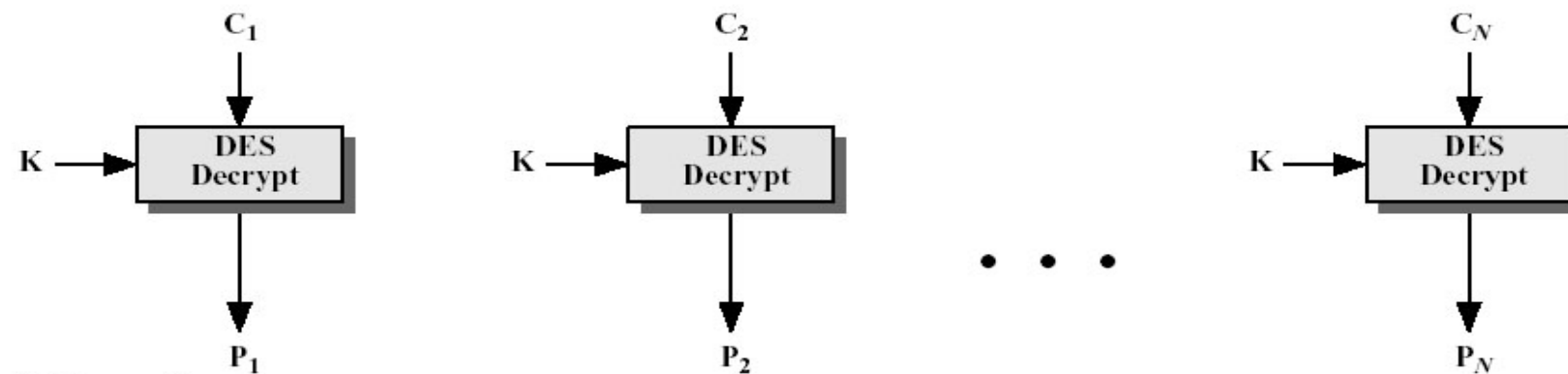
- 电子密码本**ECB** (electronic codebook mode)
- 密码分组链接**CBC** (cipher block chaining)
- 密码反馈**CFB** (cipher feedback)
- 输出反馈**OFB** (output feedback)
- 计数器 **CTR** (counter)

电子密码本 (ECB)

$$C_i = E_K(P_i) \Leftrightarrow P_i = D_K(C_i)$$



(a) Encryption



(b) Decryption

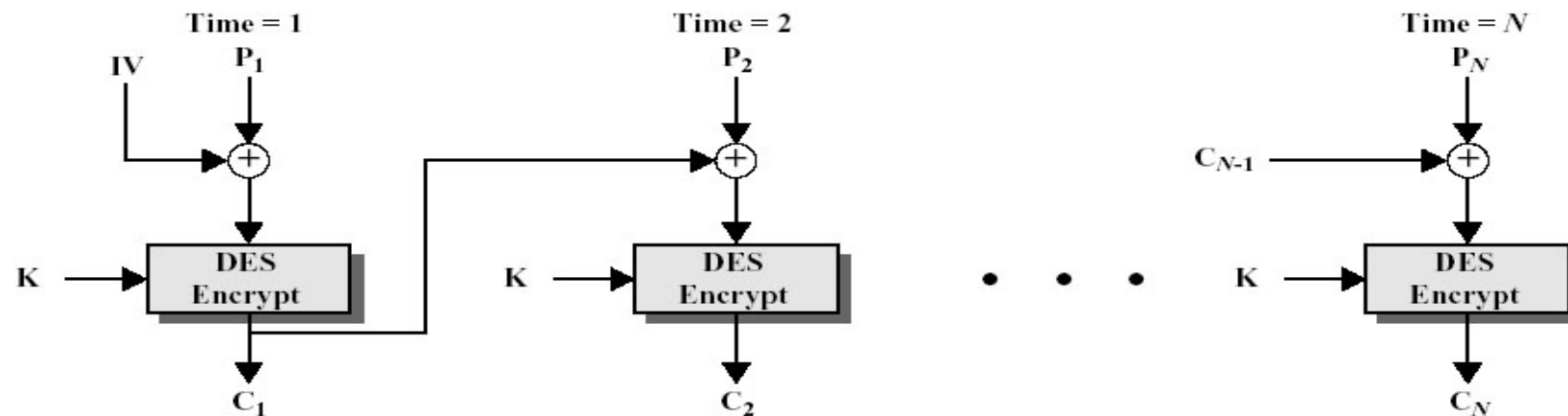


ECB特点

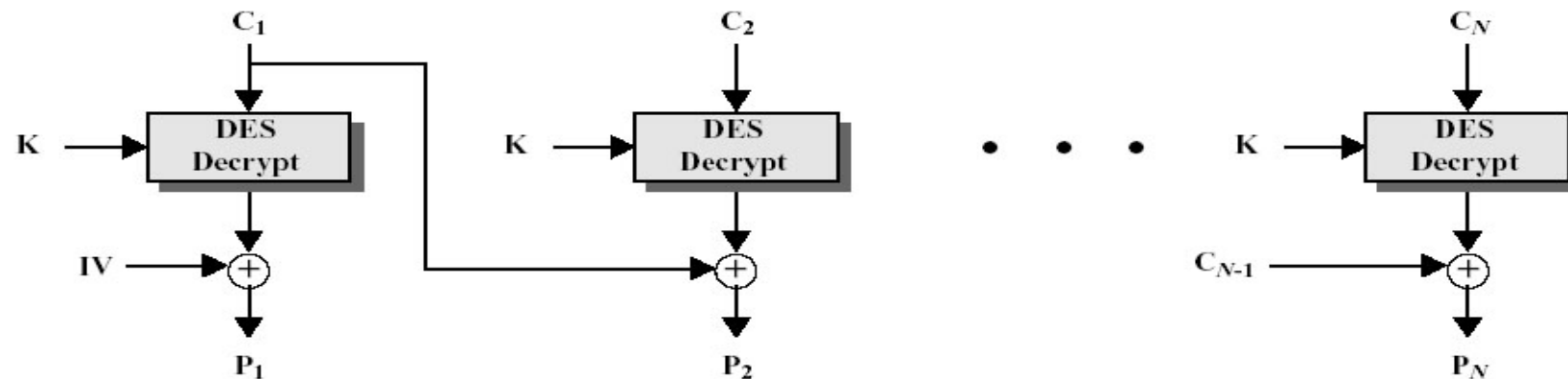
- 简单和有效
- 可以并行实现
- 有问题！
 - 相同明文 \Rightarrow 相同密文
 - 同样信息多次出现造成泄漏
 - 信息块可被重放...
- 误差特点：密文块损坏 \Rightarrow 仅对应明文块损坏

密码分组链接CBC

- $$C_i = E_K(C_{i-1} \oplus P_i) \Leftrightarrow P_i = E_K(C_i) \oplus C_{i-1}$$



(a) Encryption



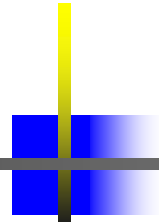
(b) Decryption

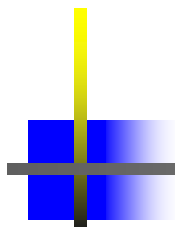


CBC特点

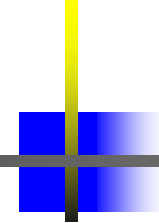
- 能隐藏明文的模式信息
 - ❑ 需要共同的初始化向量IV
 - ❑ 相同明文 \Rightarrow 不同密文
- 误差特点：一块密文块损坏，几块明文损坏？
- 答案：
明文块损坏
- 安全性好于**ECB**
- 适合于传输长度大于**64**位的报文
- 是大多系统的标准如 **SSL**、**IPSec**

两

- 
- 科罗拉多州的一位叫作**Rocke Verser**的程序员苦思冥想了近**40**天，终于设计出一个可通过互联网分段运算的密钥穷举攻击程序，这个计算机程序可以从互联网上分发和下载。
 - **Verser**把这项计算命名为**DESCHALL**，每名志愿者测试部分密钥空间，这样，正确的密钥最终会被某一个志愿者破解。大学和研究所的年轻电脑爱好者迅速成为**DESCHALL**计划的中坚力量，全美各大学宿舍里的奔腾机整夜地在运转着。

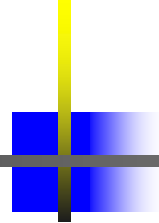


1997年6月17日，即在RSA挑战赛公布之后的第140天，DESCHALL计划实施的第96天，幸运降临到了美国盐湖城公司职员 Sanders 身上，他那台主频为奔腾90MHz、16M内存的PC 机上成功地解出了DES的明文—— The unknown Message is: “Strong cryptography makes the world a safer place”，根据Verser 的诺言，他和Verser按四六分成，共同分享那一万美元的奖金。



两点启示

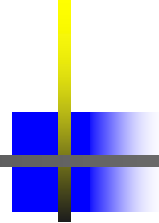
- 1** 这是第一次有人公开声明说他能破译**DES**加密的消息，美国政府和工业界将不得不严肃考虑他们所制定的密码政策。**Verser**说：“对于一个坚定的攻击者来说，**DES**已经不再安全”。
- 2** 这项计划证明了互联网的超级计算能力。并且这项计划是数万名从未见过面的志愿者使用普通的**PC**机，利用“空闲”机时来完成的。

- 
- **1998年7月电子边境基金会（EFF）使用一台价值25万美元的计算机在56小时之内破译了56-bit的DES。**
 - **1999年1月电子边境基金会（EFF）通过互联网上的10万台计算机合作，仅用22小时15分就破解了56-bit的DES。**



AES

- 为了替换安全性逐渐减弱的**DES**算法，**1997**年年初，**NIST**向全世界公开征集高级加密标准（**Advanced Encryption Standard, AES**）。对**AES**的基本要求是强度至少和三重**DES**一样，但要比三重**DES**更快。
- 经过长达三年、历经两轮的评估，最终，**Rijndael**数据加密算法脱颖而出。**NIST**在**2001**年**11**月**26**日正式公布**Rijndael**数据加密算法作为高级加密标准**AES**，于**2002**年**5**月**26**日起正式生效。

- 
- **Rijndael**数据加密算法是由比利时密码学家**Joan Daemen**和**Vincent Rijmen**开发设计的。主要的设计原则是以下三点：
 - 1. 能够抵抗所有已知的攻击；
 - 2. 设计简单；
 - 3. 编码紧凑，较好地适应平台，运算速度快。



AES加密过程

- **AES**的密钥长度可变，可以指定为**128**比特、**192**比特或**256**比特，数据分组长度也可以指定为这三个长度。
- **AES**加密、解密过程中，数据分组在中间各步的结果称为一个状态（**State**），可以用一个**4**行、**Nb**列的矩阵表示，其中，**Nb**为数据分组的长度除以**32**。
- 矩阵的元素是字节，如分组长度为**128**比特时，数据分组可表示为**4*4**的状态矩阵。

AES加密过程

如对于128比特的明文分组按顺序划分成16个字节，按照S00, S10, S20, S30; S01, S11, S21, S31;, 的顺序映射到状态阵列中。

$$\begin{pmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{pmatrix}$$



AES加密过程

- 密钥也用同样的方法表示，数组有**4**行，**Nk**列，其中**Nk**等于密钥长度除以**32**。
- 初始化轮密钥，对其进行（**Nr-1**）轮变换，**Nr**由**Nb**、**Nk**决定。
- 轮变换包括四个不同的矩阵变换：字节替换、行移位、列混合以及**加密钥**。经过多轮变换后，得到的状态矩阵即为密文分组矩阵。



