中山大學
SUN YAT-SEN UNIVERSITY

Module I. Fundamentals of Information Security

Chapter 2
Cryptographic Techniques

Web Security: Theory & Applications

School of Data & Computer Science, Sun Yat-sen University

Outline

1. Introduction

- 针对消息内容的攻击方法：
  - 伪造消息
  - 窜改消息内容
  - 改变消息顺序
  - 消息重放或者延迟
- 消息认证
  - 如果接收方计算的认证信息与收到的匹配，则：
    - 接收者可以确信消息未被改变；
    - 接收者可以确信消息来自所声称的发送者；
    - 如果消息中含有序列号，则可以保证正确的消息顺序。
  - 单一的消息认证技术不一定能够提供机密性保护和不可抵赖性。

1. Introduction

❑ 消息认证的方式

- Message Authenticated Encryption
  - ✧ 消息认证加密方法。对整个消息进行加密，以密文作为消息的认证标识 (tag)。消息加密可以采用现有的密码体制实现。
- Message Authentication Codes
  - ✧ 消息认证码。使用一个公开函数，加上一个密钥，为消息产生一个固定长度的小数据块 (即为消息认证码 MAC，或称密码校验和 Cryptographic Checksum) 作为消息的认证标识并附加到消息中一起传输。
- Hash
  - ✧ 哈希法使用一个公开函数，将任意长度的消息映射到一个固定长度的散列值，作为消息认证标识并附加到消息中一起传输。
- Digital Signatures
  - ✧ 基于非对称密码技术的数字签名将签名与被签的文件“绑定”，提供了不可抵赖、不能伪造和容易验证等功能。

2. Message Authentication Code

❑ 概述

- A message authentication code consists of three algorithms:
  - ✧ A key generation algorithm selects a key from the key space uniformly at random.
  - ✧ A signing algorithm efficiently returns a tag given the key and the message.
  - ✧ A verifying algorithm efficiently verifies the authenticity of the message given the key and the tag. That is, return *accepted* when the message and tag are not tampered with or forged, and otherwise return *rejected*.
- For a secure unforgeable message authentication code, it should be computationally infeasible to compute a valid tag of the given message without knowledge of the key, even if for the worst case, we assume the adversary can forge the tag of any message except the given one.

2. Message Authentication Code

❑ 概述

- Formally, a **MAC** is a triple of efficient algorithms (G, S, V) satisfying:
  - ✧ G (key-generator) gives the key k on input 1^n, where n is the security parameter, 1^n refers to a unary number.
  - ✧ S (signing) outputs a tag t on the key k and the input string x.
  - ✧ V (verifying) outputs *accepted* or *rejected* on inputs: the key k, the string x and the tag t. S and V must satisfy the following:

$$\Pr[k \leftarrow G(1^n), V(k, x, S(k, x)) = \textit{accepted}] = 1.$$

- A MAC is **unforgeable** if for every efficient adversary A:

$$\Pr[k \leftarrow G(1^n), (x, t) \leftarrow A^{S(k, \cdot)}(1^n), x \notin \text{Query}(A^{S(k, \cdot)}, 1^n), V(k, x, t) = \textit{accepted}] < \text{negl}(n),$$

where $A^{S(k, \cdot)}$ denotes that A has access to the oracle $S(k, \cdot)$, and $\text{Query}(A^{S(k, \cdot)}, 1^n)$ denotes the set of the queries on S made by A, which knows n. Clearly we require that any adversary cannot directly query the string x on S, since otherwise a valid tag can be easily obtained by that adversary.

2. Message Authentication Code

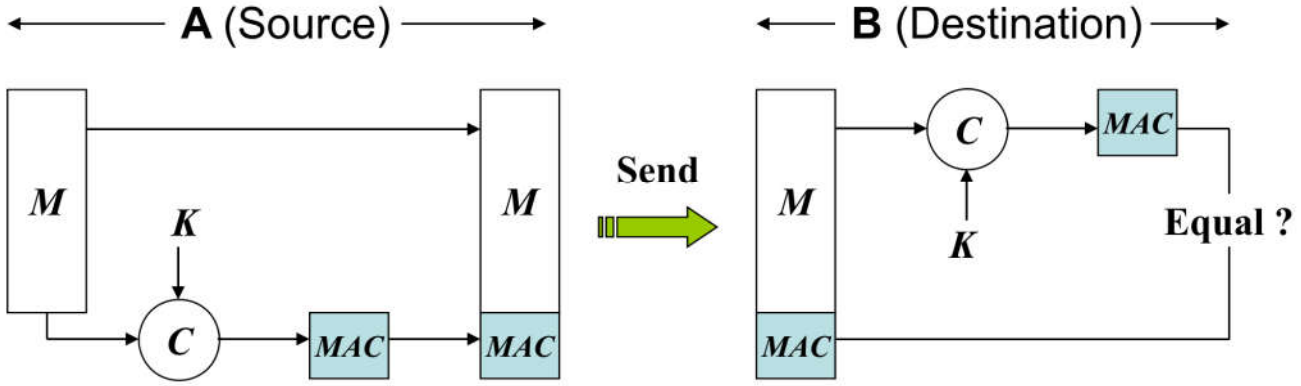
❑ 概述

- 基本结构
  - ✧ MAC 方法使用一个双方共享的密钥，为目标消息生成一个固定长度的小数据块，并加入到消息中。该数据块称为消息认证码 MAC，或密码校验和 (Cryptographic Checksum)。
  - ✧ MAC 函数类似于加密函数，但不需要可逆性，因此受到攻击的弱点在数学上比加密算法要少。
- 使用 MAC 的理由：
  - ✧ 保密性与真实性不同，加密算法主要提供保密性；
  - ✧ 加密算法的代价偏大；
  - ✧ 认证函数与保密函数分离能够提供功能上的灵活性；
  - ✧ 认证码可以提高对消息报文的真实性的保护期限；
  - ✧ 很多信息需要的是真实性，而不是保密性 (比如公告信息、海量广播式信息)。

2. Message Authentication Code

❑ 概述

- 只用于消息认证的 MAC 模型
  - ✧ 用户 *Alice* 和用户 *Bob* 共享认证密钥 *K*。*C* 为 MAC 函数。对于消息 *M*，*MAC* = *C*(*K*, *M*)。
  - ✧ 信道传输 *M* || *MAC*，模型只提供消息认证，没有提供保密性。



2. Message Authentication Code

❑ 概述

- 用于与明文相关的消息认证和保密性的 MAC 模型
  - ✧ 用户 *Alice* 和用户 *Bob* 共享认证密钥 K 和加密密钥 K_2。 C 为 MAC 函数，E 和 D 分别为加解密函数。
  - ✧ 对于明文消息 M，模型先计算与 M 相关的消息认证码 MAC，附加在明文 M 的末尾，再加密形成密文 M' 传输。
    - ◌ $MAC = C(K, M)$， $M' = E(K_2, M \| MAC)$；
    - ◌ 信道传输 M'。
  - ✧ 接收端先解密再认证。

2. Message Authentication Code

❑ 概述

- 用于与密文相关的消息认证和保密性的 MAC 模型
  - ✧ 用户 *Alice* 和用户 *Bob* 共享认证密钥 K 和加密密钥 K_2。 C 为 MAC 函数，E 和 D 分别为加解密函数。
  - ✧ 模型先对明文消息 M 加密，形成密文 M'，再计算与 M' 相关的消息认证码 MAC，附加在密文 M' 的末尾传输。
    - ◌ $M' = E(K_2, M)$，$MAC = C(K, M')$；
    - ◌ 信道传输 $M' \| MAC$；
  - ✧ 接收端先认证再解密。

2. Message Authentication Code

❑ 概述

- MAC 不等于数字签名。
  - ✧ MAC 和数字签名都用于消息认证。MAC 的通讯双方共享同一个私有密钥，而数字签名通常利用公钥密码技术实现。
- MAC 结构的重要性
  - ✧ 密钥足够长+加密算法足够好 ≠ 安全。
  - ✧ 攻击案例：
    - ○ 公开明文 $M = (X_1, X_2, \ldots, X_{t-1}, X_t)$
    - ○ 对 M 产生校验和 $\Delta M = X_1 \oplus X_2 \oplus \ldots \oplus X_{t-1} \oplus X_t$
    - ○ $MAC = E_K(\Delta M)$，K 认证密钥，E 加密算法
    - ○ 攻击者选择 $M' = (Y_1, Y_2, \ldots, Y_{t-1}, Y_t)$，使得 Y_t 满足

      $$Y_t = Y_1 \oplus Y_2 \oplus \ldots \oplus Y_{t-1} \oplus \Delta M$$
    - ○ 于是 $\Delta M' = \Delta M \Rightarrow E_K(\Delta M) = E_K(\Delta M')$
    - ○ 尽管攻击者不知道 K，仍然可以伪造消息 M'，使得 M' 和 M 具有相同的 MAC 值，可以通过消息认证。

