东南大学网络空间安全学院
密码学与安全协议

第六讲 消息认证和散列函数

黄　杰
信息安全研究中心

本讲内容

- 消息认证和hash函数
  - 消息加密函数(Message Encryption)
  - 消息认证码MAC (Message Authentication Code)
  - 散列函数(Hash Function)
- Hash算法
  - MD5
  - SHA-1
  - HMAC
- Hash算法的攻击方法
- Hash算法攻击的理论基础

消息认证与数字签名

- 问题
  - 消息认证和数字签名的区别

安全攻击

- 泄密：将消息透露给非授权第三方。
- 传输分析：分析通信双方的通信模式。

破坏机密性

- 伪装：伪造信息，声称来自合法实体。
- 内容修改：对消息插入、删除、转化或修改。
- 顺序修改：修改顺序。
- 计时修改：对消息延时或重放。

破坏完整性

- 发送方否认：发送方否认发送过消息。
- 接收方否认：接收方否认收到过消息。

破坏抗抵赖性

认证函数

- 可用来产生认证符的函数分为三类：

(1) 消息加密函数(Message Encryption)

用完整信息的密文作为对信息的认证。

(2) 消息认证码MAC (Message Authentication Code)

MAC是消息和密钥的函数，产生一个固定长度的值作为认证标识

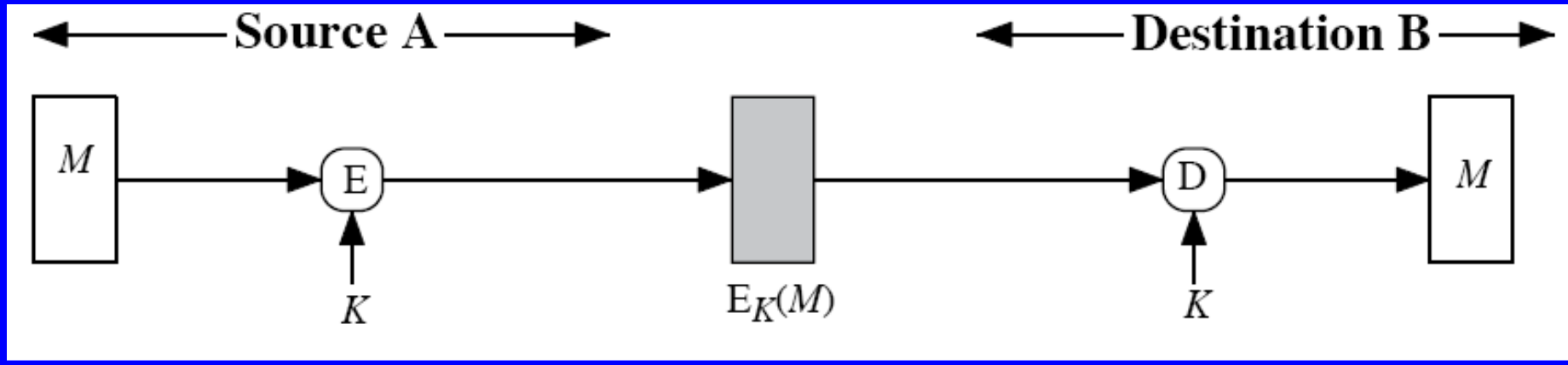
(3) 散列函数(Hash Function)

是一个公开的函数，它将任意长的信息映射成一个固定长度的信息。

消息加密

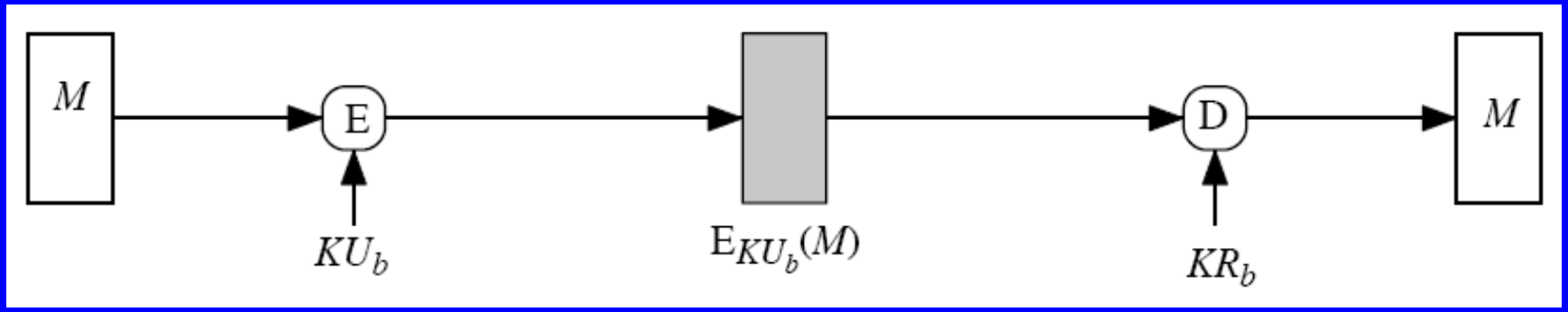
- 消息的自身加密可以作为一个认证的度量。
- 对称密钥模式和公开密钥模式有所不同。

对称加密

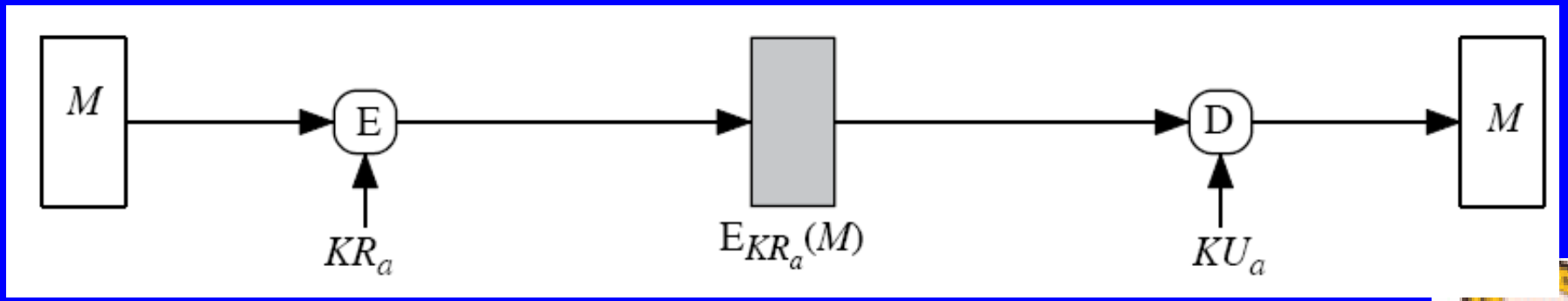


- 如果明文具有一定的语法结构，接收方可以判断解密后明文的合法性，从而确认消息来自发送方而且中间未受到篡改。
- 如果明文为二进制文件，则难以判断解密后的消息是正确的明文。