对AES的评价

- **AES**算法对密钥的选择几乎没有限制，基本上不存在弱密钥和半弱密钥。
- 而且在抵抗穷举攻击、线性攻击、差分攻击、积分密码攻击和插入攻击方面，均表现出很好的性能。



对称算法的不足

(1) 密钥管理量的困难

传统密钥管理：两两分别用一个密钥时，则 n 个用户需要 $C(n,2)=n(n-1)/2$ 个密钥，当用户量增大时，密钥空间急剧增大。如：

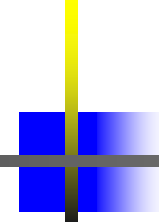
$n=100$ 时， $C(100,2)=4,995$

$n=5000$ 时， $C(5000,2)=12,497,500$

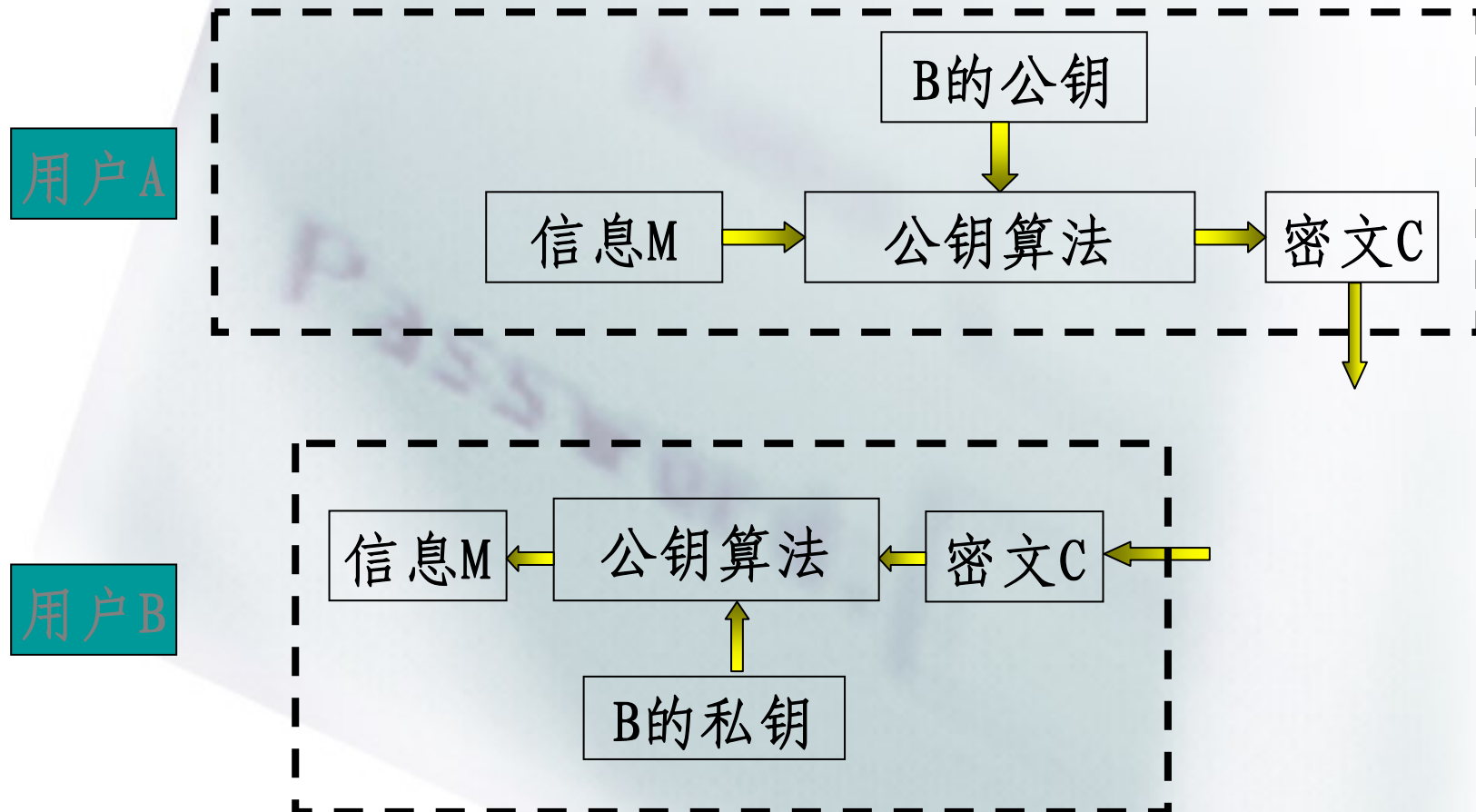
(2) 密钥必须通过某一信道协商，对这个信道的安全性的要求比正常的传送消息的信道的安全性要高。

(3) 数字签名的问题

传统加密算法无法实现抗抵赖的需求。

- 
- 公开密钥学的概念，是由 **Diffie** 和 **Hellman** 于 **1976** 年在美国国家计算机会议上首先提出的。几个月后，他们发表了论文“密码学的新方向”（“**New Directions in Cryptography**”）进一步阐述他们的思想。这篇开创性的论文，被公认为是密码学发展中的里程碑。
 - **1977** 年 **Rivest, Shamir & Adleman** 提出了 **RSA** 公钥算法

公钥密码学概述





公开密钥密码的重要特性

- ① 加密与解密由不同的密钥完成
- ② 知道加密算法,从加密密钥得到解密密钥在计算上是不可行的
- ③ 有的算法中,两个密钥中任何一个都可以用作加密而另一个用作解密



用公钥密码实现保密

- 用户拥有自己的密钥对(K_U, K_R)
- 公钥 K_U 公开, 私钥 K_R 保密
- $A \rightarrow B: Y = E_{K_{Ub}}(X)$
- $B: D_{K_{Rb}}(Y) = D_{K_{Rb}}(E_{K_{Ub}}(X)) = X$

用公钥密码实现签名

• 签名:

$$A \rightarrow ALL: Y = E_{KRa}(X)$$

$$ALL: D_{KUa}(Y) = D_{KUa}(E_{KRa}(X)) = X$$

• 签名+保密:

$$A \rightarrow B: Z = E_{KUb}(E_{KRa}(X))$$

$$B: D_{KUa}(D_{KRb}(Z)) = X$$

条件是两个密钥中任何一个都可以用作加密
而另一个用作解密



公钥密钥的应用范围

- 加密/解密
- 数字签名
- 密钥交换

算法	加/解密	数字签名	密钥交换
RSA	是	是	是
Dieffie-Hellman	否	否	是
DSS	否	是	否

公钥密码基于的数学难题

- 背包问题
- 大整数分解问题 (The Integer Factorization Problem, RSA体制)
- 离散对数问题
 - 有限域的乘法群上的离散对数问题(The Discrete Logarithm Problem, ElGamal体制)
 - 定义在有限域的椭圆曲线上的离散对数问题 (The Elliptic Curve Discrete Logarithm Problem, 类比的ElGamal体制)



RSA 算法

- 第一个出现的公开密钥算法是 **Ralph Merkle** 和 **Martin Hellman** 开发的背包算法。在这后不久，**Ron Rivest**，**Adi Shamir** 和 **Leonard Adleman** 开发了一个更完善的公开密钥算法。这个算法以这三位发明者的名字命名为 **RSA**。
- 在已提出的公开密钥算法中，**RSA** 是最为流行，也是最容易理解和实现的一个算法。密码分析者对其进行了多年深入的密码分析，但是仍然既不能否认也不能肯定 **RSA** 的安全性。**RSA** 算法的可信度由此可见一斑。
- 只在美国申请专利，且已于 **2000年9月** 到期



RSA 算法描述

RSA加、解密算法

- 公开密钥为 $\{e, n\}$, 其中:
 - n 是两个大素数 p 和 q 的乘积, 推荐 p, q 等长
 - 随机选取 e , 要求 e 与 $(p-1)(q-1)$ 互素
- 私有密钥 $\{d, p, q\}$, 注意 p 和 q 是严格保密的。
 - 计算 $d = e^{-1} \bmod (p-1)(q-1)$
 - 也使得 $ed \equiv 1 \bmod (p-1)(q-1)$

RSA 实现

- 用户B产生两个大素数 p 和 q , $p \neq q$
- B计算 $n=pq$, 并计算 $\phi(n)=(p-1)(q-1)$
- B选择随机数 $e, (0 < e < \phi(n))$, 使 $\gcd(e, \phi(n))=1$
- B使用Euclidean算法（欧几里德扩展算法）
计算 $d \equiv e^{-1} \bmod \phi(n)$
- 公钥: $K_U=\{e,n\}$, 私钥: $K_R=\{d,p,q\}$

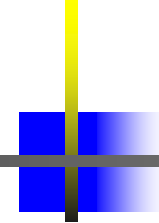
RSA加密

- 公钥: $KU=\{e,n\}$, 私钥: $KR=\{d,p,q\}$
- 选好这些参数后, 将明文 P 划分成块, 块长度满足 $2^k < n \leq 2^{k+1}$, 目的是让明文块的十进制值小于 n 。
- 加密 P 时, 计算 $C=P^e(\bmod n)$
- 解密 C 时, 计算 $P=C^d(\bmod n)$ 。
- 由于模运算的对称性, 可以证明, 在确定的范围内, 加密和解密函数是互逆的。



RSA签名

- 对明文 P 签名时，将明文划分成块，计算 $C = P^d \pmod{n}$
- 验证签名时，计算 $C^e \pmod{n}$ 看是否等于 P ，是的话，则表明签名正确。



RSA Example

1. 选择素数: $p=17$ & $q=11$
2. 计算 $n = pq = 17 \times 11 = 187$
3. 计算 $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. 选择 e : $\gcd(e, 160) = 1$; 选择 $e=7$
5. 确定 d : $de = 1 \pmod{160}$ 并且 $d < 160$, 可得 $d=23$
因为 $23 \times 7 = 161 = 1 \pmod{160}$
6. 公钥 $KU = \{7, 187\}$
7. 私钥 $KR = \{23, 17, 11\}$