2. Message Authentication Code

❑ MAC 算法

- 对 MAC 算法的安全性要求：
  - ✧ 条件：攻击者知道 MAC 函数 C 但不知道密钥 K。
  - ✧ 要求：
    - ○ 在已知消息 M 和MAC 值 $C(k, M)$ 的情况下，构造 M' 使得

      $$C(k, M') = C(k, M),$$

      在计算上不可行 (计算上无碰撞)。
    - ○ $C(k, M)$ 均匀分布：随机选择 M 和 M'，设 $|MAC|$ 是 MAC 值的位数，则

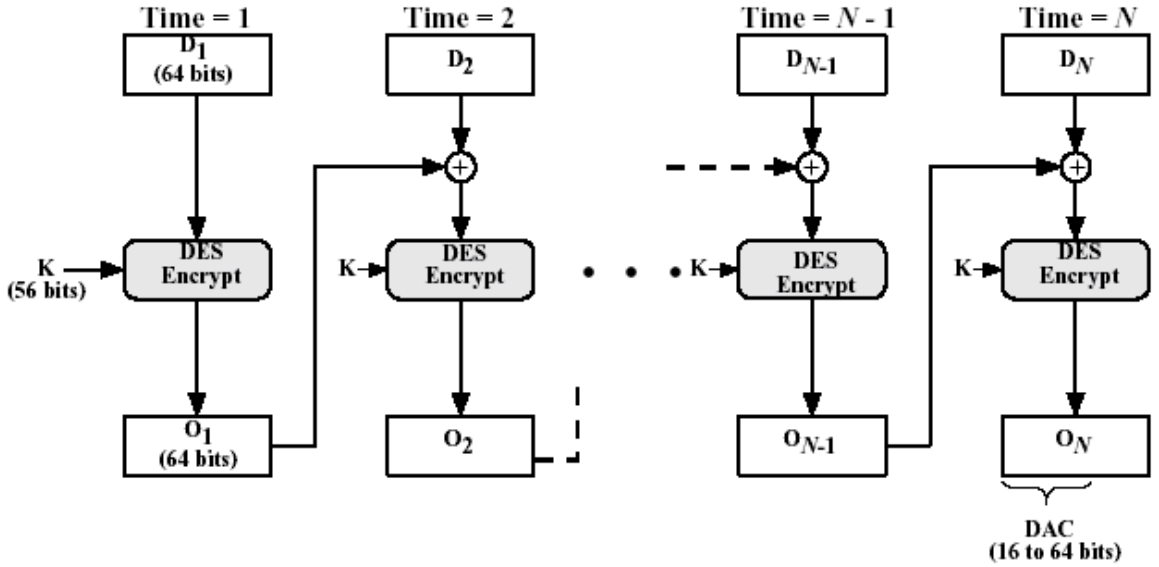
      $$\Pr[C(k, M) = C(k, M')] = 2^{-|MAC|}.$$
    - ○ MAC 值应平均依赖 M 的所有二进制位：如果 f 是 M 的一个变换 (例如对 M 的若干二进制位取反)，那么

      $$\Pr[C(k, M) = C(k, f(M))] = 2^{-|MAC|}.$$

2. Message Authentication Code

❑ MAC 算法

- ANSI x9.17 标准 (FIPS PUB 113)
  - ✧ CBC 模式结构，初始向量为0。消息按64-bit分组，不足时补0。
  - ✧ 基于 DES 加密算法，也适用于其他加密算法。
  - ✧ 认证标识可以由双方约定只取结果的高 *r* 位。美国联邦电信采用24bits (FTSC-1026)，美国金融系统则采用32bits (ABA 1986)。



2. Message Authentication Code

❑ MAC 算法

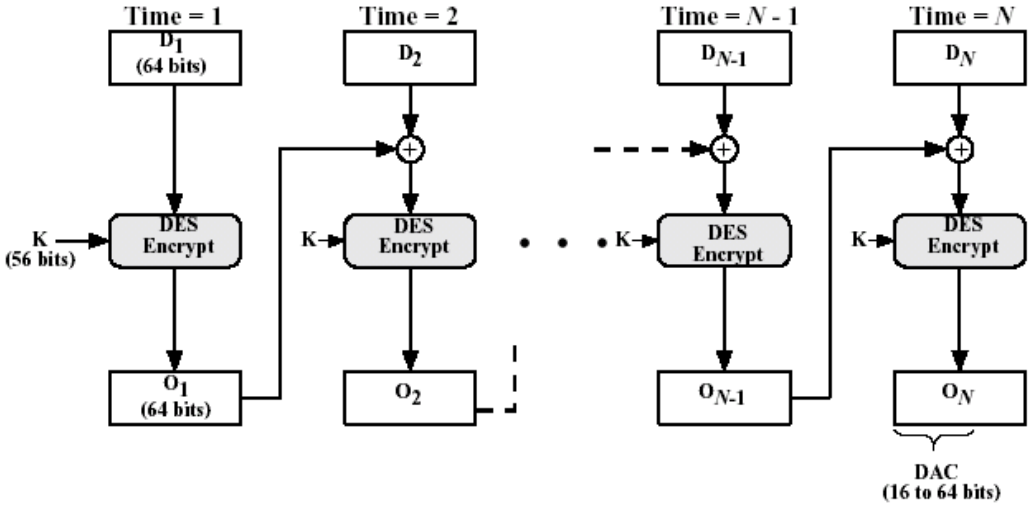
- ANSI x9.17 标准 (FIPS PUB 113)
  - ✧ 算法：

(1) $M = (D_1, D_2, \ldots, D_N)$

(2) $\Delta M_1 = E_K(D_1)$

(3) $\Delta M_{j+1} = E_K(D_{j+1} \oplus \Delta M_j)$, $1 \le j < N-1$

(4) $MAC = \Delta M_N$



中山大學 SUN YAT-SEN UNIVERSITY

3. Hash 方法

❑ 概述

- 散列 (Hashing) 通过某种特定的函数 (Hash 函数) 将要检索的项与用来检索的索引 (Hash Value 散列值) 关联起来，生成一种便于搜索的数据结构 (Hash List 散列表)。
- 在安全数据库设计中，散列算法也被用来加密保存在数据库中的密码字串。数据库中保管的是密码的散列值或密码指纹，而非密码本身，密码验证通过密码指纹验证实现。由于散列算法所计算出来的散列值具有不可逆 (无法逆向倒推原数值) 的性质，在数据库受到攻击的情况下，仍然能够有效地保护密码。
- 散列方法应用在信息安全领域，将源数据经过散列算法计算出数据指纹 (Data Fingerprint)，用于识别经过传播途径得到的数据是否有误 (通信误码或被窜改)，以保证数据的真实性。

3. Hash 方法

❑ Hash 函数

- Hash function : $h = H(M)$
  - ✧ M：变长消息，$H(M)$：定长的散列值。
  - ✧ Hash 函数 H 是一个公开函数，将任意长度的消息 M 变换为固定长度的散列码 h。如 MD5 输出128位，SHA-1输出160位。
  - ✧ Hash 函数是一种算法，算法的输出内容称为散列码或者消息摘要 (Message Digest)。消息摘要与原始消息唯一对应，不同的原始消息必须对应于不同的消息摘要，因此 Hash 函数能用来检测消息的完整性， 验证消息从建立开始到收到为止没有被改变或破坏。
  - ✧ Hash 函数又称为：哈希 (Hash) 函数、散列函数、数字指纹 (Digital Fingerprint)、压缩 (Compression) 函数、紧缩 (Contraction) 函数、数据认证码 DAC (Data Authentication Code)、篡改检验码 MDC (Manipulation Detection Code) 等。

3. Hash 方法

❑ Hash 函数

- 对 Hash 函数 $H(X)$ 的要求
  - ✧ $H(X)$ can be applied to any size data.
  - ✧ $H(X)$ produces a fixed-length output.
  - ✧ $H(X)$ is relatively easy to compute for any given X.
  - ✧ One-way property
    - ◌ For any given h, it is computationally infeasible to find X such that $H(X) = h$.
  - ✧ Weak collision resistance
    - ◌ For any given X, it is computationally infeasible to find $Y \neq X$ such that $H(Y) = H(X)$ (complexity of 2^n, n is the number of bits in the hash output).
  - ✧ Strong collision resistance
    - ◌ It is computationally infeasible to find any pair (X, Y) such that $H(X) = H(Y)$ (complexity of $2^{n/2}$, n is the number of bits in the hash output).