公钥加密

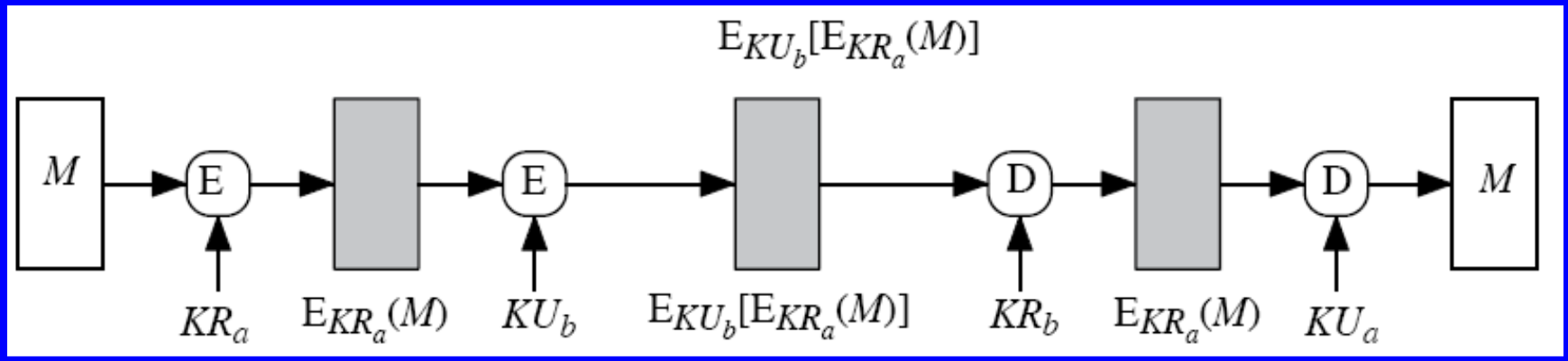


保密性



认证和签名

公钥加密



- 既能实现保密性，又能完成认证和签名。
- 一次通信中要进行四次复杂的公钥算法。

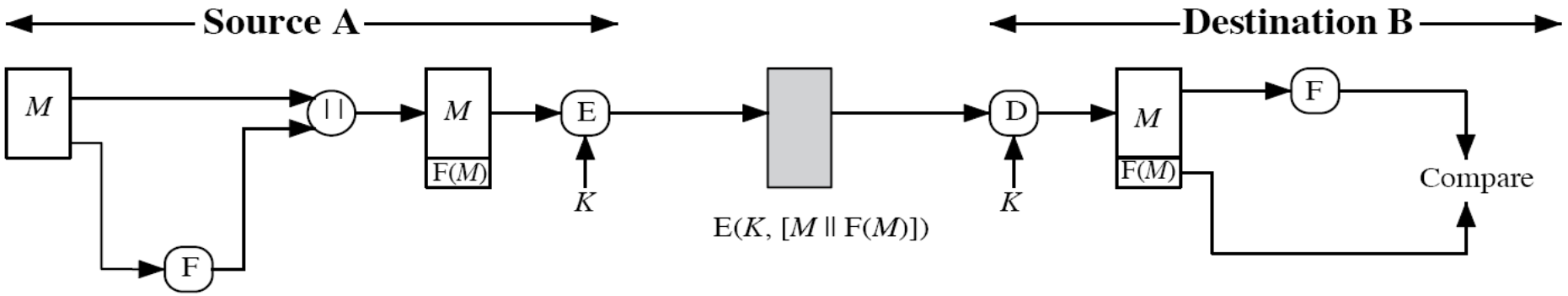
消息认证码

- 原因
  - 有许多应用将同一消息广播给很多接受者；
  - 通信某一方处理的负荷很大，只能验证消息；
  - 明文形式的计算机程序进行认证；

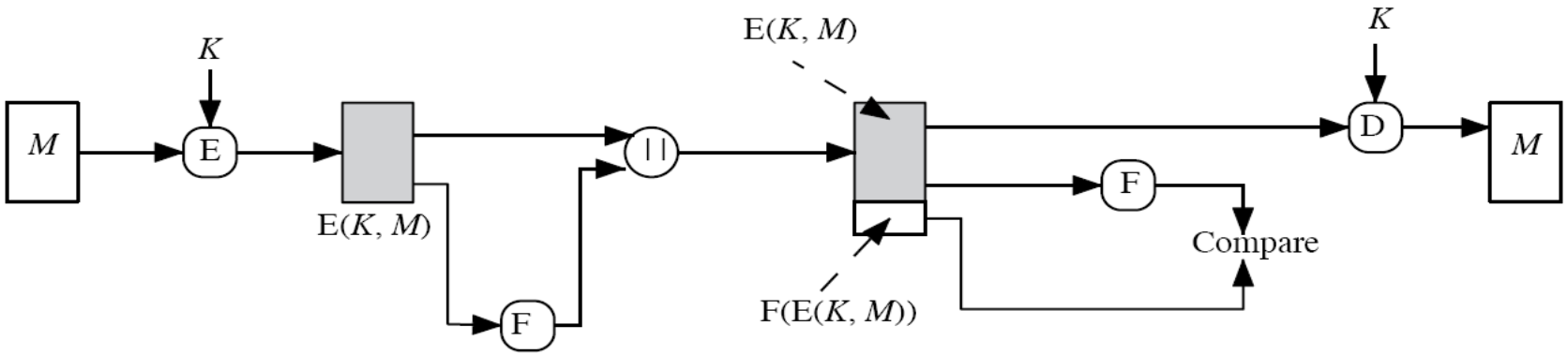
  - 有些应用不关心保密性，而关心消息认证；
  - 将认证和保密性分开；
  - 解密的消息不再受到保护，但MAC可以提供长期的认证。

消息认证码

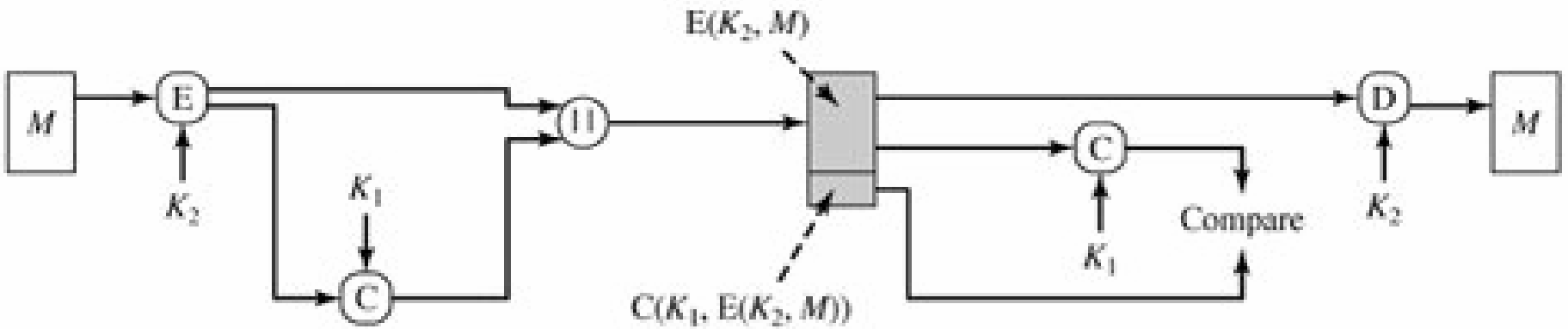
- 利用密钥生成一个固定长度的短数据块，称为消息认证码MAC，并将MAC附加在消息之后。接收方通过计算MAC来认证该消息。
- 计算公式：$MAC=C_K(M)$
  - M：长度可变的消息
  - K：收发双方共享的密钥
  - $C_K(M)$：定长的认证符



(a) Internal error control



(b) External error control



消息认证码的讨论

- 保密性与真实性是两个不同的概念
- 从根本上说,信息加密提供的是保密性而非真实性
- 加密代价大(公钥算法代价更大)
- 某些信息只需要真实性,不需要保密性
  - 广播的信息难以使用加密(信息量大)
  - 网络管理信息等只需要真实性
  - 政府/权威部门的公告

hash函数

- 形式：h=H(M)
- 它以任意长度的消息做自变量，产生规定长度的消息摘要。
- 对于任意x，计算H(x)是容易的，软硬件均可实现。
- 性质1(抗弱碰撞)，指对给定消息x，找到y，y≠x，且H(x)= H(y)在计算上是不可行的。