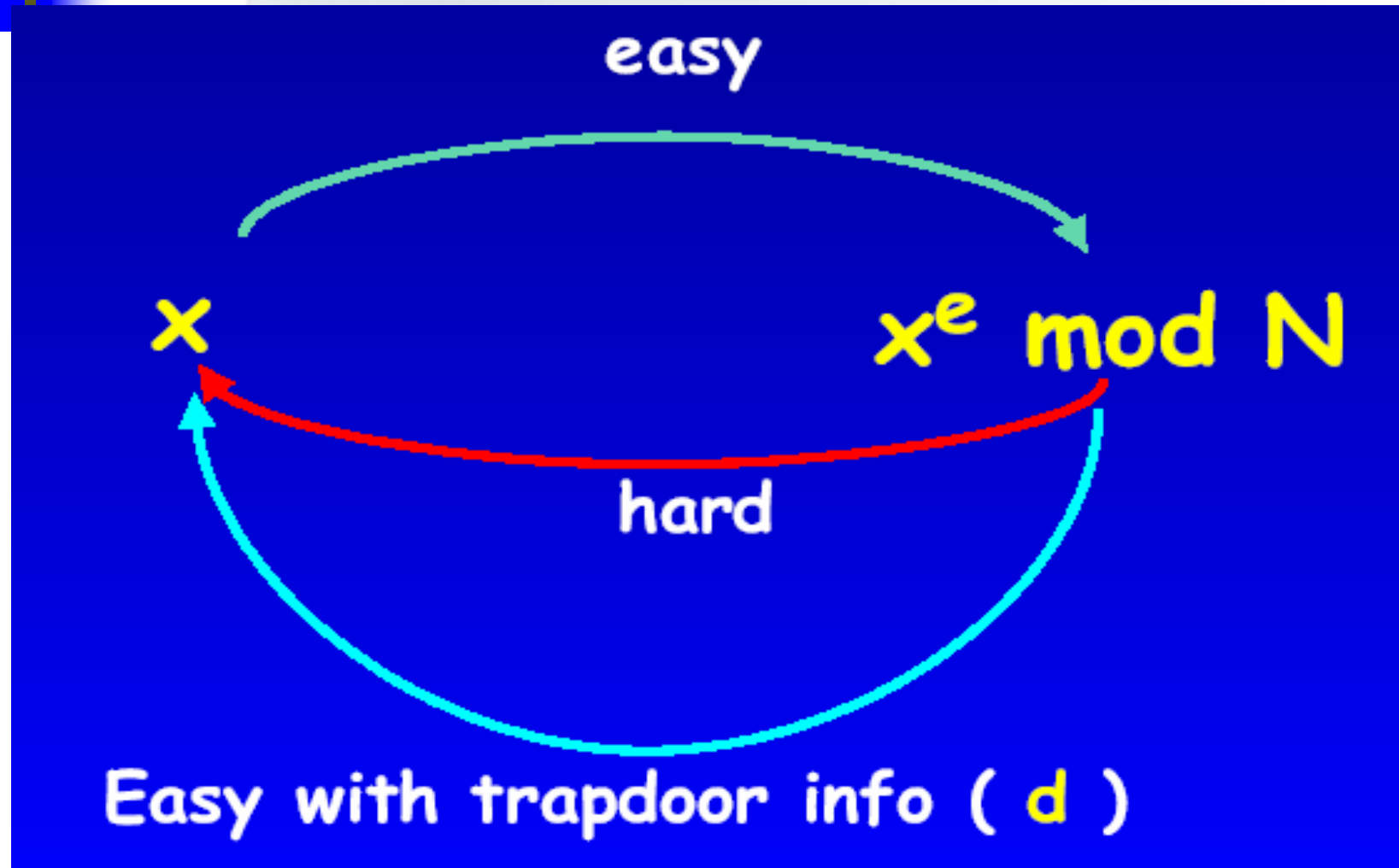


RSA Example

- **RSA**的加解密为:
- 给定消息 $M = 88$ ($88 < 187$)
- 加密:
$$C = 88^7 \bmod 187 = 11$$
- 解密:
$$M = 11^{23} \bmod 187 = 88$$

一个小软件

RSA的基本思想





RSA安全性依据

对攻击的人来说，试图从 $x^e \pmod n$ 求 x ：

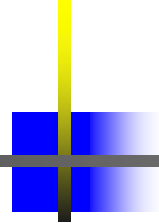
计算途径：分解 $n=pq$ ，计算 $\phi(n)=(p-1)(q-1)$ ，从而用欧氏算法解出解密私钥 d 。

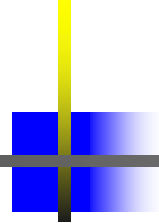
这在计算上是不可行的，有人猜想：攻破**RSA**的难度和分解 n 的难度是等价的。然而，这个猜想至今没有给出可信的证明。

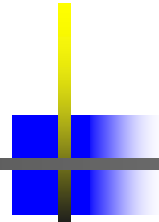
因子分解：RSA-129的故事

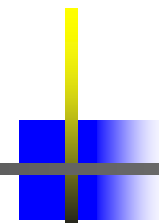
- 下面仅仅是一个乘法算式。但每一个明白他含义的读者都被震惊了，都知道它给密码学带来了什么。尽管这个算式的作者达六百多人。

**11438 16257 57888 86766 92357 79976 14661 20102
18296 72124 23625 62561 84293 57069 35245 73389
78305 97123 56395 87050 58989 07514 75992 90026
87954 3541 = 34905 29510 84765 09491 47849 61990
38981 33417 76463 84933 87843 99082 0577 * 32769
13299 32667 09549 96198 81908 34461 41317 76429
67992 94253 97982 88533**

- 
- **1977年RSA**被提出时，他们认为每秒钟百万次运算的计算机可以在**4**小时之内因子分解一个**50**位的数，但是分解一个**100**位的数要花几乎一个世纪，而**200**位的数大约要花**40**亿年。
 - **RSA**公司的研究人员用一个**129**位的数**N**和一个**4**位数**e**对一个消息作了编码。在《科学美国人》上刊登了那个密文，同时给出了**N**和**e**。**RSA**公司还悬赏**100**美元，奖给第一个破译这密码的人。**RSA**公司估计因子分解一个**129**位的数大约要花**23000**年。

- 
- 分解**RSA-129**这一项行动计划依靠了因特网，但核心动力是二次筛选法。二次筛选法是**1981**年乔治亚大学的**Garl Pomerance**提出的。
 - 这一行动计划由牛津大学的**Paul Leyland**，爱荷华州立大学的**Michael Graff**以及**RSA**公司的**Derek Atkins**发起，他们通过互联网发送程序和数据库，然后上网上征求志愿者。
 - 行动发起后，经过八个月的努力，最后于**1994**年**4**月为**RSA-129**找到了**64**位数和**65**位数两个素数因子

- 
- 破译离问题发布只有十七年。
 - 他们破译了下面这个**128**位数字的密文：
 - **96869 61375 46220 61477 14092 22543 55882
90575 99911 24574 31987 46951 20930 81629
82251 45708 35693 14766 22883 98962 80133
91990 55182 99451 57815 154**
 - 得到的明文是：“**The magic words are squeamish ossifrage**”（谜底是神经兮兮的秃鹰）

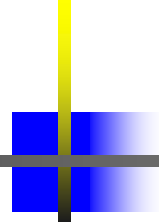
- 
- 值得注意的是，**RSA-129**分解的成功并不危及现行的以**RSA**为基础的编码，用以因子分解所需要的计算能力随着位数的增加而急剧增加。
 - **Rivest**指出：“我们建议用**200**位到**300**位的数。”除非计算数论有惊人的突破，因子分解 在一个长时间内仍然是个难题。
 - 和**DES**等对称密码算法相比，**RSA**的速度慢很多。**Applied Cryptography**一书中提到，硬件实现时，**RSA**大约比**DES**慢**1000**倍，软件实现时，比**DES**慢大约**100**倍。

Diffie-Hellman 密钥交换

- Diffie & Hellman in 1976
- 密钥交换方案，允许两个用户可以安全地建立一个秘密信息。且该秘密信息仅为两个参与者知道。
- 算法的安全性依赖于有限域上计算离散对数的难度
- 在美国的专利1997年4月29日到期

Diffie-Hellman密钥交换过程

- **A和B协商一个大素数 n 和 g ， g 是模 n 的本原元。这两个数可以公开，不需要保密。**
- **A选择一个大随机数 x ，计算 $X = g^x \bmod n$ ，将 X 发送给B。**
- **B选择一个大随机数 y ，计算 $Y = g^y \bmod n$ ，将 Y 发送给A。**
- **A计算 $Y^x \bmod n$ 。 B计算 $X^y \bmod n$ 。**
- **由于 $Y^x \bmod n = X^y \bmod n = g^{xy} \bmod n$ ，A和B之间建立起新的共享密钥。**

- 
- **Diffie-Hellman**算法的安全性依赖于在有限域上计算离散对数的困难性。由于 x 和 y 分别被**A**和**B**秘密的拥有，窃听者能够得到的值只有 n 、 g 、 X 、 Y 。
 - 除非窃听者能够根据 n 、 g 、 X 、 Y ，通过计算离散对数来得到 x 、 y 的值，否则，窃听者无法得到秘密密钥。

中间人攻击

- A计算 $g^{X_a} \bmod p$, 发给B
- O截获之, 选 X_o , 计算 $g^{X_o} \bmod p$, 冒充A \rightarrow B
- B计算 $g^{X_b} \bmod p$, 发给A
- O截获之, 冒充B \rightarrow A: $g^{X_o} \bmod p$
- A计算: $g^{X_o X_a} \bmod p$
- B计算: $g^{X_o X_b} \bmod p$
- O计算: $g^{X_a X_o} \bmod p$, $g^{X_b X_o} \bmod p$
- O坐在中间, 分别向A冒充B, 向B冒充A。
- O必须实时截获并冒充转发



Elliptic Curve Cryptography

- 多数公钥密码 (**RSA, D-H**) 使用非常大的数或多项式, 给密钥和信息的存储和处理带来很大的运算量
- 椭圆曲线是一个代替, 可以用更小的尺寸得到同样的安全性
- **1985年, Neal Koblitz和Victor Miller** 分别独立地提出了椭圆曲线密码体制
- **ANSI、IEEE、ISO和NIST**都制定了**ECC**标准草案



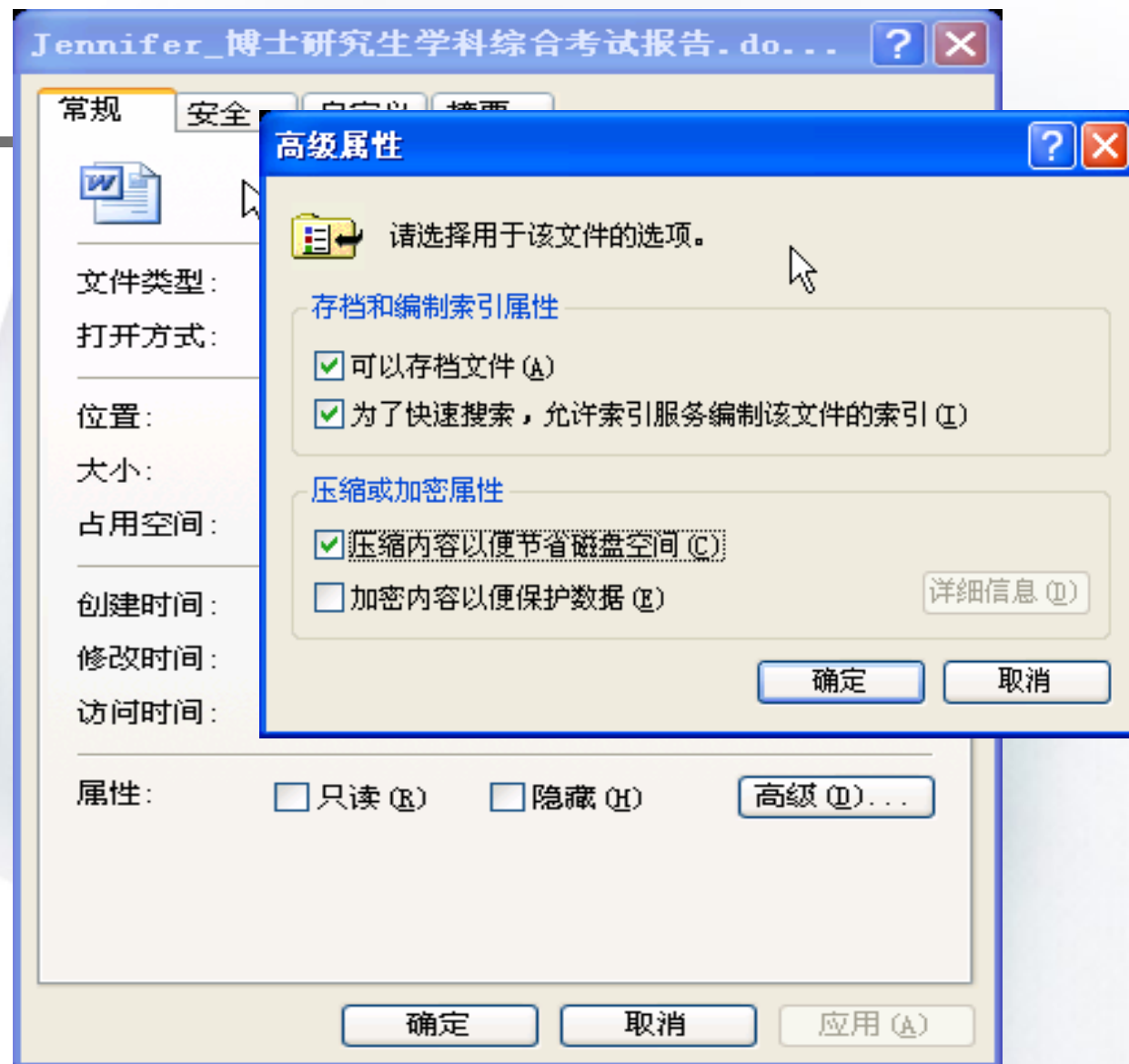
ECC和RSA性能比较

ECC密钥长度	RSA密钥长度	MIPS-年
160	1024	1012
320	5120	1036
600	21000	1078
1200	120000	10168