3. Hash 方法

❑ Hash 函数

- 对 Hash 函数 $H(X)$ 的要求
  - ✧ $H(X)$ 能用于任意大小的分组。
  - ✧ $H(X)$ 能产生定长的输出。
  - ✧ 对任何给定的 X，$H(X)$ 要相对易于计算，使得硬件和软件实现成为实际可能。
  - ✧ 对任何给定的码值 h，寻找 X 使得 $H(X) = h$ 在计算上是不可行的，即单向性 (One-way)。
  - ✧ 对任意给定的分组 X，寻找不等于 X 的 Y，使得 $H(X) = H(Y)$ 在计算上是不可行的，即弱抗冲突性 (Weak Collision Resistance/ Weak Collision-free, complexity of 2^n, n is the number of bits in the hash output)。
  - ✧ 找到任何满足 $H(X) = H(Y)$ 的一对 (X, Y) 在计算上是不可行的，即强抗冲突性 (Strong Collision Resistance/Strong Collision-free, complexity of $2^{n/2}$)。

3. Hash 方法

❑ Hash 函数的分类

- 根据碰撞规避能力的分类：
  - ✧ 弱无碰撞；
  - ✧ 强无碰撞 (包含了弱无碰撞)。
- 根据是否使用密钥的分类：
  - ✧ 不带密钥的 Hash 函数：Hash 值的产生过程不使用密钥，此时的 Hash 值也称为 MDC (Message Detection Code 消息检测码)；
  - ✧ 带密钥的 Hash 函数：Hash 值的产生由通信双方共享的密钥所控制，用于消息认证时称为 HMAC (Keyed-hash Message Authentication Code)。

3. Hash 方法

❑ 对散列方法的攻击

- 生日攻击理论

(1) 对于正整数 m，若 $k \geq 1.18\times 2^{m/2} \approx 2^{m/2}$，则 k 个在 $[1, 2^m]$ 的随机数中有两个数相等的概率不低于0.5；

(2)对于正整数 k, n，若 $k \geq 0.83\times n^{1/2}$，则两个由在 $[1, n]$ 的 k 个随机数构成的集合存在交集的概率不小于0.5。

✧ 例：求有两个人的生日相同的概率不低于0.5的最少人数？

◌ 解：由生日攻击理论 (1)，计算 $m = \log_2 365$，则

$$k \geq 1.18\times 2^{m/2} \approx 23。$$

✧ 因此，当 Hash 算法选用 m 位的 Hash 值时，k 组消息 (选择 $k=2^{m/2}$) 中至少有一对消息产生相同 Hash 值的概率超过0.5。

◌ MD5 算法输出摘要为128位，要找出至少2个消息的 Hash 值相同的概率大于0.5，则利用生日攻击需要进行的计算量 k 或时间复杂度为 2^{64}。类似地， SHA-1 算法输出摘要为160位，攻击需要进行的计算量为 2^{80}。

3. Hash 方法

❑ 对散列方法的攻击

- 生日攻击理论
  - ✧ 例：*Darth* 准备一份合法文档 M 申请 *Bob* 的签署，另外准备好一份伪造文档 M'。*Darth* 对 M 做32处符合其原文语义的微小改动，得到文档 $M_1, M_2, \ldots, M_{32}$，计算其全部2^{32}个组合的64位 Hash 值，记为集合 V。同时对 M' 也做32处符合其原文语义的微小改动，得到文档 $M'_1, M'_2, \ldots, M'_{32}$，计算其全部$2^{32}$个组合的64位 Hash 值，记为集合 V'。根据生日攻击理论 (2)，存在一个 Hash 值 t, 使得 $t \in V \cap V'$ 的概率不低于0.5。因此，存在一个文档 $M_1, M_2, \ldots, M_{32}$ 的组合 M_0 以及文档 $M'_1, M'_2, \ldots, M'_{32}$ 的组合 M'_0，它们具有相同的 Hash 值 t 的概率不低于0.5。
    - ◌ *Darth* 将 M_0 提交，*Bob* 审阅后生成64位 Hash 值 t，对该值签名后返还给 *Darth*。*Darth* 将 M_0 替换成 M'_0，附加 Hash 值 t 后发布。
  - ✧ 对策：采用足够长的 Hash 值：64 -> 128 -> 160 -> 256

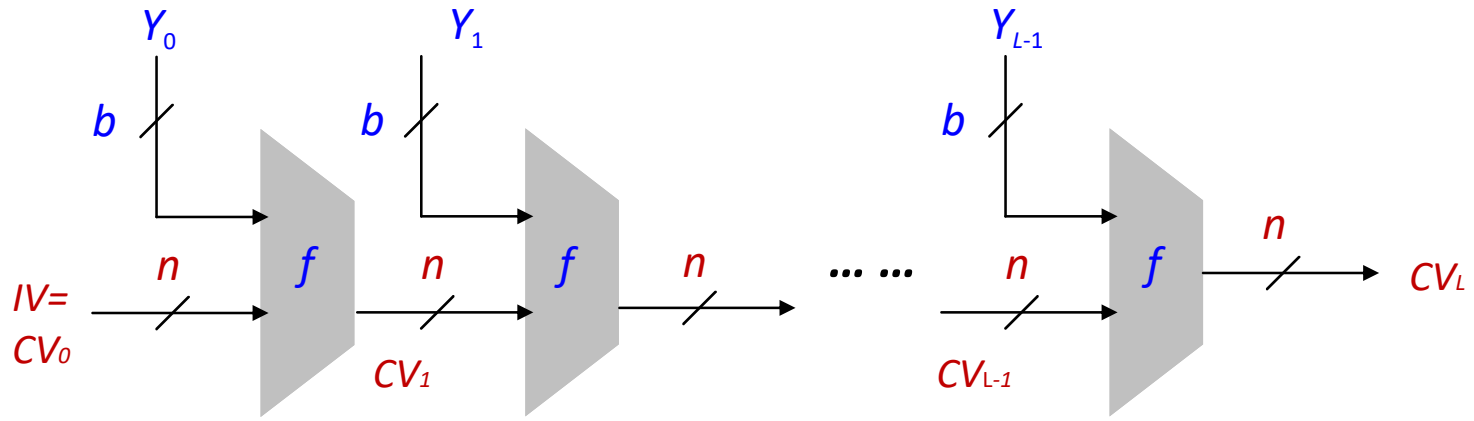
3. Hash 方法

❑ Hash 函数通用模型

- *Ralph Merkle* 于1989年提出Hash 函数通用模型，并得到广泛采用。
  - ✧ *R. Merkle* co-invented the *Merkle-Hellman* knapsack cryptosystem, invented cryptographic hashing, and invented *Merkle* trees.
- 模型结构
  - ✧ 把原始消息 M 分成一些固定长度的块 Y_i；
  - ✧ 最后一块 padding 并使其包含消息 M 的长度；
  - ✧ 设定初始值 CV_0；
  - ✧ 采用压缩函数 f，$CV_i = f(CV_{i-1}, Y_{i-1})$；
  - ✧ 最后一个 CV_i 为 hash 值。

3. Hash 方法

❑ Hash 函数通用模型



- Y_i – *i*-th input block (第 *i* 个输入数据块)
- *b* – length of input block (输入块的长度, 512/1024-SHA512)
- *f* – compression algorithm (压缩算法）
- *IV* – initial value (初始值)
- CV_i – chaining value (链接值)
- *n* – length of hash code (散列码的长度, 128/160/256/512)

