

计算机网络安全技术

清华大学

- 课程代号：40240572
- 课程对象：本科生
- 授课教师：尹 霞
- 开课单位：计算机系网络所

认证技术：认证 = 比较

➤ 消息认证基本概念

➤ 认证函数

- 消息加密、消息认证码、Hash函数

➤ Hash算法

- 安全hash函数一般结构
- 主要Hash算法概述
- MD5、SHA、RIPEMD-160
- 各类算法比较

➤ 数字签名算法DSS

消息认证



➤ 电子身份认证简介

➤ 网站身份认证技术

- HTTP的Basic认证
- 基于表单的身份认证
- 其它增强认证

➤ 其它身份认证应用

- 操作系统认证
- 网络接入认证

身份认证



消息认证技术



清华大学 110周年校庆
110th ANNIVERSARY
TSINGHUA UNIVERSITY

消息认证基本概念

认证要求、认证定义、认证函数



网络环境中的攻击

- 泄密：将消息透露给没有合法密钥的接收方。
- 传输分析
 - 分析通信双方的通信模式。
 - 在面向连接的应用中，确定连接的频率和持续时间。
 - 在面向连接或无连接的环境中，确定双方的消息数量和长度。
- 伪装
 - 欺诈源向网络中插入一条消息。
 - 攻击者产生一条消息并声称这条消息来自某个合法实体。
 - 非消息接收方发送关于收到或者未收到消息的欺诈应答。

网络环境中的攻击

- 内容修改：对消息内容的修改，包括插入、删除、转换和修改。
 - 顺序修改：对通信双方消息顺序的修改，包括插入、删除和重排。
 - 计时修改：对消息的延时和重放。
 - 发送方否认：发送方否认发送过某条消息。
 - 接收方否认：接收方否认接收过某条消息。
-
- 对付泄密、传输分析属于消息保密性范畴。
 - 对付伪装、内容修改、顺序修改、计时修改属于消息认证范畴。
 - 对付发送方否认可以使用数字签名。
 - 对付接收方否认需要使用数字签名和为抗击此类攻击而设计的协议。

消息认证的概念

- 消息认证就是验证所收到的消息确实是来自真正的发送方且未被修改的消息。
 - 消息认证也可以验证消息的顺序和及时性。
 - 数字签名是一种认证技术，可以用来抗击发送方否认。
 - 各种认证协议属于认证技术，被应用到各个方面。
- 任何消息认证在功能上面可以看作两层：
 - 下面一层中一定有某种产生认证符的函数，认证符是一个用来认证消息的值。
 - 上面协议中将该函数作为原语使接收方可以验证消息的真实性。

认证函数

- 产生认证符的函数称为认证函数
- 认证函数可以分成三类：
 - 消息加密：整个消息的密文作为认证符。
 - 消息认证码MAC：MAC是消息和密钥的公开函数，它产生定长的值，该值作为认证符。
 - Hash函数：它是将任意长的消息映射为定长的hash值的公开函数，以该hash值作为认证符。



认证函数1：消息加密



消息加密

- 消息加密本身提供了一种认证手段
- 在传统密码和公钥密码中，对消息加密的分析是不同的

对称加密

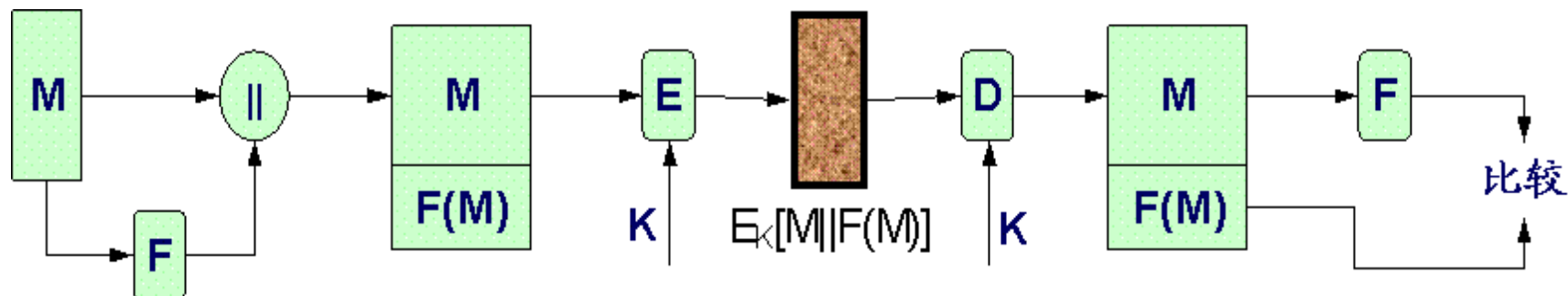
- 发送方A用A和B共享的密钥K，对发送到接收方B的消息M进行加密，如果没有其他人知道该密钥，就可以提供保密性
- 同时，B可以在接收到消息后，用密钥K解密，确认该消息是由A发出的
 - 但这不是绝对的!!! B需要确认密文是否真正来自A
- 解决对称加密中消息认证的方法是要要求明文具有某种易于识别的结构，而且，不通过加密函数是不能重复这种结构的

对称加密的例子

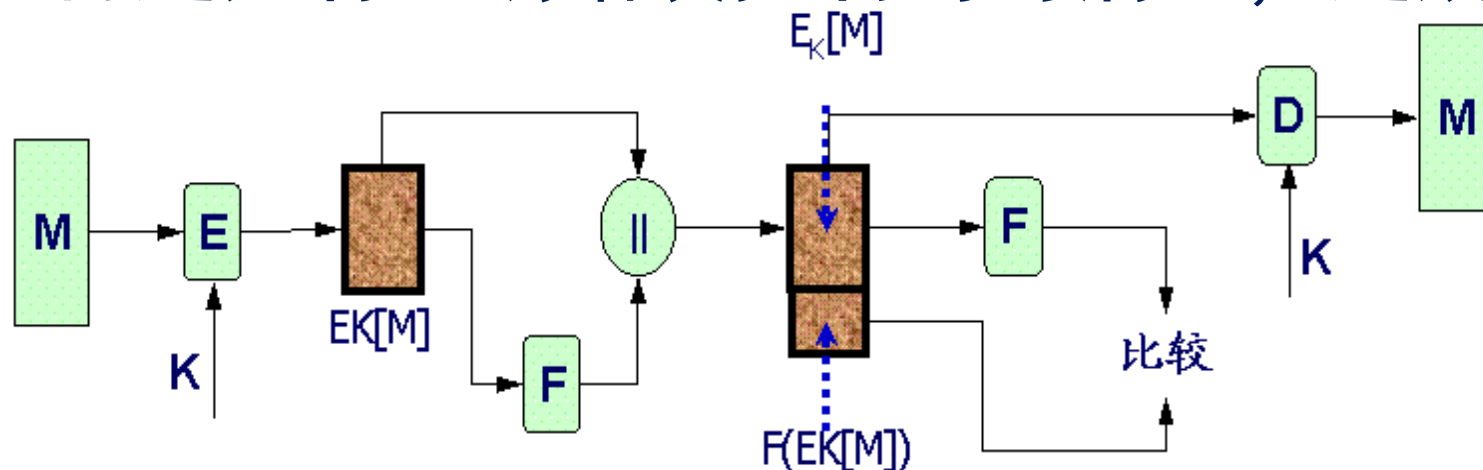
- 加密前对每个消息附加一个错误检测码，也称之为帧校验序列FCS或者校验和
 - A准备发送明文消息M，那么A将M作为函数F的输入，产生FCS，将FCS附加在M后并对M和FCS一起加密
 - 在接收端，B解密其收到的信息，并将其看作是消息和附加FCS；B用相同的函数F重新计算FCS，若所得FCS和收到FCS相等，B认为消息是真实的
 - FCS和加密函数的顺序是很重要的

对称加密的例子

- 先计算FCS，再加密：可以提供认证

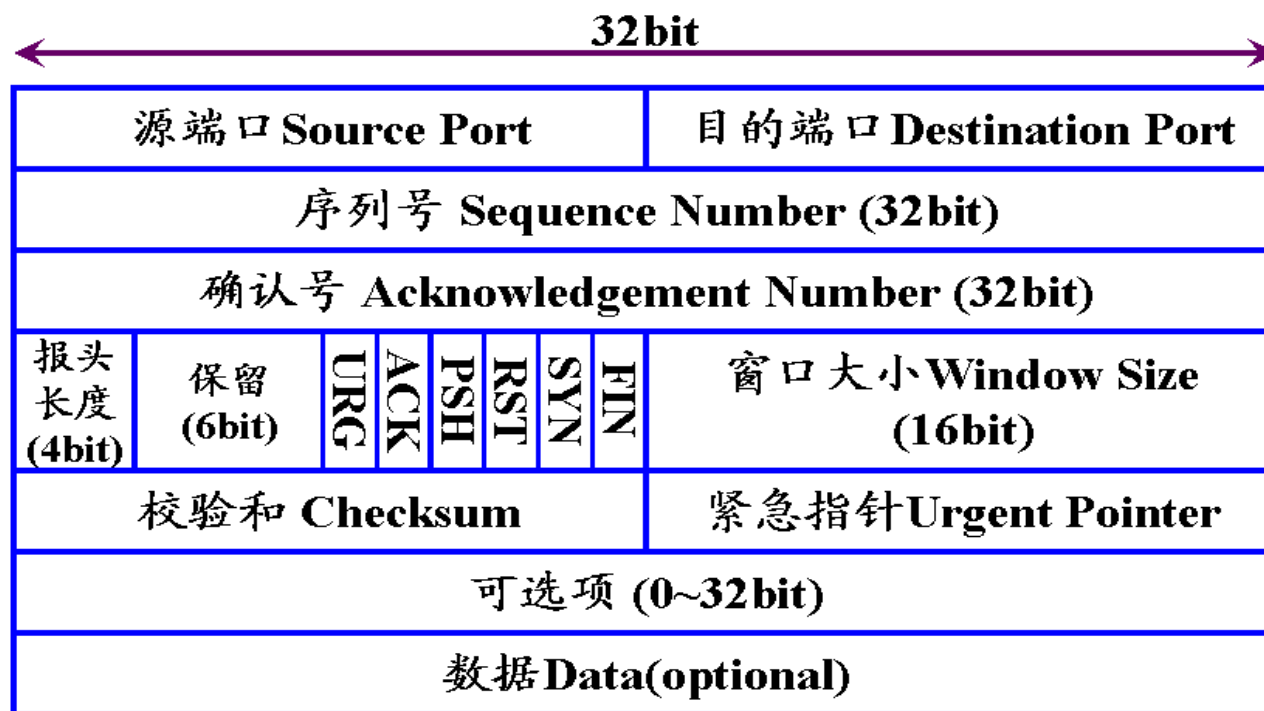


- 先加密，再计算FCS：
攻击者可以构造具有正确错误控制码的消息，造成混淆



对称加密的例子： TCP协议

- 可以对除了IP报头之外的所有数据进行加密
- 如果攻击者利用一条消息代替加密后的TCP段，解密后也不能恢复出原来的IP报头
- 这种方法中，头不仅包含了校验和，而且还有序列号，用于防止延时、删除段和改变段的顺序。



公钥加密

- 公钥加密只提供保密性，不能提供认证
- 如果既要提供保密性，又要提供认证，发送方A可以先用其私钥加密(数字签名)，然后用B的公钥加密
- 这种方法的缺点是执行了四次附加的公钥算法运算

各种加密方法的特点

- 对称加密

- 提供保密性
 - 只有A和B共享密钥K
- 提供认证
 - 只能发自A
 - 传输中未被改变
 - 需要某种数据形式/冗余
- 不能提供数字签名
 - 接收方可以伪造信息
 - 发送方可以否认信息

- 公钥加密

- 提供保密性
 - 只有B拥有用于解密的密钥KRb，但是任何一方都可用KUb对消息加密并假称是A
- 提供认证和签名
 - 只有A拥有用于加密的密钥KR_a
 - 传输中未被改变
 - 需要某种数据组织形式/冗余
 - 任何一方可用KU_a来验证签名
- 提供保密性、认证+签名
 - 提供保密性（因为KUb）
 - 提供认证和签名（因为KR_a）



认证函数2：消息认证码MAC

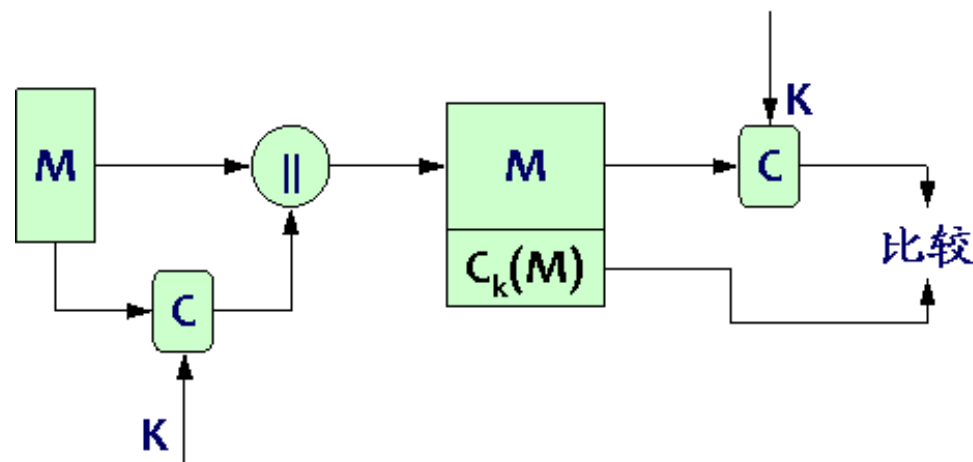


消息认证码

- 消息认证码MAC也是一种认证技术，它利用密码生成一个固定长度的短数据块，并将该数据块附加在消息之后
- MAC是消息和密钥的函数： $MAC = C_k(M)$
 - M：输入消息；
 - C：MAC函数
 - K：收发双方的共享密钥
 - MAC：消息认证码
- MAC函数与加密类似，区别就是MAC算法不要求可逆性，而加密算法必须可逆