- 性质2(抗强碰撞)，指满足H(x)= H(y)的（x，y）在计算上是不可行的。

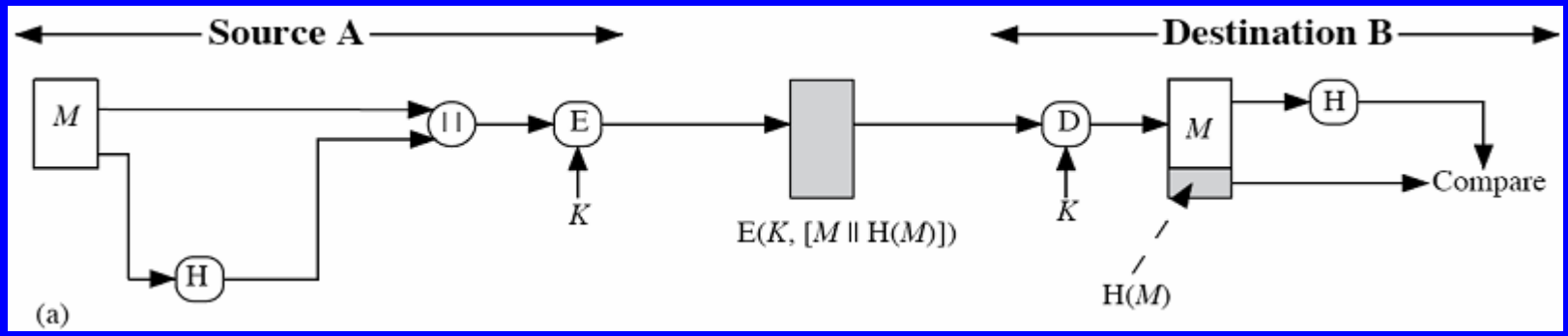
  注：抗强碰撞自然含抗弱碰撞！

- 性质3(单向的)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函数的特点

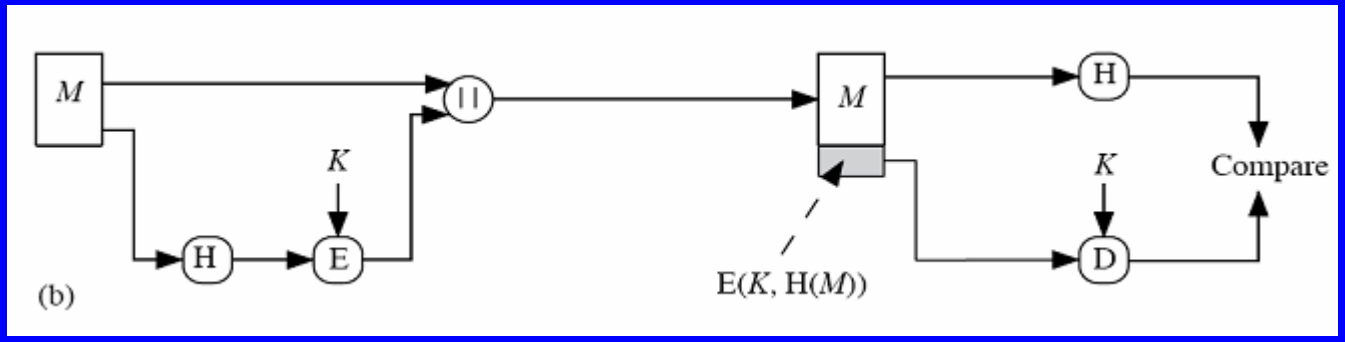
- 将一个长度可变的消息M转换为一个固定长度的输出，称之为散列码或摘要
- 与MAC的区别：没有使用密钥，函数输入参数只有一个
- 可以检测出消息是否发生变化

- 和加密结合可以用于认证
  - 加密消息及散列码：E(K,M||H(M))



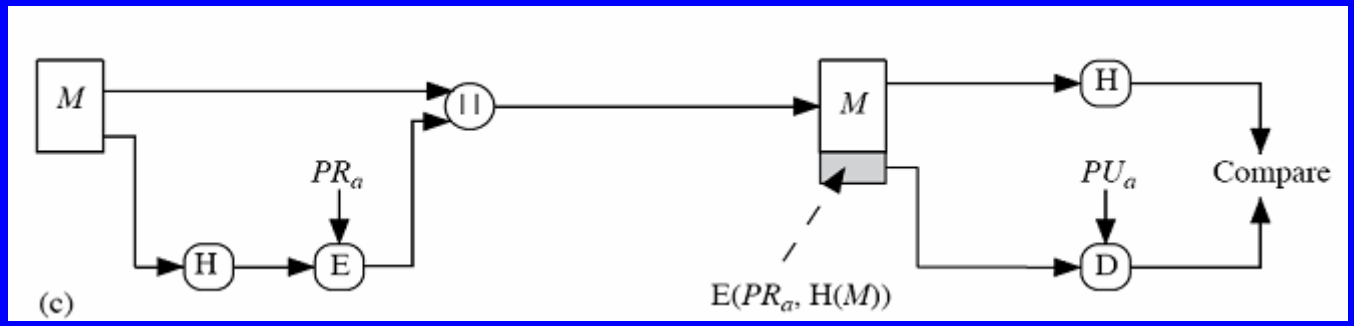
- 对称加密散列码：M || E(K, H(M))

  E(K, H(M)) 即为一MAC函数

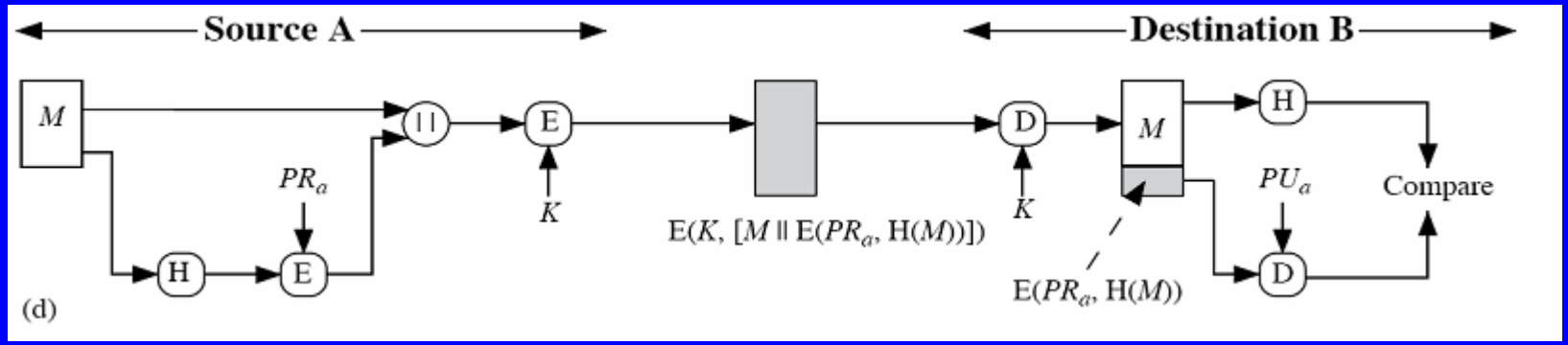


- 公钥加密散列码：M || E(PRa, H(M))

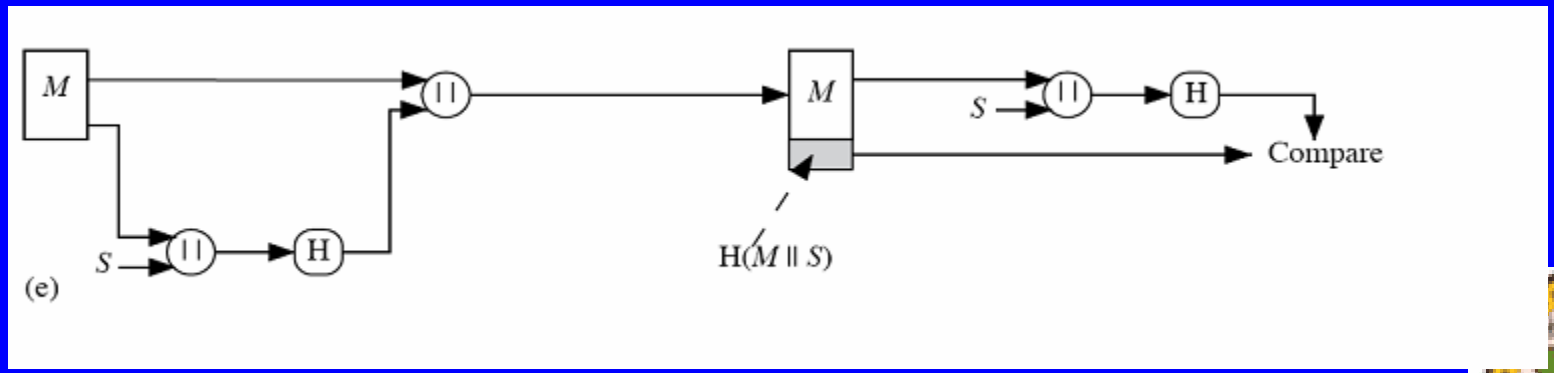
  典型的数字签名方案



- 对称密码加密签名：E(K, M || E(PRa, H(M))

