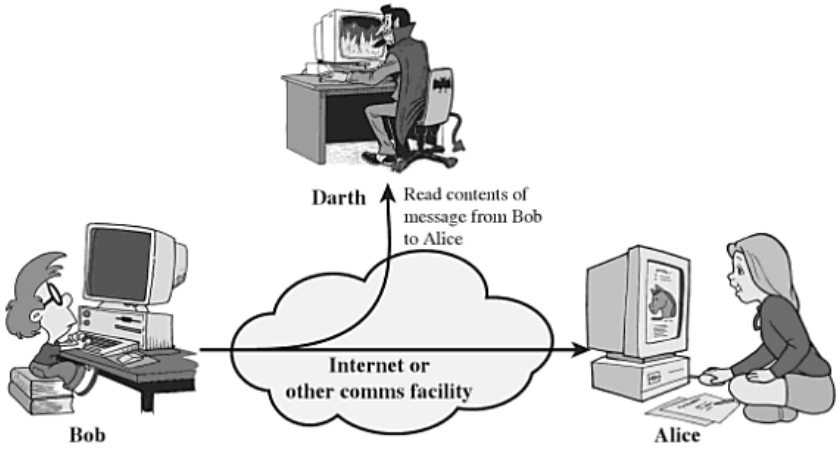
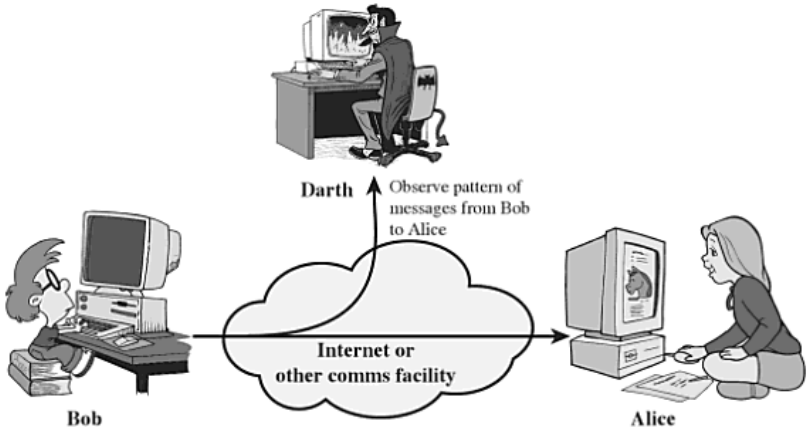
消息认证码和散列函数
MAC & HASH