应用举例：EFS

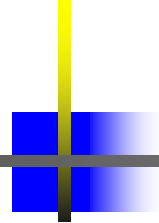
- 只有在**NTFS**分区上的文件和文件夹才能被加密。**Windows 2000、Windows XP Professional**（不包括**Home**版），还有**Windows Server 2003**。
- 加密一个文件只需设置文件的加密属性或把文件放入设置了加密属性的文件夹之中。文件将会在不打扰用户的情况下在后台加密文件，用户也不必使用密码去解密文件。
- **EFS**使用扩展的数据加密标准（**DESX**），**56**位加密算法。北美用户通过从微软订购增强的**CryptoPAK**可以获得**128**位加密算法。
- 系统文件不能被加密。
- 另外，除加密外，只要使用“对象权限”，都需要**NTFS**文件系统。

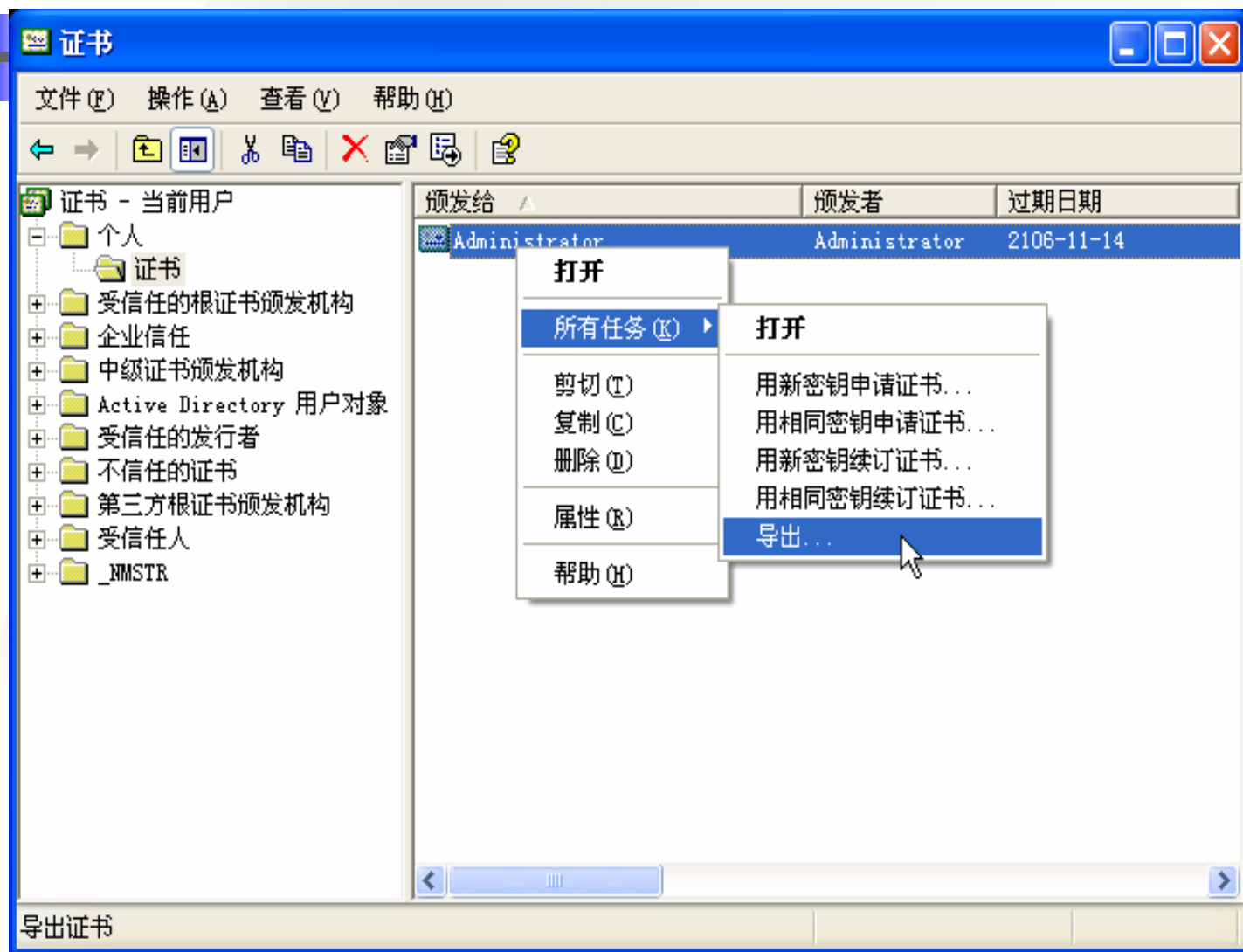


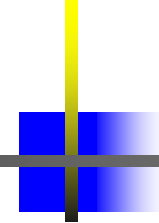


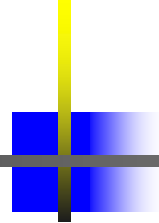
EFS保护文件工作原理

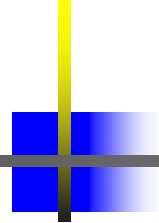
- 基于非对称公钥算法和对称加密算法的混合机制。
- 文件使用对称算法加密
- 文件的加密密钥使用用户证书的公钥加密，并与加密的文件一起存储。
- 用户的私钥可解密出文件的加密密钥，然后解密文件。

- 
- 很明显，加密文件的时候使用公钥，解密文件的时候则使用相对应的私钥。那么无论是丢失了公钥还是私钥，都会给使用带来麻烦。为了保证数据安全，最好能在加密文件之后立即将自己的密钥备份出来。
 - 运行“**certmgr.msc**”打开证书管理器，在“当前用户/个人/证书”路径下，应该 可以看见一个以你的用户名为名称的证书。右键点击，在“所有任务”中点击“导出”。并要选择“导出私钥”，导出的证书将是一个**pfx**为后缀的文件。



- 
- 当用户的密钥丢失后，例如重装了操作系统，或者无意中删除了某个帐户，我们只要找到之前导出的**pf**x文件，用鼠标右键点击，并选择“安装**PFX**”，之后会出现一个导入向导，按照导入向导的提示完成操作（注意，如果你之前在导出证书时选择了用密码保护证书，那么在这里导入这个证书时就需要提供正确的密码，否则将不能继续），而之前加密的数据也就全部可以正确打开。

- 
- 命令行使用方式：如果想加密D盘下的**test**文件夹，就输入：**cipher/e D:\test**。解密时则输入：**cipher/d D:\test**。
 - **EFS**诞生不久就出现了针对它的破解软件——**Advanced EFS Date Recovery (AEDR)**，破解成功率很高。**AEDR**采用特殊的方法找到用户密钥，并加以利用，如伪装成加密文件的帐户对文件进行查看等操作。

- 
- 人们总是对**EFS**敬而远之，一方面，**EFS**加密方式不像其他加密软件那样容易理解。另一方面，由于**EFS**的高安全性，如果用户操作不当则很可能导致数据丢失。
 - 可选的其他方法
 - ❑ **PGP**加密，加密盘
 - ❑ **WinRAR**将文件压缩的时候设置密码



消息鉴别

- 消息鉴别 (**Message Authentication**):

是一个证实收到的消息来自可信的源点且未被篡改的过程。

第一，验证信息的发送者是真正的，而不是冒充的，此为信源识别；

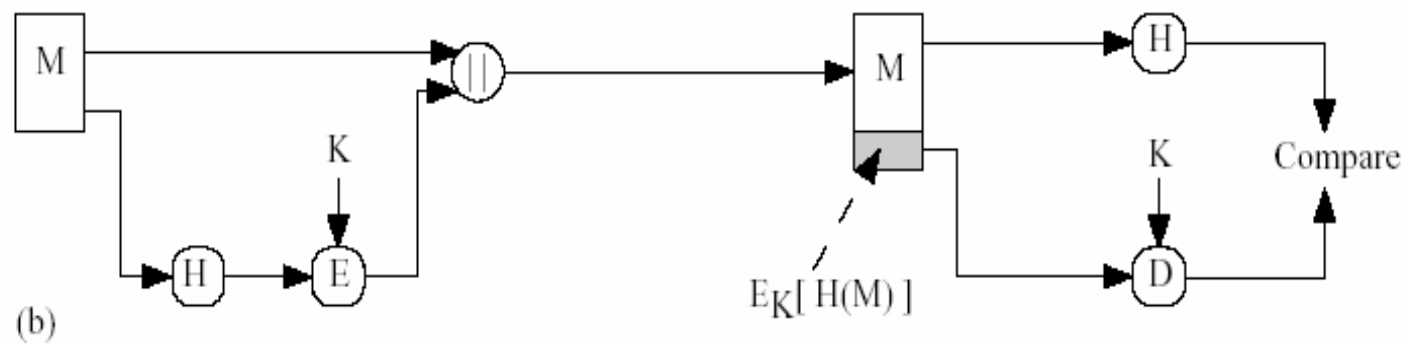
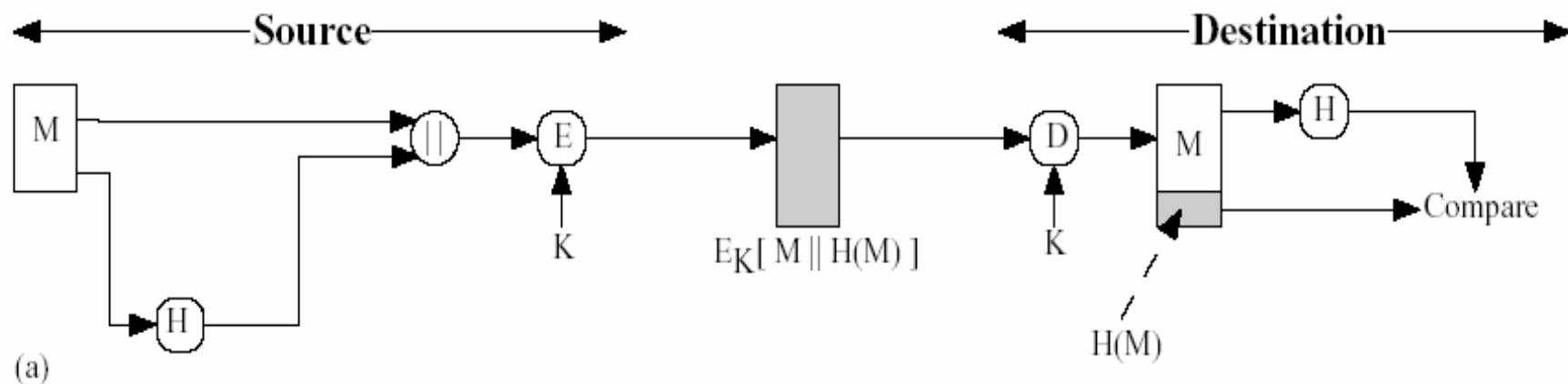
第二，验证信息的完整性，在传送或存储过程中未被篡改，重放或延迟等。