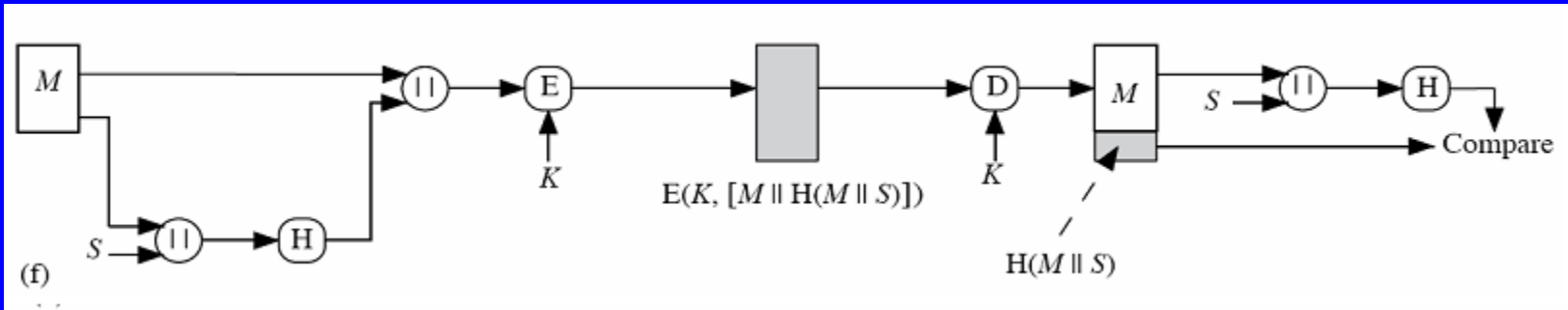


- 带秘密值的散列码：M || H(M || S)



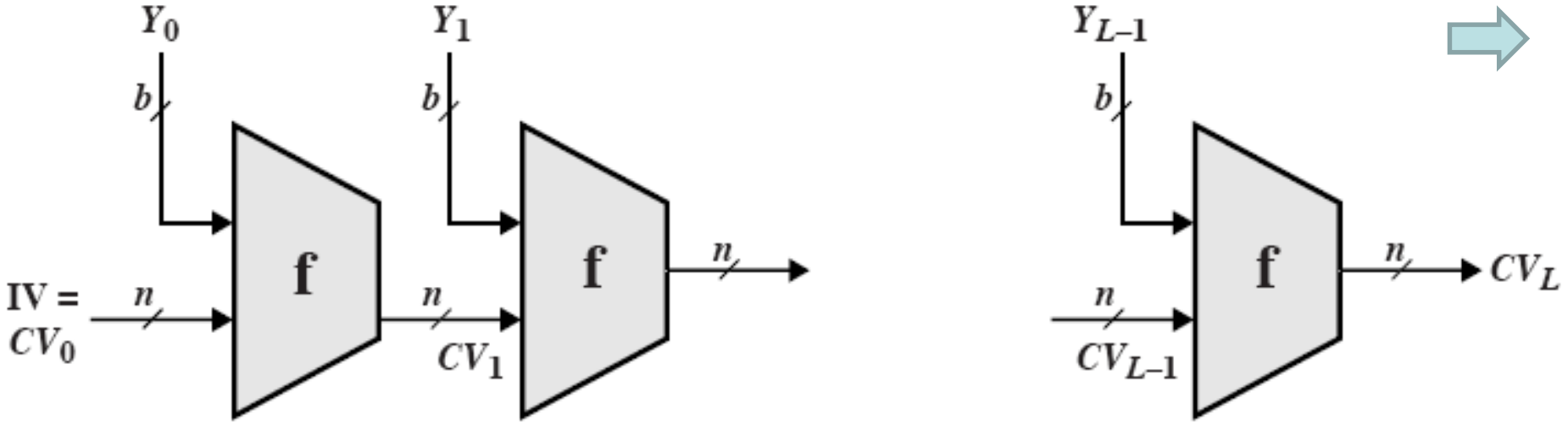
- 对称密码加密带秘密值的散列码

  E(K, M || H(M || K))



HASH算法

Hash函数的一般结构



IV	=	Initial value
CV	=	chaining variable
Y_i	=	ith input block
f	=	compression algorithm
L	=	number of input blocks
n	=	length of hash code
b	=	length of input block

计算过程：

CV_0=IV=初始n位值

CV_i=f（CV_{i-1}，Y_{i-1}），1<=i<=L

H(M)=CV_L

MD5消息摘要算法

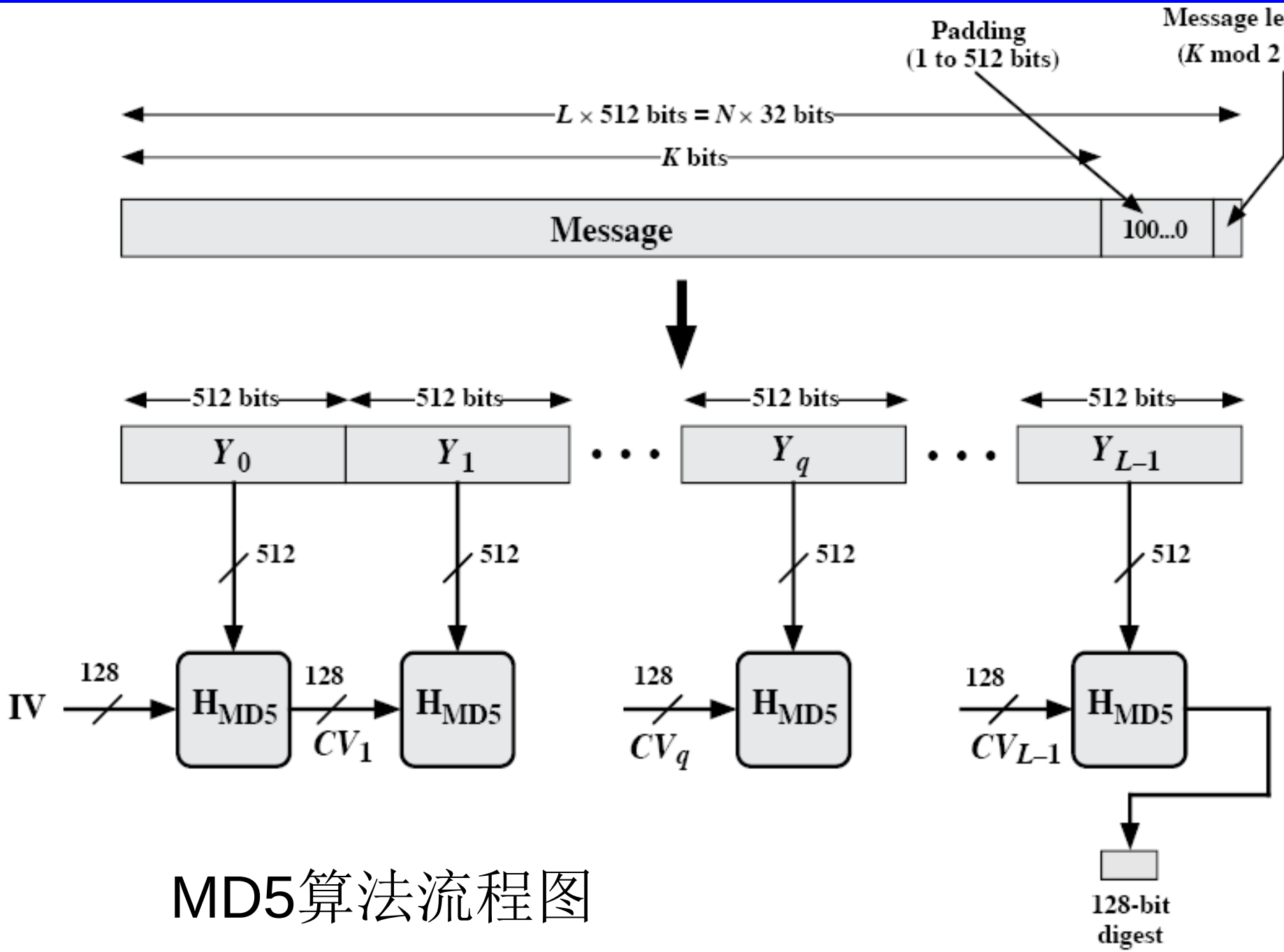
- Merkle于1989年提出hash function一般模型
- Ron Rivest于1990年提出MD4
- 1992年，Ron Rivest 开发了MD5 (RFC 1321)
- MD5把数据分成512-bit块，MD5的hash值是128-bit
- 在最近数年之前，MD5是最主要的hash算法
- 现行美国标准SHA-1以MD5的前身MD4为基础

MD5算法流程

- 填充消息
- 附加消息长度
- 初始化缓冲区
- 分组处理消息
- 输出消息摘要

MD5算法流程

- 填充消息
  - $M \rightarrow M_1$
  - $|M_1| \equiv 448 \bmod 512$
  - $|M_1| > |M|$:如果$|M| \equiv 448 \bmod 512$,则$|M_1| = |M|+512$
  - 填充内容: 100…0
- 附加消息长度
  - $M_1 \rightarrow M_2$，消息长度以64位表示
  - 若$|M_1| > 2^{64}$,则仅取低64位
  - $|M_2|$为512的倍数，可分为长度为512的L个分组 $Y_0, Y_1, \ldots, Y_{L-1}$