消息认证码概述



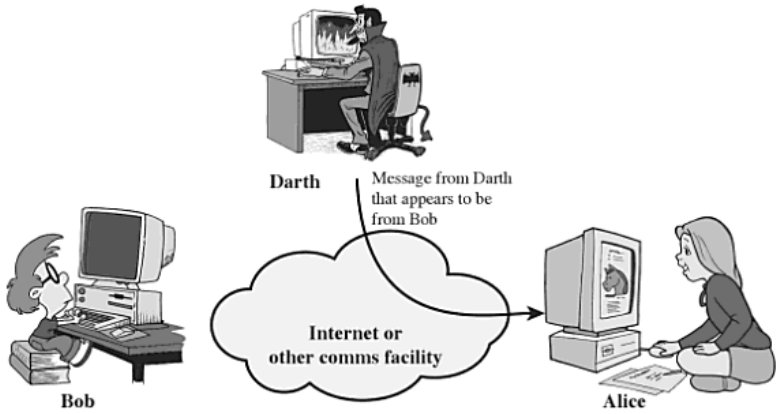
消息窃取



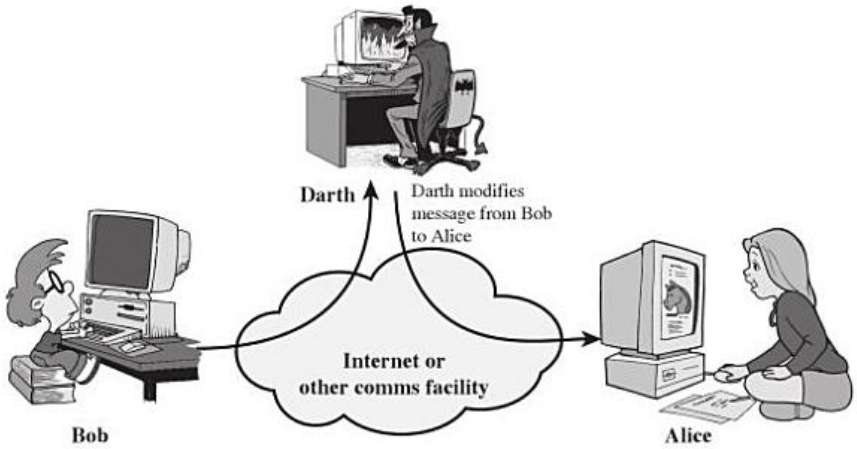
流量分析

被动攻击

消息认证码概述



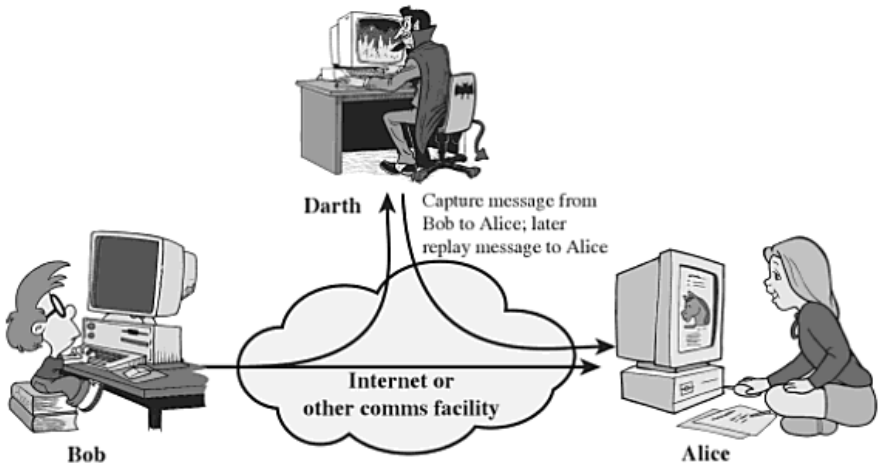