散列函数Hash Function

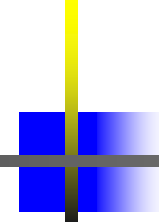
- 输入为任意长度的消息**M**；输出为一个固定长度的散列值，也称为**消息摘要（Message Digest）**。
- 消息中的任何一位或多位的变化都将导致该散列值的变化。
- 又称为：哈希函数、数字指纹（**Digital finger print**）、压缩（**Compression**）函数、数据鉴别码**DAC（Data authentication code）**、篡改检验码**MDC(Manipulation detection code)**

散列函数的基本用法



RSA签名算法的弱点

- 任何人能通过对某一 y 计算 $x=E_{k_{UB}}(y)$ 伪造一个B对随机消息 x （如果 x 可以理解为明文）的签名，因为 $y=Sig_{k_{RB}}(x)$ 。
- 如果消息 x_1, x_2 的签名分别是 y_1 和 y_2 ，则拥有 x_1, x_2, y_1, y_2 的任何人可伪造B关于消息 $x_1 x_2$ 的签名 $y_1 y_2$ ，因为 $Sig_{k_{RB}}(x_1 x_2) = Sig_{k_{RB}}(x_1) Sig_{k_{RB}}(x_2) \bmod n$
- 签名者每次只能签 $\log_2 n$ 比特长的消息，获得同样长的签名。如果消息很长，需要逐组签名。然而，这样，速度慢，签名长。



解决办法

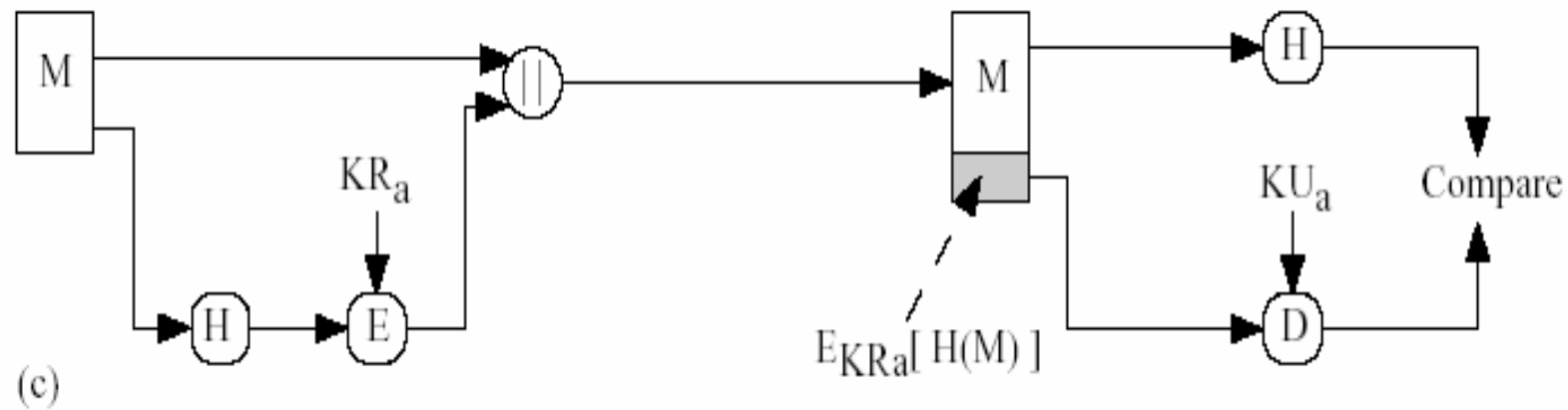
- 解决办法：引入可公开的密码散列函数(**Hash function**)。它将取任意长度的消息做自变量，结果产生规定长度的消息摘要。然后签名消息摘要。对数字签名来说，散列函数**h**是这样使用的：

消息： **x** 任意长

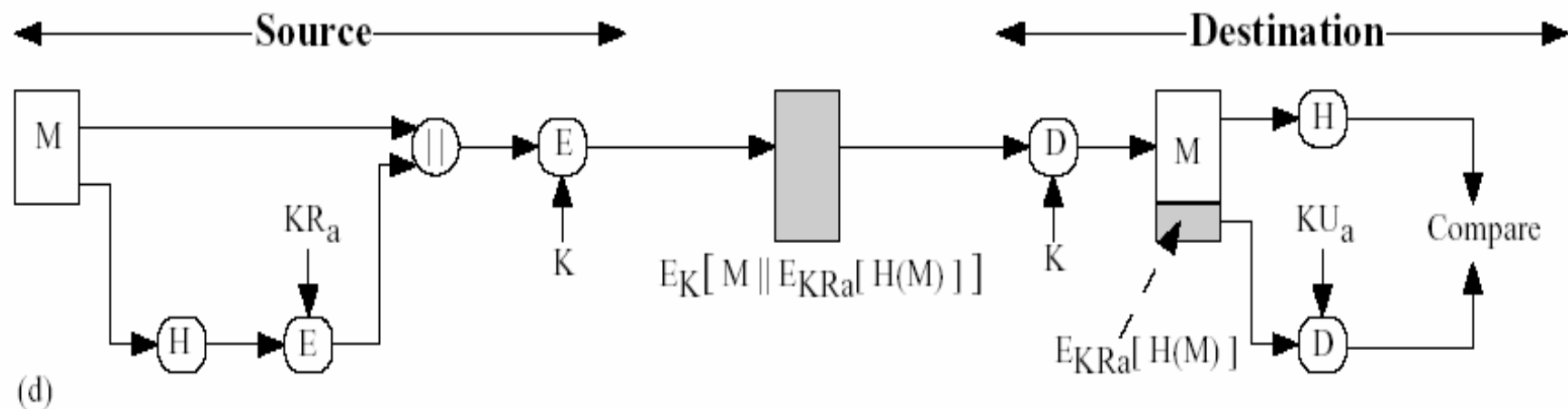
消息摘要： **$Z=h(x)$** **160bits**

签名： **$y=\text{sig}_k(Z)$** **320 bits**

散列函数的基本用法



散列函数的基本用法





安全威胁一

破译者以一个有效签名 (x, y) 开始，此处 $y = \text{sig}_k(h(x))$ 。首先他计算 $Z = h(x)$ ，并企图找到一个 x' 满足 $h(x') = h(x)$ 。若他做到这一点，则 (x', y) 也将为有效签名。为防止这一点，要求函数 h 具有无碰撞特性。

弱无碰撞：散列函数 h 称为是弱无碰撞的，是指对给定消息 $x \in X$ ，在计算上几乎找不到异与 x 的 $x' \in X$ 使 $h(x) = h(x')$ 。

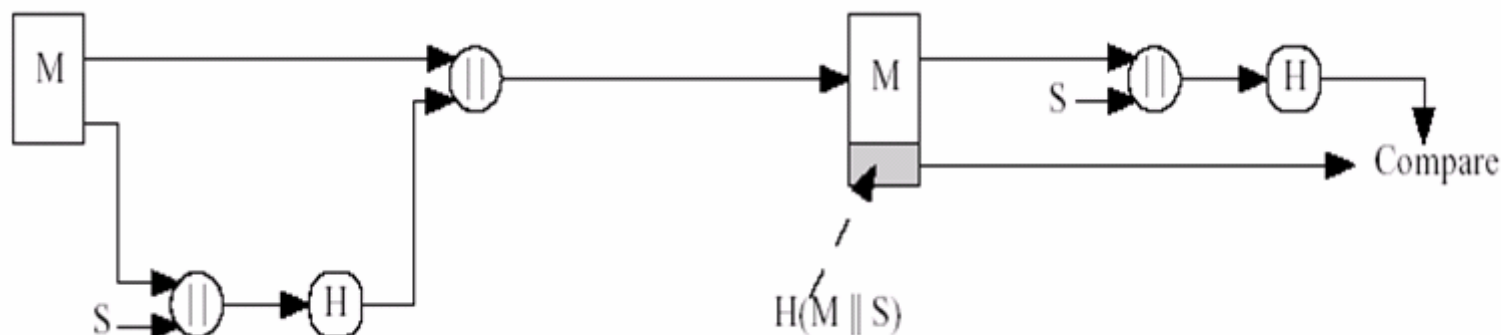


安全威胁二

任意找到两个消息 $x \neq x'$, 满足 $h(x)=h(x')$, 然后破译者把 x 给**Bob**且使他对 x 的摘要 $h(x)$ 签名, 从而得到 y , 那么 (x', y) 是一个有效的伪造。

强无碰撞: 散列函数 h 被称为是强无碰撞的, 是指在计算上几乎不可能找到相异的 x, x' 使得 $h(x)=h(x')$ 。

安全威胁三



秘密值**S**本身并不发送, 如果散列函数不是单向的, 攻击者截获到**M**和 **$H(M||S)$** . 然后通过某种逆变换获得 **$M||S$** , 因而攻击者就可以得到**S**.

单向: 称散列函数**h**为单向的, 是指计算**h**的逆函数 **h^{-1}** 在计算上不可行。

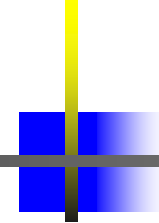


HASH 函数 $H = h(M)$

目的: “fingerprint” of message

满足:

- 1、 h 可以作用于一个任意长度的数据块;
- 2、 h 产生一个固定长度的输出;
- 3、对任意给定的 x , $h(x)$ 计算相对容易, 无论是软件还是硬件实现。
- 4、对任意给定码 H , 找到 x 满足 $h(x)=H$ 具有计算不可行性; (单向性)
- 5、对任意给定的 x , 找到满足 $h(y)=h(x)$ 的 $y \neq x$ 具有计算不可行性。
- 6、找到任意数据对 (x,y) , 满足 $h(x) = h(y)$ 是计算不可行的。



MD5 算法

- 输入：任意长度的消息
 - 输出：**128**位消息摘要
 - 处理：以**512**位输入数据块为单位
-
- **Merkle于1989年提出hash function模型**
 - **Ron Rivest于1990年提出MD4**
 - **1992年, MD5 (RFC 1321) developed by Ron Rivest at MIT**
 - **在最近数年之前,MD5是最主要的hash算法**
 - **现行美国标准SHA-1以MD5的前身MD4为基础**

Secure Hash Algorithm: SHA-1

- 输入：最大长度为 $2^{64}-1$ 位的消息；
 - 输出：**160**位消息摘要；
 - 处理：输入以**512**位数据块为单位处理；
-
- 1992年NIST制定了SHA(128位)
 - 1994年修改产生SHA-1(160位)
 - 1995年SHA-1成为新的标准,作为SHA-1(FIPS PUB 180-1)
 - 美国DSA签名方案使用的标准算法 ,Internet RFC3174
 - 基础是MD4



生日攻击

- 假定使用**64**位的散列码,是否安全?
- 如果采用传输加密的散列码和不加密的报文**M**,对手需要找到**M'**,使得 $H(M')=H(M)$,以便使用替代报文来欺骗接收者.
- 一种基于生日悖论的攻击可能做到这一点.