Padding
(1 to 512 bits)
Message length
(K mod 2 64)
L × 512 bits = N × 32 bits
K bits
Message
100...0
512 bits
512 bits
512 bits
512 bits
Y 0
Y 1
Y q
Y L–1
512
512
512
512
IV
128
H MD5
128
CV 1
128
CV q
128
CV L–1
128-bit
digest

MD5算法流程图

初始化缓冲区

- 缓冲区长度：128位
- 用四个32位的寄存器(A，B，C，D)表示

  A = 01 23 45 67 (0x67452301)

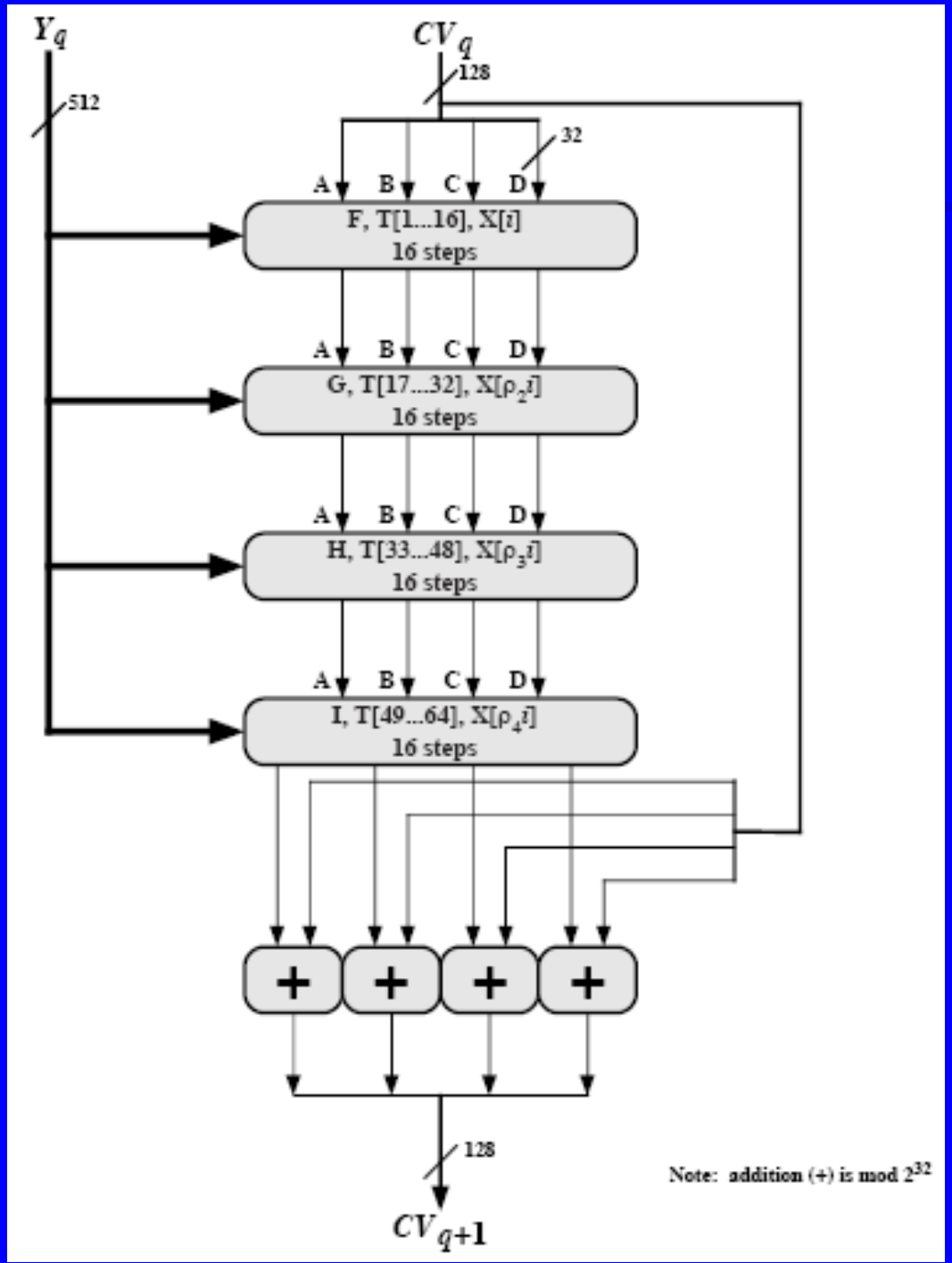
  B = 89 AB CD EF (0xEFCDAB89)

  C = FE DC BA 98 (0x98BADCFE)