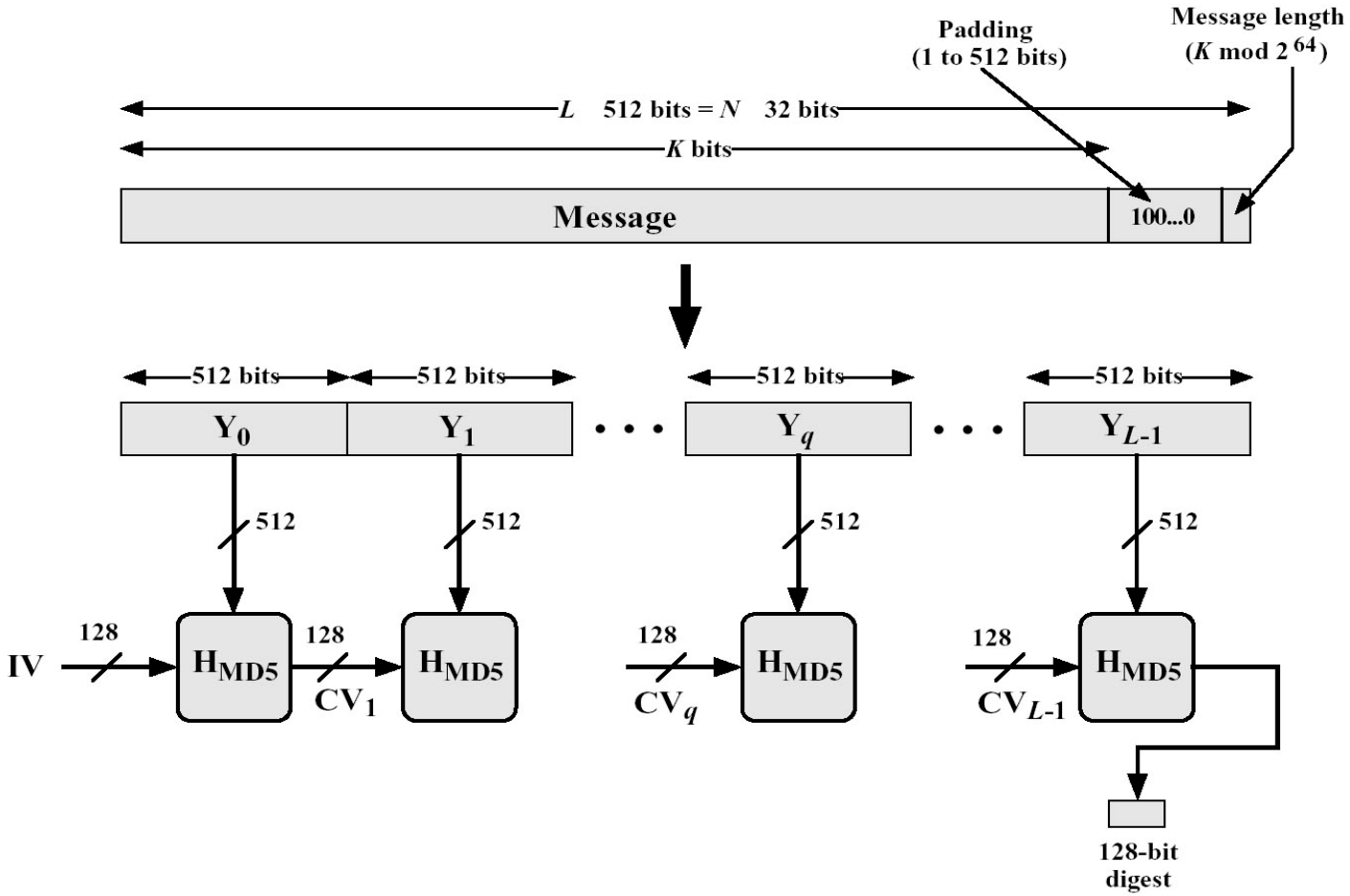
4. MD5 Algorithm

❑ 概述

- MD5 即 Message-Digest Algorithm 5 (信息-摘要算法 5)
  - ✧ MD4 (1990)、MD5(1992, RFC 1321) 由 *Ron Rivest* 发明，是广泛使用的 Hash 算法，用于确保信息传输的完整性和一致性。
  - ✧ MD5 使用 *little-endian* (小端模式)，输入任意不定长度信息，以 512-bit 进行分组，生成四个32-bit 数据，最后联合输出固定 128-bit 的信息摘要。
  - ✧ MD5 算法的基本过程为：填充、分块、缓冲区初始化、循环压缩、得出结果。
  - ✧ MD5 不是足够安全的。
    - ◌ *Hans Dobbertin* 在1996年找到了两个不同的 512-bit 块,它们在 MD5 计算下产生相同的 hash 值。
    - ◌ 至今还没有真正找到两个不同的消息，它们的 MD5 的 hash 值相等。

4. MD5 Algorithm

❑ MD5 的基本流程



4. MD5 Algorithm

❑ MD5 算法逻辑

- 填充 padding
  - ✧ 在长度为 K bits 的原始消息数据尾部填充长度为 P bits 的标识 100…0，$1 \le P \le 512$ (即至少要填充1个bit)，使得填充后的消息位数为：$K + P \equiv 448 \pmod{512}$.
    - ◌ 注意到当 $K \equiv 448 \pmod{512}$ 时，需要 $P = 512$.
  - ✧ 再向上述填充好的消息尾部附加 K 值的低64位 (即 $K \bmod 2^{64}$)，最后得到一个长度位数为 $K + P + 64 \equiv 0 \pmod{512}$ 的消息。
- 分块
  - ✧ 把填充后的消息结果分割为 L 个 512-bit 分组：$Y_0, Y_1, \ldots, Y_{L-1}$。
  - ✧ 分组结果也可表示成 N 个32-bit 字 $M_0, M_1, \ldots, M_{N-1}$，$N = L \times 16$。
- 初始化
  - ✧ 初始化一个128-bit 的 MD 缓冲区，记为 CV_q，表示成4个32-bit 寄存器 (A, B, C, D)；$CV_0 = IV$。迭代在 MD 缓冲区进行，最后一步的128-bit 输出即为算法结果。

4. MD5 Algorithm

❑ MD5 算法逻辑

- 初始化
  - ✧ 寄存器 (*A*, *B*, *C*, *D*) 置16进制初值作为初始向量 *IV*，并采用小端存储 (little-endian) 的存储结构：

- *A* = 0x67452301
- *B* = 0xEFCDAB89
- *C* = 0x98BADCFE
- *D* = 0x10325476

Word *A*	01	23	45	67
Word *B*	89	AB	CD	EF
Word *C*	FE	DC	BA	98
Word *D*	76	54	32	10

- ◌ Little-Endian 将低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。相反 Big-Endian 将高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。存储结构与 CPU 体系结构和语言编译器有关。PowerPC 系列采用 Big Endian 方式存储数据，而 Intel x86系列则采用 Little Endian 方式存储。

4. MD5 Algorithm

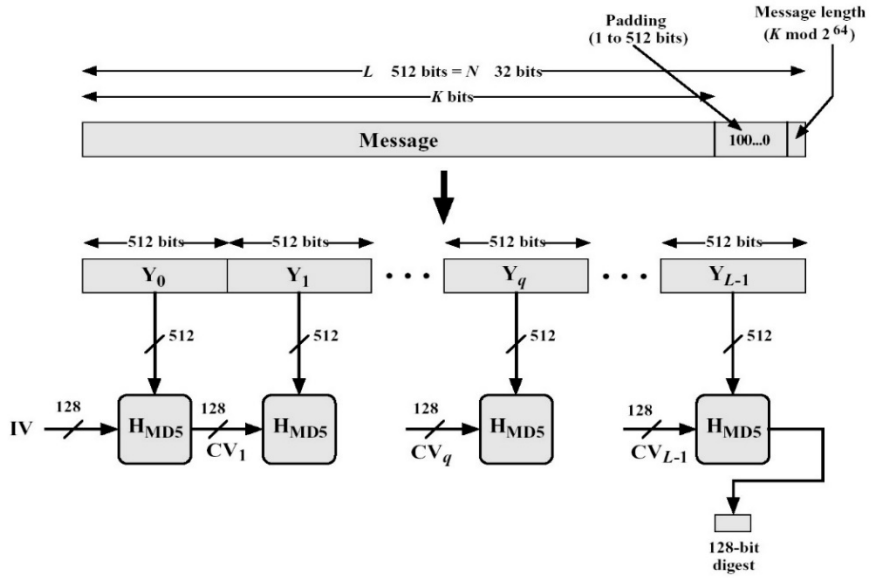
❑ MD5 算法逻辑

- 总控流程
  - ✧ 以512-bit 消息分组为单位，每一分组 Y_q (q = 0, 1, …, L-1) 经过4个循环的压缩算法，表示为：

$$CV_0 = IV$$

$$CV_i = H_{MD5}(CV_{i-1}, Y_i)$$

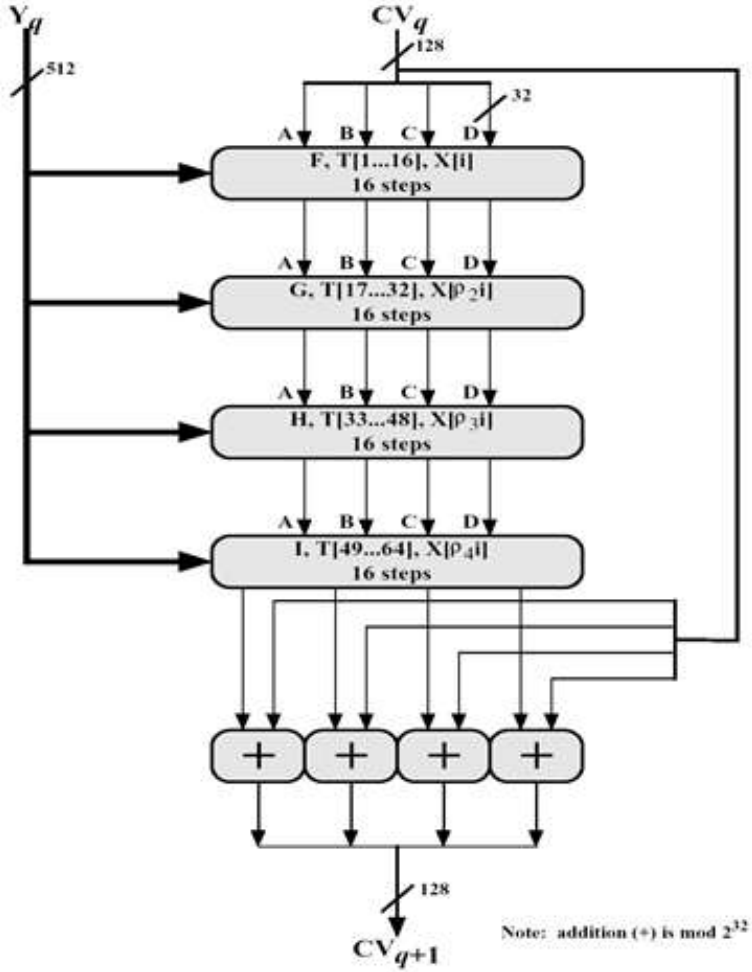
  - ✧ 输出结果：$MD = CV_L$.