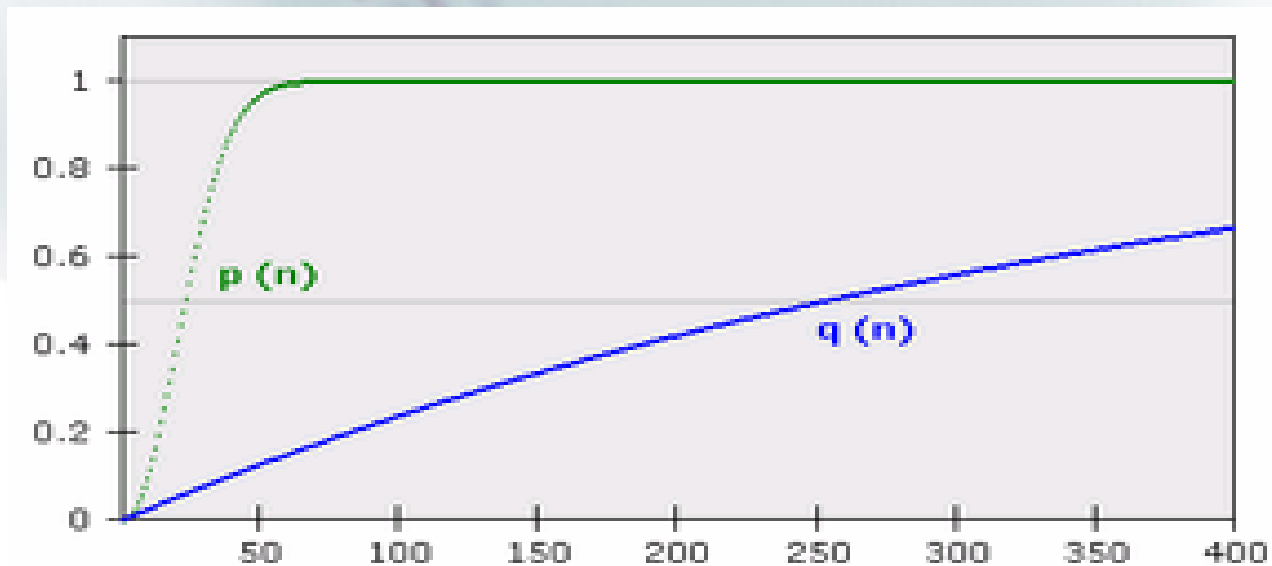


生日悖论

- 生日悖论是指，如果一个房间里有**23**个或**23**个以上的人，那么至少有两个人的生日相同的概率要大于**50%**。对于**60**或者更多的人，这种概率要大于**99%**。
- 这个数学事实与一般直觉相抵触。

理解生日悖论

- 的关键在于领会相同生日的搭配可以是相当多的。**23**个人可以产生 $23 \times 22/2 = 253$ 种不同的搭配，而这每一种搭配都有成功相等的可能。
- 换一个角度，如果你进入了一个有着**22**个人的房间，这时候只能产生**22**种不同的搭配。



和散列相关的问题

- 给定一个散列函数**H**，如果是**m**位输出的话，有 **2^m** 个可能的输出。
- 对**k**个随机输入，有两个输入的散列值相同的概率，经过计算后，近似等于 **$k/2^m$** ，想要概率大于**0.5**，只需要 **$k > 2^{m/2}$**
- 对长度为**m**位的散列码，共有 **2^m** 个可能的散列值，若要使任意的**x, y**， **$h(x)=h(y)$** 的概率为**0.5**，只需 **$k=2^{m-1}$**
- 如果让**H**接受随机输入集合**X**和随机输入**Y**，取什么值才能使两个集合间至少有一个匹配的概率 **>0.5** ？
 $k = 2^{m/2}$

Birthday Attacks: example

- **A**准备两份合同**M**和**M'**，**B**会同意**M**，但绝不会同意**M'**，因为**M'**会取走他的财产。
 - **A**对**M**和**M'**各做**32**处微小变化(保持各自的原意),分别产生 2^{32} 个**64**位hash值
 - 根据前面的结论,超过**0.5**的概率能找到一个**M**和一个**M'**,它们的hash值相同
 - **A**提交**M**,经**B**审阅后产生**64**位hash值并对该值签名,返回给**A**
 - **A**用**M'**替换**M**
- 对策: 必须足够长 (128\160)



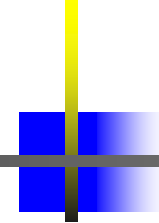
另一个例子

- **Alice** 向 **Bob**买东西。
- **Alice**向**Bob**提供了一份文档和验证文档的散列码，文档内容是 **Alice** 同意为每个小饰品付给 **Bob** 5 美元。**Bob** 认为散列就代表了文档，所以他只存储了密码散列（没有存文档）。
- **Alice** 希望只为每个小饰品付 1 美元，所以她创建第二份文档，所产生的散列值和 5 美元的那份相同，然后告上法庭，控诉 **Bob** 多收了她的钱。当她出庭时，**Bob** 将出示散列值，因为**Bob**相信 **Alice** 的文档不能散列出那个值，因为这不是她显示给他的原始文档。



Alice如何做

- **Alice** 使用生日攻击。她创建了两份文档，一份写着每个小饰品 **5** 美元，另一份则写着每个小饰品 **1** 美元。然后，在每份文档中，她标识出 **n** 处可以进行表面更改的地方（例如，那些可以用制表符取代空格的地方）。
- 好的 **n** 值通常是最终散列输出的位长度的一半加 **1**（所以如果我们指定散列输出的位长度为 **m**，则 **$n = m/2 + 1$** ）。



代价与时间

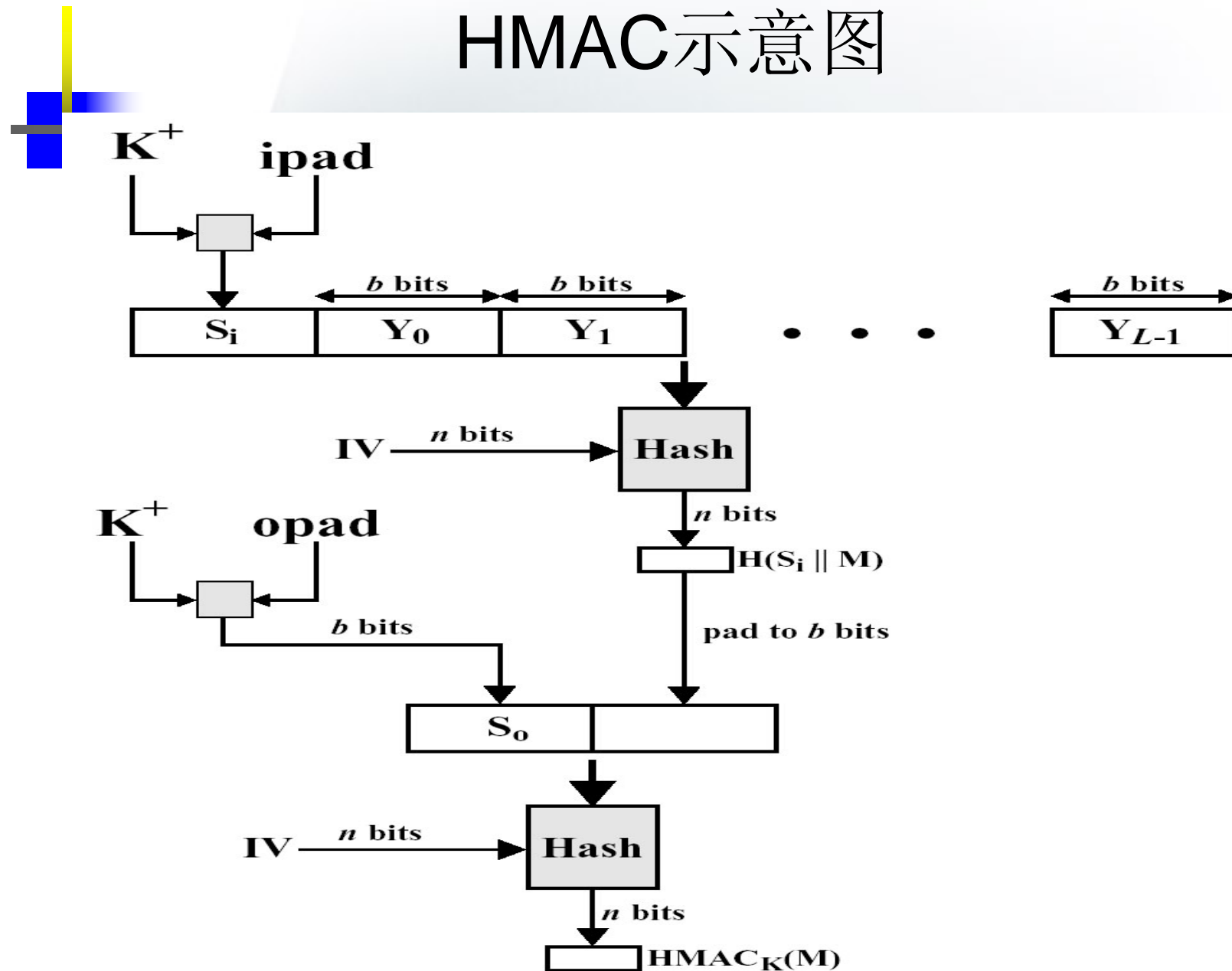
- 对于 **64** 位散列算法，她将在每份文档中选择 **33** 个地方。然后她反复尝试每份文档不同的排列，创建和存储散列值。一般来说，预计她将在散列了大约 $2^{m/2}$ 条消息之后找到散列出相同值的两份文档。
- 这比蛮力攻击要有效得多，如果使用蛮力攻击，预计她必须散列的消息数为 2^{m-1} 。如果 **Alice** 执行一次成功的蛮力攻击需要一百万年，那么她也许一周以内就可以完成一次成功的生日攻击。
- **Bob** 应该要求 **Alice** 使用一种算法，它所产生的摘要大小使得她不能在任何合理的时间之内完成生日攻击。



Keyed Hash Functions as MACs

- 使用**hash function** 而不是分组密码算法构造**MAC**
 - ❑ **hash functions** 一般速度快
 - ❑ 没有出口限制
- **original proposal:**
 $\text{KeyedHash} = \text{Hash}(\text{Key} \mid \text{Message})$
 - ❑ **some weaknesses were found with this**
- **eventually led to development of HMAC**
- **HMAC作为RFC2104并在SSL中使用**

HMAC示意图



符号

- **H** = 嵌入散列函数 (**MD5,SHA-1,RIPEMD-160**)
- **M** = 消息 (包括散列函数所需填充位)
- **Y_i** = **M**的第*i*个数据块, **$0 \leq i \leq L-1$**
- **L** = **M**的数据块数
- **b** = 数据块的位数
- **n** = 嵌入散列函数产生的散列码长度位数
- **K** = 保密密钥。如果密钥长度大于**b**,则密钥送入散列函数
- 形成一个**n**位的密钥; 推荐程度大于等于**n**
- **K^+** = **K**在左部添加**0**使得其长度为**b**位
- **ipad** = **00110110**重复 **b/8**次
- **opad** = **01011010**重复**b/8**次