  D = 76 54 32 10 (0x10325476)
- 括号内为每个寄存器的实际数值，前面为实际的存储顺序。

MD5算法流程

- 分组处理消息
  - $CV_0=IV$
  - $CV_i=H_{MD5}(CV_{i-1},Y_i)$
- 输出消息摘要
  - $MD = CV_L$



一个消息分组的处理过程：

算法的核心是压缩函数，由四轮运算组成，

每轮进行16步迭代。

每轮使用不同的逻辑函数，记为F，G，H，I。

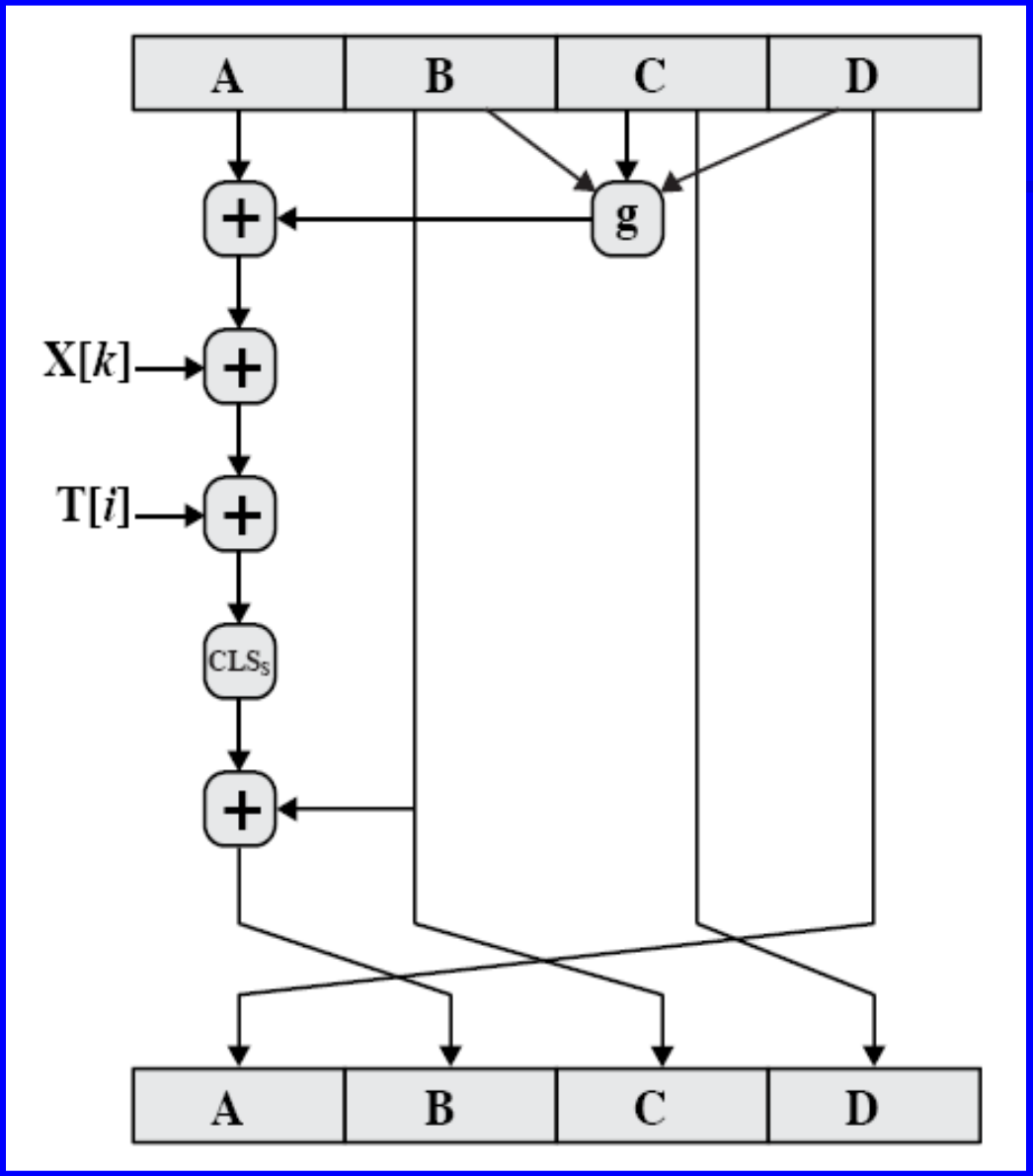
T是事先产生的表格，包含64项，每项32位。

512位消息在每轮分为16个分组，每个分组为32位。

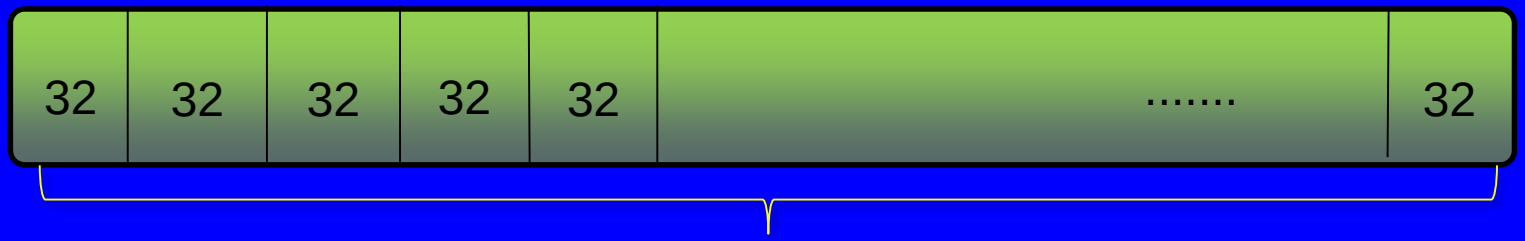
MD5压缩函数

- 每一轮包含对缓冲区ABCD的16步操作所组成的一个序列。
  - a←b + ((a + g(b,c,d) + X[k] +T[i])<<<s)

- 其中，
  - a,b,c,d = 缓冲区的四个字，以一个给定的次序排列；
  - g = 基本逻辑函数F,G,H,I之一；
  - <<<s = 对32位字循环左移s位
  - X[k] = M[q×16 + k] = 在第q个512位数据块中的第k个32位字
  - T[i] = 表T中的第i个32位字；
  - + = 模2^{32}的加；



$a \leftarrow b + ((a + g(b,c,d) + X[k] + T[i]) <<< s)$



四个非线性函数

1. F(X ,Y ,Z) = (X & Y) | ((~X) & Z)
2. G(X ,Y ,Z) = (X & Z) | (Y & (~Z))
3. H(X ,Y ,Z) =X ^ Y ^ Z
4. I(X ,Y ,Z) =Y ^ (X | (~Z))

注：（&是与（And），|是或（Or），~是非（Not），^是异或（Xor））

四轮操作

FF(a,b,c,d,Mj,s,ti)

a=b+((a+F(b,c,d)+Xj+ti)<<<s)

GG(a,b,c,d,Mj,s,ti)

a=b+((a+G(b,c,d)+Xj+ti)<<<s)

HH(a,b,c,d,Mj,s,ti)

a=b+((a+H(b,c,d)+Xj+ti)<<<s)

II(a,b,c,d,Mj,s,ti)

a=b+((a+I(b,c,d)+Xj+ti)<<<s)

第一轮

FF(a ,b ,c ,d ,M0 ,7 ,0xd76aa478)
FF(d ,a ,b ,c ,M1 ,12 ,0xe8c7b756)
FF(c ,d ,a ,b ,M2 ,17 ,0x242070db)
FF(b ,c ,d ,a ,M3 ,22 ,0xc1bdceee)
FF(a ,b ,c ,d ,M4 ,7 ,0xf57c0faf)
FF(d ,a ,b ,c ,M5 ,12 ,0x4787c62a)
FF(c ,d ,a ,b ,M6 ,17 ,0xa8304613)
FF(b ,c ,d ,a ,M7 ,22 ,0xfd469501)
FF(a ,b ,c ,d ,M8 ,7 ,0x698098d8)
FF(d ,a ,b ,c ,M9 ,12 ,0x8b44f7af)
FF(c ,d ,a ,b ,M10 ,17 ,0xffff5bb1)
FF(b ,c ,d ,a ,M11 ,22 ,0x895cd7be)
FF(a ,b ,c ,d ,M12 ,7 ,0x6b901122)
FF(d ,a ,b ,c ,M13 ,12 ,0xfd987193)
FF(c ,d ,a ,b ,M14 ,17 ,0xa679438e)
FF(b ,c ,d ,a ,M15 ,22 ,0x49b40821)

第二轮

GG(a ,b ,c ,d ,M1 ,5 ,0xf61e2562)
GG(d ,a ,b ,c ,M6 ,9 ,0xc040b340)
GG(c ,d ,a ,b ,M11 ,14 ,0x265e5a51)
GG(b ,c ,d ,a ,M0 ,20 ,0xe9b6c7aa)
GG(a ,b ,c ,d ,M5 ,5 ,0xd62f105d)
GG(d ,a ,b ,c ,M10 ,9 ,0x02441453)
GG(c ,d ,a ,b ,M15 ,14 ,0xd8a1e681)
GG(b ,c ,d ,a ,M4 ,20 ,0xe7d3fbc8)
GG(a ,b ,c ,d ,M9 ,5 ,0x21e1cde6)
GG(d ,a ,b ,c ,M14 ,9 ,0xc33707d6)
GG(c ,d ,a ,b ,M3 ,14 ,0xf4d50d87)
GG(b ,c ,d ,a ,M8 ,20 ,0x455a14ed)
GG(a ,b ,c ,d ,M13 ,5 ,0xa9e3e905)
GG(d ,a ,b ,c ,M2 ,9 ,0xfcefa3f8)
GG(c ,d ,a ,b ,M7 ,14 ,0x676f02d9)
GG(b ,c ,d ,a ,M12 ,20 ,0x8d2a4c8a)

第三轮

HH(a ,b ,c ,d ,M5 ,4 ,0xfffa3942)
HH(d ,a ,b ,c ,M8 ,11 ,0x8771f681)
HH(c ,d ,a ,b ,M11 ,16 ,0x6d9d6122)
HH(b ,c ,d ,a ,M14 ,23 ,0xfde5380c)
HH(a ,b ,c ,d ,M1 ,4 ,0xa4beea44)
HH(d ,a ,b ,c ,M4 ,11 ,0x4bdecfa9)
HH(c ,d ,a ,b ,M7 ,16 ,0xf6bb4b60)
HH(b ,c ,d ,a ,M10 ,23 ,0xbebfbc70)
HH(a ,b ,c ,d ,M13 ,4 ,0x289b7ec6)
HH(d ,a ,b ,c ,M0 ,11 ,0xeaa127fa)
HH(c ,d ,a ,b ,M3 ,16 ,0xd4ef3085)
HH(b ,c ,d ,a ,M6 ,23 ,0x04881d05)
HH(a ,b ,c ,d ,M9 ,4 ,0xd9d4d039)
HH(d ,a ,b ,c ,M12 ,11 ,0xe6db99e5)
HH(c ,d ,a ,b ,M15 ,16 ,0x1fa27cf8)
HH(b ,c ,d ,a ,M2 ,23 ,0xc4ac5665)

第四轮

II(a ,b ,c ,d ,M0 ,6 ,0xf4292244)
II(d ,a ,b ,c ,M7 ,10 ,0x432aff97)
II(c ,d ,a ,b ,M14 ,15 ,0xab9423a7)
II(b ,c ,d ,a ,M5 ,21 ,0xfc93a039)
II(a ,b ,c ,d ,M12 ,6 ,0x655b59c3)
II(d ,a ,b ,c ,M3 ,10 ,0x8f0ccc92)
II(c ,d ,a ,b ,M10 ,15 ,0xffeff47d)
II(b ,c ,d ,a ,M1 ,21 ,0x85845dd1)
II(a ,b ,c ,d ,M8 ,6 ,0x6fa87e4f)
II(d ,a ,b ,c ,M15 ,10 ,0xfe2ce6e0)
II(c ,d ,a ,b ,M6 ,15 ,0xa3014314)
II(b ,c ,d ,a ,M13 ,21 ,0x4e0811a1)
II(a ,b ,c ,d ,M4 ,6 ,0xf7537e82)
II(d ,a ,b ,c ,M11 ,10 ,0xbd3af235)
II(c ,d ,a ,b ,M2 ,15 ,0x2ad7d2bb)
II(b ,c ,d ,a ,M9 ,21 ,0xeb86d391)