4. MD5 Algorithm

❑ MD5 算法逻辑

- MD5 压缩函数 H_{MD5}
  - H_{MD5} 从 *CV* 输入128位，从消息分组输入512位，完成4轮循环后，输出128位，用于下一轮输入的 *CV* 值。
  - 每轮循环分别固定不同的生成函数 *F*, *G*, *H*, *I*，结合指定的 *T* 表元素 *T*[] 和消息分组的不同部分 *X*[] 做16次迭代运算，生成下一轮循环的输入。
  - 4轮循环总共有64次迭代运算。



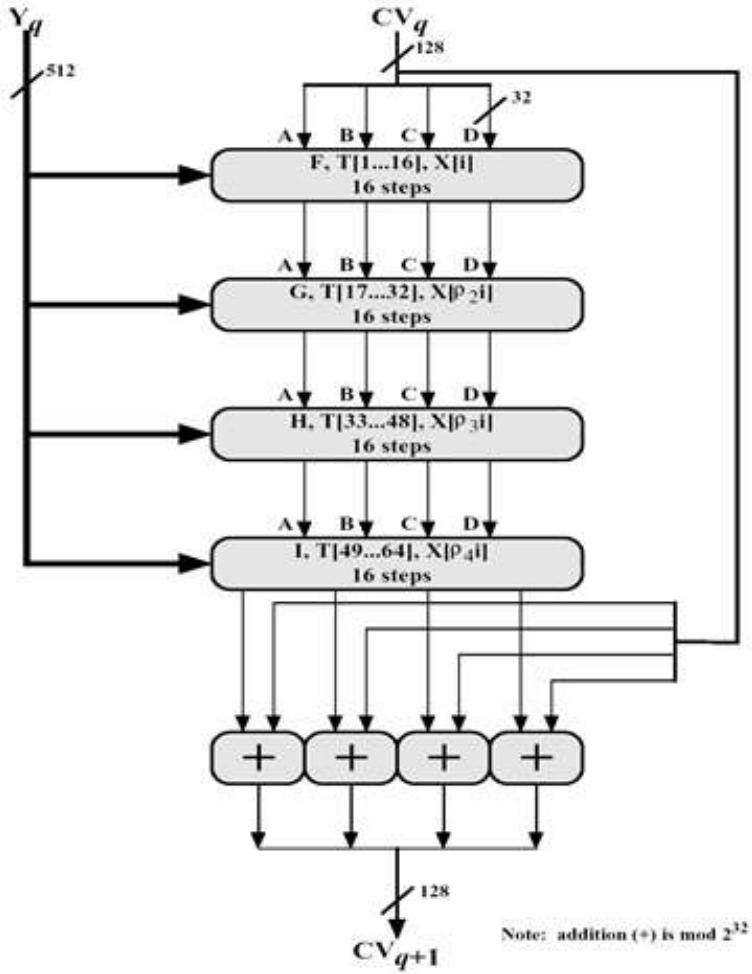
MD5 第 *q* 分组的4轮循环逻辑 (压缩函数)

4. MD5 Algorithm

❑ MD5 算法逻辑

- MD5 压缩函数 H_{MD5}
  - 4轮循环中使用的生成函数(轮函数) g 是一个32位非线性逻辑函数，在相应各轮的定义如下：

轮次	Function g	$g(b, c, d)$
1	$F(b, c, d)$	$(b \wedge c) \vee (\neg b \wedge d)$
2	$G(b, c, d)$	$(b \wedge d) \vee (c \wedge \neg d)$
3	$H(b, c, d)$	$b \oplus c \oplus d$
4	$I(b, c, d)$	$c \oplus (b \vee \neg d)$



MD5 第 q 分组的4轮循环逻辑 (压缩函数)

4. MD5 Algorithm

❑ MD5 算法逻辑

- MD5 压缩函数 H_{MD5}
  - ✧ 每轮循环中的一次迭代运算逻辑

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

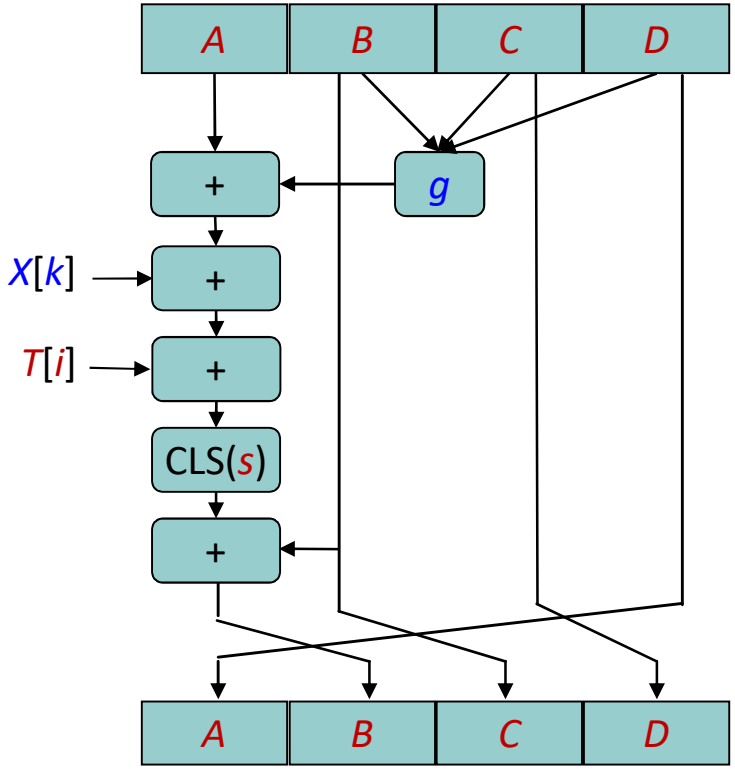
    (2) 缓冲区 (A, B, C, D) 作循环轮换：

    $(B, C, D, A) \leftarrow (A, B, C, D)$
  - ✧ 说明：
    - a, b, c, d : MD 缓冲区 (A, B, C, D) 的当前值。
    - g : 轮函数 $(F, G, H, I$ 中的一个)。
    - $<<<s$: 将32位输入循环左移 (CLS) s 位。
    - $X[k]$: 当前处理消息分组的第 k 个 (k = 0..15) 32位字，即 $M_{q\times16+k}$。
    - $T[i]$: T 表的第 i 个元素，32位字；T 表总共有64个元素，也称为加法常数。
    - + : 模 2^{32} 加法。

4. MD5 Algorithm

❑ MD5 算法逻辑

- MD5 压缩函数 H_{MD5}
  - ✧ 每轮循环中的一次迭代运算逻辑



4. MD5 Algorithm

❑ MD5 算法逻辑

- MD5 压缩函数 H_{MD5}
  - ✧ 各轮循环中第 i 次迭代 (i = 1..16) 使用的 $X[k]$ 的确定：

    设 $j = i - 1$：
    - ◌ 第1轮迭代：$k = j$.
      - 顺序使用 X[0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15]
    - ◌ 第2轮迭代：$k = (1 + 5j) \bmod 16$.
      - 顺序使用 X[1, 6,11, 0, 5,10,15, 4, 9,14, 3, 8,13, 2, 7,12]
    - ◌ 第3轮迭代：$k = (5 + 3j) \bmod 16$.
      - 顺序使用 X[5, 8,11,14, 1, 4, 7,10,13, 0, 3, 6, 9,12,15, 2]
    - ◌ 第4轮迭代：$k = 7j \bmod 16$.
      - 顺序使用 X[0, 7,14, 5,12, 3,10, 1, 8,15, 6,13, 4,11, 2, 9]

4. MD5 Algorithm

❑ MD5 算法逻辑

- MD5 压缩函数 H_{MD5}
  - ✧ *T* 表的生成
    - ◌ $T[i] = int(2^{32} \times |sin(i)|)$
      - *int* 取整函数，*sin* 正弦函数，以 *i* 作为弧度输入。
  - ✧ 各次迭代运算采用的 *T* 值：

$T[\ 1..\ 4] = \{\ 0xd76aa478,\ 0xe8c7b756,\ 0x242070db,\ 0xc1bdceee\ \}$

$T[\ 5..\ 8] = \{\ 0xf57c0faf,\ 0x4787c62a,\ 0xa8304613,\ 0xfd469501\ \}$

$T[\ 9..12] = \{\ 0x698098d8,\ 0x8b44f7af,\ 0xffff5bb1,\ 0x895cd7be\ \}$

$T[13..16] = \{\ 0x6b901122,\ 0xfd987193,\ 0xa679438e,\ 0x49b40821\ \}$

$T[17..20] = \{\ 0xf61e2562,\ 0xc040b340,\ 0x265e5a51,\ 0xe9b6c7aa\ \}$

$T[21..24] = \{\ 0xd62f105d,\ 0x02441453,\ 0xd8a1e681,\ 0xe7d3fbc8\ \}$

$T[25..28] = \{\ 0x21e1cde6,\ 0xc33707d6,\ 0xf4d50d87,\ 0x455a14ed\ \}$

$T[29..32] = \{\ 0xa9e3e905,\ 0xfcefa3f8,\ 0x676f02d9,\ 0x8d2a4c8a\ \}$

4. MD5 Algorithm

❑ MD5 算法逻辑

- MD5 压缩函数 H_{MD5}
  - ✧ 各次迭代运算采用的 *T* 值：

T[33..36] = { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }

T[37..40] = { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }

T[41..44] = { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }

T[45..48] = { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }

T[49..52] = { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }

T[53..56] = { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }

T[57..60] = { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }

T[61..64] = { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

4. MD5 Algorithm

❑ MD5 算法逻辑

- MD5 压缩函数 H_{MD5}
  - ✧ 各次迭代运算采用的左循环移位的 *s* 值：

    s[1..16] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22 }

    s[17..32] = { 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20 }

    s[33..48] = { 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23 }

    s[49..64] = { 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 }

  - ✧ 每次循环使用相同的迭代逻辑和 4×16 次运算的预设参数表。
  - ✧ Ref. to : RFC 1321

4. MD5 Algorithm

❑ MD5 算法逻辑

- 关于 Endianess 的问题
  - ✧ Endianess 涉及多字节数据类型与其内存对应存储结构的定义，与 CPU 架构和语言编译器有关。比如 x86 是 little-endian，PowerPC 则是 big-endian；C 编译器多采用 little-endian，Java 则是 big-endian；一些 ARM 架构可以支持 little-big 之间的转换。
  - ✧ Endianess 问题可能导致混合数据类型数据运算过程中内存数值转换的错误。
  - ✧ MD5 算法中从原始消息数据中顺序取4个字节，按 little-endian 转移到1个32位无符号整型变量，每16个32位无符号整型变量再顺序组成1个512位消息分组即时进行 H_{MD5} 压缩，同时累计原始消息数据的位数，保存在64位无符号整型变量 *count* 中。直到余下的原始消息数据不够组成1个消息分组而无法即时处理时，可以获得 *count* 的最终值。注意到 *count* 的值随着压缩的进行逐渐累加形成，而不需要进行预先统计。