HMAC计算过程

- ① 对密钥K左边补0以产生一个hash用块 K^+
- ② K^+ 每个字节与ipad(00110110)作XOR以产生 S_i
- ③ 对 $(S_i || M)$ 进行hash
- ④ K^+ 每个字节与opad(01011010)作XOR以产生 S_o
- ⑤ $HMACK = H[K^+ \oplus opad) || H(K^+ \oplus ipad) || M]$



PGP

- **PGP - Pretty Good Privacy**

- ☐ 作者: **Phil Zimmermann**
- ☐ 提供可用于电子邮件和文件存储应用的保密与鉴别服务。
- ☐ **pgp**已成为**Internet** 标准文档 (**RFC 3156**)
- ☐ 免费、可用于多平台。**DOS/Windows、Unix、Macintosh**
- ☐ 选用算法的生命力和安全性公众认可。
- ☐ 具有广泛的可用性



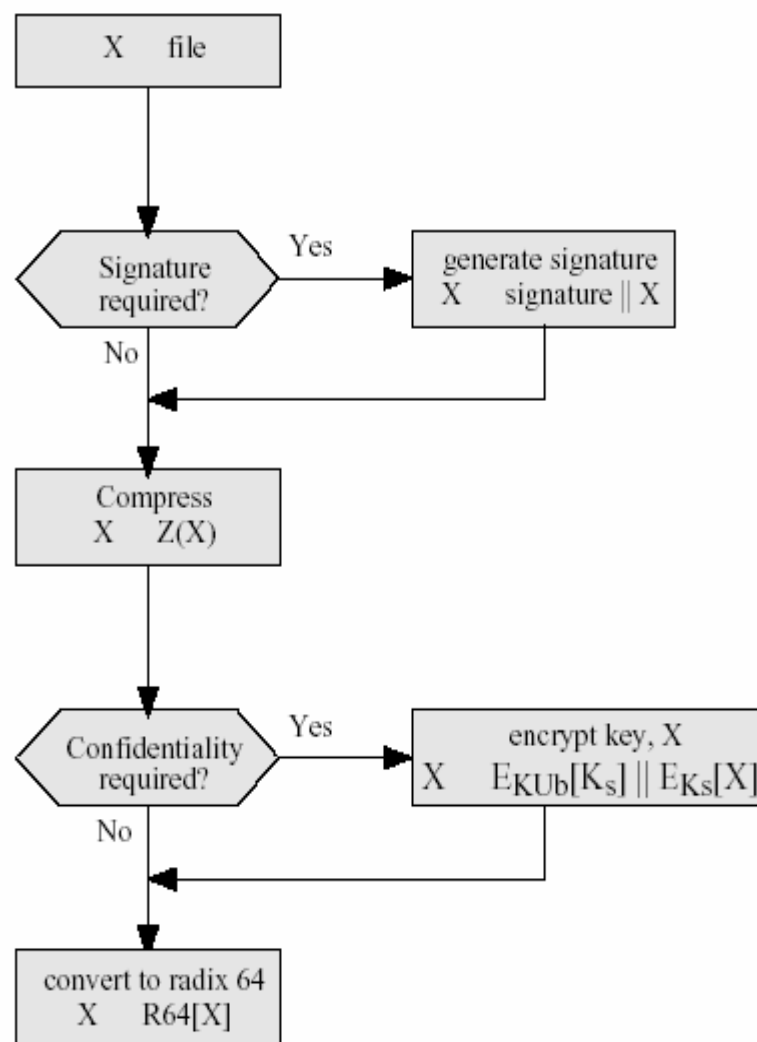
数据压缩

- 压缩的位置：发生在签名后、加密前。
- 压缩之前生成签名：
 - （1）验证时无须压缩
 - （2）压缩算法的多样性
- 在加密前压缩：压缩的报文更难分析
 - 对邮件传输或存储都有节省空间的好处。

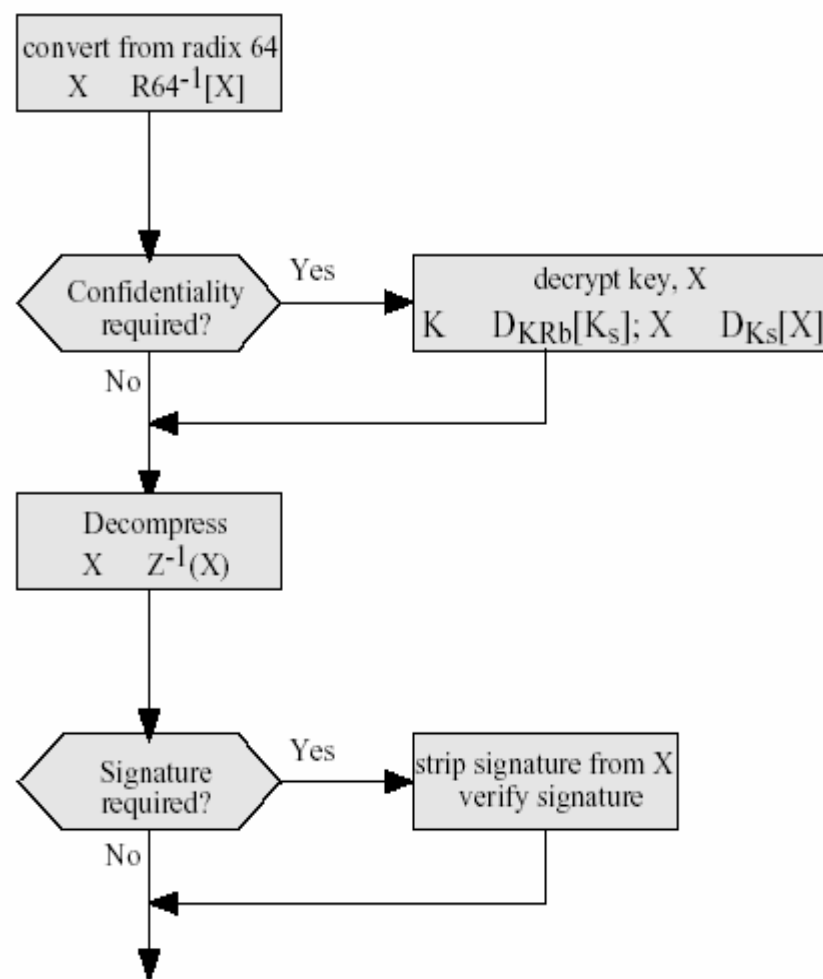


E-mail兼容性

- 加密后是任意的**8**位字节，需要转换到**ASCII**格式。
- **Radix64**将**3**字节输入转换到**4**个**ASCII**字符，并带**CRC**校验。
- 长度扩大**33%**
- 与压缩综合后，长度为：
□ $1.33 \times 0.5 \times M = 0.665 \times M$



(a) Generic Transmission Diagram (from A)



(b) Generic Reception Diagram (to B)

PGP消息的传送与接收



加密密钥和密钥环

- **PGP**使用四种类型的密钥：一次性会话常规密钥，公钥，私钥，基于口令短语的常规密钥。
- 需要某种手段来标识具体的密钥。因为一个用户拥有多个公钥/私钥对。接收者如何知道发送者是用哪个公钥来加密会话密钥？
 - ❑ 将公钥与消息一起传送。
 - ❑ 将一个标识符与一个公钥关联。对一个用户来说做到一一对应。
- 每个**PGP**实体需要维护一个文件保存其公钥私钥对，和一个文件保存通信对方的公钥。

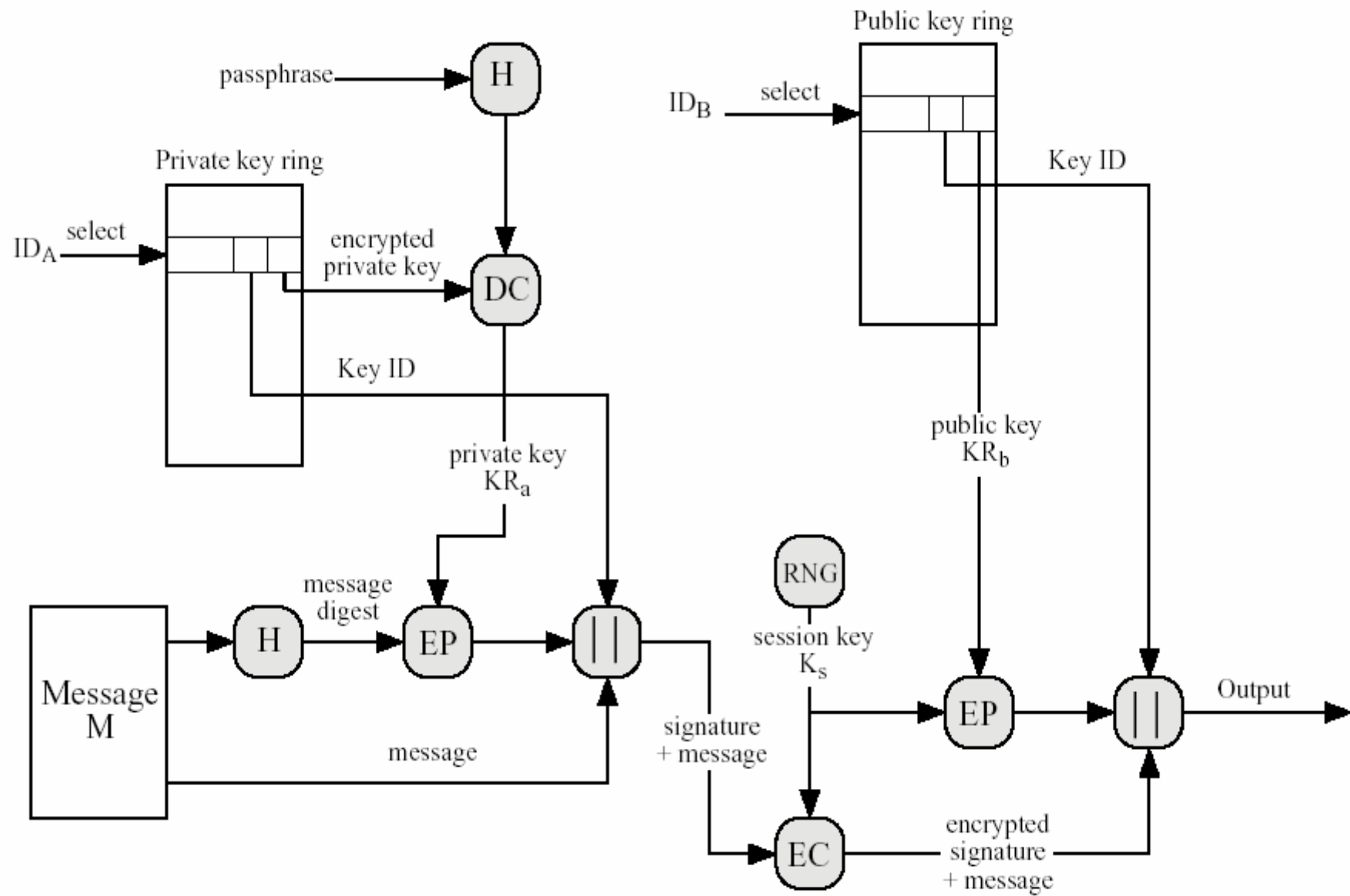


私钥说明

- **UserID**: 通常是用户的邮件地址。也可以是一个名字，或重用一個名字多次。
- **Private Key**: 使用**CAST-128(或IDEA或3DES)**加密。过程如下：
 - ❑ 当系统用**RSA**生成一个新的公钥/私钥对时，要求用户输入口令短语。对该短语使用**SHA-1**生成一个**160**位的散列码后，销毁该短语。
 - ❑ 系统用其中**128**位作为密钥用**CAST-128**加密私钥，然后销毁这个散列码，并将加密后的私钥存储。
 - ❑ 当用户要访问私钥时，必须提供口令短语。**PGP**将检索出加密的私钥，生成散列码，解密私钥。