消息伪造



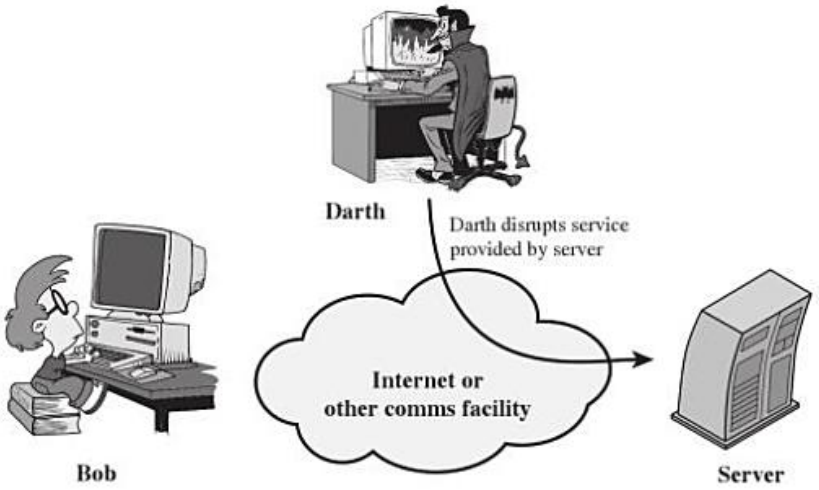
消息篡改

主动攻击

消息认证码概述



消息重放



拒绝服务攻击

主动攻击

消息认证码概述

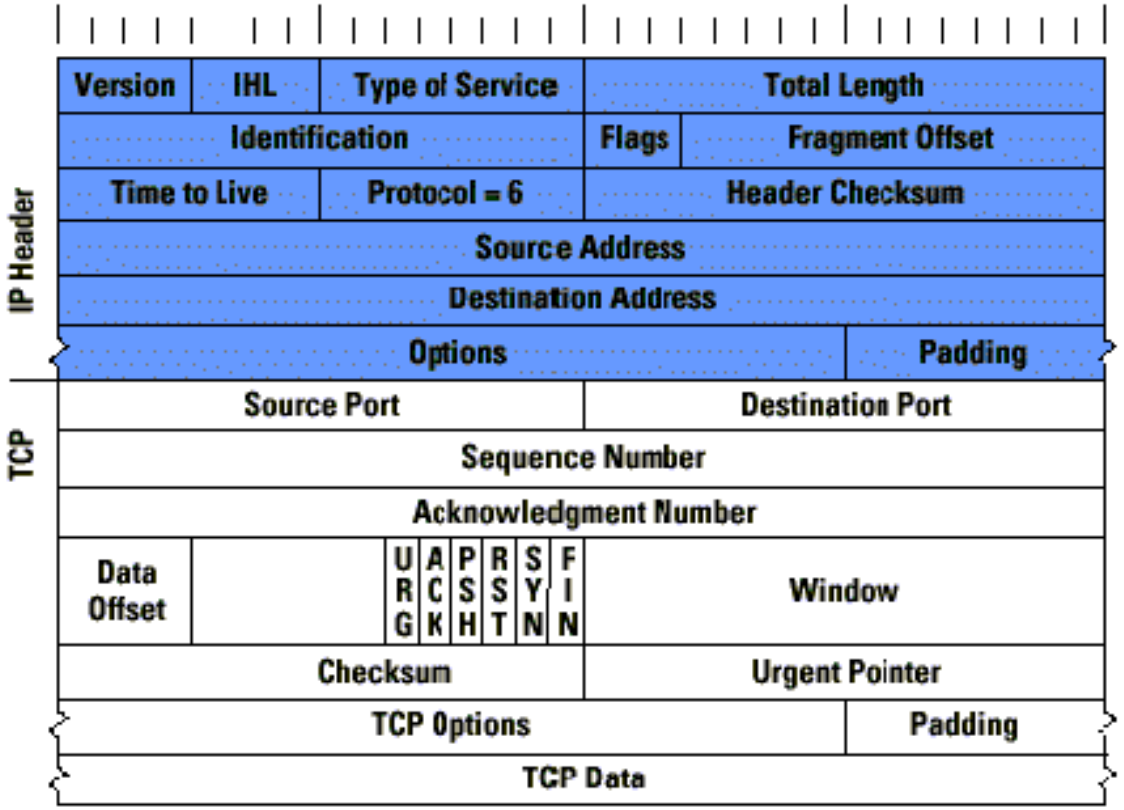
直接对数据进行加密是否可以抵抗消息篡改或伪造?