4. MD5 Algorithm

❑ MD5 算法逻辑

- 关于 Endianess 的问题
  - ✧ 此时执行规范要求填充 100...0，留下末尾64位，将计数器 *count* 的低64位按 little-endian 转移成8个字节顺序填充 (注意到按照 RFC1321，*count* 在转移字节时被看成是2个32位的字，没有交换高低32位的位置)。之前余下的不能即时压缩的原始消息数据加上填充数据构成1-2个消息分组，继续对其进行 H_{MD5} 压缩。
  - ✧ 每轮循环中对32位寄存器 *A* 的迭代运算逻辑

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

    需要按照程序语言的定义调整各个 endianness 避免进位错误。
  - ✧ 算法结束时，对32位寄存器 *A* 的值按照 little-endian 转换成4个字节，顺序输出其8个16进制数符号；同样分别处理寄存器 *B*, *C*, *D*，输出 MD5 值的其他24个16进制数符号。寄存器 *A*, *B*, *C*, *D* 联合输出的结果是32个16进制数符号 (每个16进制数符号占4个二进制位，总共128位)。

4. MD5 Algorithm

❑ 几种散列函数的比较

主要参数	MD5	SHA-1	RIPEMD-160
摘要长度	128-bit	160-bit	160-bit
分组长度	512-bit	512-bit	512- bit
步数 Round	64	80	80
消息长度	No limit	2^{64}-1 bits	2^{64}-1 bits
基本逻辑函数	4	4	5
Endianess	Little-endian	Big-endian	Little-endian
性能*	32.4Mbps	14.1Mbps	13.6Mbps

5. Other MD Algorithms

❑ SHA

- SHA 的产生
  - ✧ 1992 NIST 制定 SHA (128位)
  - ✧ 1993 SHA 成为标准 (FIPS PUB 180)
  - ✧ 1994 修改产生 SHA-1 (160位)
  - ✧ 1995 SHA-1 成为新标准 (FIPS PUB 180-1/RFC 3174)
    - ◌ 输入消息长度 2^{64}-1
    - ◌ 摘要长度 160 bits
    - ◌ 输入块长度 512bits
    - ◌ 算法基础 MD4
  - ✧ 2001 FIPS PUB 180-2
    - ◌ SHA-256, SHA-384, SHA-512.
  - ✧ 2008 FIPS PUB 180-3
  - ✧ 2012 FIPS PUB 180-4

5. Other MD Algorithms

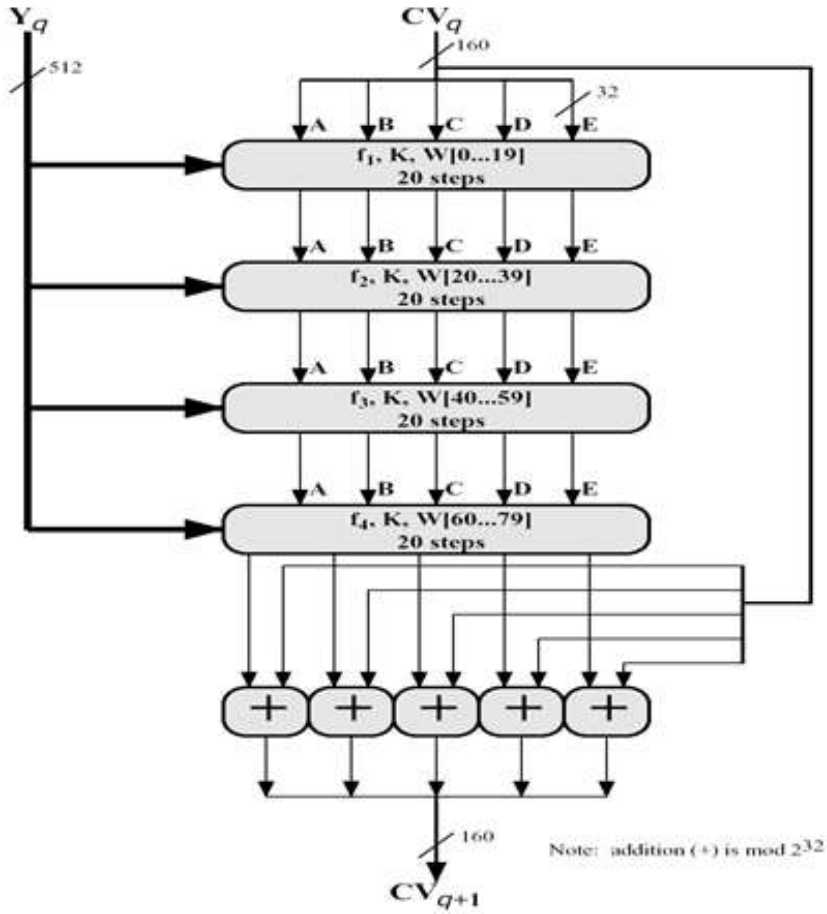
❑ SHA

- SHA-1 的算法结构与 MD4 类似
  - ✧ 第一步：padding
    - ◌ 补齐到512-bit 的倍数
  - ✧ 第二步
    - ◌ 分块长度512-bit
  - ✧ 第三步
    - ◌ 初始化 MD buffer 为160-bit 常量 (5个32-bit 长度的字)
    - ◌ 进入循环，160-bit 输入+512-bit 输入 $\Rightarrow$ 160-bit 输出
  - ✧ 第四步
    - ◌ 最后的输出为 SHA-1 的结果

5. Other MD Algorithms

❑ SHA

- SHA-1 的算法结构

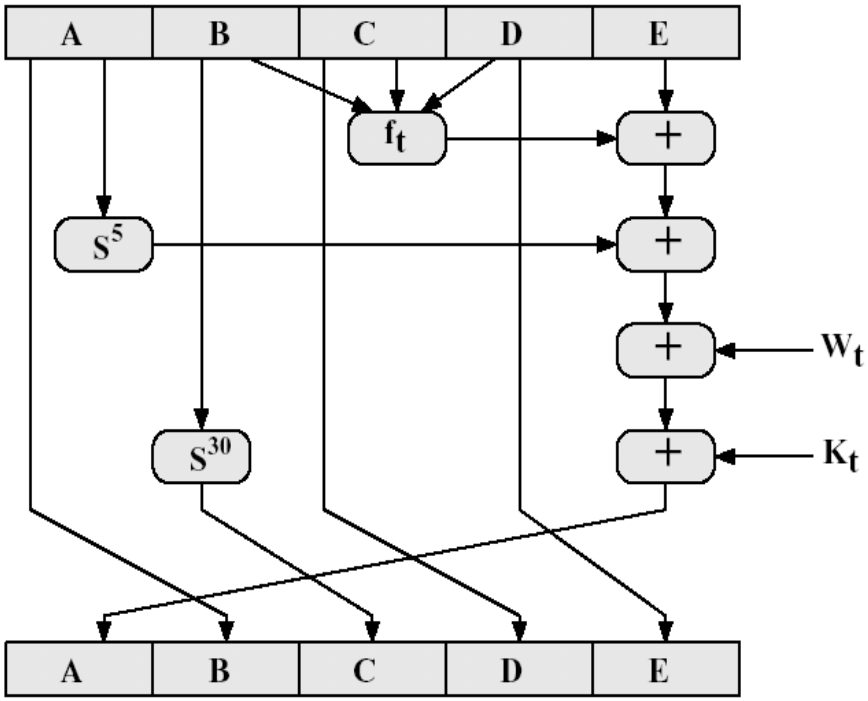


SHA-1 第 q 个分组的4轮循环逻辑 (压缩函数)

5. Other MD Algorithms

❑ SHA

- SHA-1 的算法结构
  - ✧ W_t : 从消息块导出
  - ✧ K_t : 常量



SHA-1 每轮循环中的一步运算逻辑

5. Other MD Algorithms

❑ SHA

- SHA-1 的一些讨论
  - ✧ 使用 big-endian
  - ✧ 抵抗生日攻击：160-bit hash 值
  - ✧ 没有发现两个不同的 512-bit 块,它们在 SHA-1 计算下产生相同的 hash 值
  - ✧ 速度慢于 MD5
  - ✧ 安全性优于 MD5