1、常数ti是2^32*abs(sin(i))的整数部分，i取值从1到64，单位是弧度。
2、所有这些完成之后，将a、b、c、d分别在原来基础上再加上A、B、C、D。
即a = a + A，b = b + B，c = c + C，d = d + D。
3、然后用下一分组数据继续运行以上算法。

第一轮运算

- For(k = 0; k < 16; ++k) {

$A \leftarrow B + ((A + g_1(B,C,D) + X[\rho_1(k)] + T[16\times 0 + k + 1]) <<< s_1[k \bmod 4])$

$(A,B,C,D) \leftarrow (A,B,C,D) >>> 32$

}

- $g_1(B,C,D) = (B \,\&\, C) \mid (B \,\&\, D)$
- $\rho_1(k) = k, \qquad 0 \le k < 16$
- $s_1[0\ldots3] = [7,12,17,22]$

第二轮运算

- For(k = 0; k < 16; ++k) {

$A \leftarrow B + ((A + g_2(B,C,D) + X[\rho_2(k)] + T[16\times1+k+1]) <<< s_2[k \bmod 4])$

$(A,B,C,D) \leftarrow (A,B,C,D) >>> 32$

}

- $g_2(B,C,D) = (B \,\&\, D) \mid (C \,\&\, D)$
- $\rho_2(k) = (1+5k) \bmod 16, \quad 0 \le k < 16$
- $s_2[0\ldots3] = [5,9,14,20]$

第三轮运算

- For(k = 0; k < 16; ++k) {

$A \leftarrow B + ((A + g_3(B,C,D) + X[\rho_3(k)] + T[16\times 2 + k + 1]) <<< s_3[k \bmod 4])$

$(A,B,C,D) \leftarrow (A,B,C,D) >>> 32$

}

- $g_3(B,C,D) = B \oplus C \oplus D$
- $\rho_3(k) = (5+3k) \bmod 16, \quad 0 \le k < 16$
- $s_3[0\ldots3] = [4,11,16,23]$

第四轮运算

- For(k = 0; k < 16; ++k) {

$A \leftarrow B + ((A + g_4(B,C,D) + X[\rho_4(k)] + T[16\times 3 + k + 1]) <<< s_4[k \bmod 4])$

$(A,B,C,D) \leftarrow (A,B,C,D) >>> 32$

}

- $g_4(B,C,D) = C \oplus (B \mid D)$
- $\rho_4(k) = 7k \bmod 16, \quad 0 \le k < 16$
- $s_4[0\ldots3] = [6,10,15,21]$

MD5的强度

特点：

- Hash函数的每一位都是输入的每一位的函数。
- 迭代函数的复杂性使得输出对输入的依赖非常小。
- 找到Hash码相同的两条消息所需要的代价是2^{64}数量级。
- 找到具有给定摘要的消息所要付出的代价为2^{128}数量级。

MD5存在的安全问题

- 2004年，王小云证明MD5数字签名算法可以产生碰撞
- 2007年， Marc Stevens，Arjen K. Lenstra和Benne de Weger进一步指出通过伪造软件签名，可重复性攻击MD5算法。
- 2008年，荷兰埃因霍芬技术大学科学家成功把2个可执行文件进行了MD5碰撞。
- 2008年12月一组科研人员通过MD5碰撞成功生成了伪造的SSL证书。
- 2009年，冯登国、谢涛二人利用差分攻击，降低了MD5的碰撞算法复杂度

Secure Hash Algorithm算法

- 1992年NIST制定了SHA(128位)
- 1993年SHA成为标准（FIPS PUB 180）
- 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)
- SHA-1要求输入消息长度<2^{64}
- 输入按512位的分组进行处理的
- SHA-1的摘要长度为160位
- 基础是MD4

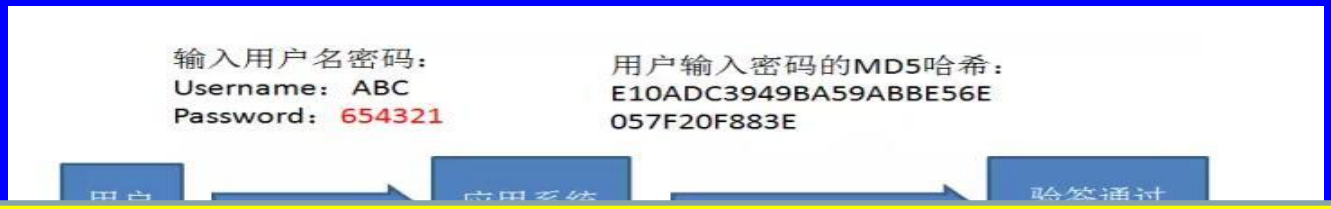
MD5、SHA-1和RIPEMD-160的比较

	MD5	SHA-1	RIPEMD-160
摘要长度	128位	160位	160位
基本处理单位	512位	512位	512位
步数	64(4 of 16)	80(4 of 20)	160(5 paired of 16)
最大消息长度	无限	2^{64}-1位	2^{64}-1位
基本逻辑函数	4	4	5
加法常数	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