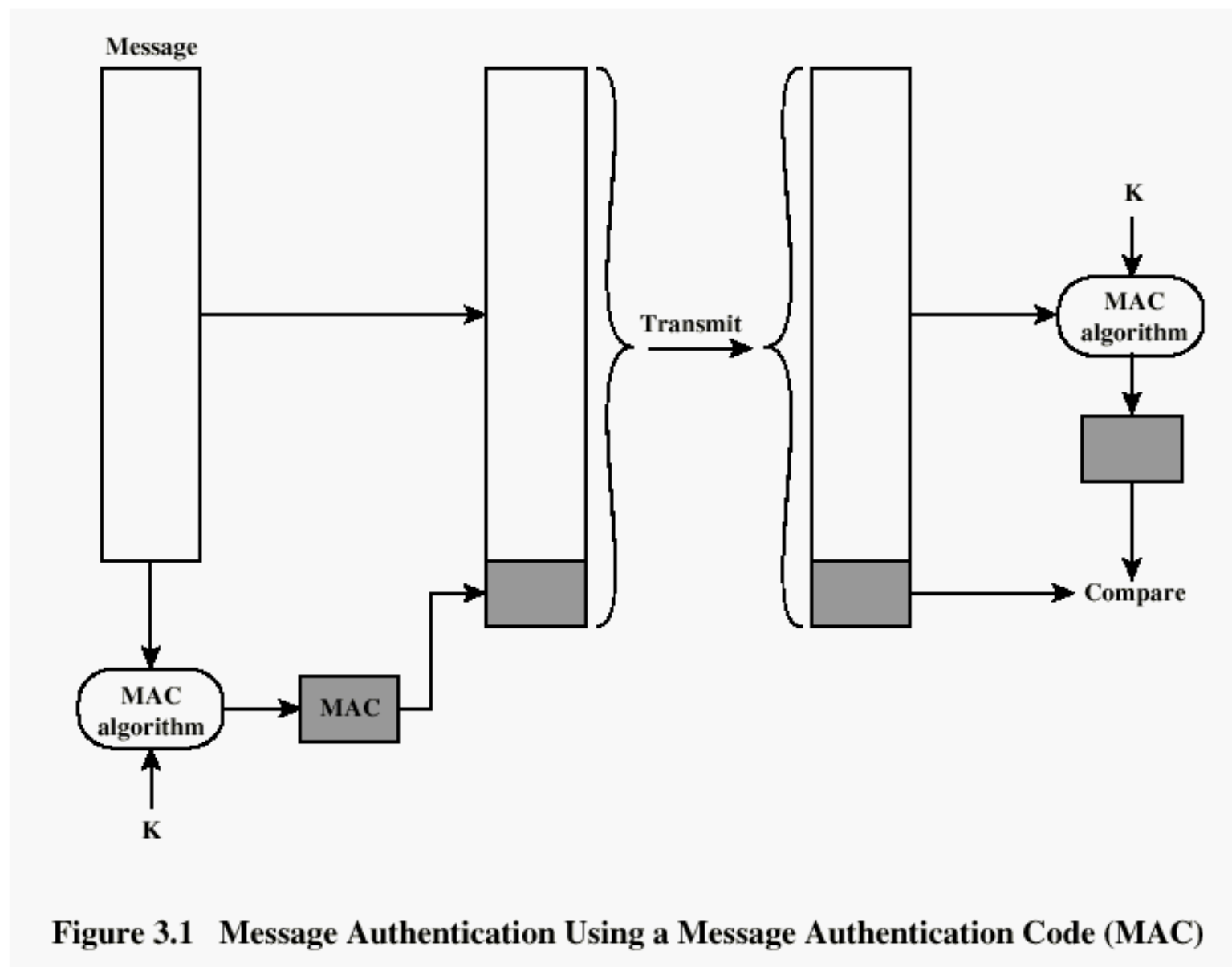
MAC认证的过程

- A和B共享密码K
- A向B发送消息时，A计算MAC，将其附加在消息后面，一起发送给接收方
- 接收方对收到的消息用相同的密钥K进行计算得到MAC，并与收到的MAC进行对比



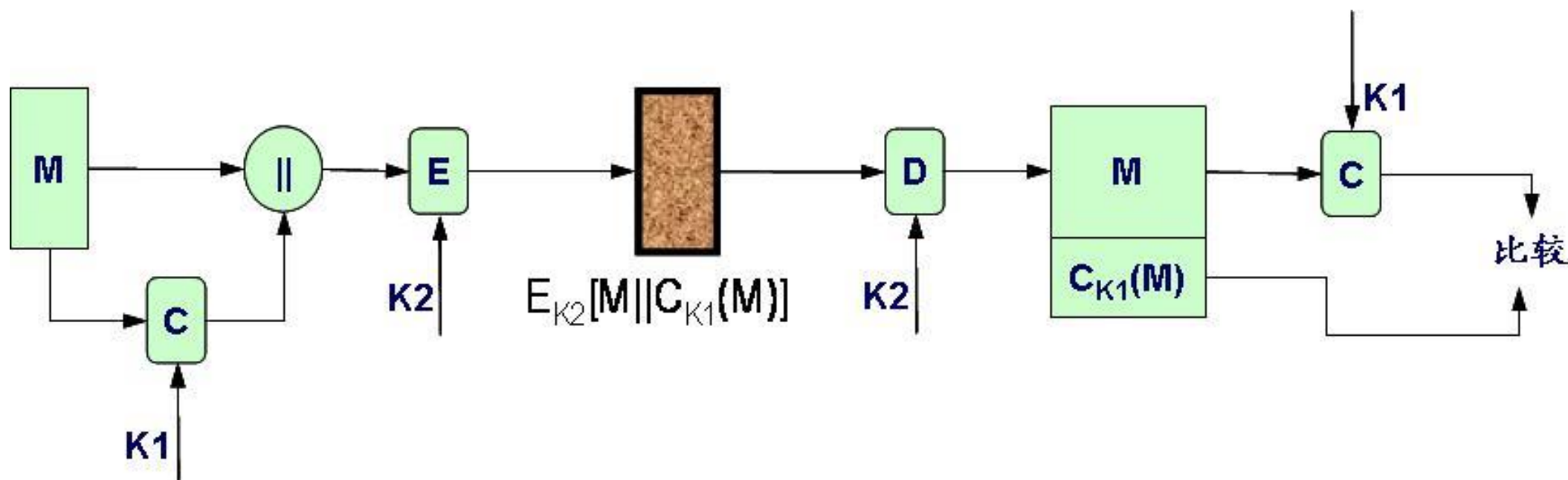
- 因收发双方共享密钥，MAC不能提供数字签名

MAC认证的过程



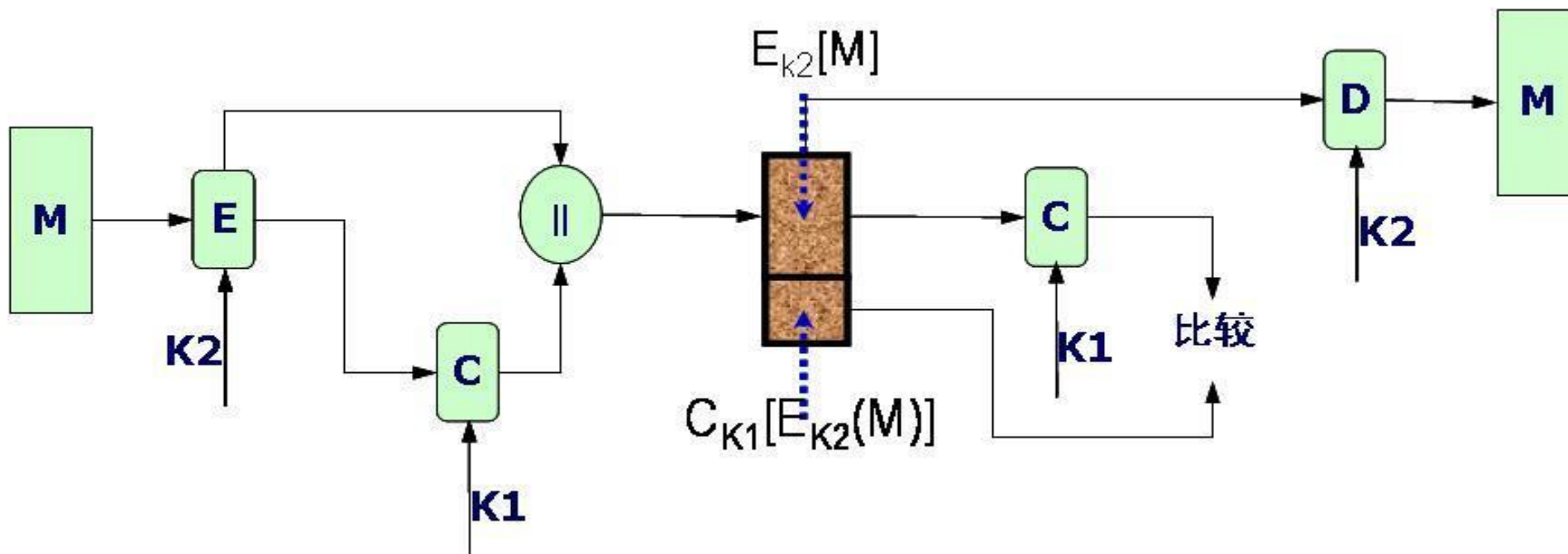
认证 + 保密：与明文有关的认证

- 在实际中，一般将认证和保密性相结合使用
- 与明文有关的认证：先将消息作为输入，计算MAC，附加在消息之后，然后对整个信息块加密



认证 + 保密：与密文有关的认证

- 与密文有关的认证：
先将消息加密，然后将密文作为输入，计算MAC，并将MAC附加在上述密文之后形成待发送的信息块



使用消息认证码MAC的情况

- 将同一消息广播给很多接收者的情况
 - 一个接收者负责验证真实性，告知其它人
- 在信息交换中，接收者希望可以随机地对消息进行认证
- 对明文形式的计算机程序进行认证是很有意义的服务
 - 运行一个计算机程序，不需要每次都进行加密解密；只在需要保护程序完成性的时候才检验消息认证码MAC
- 一些应用只关心消息的认证，而不关心消息的保密性
 - 例如：网络管理协议SNMPv3
- 将认证和保密性分开，可以使层次结构更灵活
 - 例如：可以在应用层提供认证，在传输层提供保密性



认证函数3: Hash函数

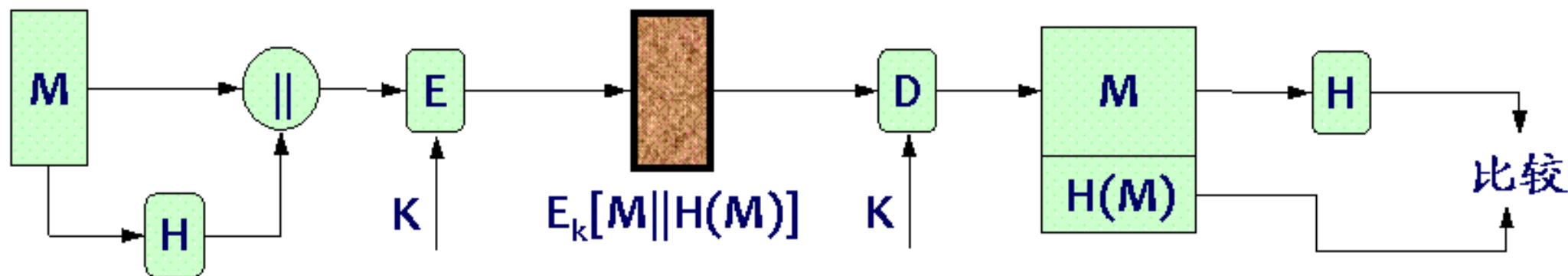


Hash函数

- 单向的hash函数是消息认证码的一种变形
- 与消息认证码一样，hash函数的输入是大小可变的消息M，输出是固定大小的hash码 $H(M)$
- 与MAC不同的是，hash码并不使用密钥，它仅仅做为输入消息的函数，hash码也称为消息摘要(Message Digest, MD)
- Hash码是所有消息位的函数，它具有错误检测能力，即改变消息的任何一位或者多位，都会导致hash码的改变

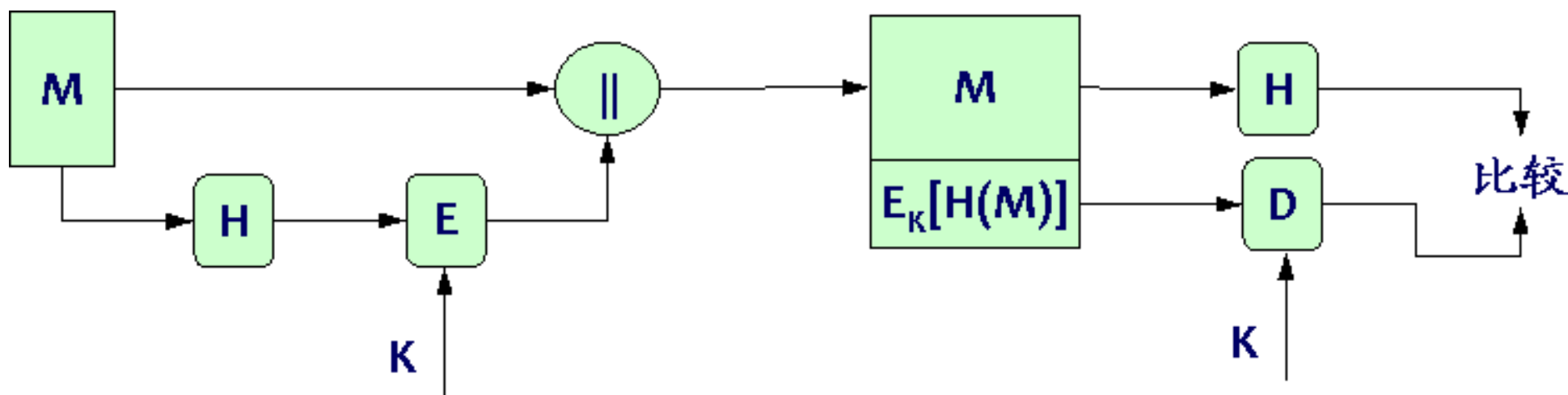
将hash码用于消息认证 (1)

- 用对称密码对消息及附在其后的hash码进行加密：
由于只有A和B共享密钥，所以消息一定是来自A且未被修改
- Hash码提供了认证所需要的结构或者冗余，并且是对整个消息和hash码加密，所以也提供了保密性



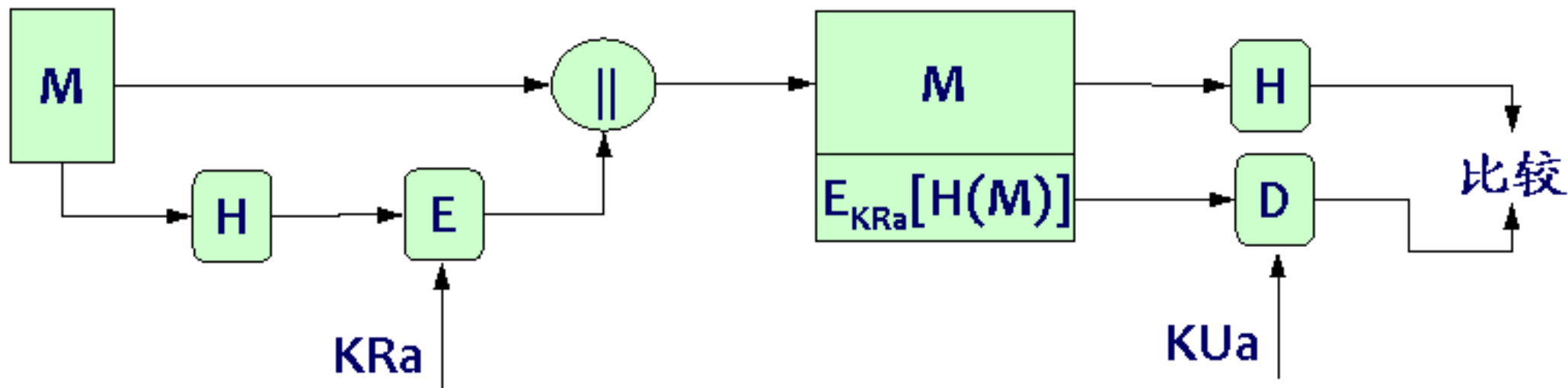
将hash码用于消息认证 (2)

- 用对称密码仅对hash加密：对不要求保密性的应用，这种方法会减少处理代价
- 此种方法提供了认证
 - Hash函数和加密函数合成就是MAC



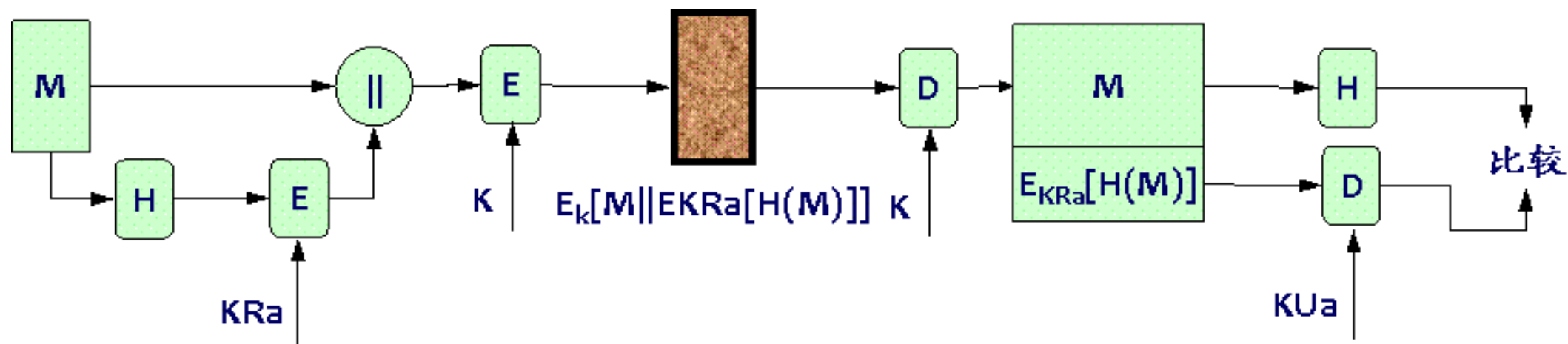
将hash码用于消息认证 (3)