5. Other MD Algorithms

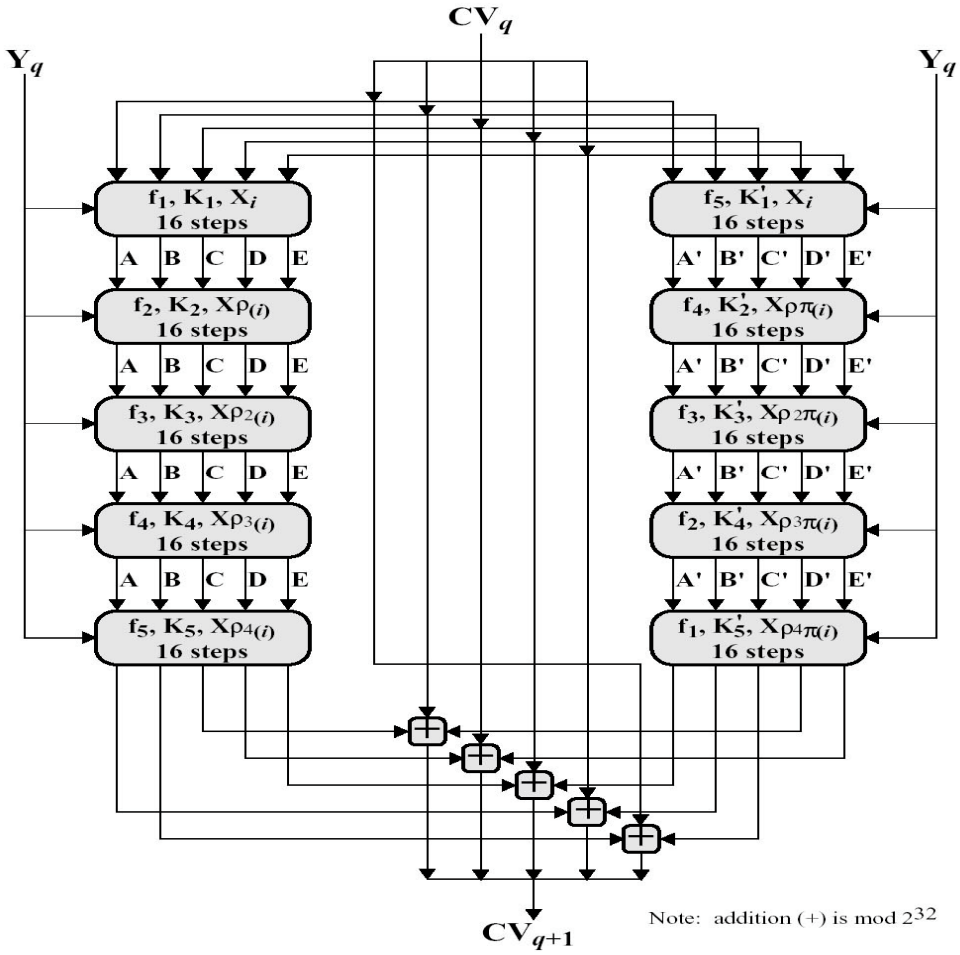
❑ RIPEMD

- 概况
  - ✧ RIPEMD (RACE Integrity Primitives Evaluation Message Digest), in the framework of the EU project RIPE (RACE Integrity Primitives Evaluation, 1988-1992), first published in 1996, is a family of cryptographic hash functions developed in Leuven, Belgium, by *Hans Dobbertin*, *Antoon Bosselaers* and *Bart Preneel* at the COSIC research group at the Katholieke Universiteit Leuven.
  - ✧ RIPEMD was based upon the design principles used in MD4, and is similar in performance to the more popular SHA-1.
  - ✧ RIPEMD-160 was designed in the open academic community, in contrast to the NSA-designed SHA-1 and SHA-2 algorithms.
    - ◌ Length counter size: 64 bits
    - ◌ Block size: 512 bits
    - ◌ Output size: 160 bits

5. Other MD Algorithms

❑ RIPEMD

- RIPEMD 的算法结构



5. Other MD Algorithms

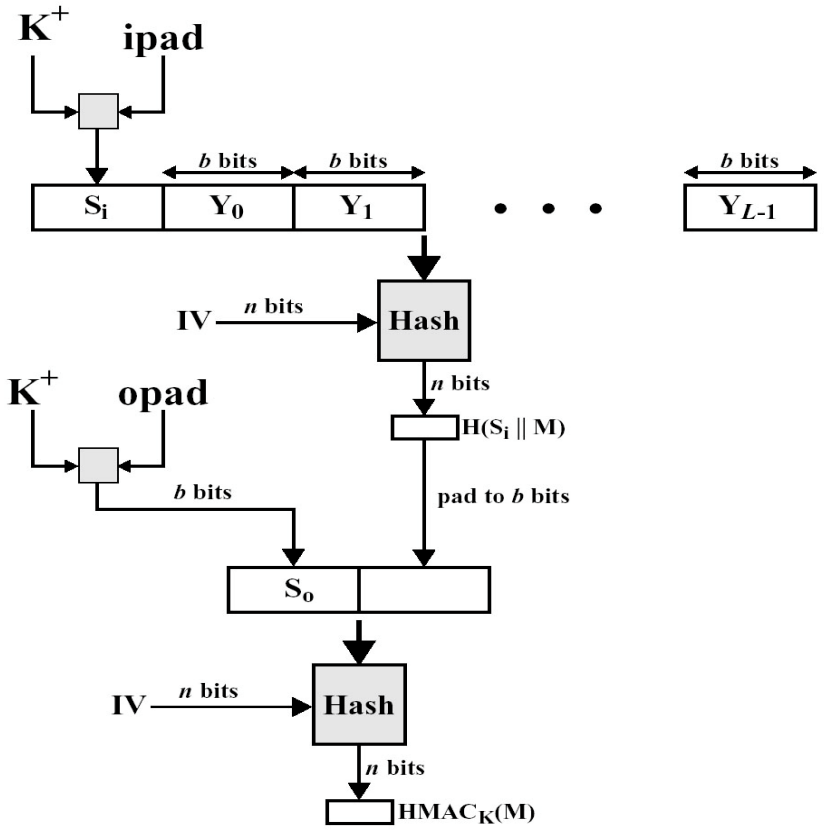
❑ HMAC

- What is HMAC
  - ✧ In cryptography, a *keyed-hash message authentication code* (HMAC) is a specific type of MAC involving a cryptographic hash function and a secret cryptographic key.
  - ✧ It may be used to simultaneously verify both the data integrity and the authentication of a message, as with any MAC.
  - ✧ Any cryptographic hash function, such as MD5 or SHA-1, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5 or HMAC-SHA1 accordingly.
  - ✧ The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key.
  - ✧ Ref. to RFC 2104.

5. Other MD Algorithms

❑ HMAC

- HMAC 的算法结构



M – message input to HMAC

H – embedded hash function

k – secrete key, $|k| \le b$

b – length of input block

n – length of hash code

ipad – 00110110 重复 *b*/8 次

opad – 01011010 重复 *b*/8 次

$$HMAC_k = H((K^+ \oplus opad) \| H((K^+ \oplus ipad) \| M))$$

5. Other MD Algorithms

❑ HMAC

- HMAC 的算法结构
  - ✧ 对密钥 k 左边补0，生成一个 b 位的数据块 K^+；
  - ✧ K^+ 与 $ipad$ 作 XOR，生成 b 位的 S_i；
  - ✧ 对 $(S_i \| M)$ 进行标准过程的 hash 压缩，得到 $H(S_i \| M)$；
  - ✧ K^+ 与 $opad$ 作 XOR，生成 b 位的 S_o；
  - ✧ 对 $S_o \| H(S_i \| M)$ 进行标准过程的 hash 压缩，得到

$$HMAC_K = H(S_o \| H(S_i \| M)).$$

- HMAC 特征
  - ✧ 可直接使用各种 Hash 算法；
  - ✧ 可使用将来的更加安全和更加快速的 Hash 算法；
  - ✧ 保持原始 Hash 算法的性能；
  - ✧ 密钥的使用简单；
  - ✧ 与 Hash 函数有同等的安全性。

5. Other MD Algorithms

❑ 消息认证的局限性

- 消息认证可以保护通信双方不受第三方攻击，但是不能处理通信双方的相互攻击。
  - ✧ 信宿方伪造消息并利用与信源方共享的认证密钥生成认证码，从而声称消息来自信源方；
  - ✧ 由于上述原因，信源方可以对发送过的某一消息进行抵赖，
- 在通信双方无法相互信任的情况下，需要引入数字签名。

6. Digital Signature

❑ What is Digital Signatures

- A **digital signature** is a mathematical scheme for presenting the authenticity of digital messages or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender (*authentication*), that the sender cannot deny having sent the message (*non-repudiation*), and that the message was not altered in transit (*integrity*).
- Digital signatures are a standard element of most cryptographic protocol suites, and are commonly used for software distribution, financial transactions, contract management software, and in other cases where it is important to detect forgery or tampering.

6. Digital Signature

❑ Scheme of Digital Signatures

- A digital signature scheme typically consists of 3 algorithms:
  - ✧ A *key generation* algorithm that selects a *private key* uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding *public key*.
  - ✧ A *signing* algorithm that, given a message and a private key, produces a signature.
  - ✧ A *signature verifying* algorithm that, given the message, public key and signature, either accepts or rejects the message's claim to authenticity.