HASH的攻击方法

- 应用场景：

- 攻击方式：

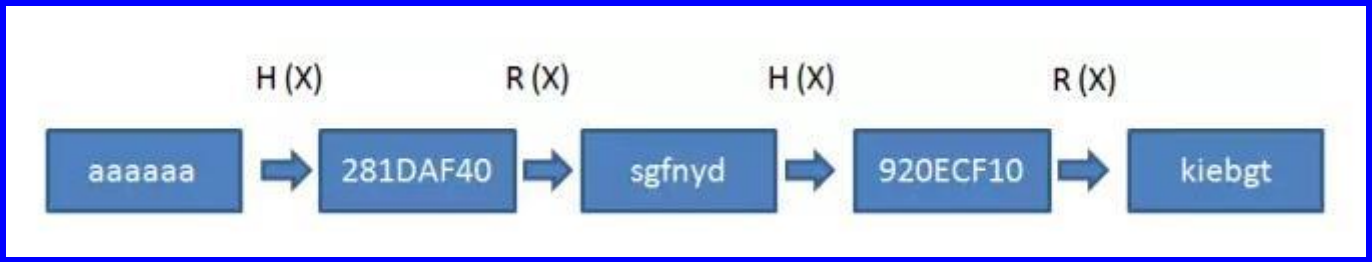


攻击方法：

枚举法　　字典法　　彩虹表法

彩虹表法

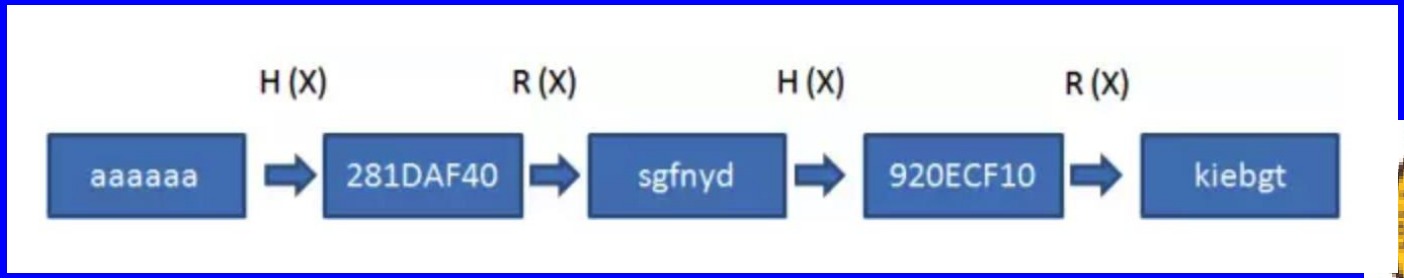
- 彩虹表：



其中：H(x)是生成信息摘要的哈希函数，如MD5

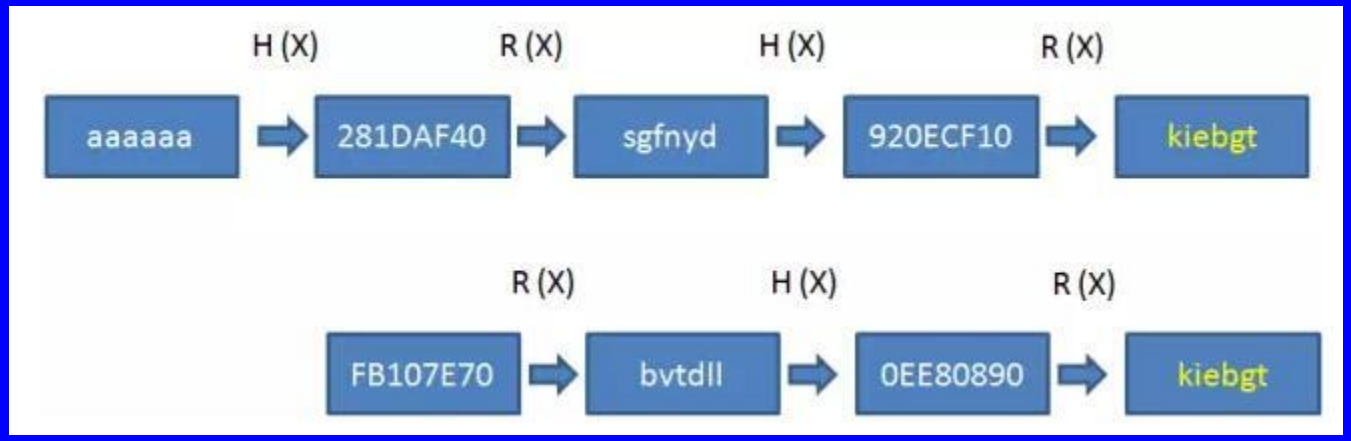
R(x)是从信息摘要转换成另一个字符串的衰减函数

注：R(X)的定义域是H(X)的值域， R(X)的值域是H(X)的定义域

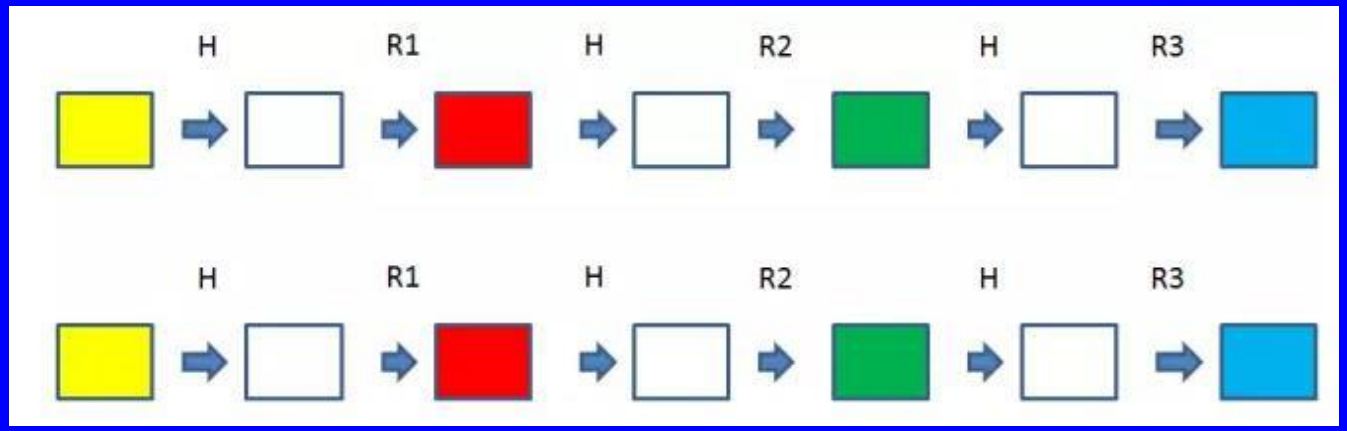


彩虹表法

- 问题：



- 解决的办法：



生日攻击

- 问题：在一个集合A中，任意取出k个数，如果 存在$y \neq x$，而$H(y) = H(x)$的概率为0.5的k是多少？

【问题1】已知：H和$H(x)$，H的输出为n种可能性，H作用于k个输入y。

问：那么至少有1个y，使得$H(y) = H(x)$的概率为0.5的k是多少？

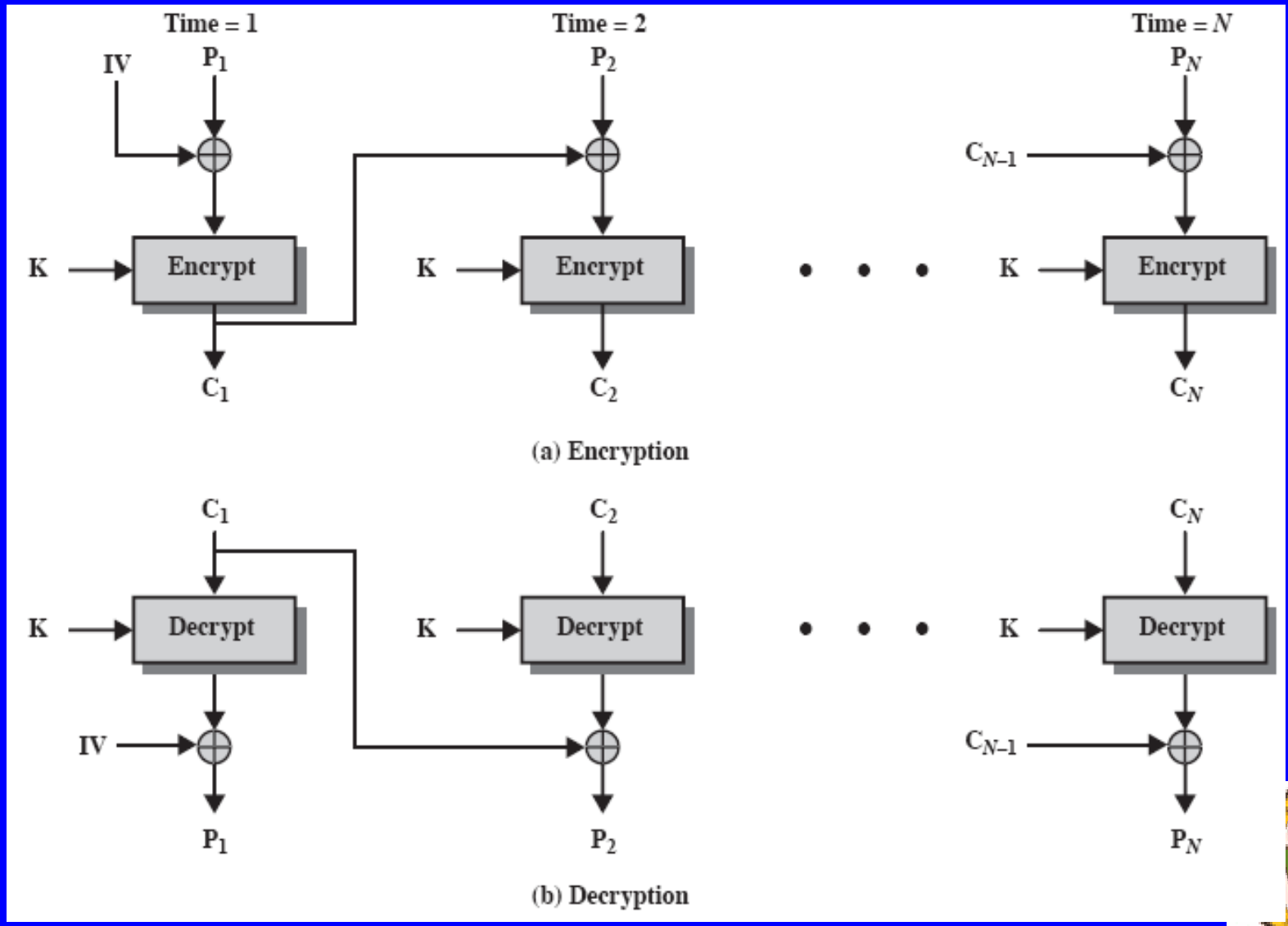
【问题2】k个人中，至少有两个人的生日相同的概率大于0.5的k值最小是多少？

【问题3】问题2一般情形的表达式？

东南大学网络空间安全学院
密码学与安全协议

谢谢！

密码分组链接CBC


Time = 1
Time = 2
Time = N
IV
P_1
P_2
P_N
C_{N-1}
K
Encrypt
C_1
C_2
C_N
(a) Encryption
Decrypt
(b) Decryption