- 用公钥密码和发送方的私钥对hash码加密
- 这种方法可以提供认证，由于只有发送方可以产生加密后的hash码，所以这种方法也提供了数字签名



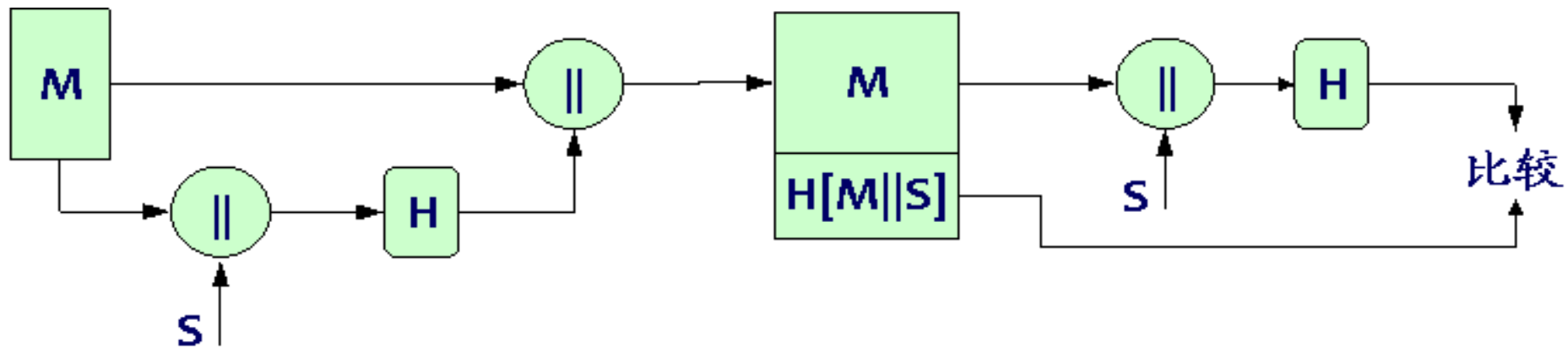
将hash码用于消息认证 (4)

- 即希望保证保密性又希望有数字签名的，先用发送方的私钥对hash码进行加密，再用对称密码中的密钥对消息和上述加密结果进行加密
- 这种技术比较常用：提供认证、数字签名和保密性



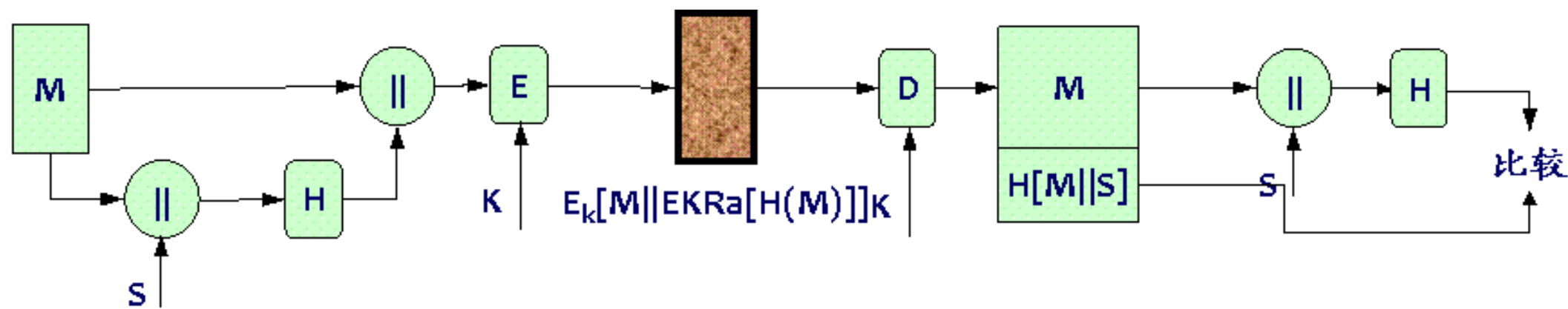
将hash码用于消息认证 (5)

- 假定通信双方共享公共的秘密值S，A将M和S联接后再计算hash值，并将其附加在M后面
- 由于B也知道S，B可以计算hash值，并验证其正确性
- 由于秘密值S不在网络中传送，所以攻击者无法修改所截获的消息，也不能伪造消息，提供了认证



将hash码用于消息认证 (6)

- 对整个消息和hash码加密：提供认证和保密性



不使用加密函数

- 如果不要保证保密性，就可以不使用加密函数，从而减少计算代价
- 基于以下原因可以考虑不使用加密函数：
 - 加密软件速度慢
 - 加密硬件成本不容忽视
 - 加密硬件的优化是针对大数据块的，小数据块则在初始化和调用上面花费了大量时间
 - 加密算法可能收到专利保护
 - 加密算法受到美国出口限制

对Hash函数的要求



- Hash : 哈希函数, 杂凑函数, 散列函数 $h = H(m)$
- H 具有如下特性:
 - 可以操作任何大小的报文 m
 - 给定任意长度的 m , 产生的 h 的长度固定
 - 给定 m 计算 $h = H(m)$ 是容易的
 - 给定 h , 寻找 m , 使得 $H(m) = h$ 是困难的 【单向性】
 - 给定 m , 要找到 $m', m' \neq m$ 且 $H(m) = H(m')$ 是计算上不可行的 【抗弱碰撞性】
 - 寻找任何 (x, y) , $x \neq y$, 使得 $H(x) = H(y)$ 是计算上不可行的 【抗强碰撞性】



Hash算法： 安全hash函数的一般结构



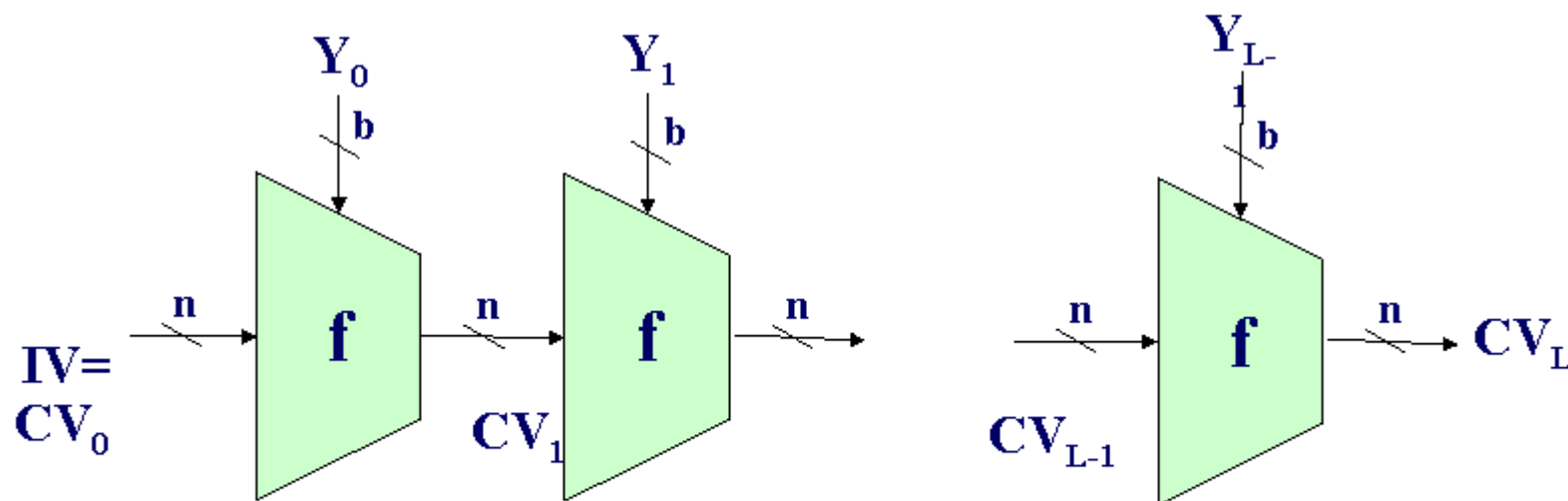
安全hash函数的一般结构

- 典型的安全hash函数的一般结构是由Merkel提出的，又被称为迭代hash函数
- MD5、SHA-1和RIPEMD-160等目前广泛使用的hash函数都是采用这种结构的
- Hash函数将输入消息分为 L 个固定长度的分组，每个分组长度为 b 位，最后一个分组不足 b 位时，需要填充成 b 位
- 输入中包含长度，增加了攻击的难度，因为攻击者必须找出：
 - 具有相同hash值且长度相等的两条消息
 - 或者，找出两条长度不等但加入消息长度后hash值相同的消息

安全hash函数的一般结构

- Hash函数公式

- $CV_0 = IV = \text{初始}n\text{位值}$
- $CV_i = f(CV_{i-1}, Y_i) \quad 1 \leq i \leq L$
- $H(M) = CV_L$ (其中输入M由 Y_0, Y_1, \dots, Y_{L-1} 组成)



IV=初始值; CV=连接变量; Y_i =第*i*个输入分组; f =压缩函数
 L =输入分组数; n =hash码的长度; b =输入分组的长度

安全hash函数的一般结构

- Hash函数中重复使用了压缩函数 f
 - 它的输入是前一步中得出的 n 位结果（即连接变量）和一个 b 位分组，输出位一个 n 位分组
 - 连接变量的初始值在算法开始的时候指定，其终值即为hash值。通常 $b > n$ ，因此称为压缩函数
- 如果压缩函数具有抗碰撞能力，那么迭代hash函数也具有抗碰撞能力；由此可见，设计安全hash函数可以归纳为设计具有抗碰撞能力的压缩函数问题，并且该压缩函数的输入是定长的
- 因为hash函数中， $b > n$ ，将分组大小为 b 位的消息映射为长度为 n 的hash码，所以任何hash函数都存在碰撞，因此要求找出在计算上面不可行的碰撞



Hash算法： 主要的Hash算法概述



Hash算法--- MD族

- MD = Message Digest （消息摘要）
- 90年代初，MD族哈希函数是在由Ron · Rivest设计成功的。MD2、MD4和MD5都产生一个128位的消息摘要。
- MD2
 - 1989年开发出MD2算法。
 - 算法中，首先进行数据补位，使字节长度是16的倍数；然后以一个16位的检验和追加到末尾；最后计算出散列值。
 - 后来Rogier和Chauvaud发现：如果忽略了检验和，将产生MD2碰撞。

Hash算法--- MD族

- MD4

- 1990年开发出MD4算法。
- 有人很快发现了MD4版本中第一、三步的漏洞，并演示了如何利用一部普通的个人电脑在几分钟内找到MD4完整版本中的碰撞。

- MD5

- 1991年，Rivest开发出更为成熟的MD5算法。
- 曾有人发现MD5算法中的伪碰撞，后来被王小云攻破了。

- RIPEMD-128/160/320

- RIPEMD由欧洲财团开发和设计的MD算法。

Hash算法--- SHA族

- SHA = Secure Hash Algorithm
- SHA系列算法是NIST根据Ron Rivest 设计的MD4和MD5开发的算法；美国国家安全局发布SHA作为政府标准。
 - SHA-0：正式地称作SHA系列，发行后不久即被指出存在弱点
 - SHA-1：1994年发布的，与MD4和MD5散列算法非常相似，被认为是MD4和MD5的后继者.
 - SHA-2：实际上分为SHA-224、SHA-256、SHA-384和SHA-512算法。
 - SHA-3：方案正在征集中。

Hash算法--- 其它

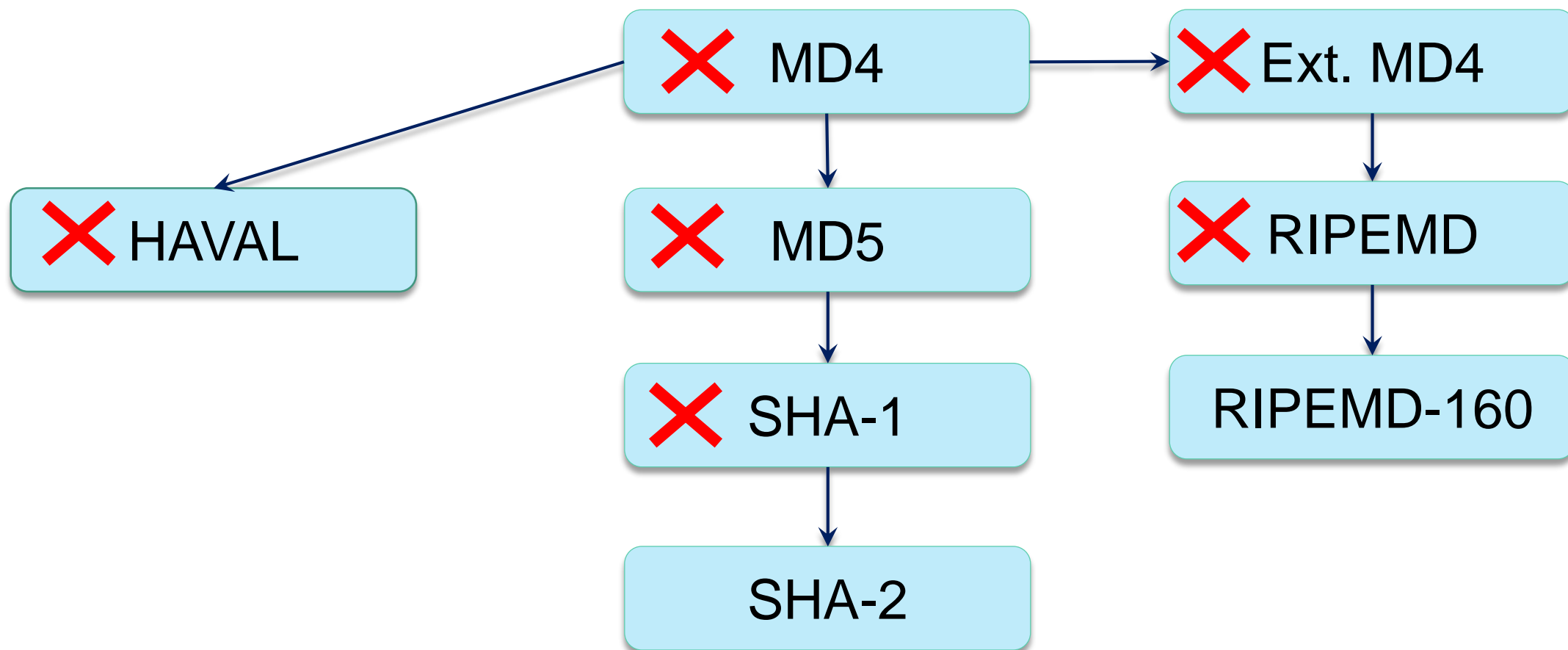
- HAVAL: 加密哈希算法

- 1992年, 由Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry三人设计。与MD5结构不同, 更象现代加密散列函数。
- HAVAL可以产生不同长度的哈希值128位, 160位, 192位, 224位和256位; 还允许用户指定轮数 (3、4或5) 。