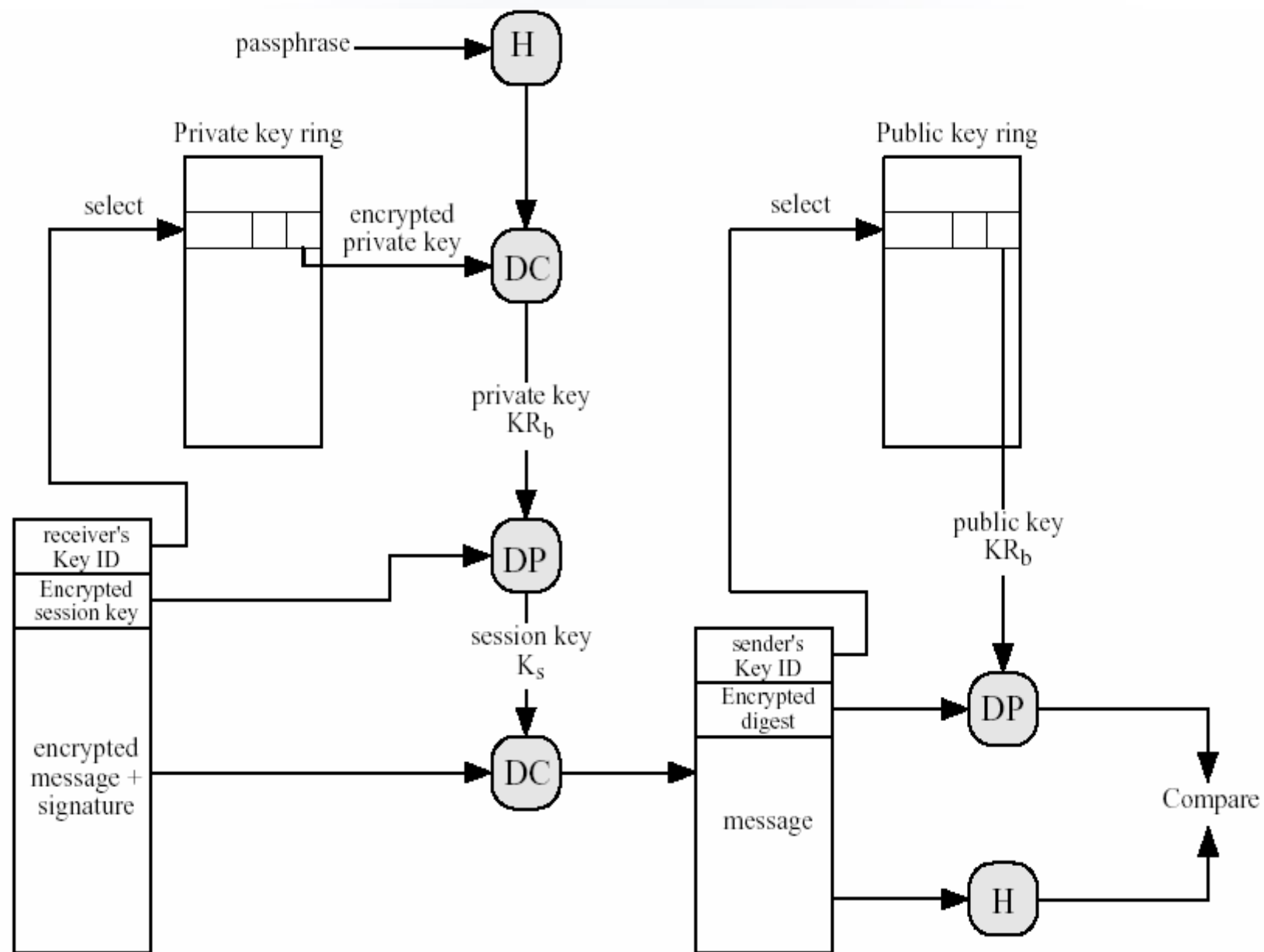
PGP — 发送方处理消息的过程

- 签名：
 - ❑ 从私钥环中得到私钥，利用**userid**作为索引
 - ❑ **PGP**提示输入口令短语，恢复私钥
 - ❑ 构造签名部分
- 加密：
 - ❑ **PGP**产生一个会话密钥，并加密消息
 - ❑ **PGP**用接收者**userid**从公钥环中获取其公钥
 - ❑ 构造消息的会话密钥和**KeyID**部分

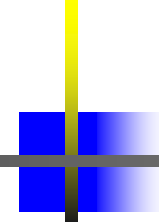


PGP — 接收方处理消息的过程

- 解密消息
 - ❑ PGP用消息的会话密钥部分中的**KeyID**作为索引，从私钥环中获取私钥
 - ❑ PGP提示输入口令短语，恢复私钥
 - ❑ PGP恢复会话密钥，并解密消息
- 验证消息
 - ❑ PGP用消息的签名部分中的**KeyID**作为索引，从公钥环中获取发送者的公钥
 - ❑ PGP恢复被传输过来的消息摘要
 - ❑ PGP对于接收到的消息作摘要，并与上一步的结果作比较



PGP报文的接收



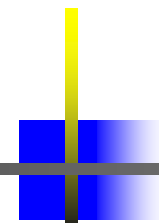
公钥管理问题

- 由于**PGP**重在广泛地在正式或非正式环境下应用，没有建立严格的公钥管理模式。
- 如果**A**的公钥环上有一个从**BBS**上获得**B**发布的公钥，但已被**C**替换成**C**的公钥，这是就存在两条通道。**C**可以向**A**发信并冒充**B**的签名，**A**以为是来自**B**；**A**与**B**的任何加密消息**C**都可以读取。
- 为了防止**A**的公钥环上包含错误的公钥，有若干种方法可用于降低这种风险。



验证公钥

- 1、物理上得到**B**的公钥。
- 2、通过电话验证公钥。
 - **B**将其公钥**email**给**A**，**A**可以用**PGP**对该公钥生成一个**160**位的**SHA-1**摘要，并以**16**进制显示。这一特点称作密钥的“指纹”。然后**A**打电话给**B**，让**B**在电话中对证“指纹”。如果双方一致，则该公钥被认可。



数字证书

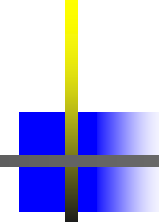
3、数字证书

□从双方都信任的个体**D**处获得**B**的公钥。

D是介绍人，生成一个签名的证书。其中包含**B**的公钥，密钥生成时间。

D对该证书生成一个**SHA-1**摘要，用其私钥加密这个摘要，并将其附加在证书后。

这个签名的证书可以由**B**或**D**直接发给**A**，也可以贴到公告牌上。



数字证书

CA

❑ 从一个信任的**CA**中心得到**B**的公钥。

PGP支持两种形式的证书

❑ **PGP**证书

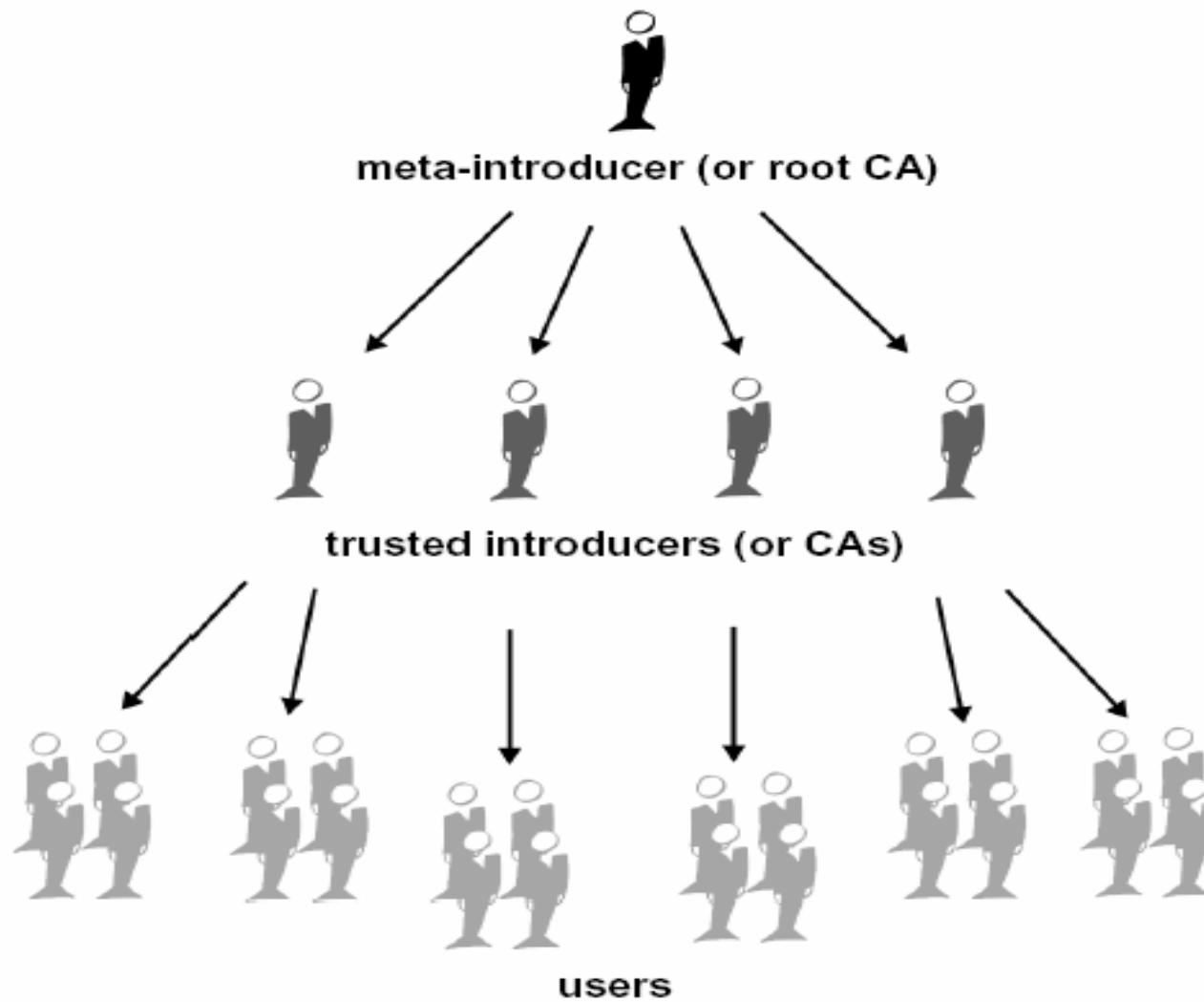
❑ **X.509**证书

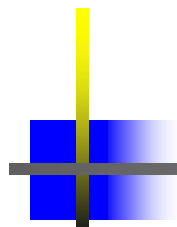


信任模型

- 不可能由一个**CA**来证明所有的证书，所有存在三种信任模型
 - ☐ **Direct Trust** 用户之间直接验证密钥
 - ☐ **Hierarchical Trust** 层次信任
 - ☐ **A Web of Trust** 网状信任

Hierarchical Trust





Q&A

谢谢！