6. Digital Signature

❑ 数字签名的特征

- 接收方能够确认或证实发送方的签名，但不能伪造，称为 R-1 条件 (Unforgeable)；
- 发送方将签名消息发给接收方后，不能否认其签发的消息，称为 S-条件 (Non-repudiation)；
- 接收方对已经收到的签名消息不能否认，即有收报认证，称为 R-2 条件；
- 第三方可以确认收-发双方的消息传送，但不能伪造这一过程，称为 T-条件。
- 数字签名与手写签名的区别：手写签名字迹因人而异，数字签名是二进制数据，内容因消息而异；
- 数字签名与消息认证的区别：消息认证由接收方用于验证信源和消息内容是否被窜改，数字签名用来解决收-发双方的利害冲突；
- 数字签名与消息加密的区别：签名的安全性要求更高，而且要求更快的验证速度。

6. Digital Signature

❑ 数字签名的分类

- 按签名的内容划分：
  - ✧ 对整体消息的签名；
  - ✧ 对压缩信息的签名。
- 按明文-密文的对应关系划分：
  - ✧ 确定性 (Deterministic) 数字签名：明文与密文一一对应，对于一个特定的消息，其签名保持确定不变，如 RSA、Rabin 等；
  - ✧ 随机化 (Randomized) 或概率式数字签名。
- 普通数字签名算法
  - ✧ RSA
  - ✧ ElGamal / DSS / DSA
  - ✧ ECDSA
- 盲签名 (Blind Signature) 算法
  - ✧ 未知消息内容的情况下签名，如电子选举、数字货币协议等。
- 群签名算法

6. Digital Signature

❑ 数字签名标准 DSS

- Digital Signature Standard, FIPS 186, NIST 1994.
- DSS 扩充版, FIPS 186-2, NIST 2000.
- FIPS 186-3 in 2009.
- FIPS 186-4 in 2013.
- DSS 是 ElGamal 和 *Schnorr* 签名方案的改进版，使用的算法记为 DSA (Digital Signature Algorithm)，由 *D. W. Kravitz* 设计。早期版本的 DSS 使用 SHA-1，安全性由离散对数求解困难性保证。
- DSS 只用于数字签名，不用于加解密或密钥分配。
- DSS 的签名验证速度比产生签名的速度慢得多。

6. Digital Signature

❑ DSA 算法描述

- The **DSA** algorithm works in the framework of public-key cryptosystems and is based on the algebraic properties of the **modular exponentiations,** together with the **discrete logarithm problem** (which is considered to be computationally intractable). Messages are **signed** by the signer's **private key** and the signatures are **verified** by the signer's corresponding **public key**. The digital signature provides message authentication, integrity and non-repudiation.

6. Digital Signature

❑ DSA 算法描述

- **Parameter generation**
  - ✧ Choose an approved cryptographic hash function *H*. In the original DSS, *H* was always SHA-1, but the stronger SHA-2 hash functions are approved for use in the current DSS. The hash output may be truncated to the size of a key pair.
    - ◌ The hash functions to be used are specified in the Secure Hash Standard (SHS), FIPS 180. FIPS **approved** digital signature algorithms **shall** be used with an appropriate hash function that is specified in the SHS.
  - ✧ Decide on a key length *L* and *N*. This is the primary measure of the cryptographic strength of the key. The original DSS constrained *L* to be a multiple of 64 between 512 and 1,024 (inclusive). NIST 800-57 recommends lengths of 2,048 (or 3,072) for keys. FIPS 186-3 specifies *L* and *N* length pairs of (1,024, 160), (2,048, 224), (2,048, 256), and (3,072, 256). *N* must be less than or equal to the output length of the hash function *H*.

6. Digital Signature

❑ DSA 算法描述

- **Parameter generation**
  - ✧ Choose an L-bit prime p. Choose an N-bit prime q such that $q \mid p-1$.
  - ✧ Choose g, $1<g<p$. g's multiplicative order modulo p is q. This means that q is the smallest positive integer such that $g^q \equiv 1 \bmod p$.
    - ◌ This may be done by setting $g = h^{(p-1)/q} \bmod p$ for some arbitrary h, $1<h<p-1$ and trying again with a different h if the result comes out as 1. Most choices of h will lead to a usable g; commonly $h = 2$ is used.
      - Since gcd(h, p)=1, applying *Euler*'s Theorem we have

        $$h^{\varphi(p)} = h^{p-1} \equiv 1 \pmod{p}, \text{ and so}$$

        $$g^q \bmod p = (h^{(p-1)/q} \bmod p)^q \bmod p = h^{(p-1)} \bmod p = 1.$$
    - ◌ Efficient algorithms such as exponentiation by squaring can be applied to compute the modular exponentiations $h^{(p-1)/q} \bmod p$.
  - ✧ The algorithm parameters (p, q, g) may be shared between different users of the system.

6. Digital Signature

❑ DSA 算法描述

- **Per-user keys**
  - ✧ Given a set of parameters, the second phase computes private and public keys for a single user:
    - ◌ Choose a secret private key x; x is an integer randomly or pseudorandomly generated, such that $0 < x < q$.
    - ◌ Calculate the public key y where $y = g^x \bmod p$.
    - ◌ Efficient algorithms such as exponentiation by squaring can also be applied for computing the modular exponentiations $g^x \bmod p$.

6. Digital Signature

❑ DSA 算法描述

- **Signing**
  - ✧ Let H be the hashing function and M the message to be signed:
    - ◌ Generate a random or pseudorandom per-message value k where $1<k<q$.
    - ◌ Calculate $r = (g^k \bmod p) \bmod q$.
    - ◌ In the unlikely case that $r = 0$, start again with a different random k.
    - ◌ Calculate $s = k^{-1} (H(M) + xr) \bmod q$, k^{-1} is the multiplicative inverse of k modulo q.
    - ◌ In the unlikely case that $s = 0$, start again with a different k.
    - ◌ The signature is (r, s).
  - ✧ The modular exponentiation $g^k \bmod p$ is the most computationally expensive part of the signing operation, and the modular inverse $k^{-1} \bmod q$ (using the *Extended Euclidean Algorithm* or applying *Euler's Theorem* as $k^{q-2} \bmod q$) is the second most expensive part. Both may be computed before the message hash is known.

6. Digital Signature

❑ DSA 算法描述

- Verifying
  - ✧ Reject the signature (r, s) if $0<r<q$ or $0<s<q$ is not satisfied.
  - ✧ Calculate $w = s^{-1} \bmod q$ where s^{-1} is the multiplicative inverse of s modulo q.
  - ✧ Calculate $u_1 = H(M)w \bmod q$.
  - ✧ Calculate $u_2 = rw \bmod q$.
  - ✧ Calculate $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$.
  - ✧ The signature is valid if and only if $v = r$.

6. Digital Signature

❑ DSA 算法描述

- Correctness of DSA
  - ✧ The signature scheme is correct in the sense that the verifier will always accept genuine signatures. This can be shown as follows:
    - ◌ First, if $g = h^{(p-1)/q} \bmod p$, it follows that $g^q = h^{(p-1)} \equiv 1 \pmod p$ by *Euler's Theorem* for $\varphi(p) = p-1$ and $\gcd(h, p) = 1$. Since $1<g<p$ and q is prime, g must have order q modulo p.
    - ◌ The signer computes
      $$s = k^{-1}(H(M) + xr) \bmod q.$$
    - ◌ Thus $k \equiv H(M)s^{-1} + xrs^{-1} \equiv [H(M)w + xrw] \pmod q$.
    - ◌ Let $k = aq + h$, where h is the least positive remainder.
    - ◌ Then $H(M)w + xrw = bq + h$, here a, b are some integers.
    - ◌ Since g has order q modulo p we have
      $g^k = g^{aq+h} \equiv g^h \pmod p$, and $g^{H(M)w+xrw} = g^{bq+h} \equiv g^h \pmod p$.
    - ◌ Therefore $g^k \equiv g^{H(M)w} g^{xrw} \equiv g^{H(M)w} y^{rw} \equiv g^{u_1} y^{u_2} \pmod p$.
    - ◌ Finally, the correctness of DSA follows from
      $$r = (g^k \bmod p) \bmod q = (g^{u_1} y^{u_2} \bmod p) \bmod q = v.$$

6. Digital Signature

❑ 针对 DSA 算法的一些批评意见

- DSA 不能用于加密或密钥分配。
- DSA 由 NSA 支持开发，算法中会不会存在陷门。
- DSA 的验证速度不如 RSA。
- RSA 已经是一个事实标准。
- DSA 未公开选择过程，还没有足够的时间进行分析证明。
- DSA 可能一定程度上侵犯了其它专利 (如 *Schnorr* 签名算法、*Diffie-Hellman* 算法)。

Outline

2.1 Cryptology Introduction

2.2 Symmetric Key Cryptographic Algorithms

2.3 Mathematical Foundations of Public-Key Cryptography

2.4 Asymmetric Key Cryptographic Algorithms

2.5 MAC and Hashing Algorithms

2.6 Typical Applications

- MD5 and Passwords
- AES and Wi-Fi Protected Access
- RSA and e-Business

2.6 Typical Applications

❑ Discussing

- MD5 and Passwords
- AES and Wi-Fi Protected Access
- RSA and e-Business

References

1. William Stallings, Cryptography and Network Security: Principles and Practice (5th Edition), Prentice Hall 2010
2. http://en.wikipedia.org/wiki/
   - Birthday attack
   - Cryptography
   - Data Encryption Standard
   - Diffie–Hellman key exchange
   - Digital signature
   - Hash function
   - Information_security
   - MD5
   - Public-key cryptography
   - RSA

End of Chapter 2