- Gost

- Gost是一套俄国标准。

主要Hash算法的发展历程





Hash算法: MD5

Message Digest



MD5 算法

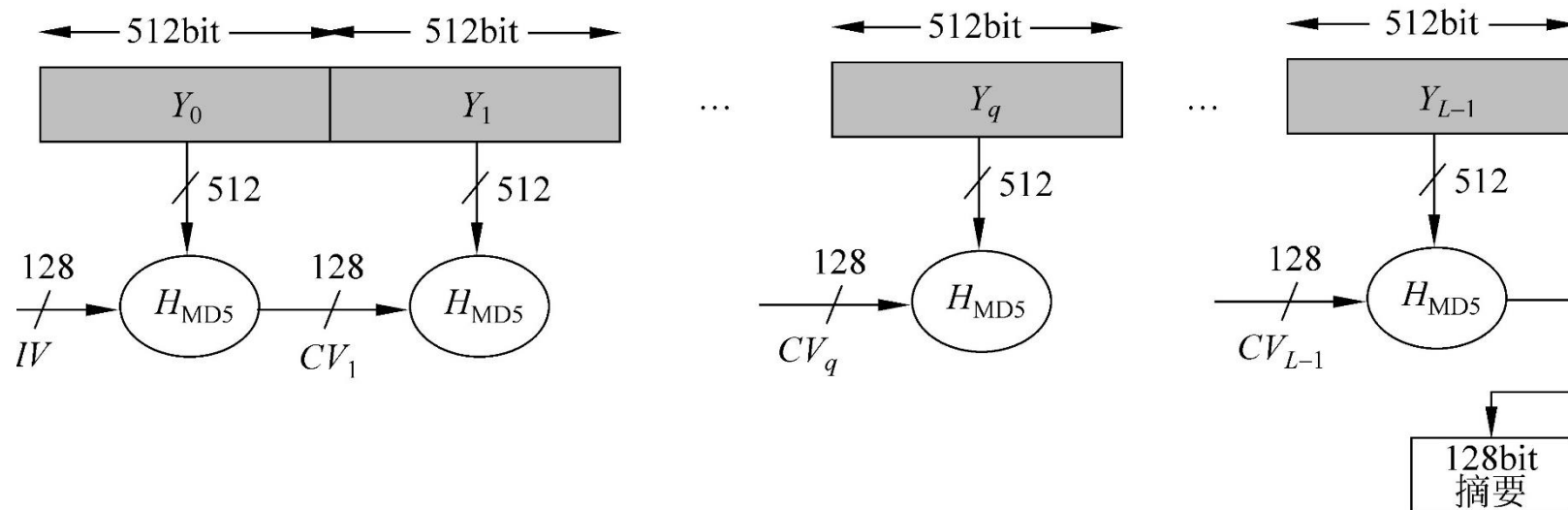
- MD5消息摘要算法(RFC 1321)是由MIT Laboratory of Computer Science 的Ron Rivest 提出的。
- MD5的输入是任意长度的消息，对输入按照512位的分组为单位进行处理，算法的输出是128位的消息摘要。
 - 输入：任意长的消息
 - 分组：512比特
 - 输出：128比特

MD5 算法的设计目标

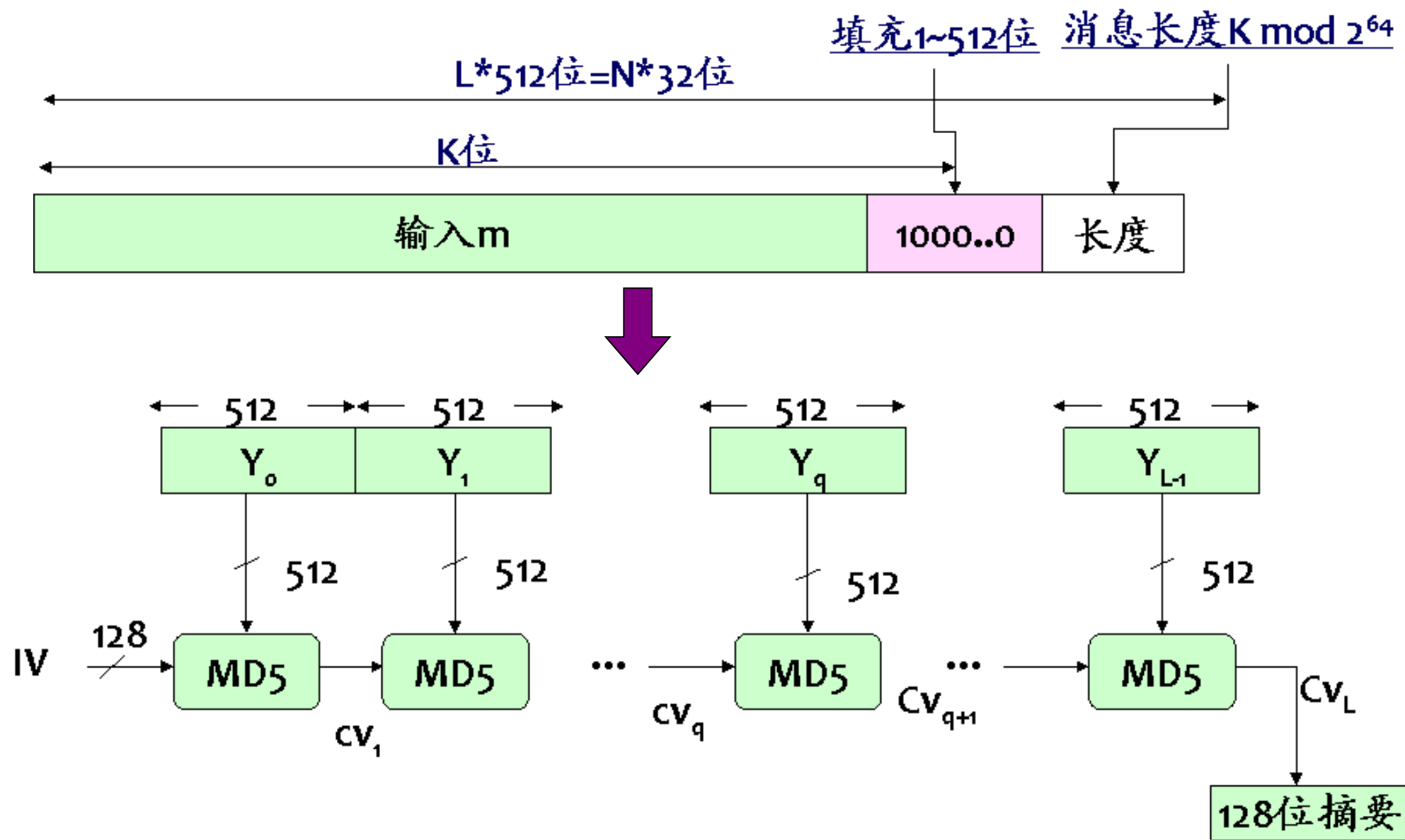
- **安全性：** 找到两个摘要相同的消息在计算上不可行。
- **速度：** 算法应该有利于快速的软件实现。特别是，算法的快速实现是针对32位机的，因此算法基于的是字长为32位的基本操作。
- **简单和简洁性：** 算法应易于描述且易于编程，不需要使用大程序或者置换表。
- **倾向于使用低端结构**
 - 某些处理器(如Pentium系列)将字的最低有效字节存于低地址字节位置（低端），而其它的处理器(如Sun系列)将字的最高有效位存于低地址字节位置（高端）。
 - 将消息作为32位的字序列进行处理的时候，这两种处理方法有着明显的区别，因为其中一种结构需要将每个要处理的字节反向。
 - Rivest注意到使用高端结构的处理器一般比较快，因而能够承受这种处理代价，所以选择使用低端结构将消息表示为32位的字序列。

MD5算法步骤

- Step1: 增加填充位
- Step2: 填充长度。
- Step3: 初始化MD 缓存
- Step4: 以512位的分组(16个字)位单位处理消息
- Step5: 输出

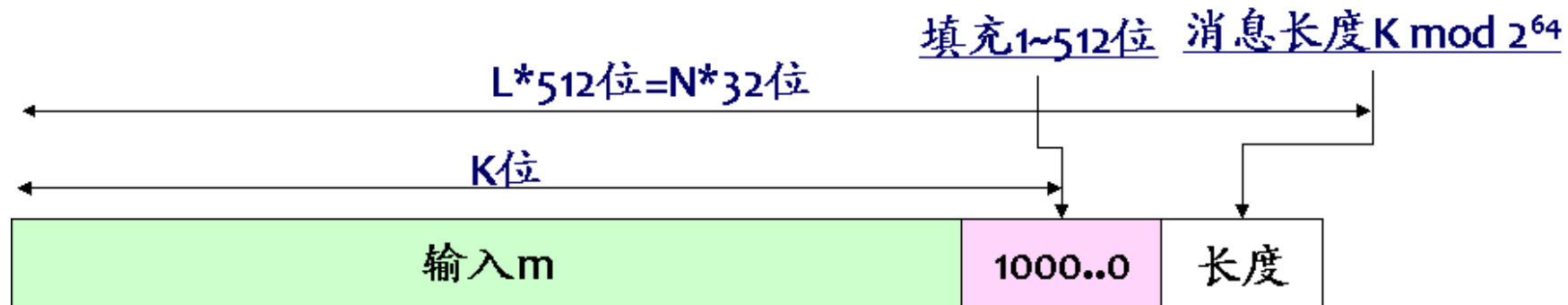


MD5 算法过程



Step1: 增加填充位

- 填充比特：第1位为1，其后各位皆为0
- 填充比特后，使之与448模512同余；
即填充后的消息比512的整数倍少64位



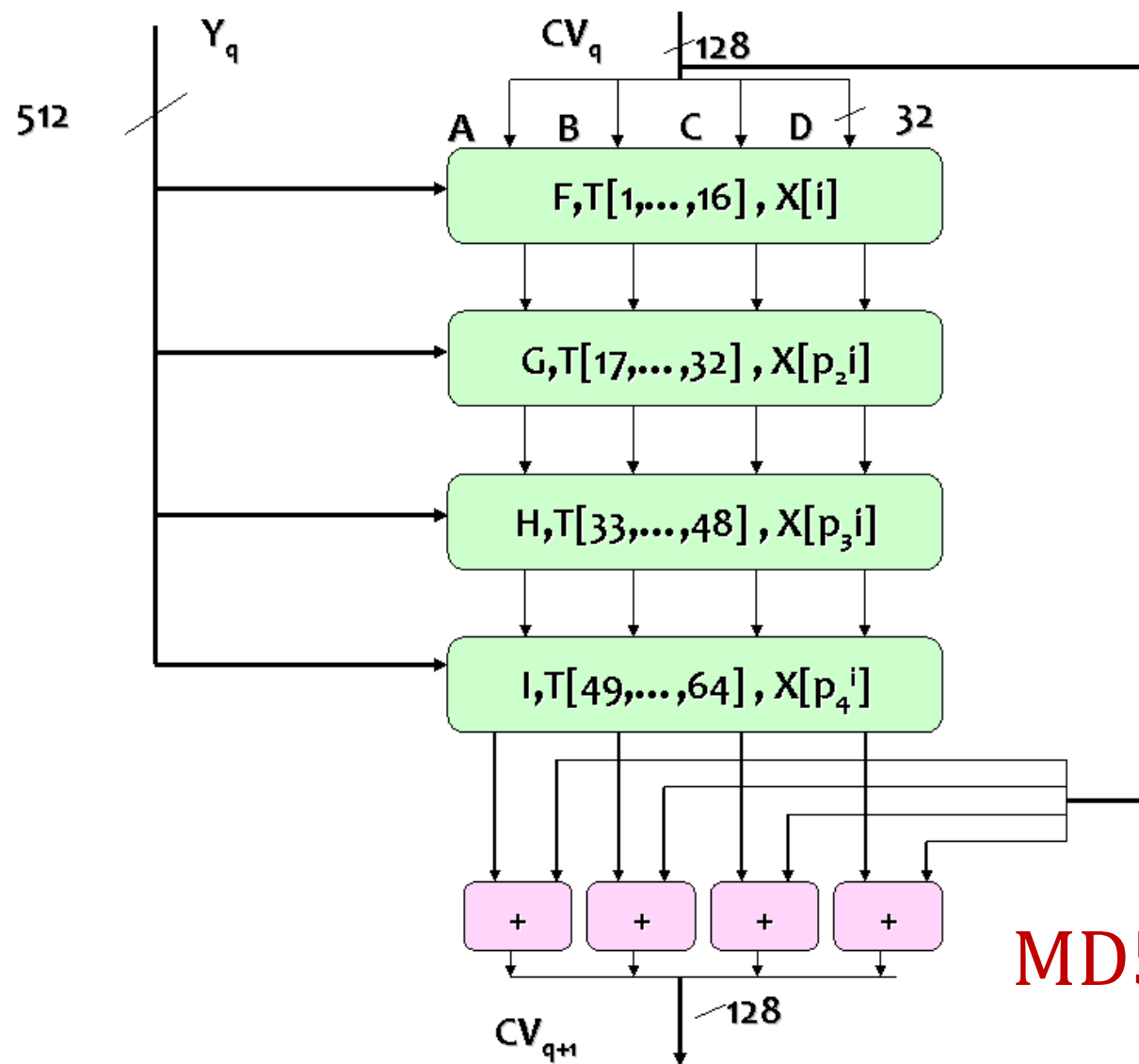
Step2: 填充长度

- 用64位表示填充前的报文长度，附加填充比特的后面；如果长度大于 2^{64} ，则对 2^{64} 取模
- 以little-endian方式表示被填充前的消息长度
 - Little-endian方式是指按数据的最低有效字节优先的顺序存储数据，即将低有效字节存于低地址字节
 - 相反的存储方式为big-endian方式
 - 增加消息长度的目的是为了加强算法安全强度
- 完成填充后，消息的总长度是512的整数倍

Step3: 初始化MD 缓存

- Hash函数的中间结果和最终结果都保存于128位的缓冲区中，缓冲区用4个32位的寄存器(A,B,C,D)表示
- A,B,C,D寄存器以低端格式存储，其初始值为：
 - A= 01234567
 - B= 89ABCDEF
 - C=FEDCBA 98
 - D=76543210

Step4: 计算Hash值



- 以512位的分组(16个字)为单位处理消息

- 由四轮运算组成的压缩函数是算法的核心，压缩函数标记为HMD5
- 使用一个随机矩阵
 $T[i] = 2^{32} \text{abs}(\sin(i))$
 $i=1, 2, \dots, 64$

MD5 的压缩函数

MD5 的压缩函数

- 压缩函数由四轮运算构成，四轮运算结构相同，但各轮使用不同的基本逻辑函数，分别称之为F/G/H/I
- 每轮的输入是当前要处理的512位的分组(Y_q)和128位缓冲区ABCD的内容
- 表T有64个元素，每轮使用表T[1...64]中的16个元素，并更新缓冲区
 - 表T是通过正弦函数构造的，T的第i个元素记为T[i]
 - T的每个元素都可以用32位表示，堆积化的32位输入数据，消除了输入数据的规律性
 - $T[i] = 2^{32} \text{abs}(\sin(i))$ $i=1,2,\dots,64$
- 第四轮的输出与第一轮CV_q的输入相加得到CV_{q+1}。
 - 加法是指缓冲区中的4个字与CV_q中对应的4个字分别模2³²相加。

从正弦函数构造的表T

T[1]=D76AA478	T[17]=E76A2562	T[33]=FFFA3492	T[49]=4BDF7221
T[2]=E8C7B756	T[18]=756FDA1	T[34]=8871F681	T[50]=
T[3]=	T[19]=	T[35]=	T[51]=
T[4]=	T[20]=	T[36]=	T[52]=
T[5]=	T[21]=	T[37]=	T[53]=
...
T[16]=	T[32]=	T[48]=	T[64]=

MD5 每轮处理512位分组的过程

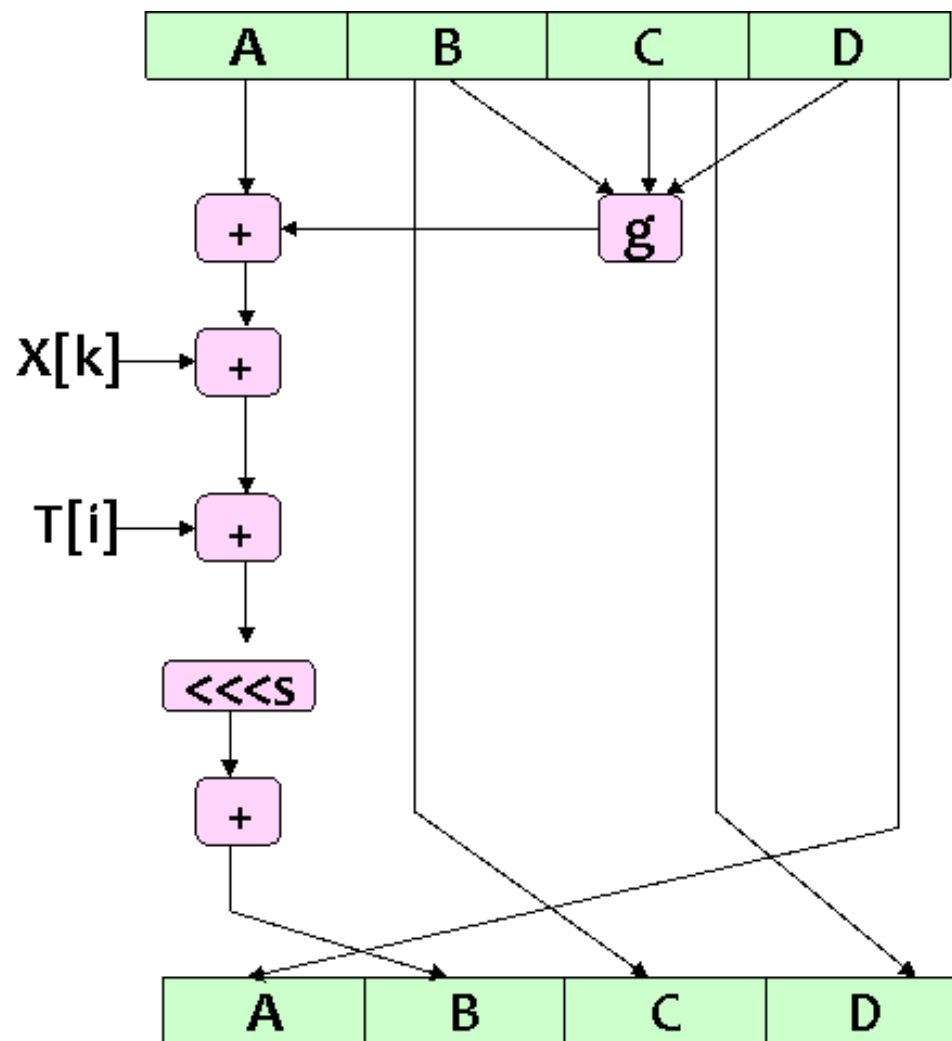
- MD5中每轮对缓冲区ABCD进行16步迭代，每步迭代为：
$$b \leftarrow a + ((a + g(b,c,d) + X[k] + T[i]) \lll s)$$
 - a, b, c, d : 缓冲区的四个字，它按照一定的次序随迭代步变化
 - g : 基本逻辑函数F/G/H/I之一
 - $\lll s$: 32位的变量循环左移 s 位
 - $X[k] = M[q*16+k]$: 消息第 q 个512位分组的第 k 个32位字
 - $T[i]$: 矩阵 T 中的第 i 个32位字
 - $+$: 模 2^{32} 加法

MD5 每轮处理512位分组的过程

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

- $X[k]$: 消息第q个512位分组的第k个32位字
- $T[i]$: 矩阵T中的第i个32位字

循环	原始函数	$g(b,c,d)$
1	F	$(b \wedge c) \vee (\neg b \wedge d)$
2	G	$(b \wedge d) \vee (c \wedge \neg d)$
3	H	$b \text{ xor } c \text{ xor } d$
4	I	$c \text{ xor } (b \vee \neg d)$



Step5：输出

- Step5：输出

- 所有的L个512位的分组处理完之后，第L个分组的输出即是128位的消息摘要。

- MD5算法的公式表示

- $CV_0 = IV$
- $CV_{q+1} = \text{SUM32} [CV_q, \text{RFI}[Y_q], \text{RFH}[Y_q], \text{RFG}[Y_q], \text{RFF}[Y_q, CV_q]]$
- $MD = CV_{L-1}$

MD5的强度

- MD5算法中，hash函数的每一位都是输入的每一位的函数；基本逻辑函数的复杂迭代使得输出对输入的依赖性非常小，即随机选择的两条消息即是具有相同的规律性，也不可能产生相同的hash码
- Ron Rivest猜测，MD5可能是128位的hash码中最强的算法，找到两条hash码相同的消息的代价是 2^{64} 数量级，找到具有给定摘要的消息所需代价为 2^{128} 数量级

MD4算法

- MD4和MD5算法都是Ron Rivest提出的，MD4出现在MD5之前；MD4的RFC发表于1990年10月，其修订版和MD5同时发表于1992年4月的RFC 1320中
- MD4和MD5算法的主要区别在于：
 - MD4使用三轮运算，每轮16步迭代；MD5使用四轮运算，每轮16步迭代
 - MD4第一轮运算没有使用加法常量，第二轮运算中每步迭代使用的加法常量相同，第三轮运算中每步迭代使用的加法常量也相同，但不同于第二轮的加法常量；MD5的64步迭代中，每步使用不同的加法常量 $T[i]$
 - MD5使用四个基本逻辑函数，每轮运算使用一个基本逻辑函数；MD4使用三个基本逻辑函数，每轮运算使用一个基本逻辑函数
 - MD5中的每步迭代结果都与前一步的结果相加，MD4中没有如此



石破惊天 MD5堡垒轰然倒塌



Hash算法: SHA

Secure Hash Algorithm



安全hash算法 — SHA

- 安全hash算法是NIST设计，于1993作为联邦信息处理标准FIPS180发布的，修订版于1995年发布FIPS 180-1，也称之为SHA-1；RFC 3174也给出了SHA-1
- SHA算法建立在MD4之上，基本框架与MD4类似
- SHA-1算法输入是长度小于264位的消息，输出是160位的消息摘要，输入消息以512位的分组为单位进行处理
- SHA-1算法也将消息按照512位分组，但hash值和连接变量长为160位

SHA-1算法步骤

- Step1: 增加填充位
 - 填充消息使之与448模512同余。即填充后的消息比512的整数倍少64位。
- Step2: 填充长度
 - 用64位表示填充前的报文长度，附加在填充后的结果后面。
- Step3: 初始化MD 缓存。
 - Hash函数的中间结果和最终结果都保存于160位的缓冲区中，缓冲区用5个32位的寄存器(A,B,C,D,E)表示，并将这些寄存器初始化。

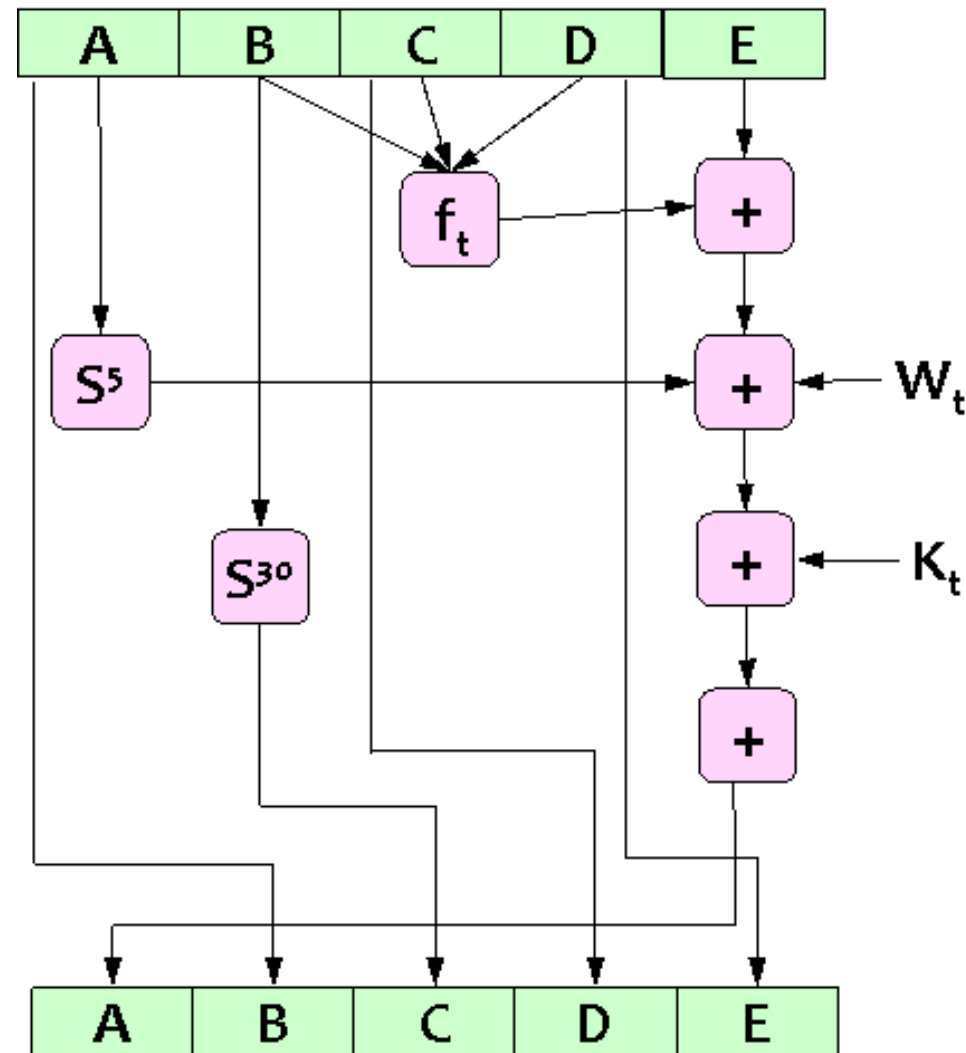
SHA-1算法步骤

- Step4: 以512位的分组(16个字)位单位处理消息。
 - 由十轮运算组成的压缩函数是算法的核心，十轮运算分成两组，每轮执行16迭代20步。
- Step5: 输出。
 - 所有的L个512位的分组处理完之后，第L个分组的输出即是160位的消息摘要。
- SHA-1算法的公式表示：
 - $CV_0 = IV$
 - $CV_{q+1} = \text{SUM32}(CV_q, ABCDE_q)$
 - $MD = CV_L$

SHA-1压缩函数

- 处理一个512位的分组要执行80步，每步的处理过程是一样的：

- A,B,C,D,E：缓冲区的5个字
- t：步骤编号， $0 \leq t \leq 79$
- $f(t,B,C,D)$ ：第t步使用的基本逻辑函数
- S_k ：32位的变量循环左移k位
- W_t ：从当前512位输入分组导出的32位字
- K_t ：加法常量。共使用了四个不同的加法常量。
- $+$ ：模 2^{32} 加法

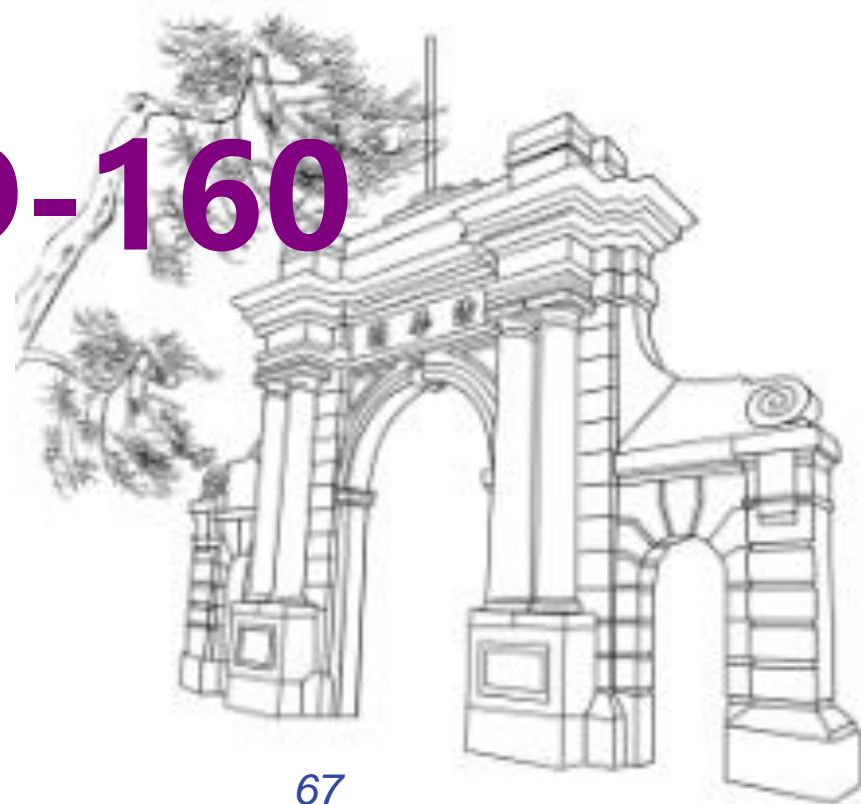


SHA系列算法碰撞攻击复杂度

Algorithm	Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Operations	Collision
SHA-0	160	160	512	$2^{64} - 1$	32	80	+,and,or,xor,rotl	Yes
SHA-1	160	160	512	$2^{64} - 1$	32	80	+,and,or,xor,rotl	2^{63} attack
SHA-256/224	256/224	256	512	$2^{64} - 1$	32	64	+,and,or,xor,shr,rotr	None
SHA-512/384	512/384	512	1024	$2^{128} - 1$	64	80	+,and,or,xor,shr,rotr	None



Hash算法: RIPEMD-160



RIPEMD-160算法

- RIPEMD-160消息摘要算法是欧洲RIPE计划下一个研究组设计的
 - 这些人曾对MD4和MD5进行过一些成功的攻击
 - 后来，他们和H.Dobbertin合作，一起完成了对两轮RIPEMD算法改进工作，最终形成了RIPEMD-160消息摘要算法
- RIPEMD-160算法的输入是任意长的消息，输出是160位的消息摘要；输入以512位的分组单位进行处理，hash值和链接变量是160位

RIPEMD-160算法步骤

- Step1: 增加填充位
 - 填充消息使之与448模512同余, 即填充后的消息比512的整数倍少64位
- Step2: 填充长度
 - 用64位表示填充前的报文长度, 附加在填充后的结果后面
- Step3: 初始化MD 缓存
 - Hash函数的中间结果和最终结果都保存于160位的缓冲区中
 - 缓冲区用5个32位的寄存器(A,B,C,D,E)表示, 并初始化寄存器

RIPEMD-160算法步骤

- Step4: 以512位的分组(16个字)位单位处理消息
 - 由十轮运算组成的压缩函数是算法的核心，十轮运算分成两组，每组五轮，每轮执行16步迭代
- Step5: 输出
 - 所有的L个512位的分组处理完之后，第L个分组的输出即是160位的消息摘要。



Hash算法： 各类算法比较

MD5、 SHA-1、 RIPEMD-160



MD5/SHA-1/RIPEMD-160比较

	MD5	SHA-1	RIPEMD-160
摘要长度	128 bits	160 bits	160 bits
基本处理单元	512 bits	512 bits	512 bits
步数	64(4 轮, 每轮 16 步)	80(4 轮, 每轮 20 步)	160(5 轮, 每轮 16 步)
最大消息长度	∞	$2^{64}-1$ 位	$2^{64}-1$ 位
基本逻辑函数	4	4	5
使用的加法常量	64	4	9
低端位/高端位结构	低位在前	高位在前	低位在前

MD5/SHA-1/RIPEMD-160比较

- 抗穷举攻击的能力

- 三个算法都不易受到弱碰撞性的攻击
- MD5由于消息摘要短，易于收到强碰撞性的攻击
- SHA-1和RIPEMD-160在将来，对强碰撞性的攻击还是安全的

- 抗密码分析的能力

- 对MD5的密码分析取得了很大进展
- 对SHA-1的密码分析比MD5要困难
- 对RIPEMD-160的密码分析比SHA-1又要困难

MD5/SHA-1/RIPEMD-160比较



- 速度

- 三个算法都基于模 2^{32} 加法和简单的位逻辑运算，所以它们在32位机上执行速度较快
- 由于MD5相对简单，迭代次数少，所以运行速度最快

- 低端位/高端位结构

- MD5和RIPEMD-160采用低位在前结构，SHA-1采用高位在前结构



清华大学 110周年校庆
110th ANNIVERSARY
TSINGHUA UNIVERSITY

数字签名算法DSS



数字签名

- 消息认证可以保证通信双方不受第三方的攻击，但是它不能处理通信双方自身发生的攻击
- 例如A使用消息认证码MAC给B发送了一条消息，可以出现下面两种情形：
 - 情况1：B可以伪造一条消息，并声称该消息发自A
 - B只需产生一条消息，用A和B共享的密钥产生认证码，并将其附在消息之后即可
 - 情况2：A可以否认曾经发送过某条消息
 - 因为B可以伪造消息，所以也无法证明A确实发送过某消息

数字签名

- 在收发双方不能完全信任的情况下，就需要其它的方法来解决，数字签名就是一个最好的解决方法
- 数字签名是公钥密码学发展过程中最重要的概念之一，它可以提供其它方法难以实现的安全性
- 数字签名相当于手写签名，须具备以下特征：
 - 它必须能够验证签名者、签名日期和时间
 - 它必须能认证被签的消息内容
 - 签名应能由第三方仲裁，以解决争执
- 数字签名可以分成两类：直接数字签名和仲裁数字签名

直接数字签名

- 直接数字签名只涉及通信双方
- 假定接收方已知发送方的公钥，则发送方可以通过用自己的私钥对整个消息或者消息的hash码加密来产生数字签名
- 之后，再用接收方的公钥(公钥密码)和共享的密钥(对称密码)对整个消息和签名进行加密，则可以获得保密性
- 直接数字签名的弱点在于方法的有效性依赖于发送方私钥的安全性

仲裁数字签名

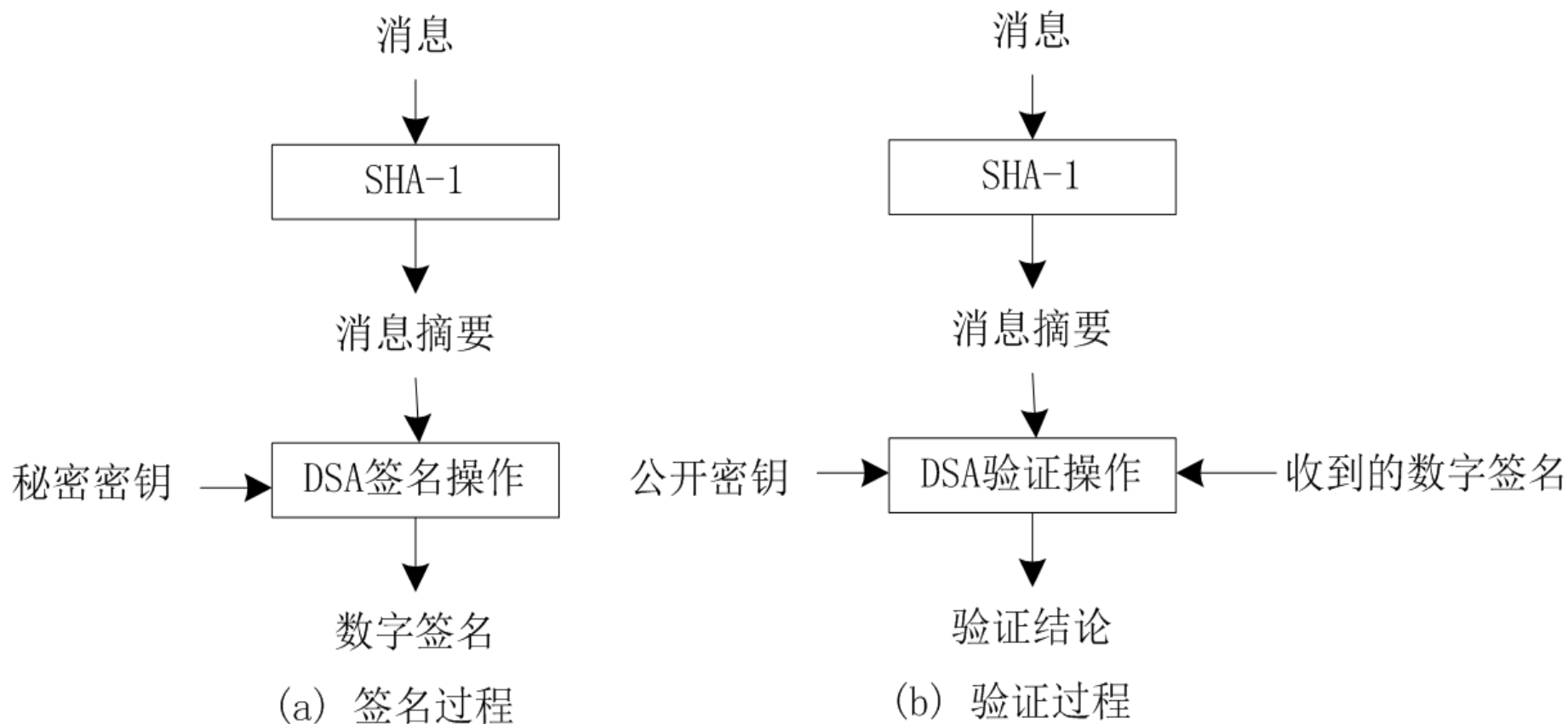
- 从发送方X到接收方Y的每条已签名的消息都先发给仲裁者A，A对消息及其签名进行检查以验证消息源及其内容，然后给消息加上日期，并发给Y，同时指明该消息已通过仲裁者的检验
- A的加入解决了直接数字签名的问题

数字签名标准DSS

- 1991年8月30日，美国国家标准与技术学会NIST提出了一个联邦数字签名标准，称之为数字签名标准DSS
 - 1993年，根据公众对其安全性的反馈做了修改
 - 1994年12月1日，被正式采用为美国联邦信息处理标准
 - 1996年，又进一步修改；2000年发布了DSS的扩展版FIP186-2
- DSS使用SHA-1算法，给出了一种新的数字签名方法，即数字签名算法DSA
 - NIST提出：“此标准适用于联邦政府的所有部门，以保护未加保密的信息——它适用于E-mail、电子金融信息传输、电子数据交换、软件发布、数据存储及其他需要数据完整性和原始真实性的应用”

DSS的签名与验证

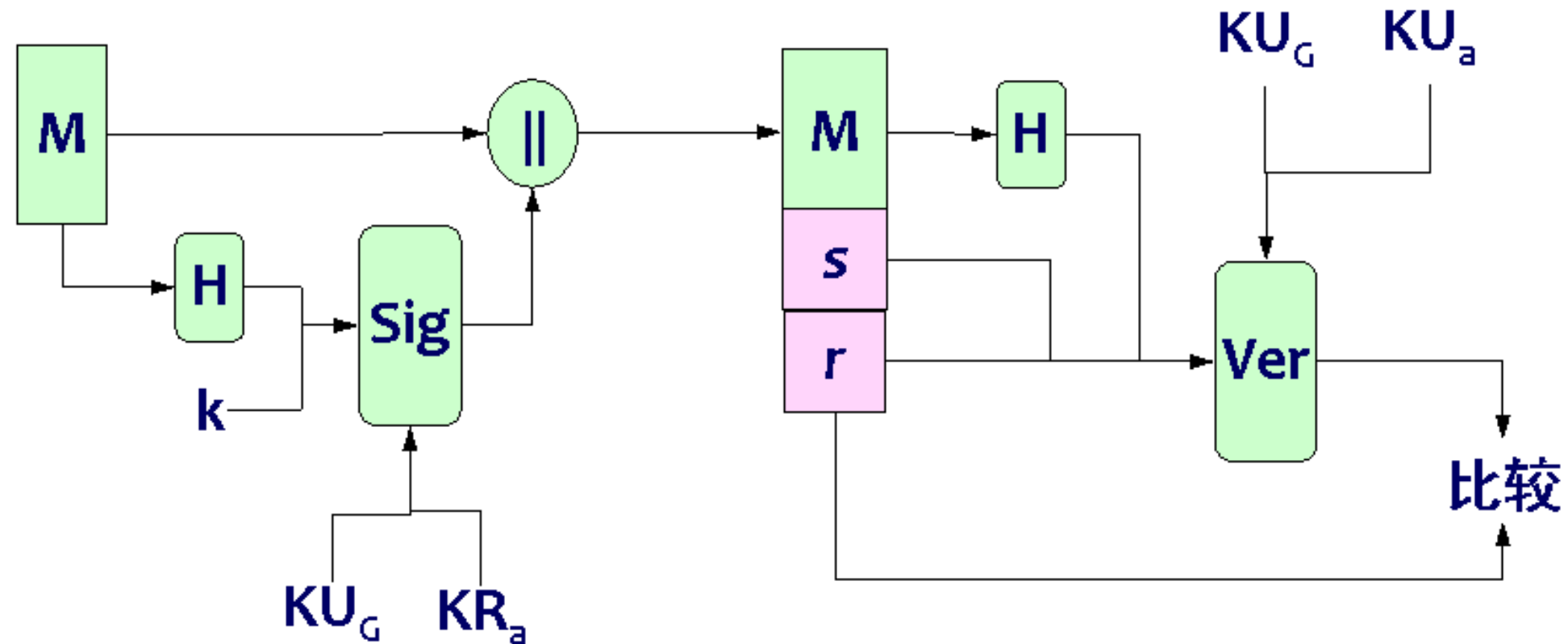
- DSS是一种公钥方法，与RSA不同，DSS不能用于加密或者密钥分配，只提供数字签名功能



数字签名标准DSS

- DSS使用hash函数，它产生的hash码和随机数 k 作为数字签名函数Sig的输入，签名函数依赖于发送方的私钥 KRa 和一组通信伙伴共同拥有的参数构成的全局公钥 KUG
- 签名函数保证只有拥有私钥的发送方才能产生有效签名
- 签名由两部分构成： s 和 r
- 接收方对接收到的消息产生hash码，这个hash码和签名一起作为验证函数Ver的输入，验证函数依赖于全局公钥 KUG 和发送方公钥 KUa ；若验证函数的输出等于签名中的 r ，则签名有效

数字签名标准DSS



参数说明

- 全局公钥 (p, q, g)
 - p : 为 L 位长的素数。其中, L 为512~1024之间且是64倍数的数。
 - q : 是160位长的素数, 且为 $p-1$ 的因子。
 - g : $g = h^{(p-1)/q} \bmod p$ 。
其中, h 是满足 $1 < h < p-1$ 且 $h^{(p-1)/q} \bmod p$ 大于1的整数。
- 用户私钥 x : x 为在 $0 < x < q-1$ 内的随机数
- 用户公钥 y : $y = g^x \bmod p$
- 用户每个消息用的秘密随机数 k , $0 < k < q$

参数 p 、 q 、 g 是公开的; x 为私钥, y 为公钥;
对于每一次签名都应该产生一次 k ; x 和 k 用于数字签名, 必须保密;

签名过程与验证过程



签名过程

用户随机选取 k ，计算：

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

(r, s) 即为消息 M 的数字签名

验证过程

接收者收到 M, r, s 后，首先验证 $0 < r < q$, $0 < s < q$ ，如通过则计算：

- $w = (s)^{-1} \bmod q$
- $u_1 = [Mw] \bmod q$
- $u_2 = [rw] \bmod q$
- $v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$

如果 $v=r$ ，则确认签名正确

认证技术：认证 = 比较

➤ 消息认证基本概念

➤ 认证函数

- 消息加密、消息认证码、Hash函数

➤ Hash算法

- 安全hash函数一般结构
- 主要Hash算法概述
- MD5、SHA、RIPEMD-160
- 各类算法比较

➤ 数字签名算法DSS

消息认证



➤ 电子身份认证简介

➤ 网站身份认证技术

- HTTP的Basic认证
- 基于表单的身份认证
- 其它增强认证

➤ 其它身份认证应用

- 操作系统认证
- 网络接入认证

身份认证



身份认证技术



(电子) 身份认证简介



什么是身份认证?

- 身份认证是指在主客体交互行为过程中确认行为参与者（主体或客体，通常称为用户）身份的解决方法。
- 在现实生活中，身份认证无处不在：
 - 印章、居民身份证、银行卡及密码、飞机票、等等……
- 在真实世界中，对用户的进行身份认证的方法基本上可以分三种：
 - what you know: 根据你所知道的信息来证明你的身份
 - what you have: 根据你所拥有的东西来证明你的身份
 - who you are: 直接根据独一无二的身体特征来证明身份

互联网世界中的电子身份认证

- （电子）身份认证是在计算机及网络中确认操作者身份的解决方法
- 互联网世界中一切信息（包括各类对象的身份信息）都是用一组特定的数据来表示的，计算机只能识别用户的数字身份，因此身份认证的过程也是针对这组特定的数据进行
- 电子身份的认证方法与现实世界的身份认证方法是类似的（根据what you know、what you have、who you are），其中，需要大量使用消息认证技术（Message Authentication）

常见的电子身份认证技术手段

方法	技术
What you know	静态口令
What you have	智能卡(IC卡)
	短信口令
	动态口令牌(基于时间戳)
	动态口令牌(基于计数器)
	USB KEY(Challenge/Response模式)
	USB KEY(电子证书模式)
Who you are	生物特征验证技术 (指纹、掌纹、声纹、视网膜、虹膜、脸型、体态、DNA等)



电子身份认证常见应用场景

- 登录操作系统

- Windows、Mac OS、Linux.....

- 上网

- TUNet客户端
- 无线网登录

- 登录网站

- 校内：INFO、网络学堂、图书馆.....
- 校外：书库、论坛、微博、淘宝、电子银行.....

- 玩游戏

-

构建安全网站的技术手段

- 正面

- 认证 (Authentication)
- 授权 (Authority)
- 审计 (Auditing)

- 反面

- 消灭各个层面的漏洞 (Bug)

- 网站是一个复杂系统，安全问题涉及很多方面

step by step: 设计安全可靠的网站用户认证模块



清華大學 110周年校庆
110th ANNIVERSARY
TSINGHUA UNIVERSITY

网站身份认证技术

step by step

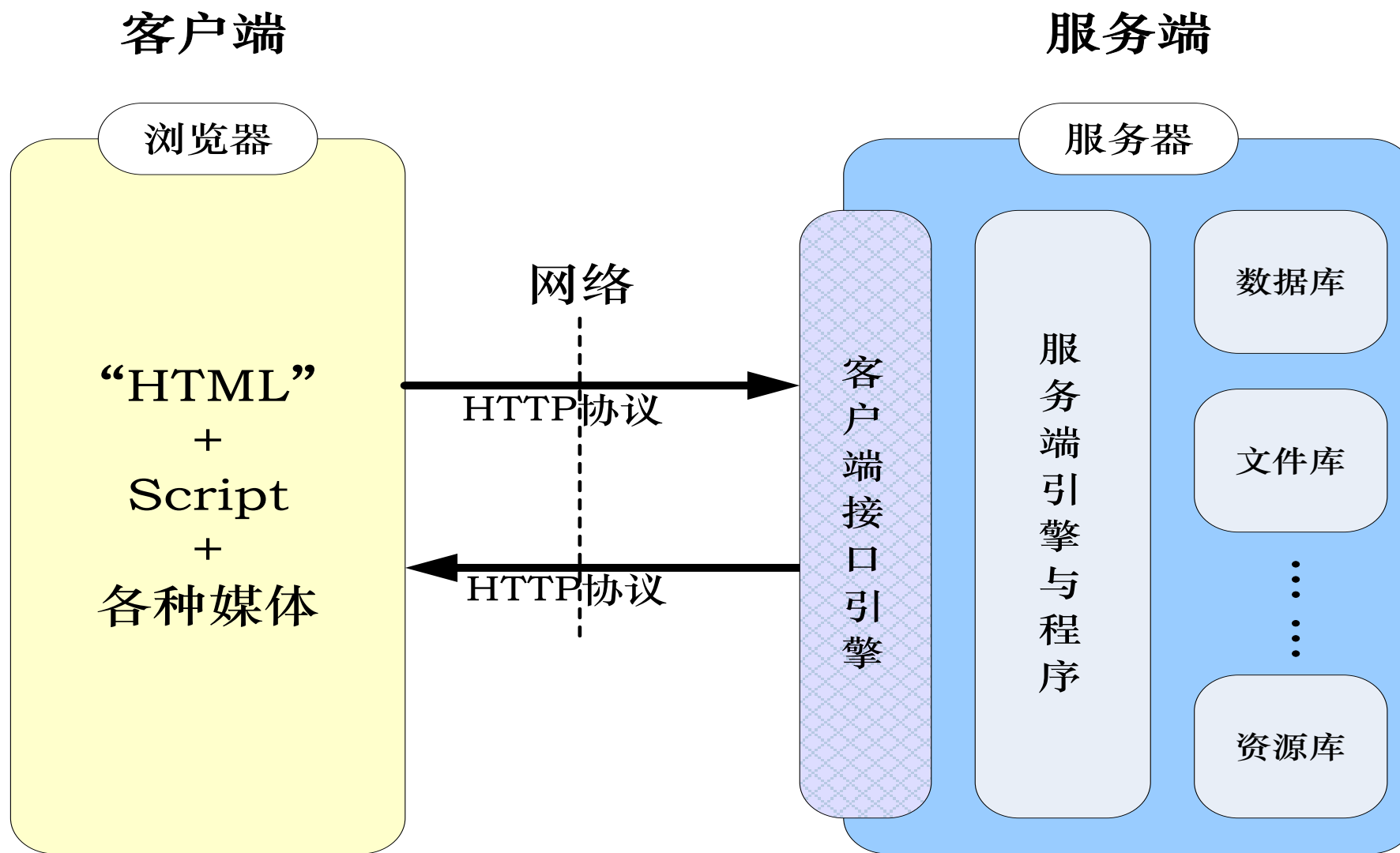
设计安全可靠的网站身份认证模块



网站系统的基本构成

- 网站Web又称为万维网（World Wide Web），基本结构是采用开放式的客户/服务器结构（Client/Server, C/S）
- 完整的网站系统分成三个部分：
 - 服务器端：Web Server
 - 客户端：浏览器Browser、专用程序
 - 通信协议：HTTP协议
- 网站系统中常用的技术
 - HTML、JavaScript、多媒体技术
 - 服务端技术

网站系统的基本构成



HTTP协议

- 超文本传输协议HTTP（Hyper Text Transfer Protocol）是基于TCP/IP的应用层协议
- HTTP的主要特点：
 - 典型的C/S工作方式，简单快速，数据传输较少
 - 可以通过HTTP协议传输任意类型的数据对象
 - HTTP是面向一次连接的无状态网络协议

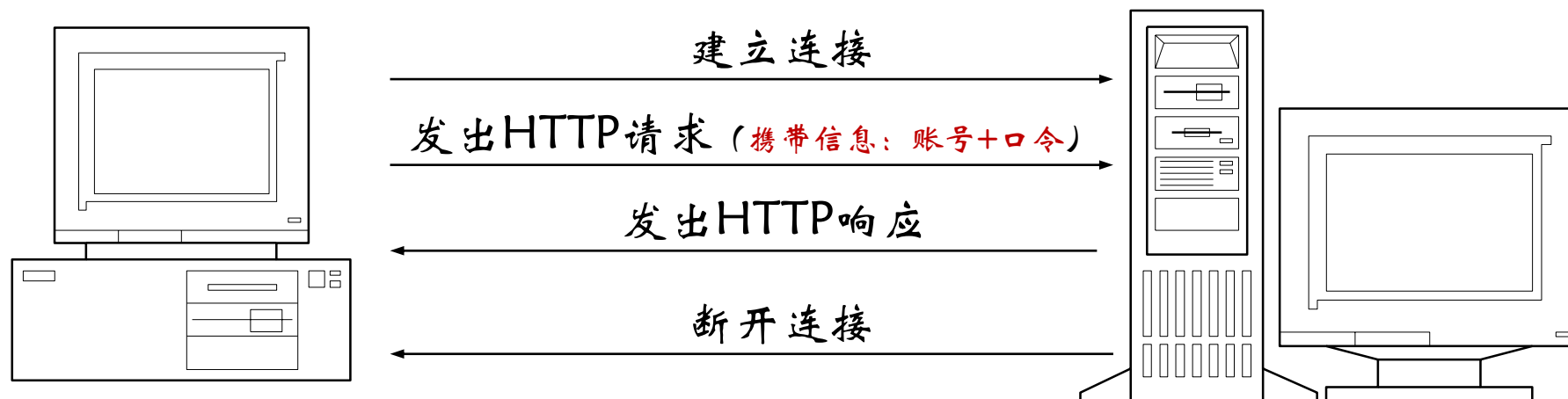


网站用户认证技术

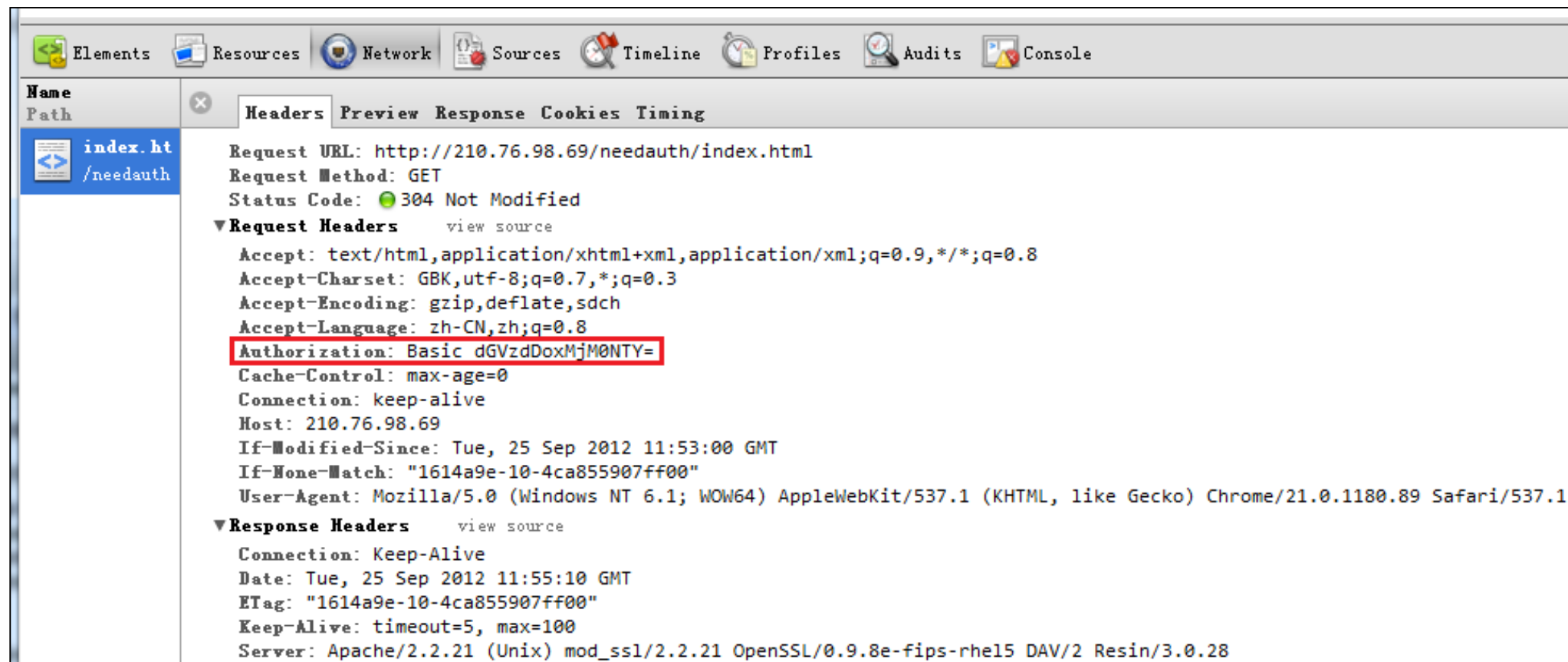
- 最简单的网站用户认证： **HTTP的Basic认证**
- 改进解决Basic认证的问题： **基于表单的身份认证**
- 若干网站身份**增强认证**技术
 - 手机短信口令
 - 动态口令
 - USB KEY
 - 数字证书

HTTP的Basic认证：机制原理

- 用户身份凭证：账号+静态口令
- 由于HTTP协议是面向一次连接的无状态网络协议，因此需要在每次发出HTTP请求时，把用户身份凭证的明文发送到服务器端，服务器与存储在服务器端的用户凭证进行比较
- 这是HTTP的Basic认证机制



HTTP的Basic认证：举例



The screenshot shows the Chrome DevTools Network tab with the 'Headers' sub-tab selected. The request is for 'index.html' from 'http://210.76.98.69/needauth/index.html' using the GET method. The status is 304 Not Modified. The 'Request Headers' section is expanded, showing various headers. The 'Authorization' header is highlighted with a red box, displaying 'Basic dGVzdDoxMjM0NTY='.

Name Path
index.ht
/needauth

Headers Preview Response Cookies Timing

Request URL: http://210.76.98.69/needauth/index.html
Request Method: GET
Status Code: 304 Not Modified

▼ **Request Headers** view source

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- Accept-Charset: GBK,utf-8;q=0.7,*;q=0.3
- Accept-Encoding: gzip,deflate,sdch
- Accept-Language: zh-CN,zh;q=0.8
- Authorization: Basic dGVzdDoxMjM0NTY=**
- Cache-Control: max-age=0
- Connection: keep-alive
- Host: 210.76.98.69
- If-Modified-Since: Tue, 25 Sep 2012 11:53:00 GMT
- If-None-Match: "1614a9e-10-4ca855907ff00"
- User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.89 Safari/537.1

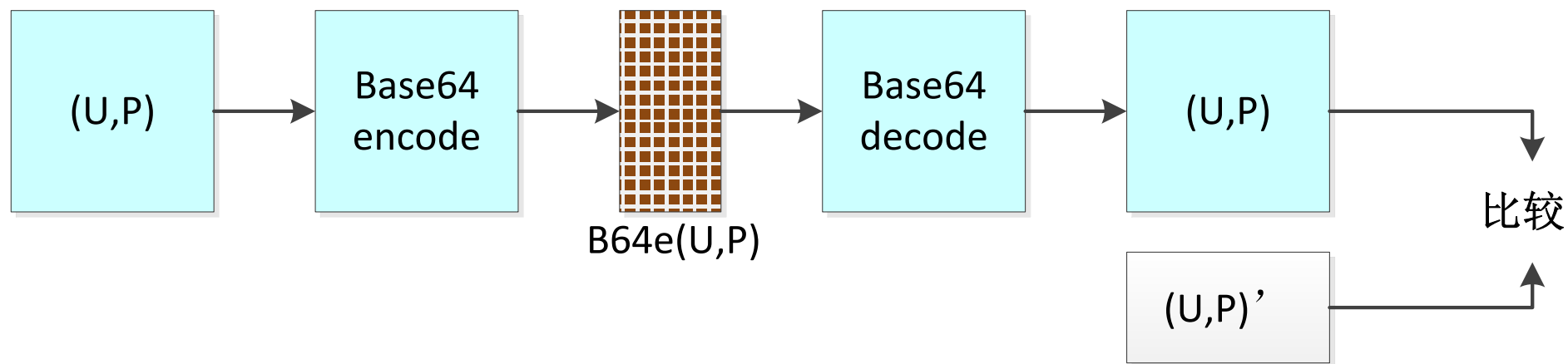
▼ **Response Headers** view source

- Connection: Keep-Alive
- Date: Tue, 25 Sep 2012 11:55:10 GMT
- ETag: "1614a9e-10-4ca855907ff00"
- Keep-Alive: timeout=5, max=100
- Server: Apache/2.2.21 (Unix) mod_ssl/2.2.21 OpenSSL/0.9.8e-fips-rhel5 DAV/2 Resin/3.0.28

```
[root@mailly bin]# echo "dGVzdDoxMjM0NTY=" | openssl base64 -d; echo  
test:123456  
[root@mailly bin]#
```

HTTP的Basic认证：认证过程

- 对账号口令 (U,P) 进行Base64编码(便于文本方式传输), 发送到Web服务器端
- 服务器端接收后进行Base64解码得到 (U,P) , 并与本地存储的账号口令 (U,P)' 进行**比较**, 认证用户身份



HTTP的Basic认证：优缺点

• 优点

- 没有复杂的交互过程
- 使用非常简单

• 缺点

- 每次都需要传递账号和口令，易于被监听、盗取，安全性低
- 本地（浏览器）还需要保存账号和口令，存在安全隐患
- 服务器端处理每个HTTP请求都要进行一次身份验证过程，效率低

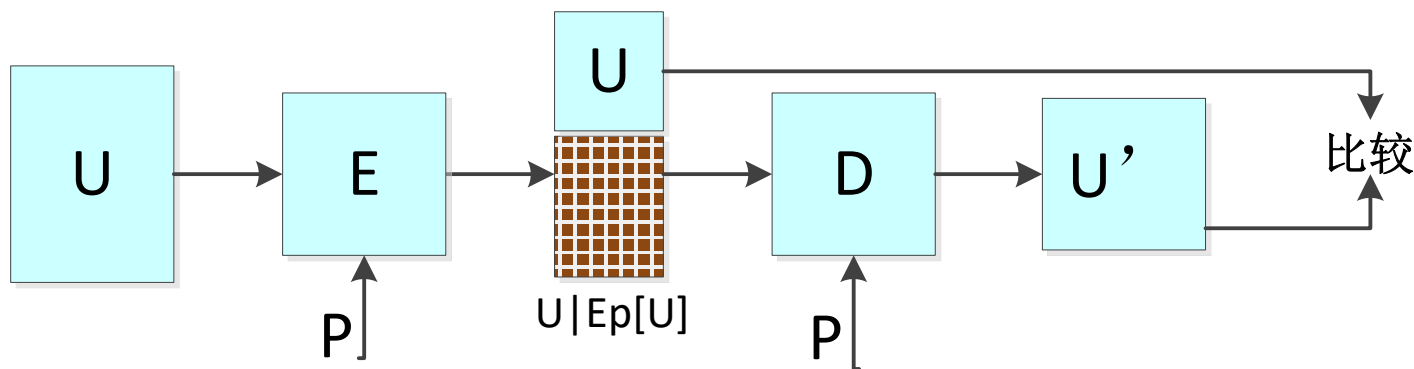
HTTP的Basic认证：改进1

- 解决账号口令明文传输可能被监听盗取的问题
 - 使用加密技术
 - 使用消息认证(MA)技术
- 解决账号口令长期本地保存以及服务器端每次都进行账号口令验证的问题
 - 使用表单验证+session的机制

HTTP的Basic认证：改进2

- 账号口令加密

- 加密需要密钥key，正好把口令作为密钥，即：k=口令
- 每次传输 $(U \parallel Ek[U])$ ，
 - 算法E可以是任意对称加密算法，如3DES、AES、RC5等

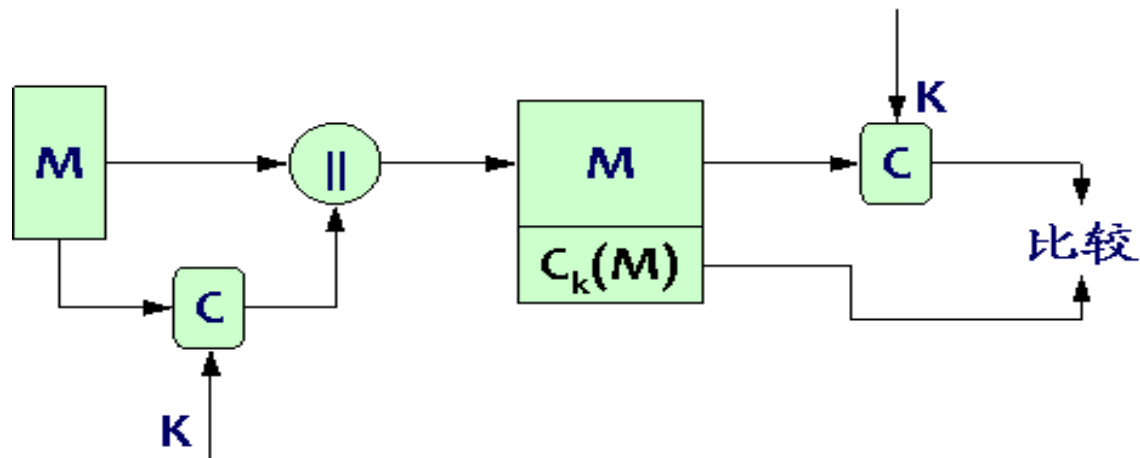


- 但是，无法躲避重放攻击(Replay)！

HTTP的Basic认证：改进3

- 使用消息认证中的MAC认证技术，

- 客户端和服务端共享密码K（口令）



- 采用挑战/响应(Challenge/Response)机制，需要进行两次HTTP请求

- 第一次请求：服务器向客户端返回一个挑战码M
 - 挑战码M由服务器随机生成，可以有效避免重放攻击
- 第二次请求：客户端使用MAC认证发送MAC码，服务器端进行验证
 - C可用hash算法，如SHA1

基于表单的身份认证：目标

- 为了解决Basic认证的问题：
 - 账号口令长期本地保存
 - 服务器端每次都进行账号口令验证



The screenshot shows the Tsinghua University Information Portal (清华大学信息门户). The header includes the university's name and navigation links. The main navigation bar contains several categories: 全面从严治党教育, 学风建设年, “不忘初心 牢记使命”主题教育, 一站式服务工程, and 校机关服务导引. Below this is a login section with fields for 用户名 (Username) and 密码 (Password), a 登录 (Login) button, and a 忘记密码 (Forgot Password) link. To the left of the login fields is a 意见与反馈 (Opinion and Feedback) button. Below the login section is a list of 办公通知 (Office Notices) and 招标招租 (Bidding and Leasing) items, each with a timestamp. To the right of the login section is a large banner for the 70th anniversary of the founding of the People's Republic of China, featuring a large crowd of people in white uniforms and a red banner.

清华大学信息门户

清华主页 | 清华新闻 | 清华党建

教工版 | 学生版 | English | 意见反馈

全面从严治党教育 | 学风建设年 | “不忘初心 牢记使命”主题教育 | 一站式服务工程 | 校机关服务导引

意见与反馈

使用说明

用户名: 密码: 忘记密码

Search

办公通知 | 招标招租

2019年清华大学特等奖学金（研究生）参评学生申请材料公... 10-12 16:55

2019年清华大学教职工文艺汇演通知 10-12 16:04

关于开展办公室业务培训的通知 10-12 14:22

关于10月16日召开各单位党委（总支）书记例会的通知 10-12 14:21

关于启动顺义区“北京市科技成果转化统筹协调与服务平台”... 10-12 12:19

新遴选专利代理机构通知 10-12 11:55

关于开展“小金库”自查自纠工作的通知 10-12 11:13

青春，因祖国而绽放！清华大学师生献礼新中国成立70...

70载风雨兼程的前行，70年不舍昼夜的奋进，伟大的中国人民用智慧的大脑和勤劳的双手创造了举世瞩目的东方奇迹。历史的车轮滚滚向前，新时代的青年团结奋进，以青...

1 2 3 4 5 6

基于表单的身份认证： Web的Session机制

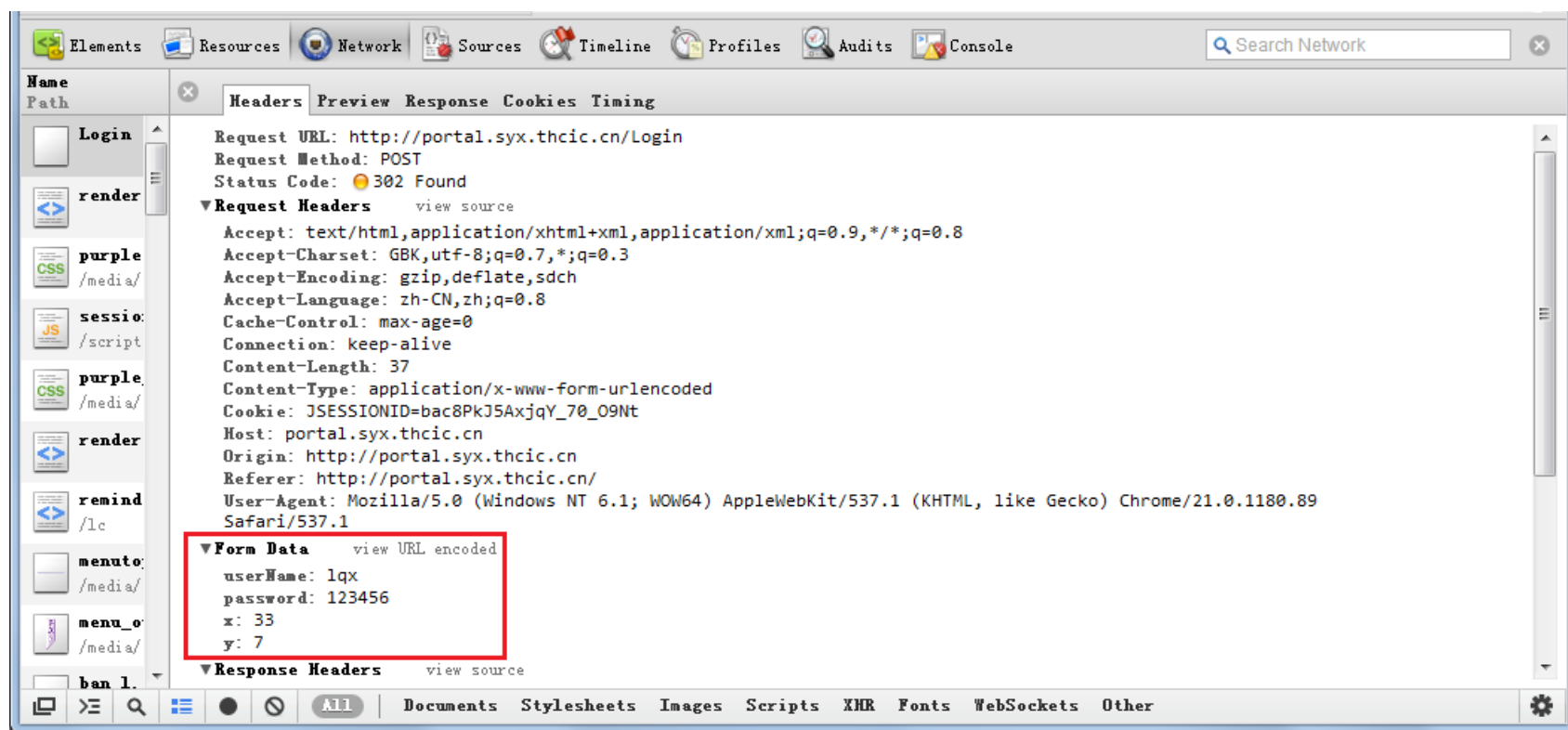
- 一个Session包括特定的客户端、特定的服务器端以及特定的操作时间段
- Session的工作原理：
 - 当某个Session首次启用时，服务器会产生一个唯一的标识符发送到客户端，客户端收到后会存储下来
 - 唯一标识符通常是一长串随机字符串，通常存于内存或本地文件中
 - 在客户端浏览器上，唯一标识符通常用Cookie技术进行存储
 - Cookie包含：key/value值对、作用域、作用路径、有效时间
 - 在Session存活期间，客户端每次向服务器发送的HTTP请求都会包含上述唯一标识符，使得服务器能够把前后多次请求关联起来
 - 当Session结束时，服务器端和客户端都应当销毁上述唯一标识符

基于表单的身份认证：过程

- 基于表单的Web身份认证过程通常包括三个步骤：
 - 第一步：客户端向服务器发送请求，服务器返回包含表单的页面
 - 第二步：用户按要求填写表单的内容完成后，客户端把表单内容发送到服务器；服务器获取表单中的内容后，进行验证，验证通过则启动Session并返回给客户端
 - 服务器端可以完成接收到客户端发送的表单数据，即：服务器端能获得用户账号口令的明文，可以直接进行比较验证
 - 第三步：客户端后续的请求包含Session的唯一标识符，服务端验证唯一标识符的合法性
 - 通常基于一个以标识符为key的哈希表数据结构

基于表单的身份认证：优缺点

- 优点：简单、方便、易用
- 缺点：安全性低，账号和口令明文在网络上传输，很大程度上存在被监听盗取的可能



基于表单的身份认证：改进1

- 引入Challenge/Response机制，避免口令明文通过网路传输，并且能避免重放攻击：
 - 第一步，服务器返回表单页面时，包含一个Challenge（通常为随机字符串+时间戳，记为M）
 - 第二步，用户完成表单填写提交请求之前，以口令为加密密钥（ $k=\text{口令}$ ）计算 $E_k[M||U]$ ，并将账号U和 $E_k[M||U]$ 发送到服务器，服务器收到后，用存储的口令 k' 计算 $E_{k'}[M||U]$ ，与收到的 $E_k[M||U]$ 比较，完成用户的身份验证

基于表单的身份认证：改进2

- 在第二步传输账号口令时，
使用传输层SSL协议传输HTTP请求
 - SSL协议(Secure Sockets Layer) 是为网络通信提供安全及数据完整性的一种安全协议，SSL在传输层对网络连接进行加密，确保数据在网络上的传输过程中不会被截取、窃听和篡改
 - SSL协议可以验证客户端和服务端软件的身份，确保数据发送到正确的客户机和服务器
 - 在Web中启用SSL协议只需把URL改为<https://>即可
 - SSL协议在“网络安全协议”中会有详细介绍

基于表单的身份认证：常见的不安全做法

- 常见的不安全做法
 - 验证成功后，把账号口令以Cookie机制存放在浏览器端
 - 验证成功后，把账号口令进行加密后 $E_k[U||P]$ 以Cookie机制存放在浏览器客户端
- 带来的问题：口令泄露，重放攻击
- 如果非要这么用
 - 存放内容加上时间戳 t ，加密 $E_k[t||U||P]$
 - 服务器端每次需要验证时间戳

网站身份增强认证的其它方式

- 仅靠静态口令认证用户安全强度不足，需要更高强度的认证方式
- 通常在静态口令的基础上加上其他因素的认证方式，形成双因素认证或多因素认证方式
 - 手机短信口令
 - 动态口令
 - USB KEY
 - 数字证书

手机口令

- 通常不会单独使用手机口令，而是与静态口令配合形成双因素认证方式
- 用户建立帐号时，需要与一个手机号码绑定
- 用户登录网站后，进行某些关键操作时，会触发系统向手机发送一条包含随机验证码的短信
- 当用户正确输入该随机验证码后，才可以继续刚才的关键操作



动态口令

- 动态口令牌通常也与静态口令验证形成双因素身份验证模式，广泛应用于VPN、网上银行、电子商务等领域
- 动态口令牌是客户手持用来生成动态密码的终端（手机上运行的程序或者是专用硬件终端）
 - 有的基于时间同步方式，每n秒变换一次有效的动态口令
 - 有的基于计数器同步方式，每次触发计数器累进会产生一个一次有效的动态口令

基于时间同步的动态口令原理

- 服务器与令牌端基于同样的算法，且双方的时间经过初始化校准（记录时间偏移）
- 算法原理
 - 服务器与令牌端共享一个key和令牌序列号sn
 - 每次计算先对时间t按区间n划分取整（上取整或下取整）
 - 计算 $F(H(t||key||sn))$ ，其中H是散列函数，F是压缩函数，把散列结果转换为6位或8为十进制数
- 用户验证时，提交令牌端计算的数字，服务器端按同样的算法计算（考虑到网络造成的时间差可以计算 $t-n, t$ 两个值）进行比较验证

USB KEY

- USB KEY是一种USB接口的硬件设备，它内置单片机或智能卡芯片，可以存储用户的密钥或数字证书
- USB KEY可以用于身份认证、数据加密和数字签名
 - USB KEY中的私钥不可导出，仅能用于内置的加密或签名运算
- 利用USB KEY实现对用户身份认证的原理是：
 - 服务器端产生随机消息M送到USB KEY内
 - USB KEY用内部私钥对M进行签名
 - 服务器使用用户的公钥验证签名

数字证书

- 数字证书就是互联网通讯中标志通讯各方身份信息的一系列数据，提供了一种在Internet上验证身份的方式
- 数字证书由权威机构，又称为证书授权（Certificate Authority）中心发行
- 数字证书是一个经证书授权中心数字签名的文件
 - 最简单的证书包含一个拥有者的公钥、名称以及证书授权中心的数字签名
- 数字证书拥有者对应的私钥应该另外妥善存储



其它身份认证应用



操作系统登录身份认证

• Windows系统

- 用户帐号信息存在SAM数据库中(Security Account Manager)
- 用户口令会进行加密处理存放到SAM中
- 用户输入帐号口令后，系统对输入的信息进行加密处理，与SAM中的数据进行**比较**，成功则完成后续登录过程

• Linux操作系统

- 用户帐号信息通常存放于/etc/passwd和/etc/shadow文件中
- 用户口令经过散列处理后进行存储，形如
test:\$1\$ZcUVmoZK\$gG/300k.Qqjt4UY4ks9UF/:12766:0:99999:7:::
- 用户输入帐号口令后，系统对输入的用户口令进行同样的散列处理，与shadow文件中的信息进行**比较**，成功则完成后续登录过程

802.1X网络准入身份认证

- 802.1X是一个认证协议，是一种基于端口对用户进行认证的方法和策略
- 对于一个端口：
 - 如果认证成功，就“打开”这个端口，允许所有的网络包通过
 - 如果认证不成功，就使这个端口保持“关闭”，此时只允许802.1X 的认证数据包通过

802.1x认证过程

- 802.1X主要是基于MD5的Challenge/Response认证



TUNet用户登录认证

- TUNet是学校用于登录校园网的客户端软件
- TUNet登录认证原理
 - 客户端C向服务器S发起TCP连接
 - 连接成功后，C向S发送“Hello”请求
 - S产生8字节随机数R，发送给C
 - C计算 $M = E_k(R)$ ，其中E为3DES算法， $k = \text{MD5}(\text{password})$ ，把M发送给S
 - S同样计算得到M，与C发送过来的比较
- 服务器每次发送给客户端的响应都包含一个新的8字节随机数 R_n ，下一次请求需要包含 $H(M || R_n)$ 以进行验证

SSH协议登录认证



- SSH是安全登录协议，替换原来的TELNET协议
- SSH全程通信数据加密
- SSH协议基于Diffie-Hellman密钥交换过程，并在第一次密钥交换之后，记录服务器的Diffie-Hellman参数(q,a)，形如
 - 192.168.1.1 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEA7ehjuoPsPeNHohtxIE2okxlHP1D8wsRqnca/7xhMnUv3V0fz
RS/ktKgclFcf9JIOLR3iWkZTPfJyepVfP9mJT4Nr77NJREBYl42W4IPpbhn5U6Eb259g/gecqADzmbh
dI6AOD7C0Qj5i/UU7rpK0pKSrhv/5AATER9KgeELwU60=
- SSH支持对服务器和客户端身份真实性进行验证

计算机网络安全技术

清华大学

Activity is the only road to knowledge
Computer Network Security @ 2020Fall