$$c = Enc_k(m) \qquad m' = Dec_k(c')$$

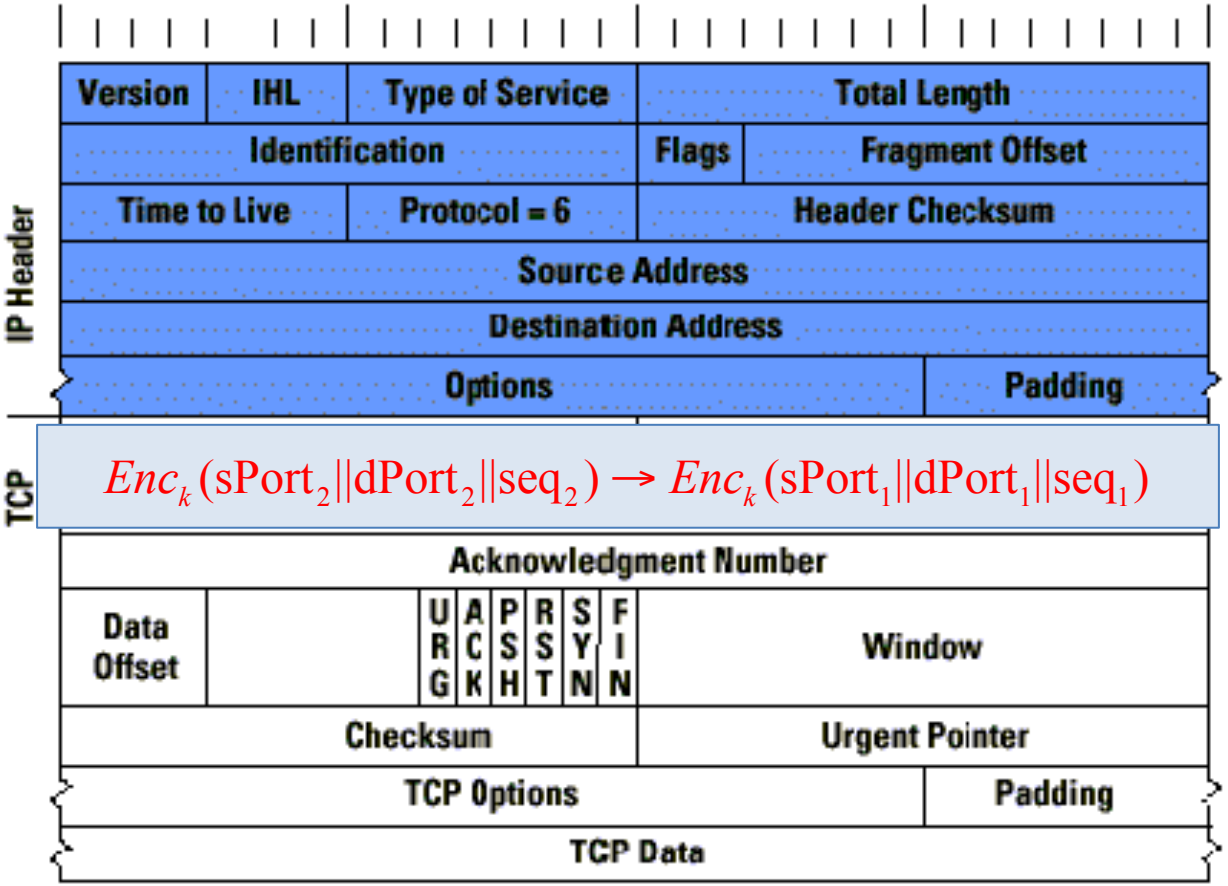
如果不限制消息的内容或格式，直接加密消息不能抵抗消息篡改或伪造。

消息认证码概述

限制消息的内容或格式，直接加密消息是否可以能抵抗消息篡改或伪造?

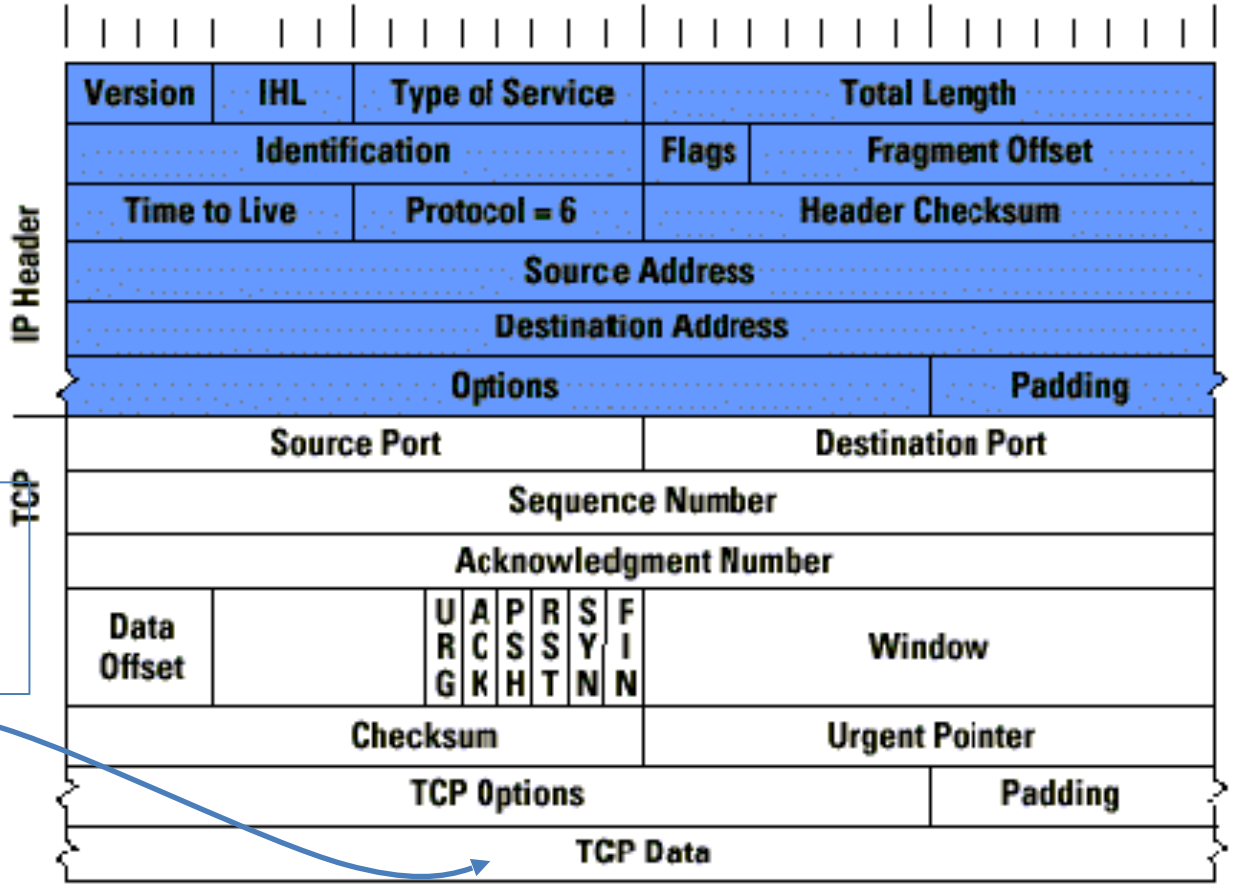


消息认证码概述



加密算法不能抵抗消息篡改或伪造

消息认证码概述



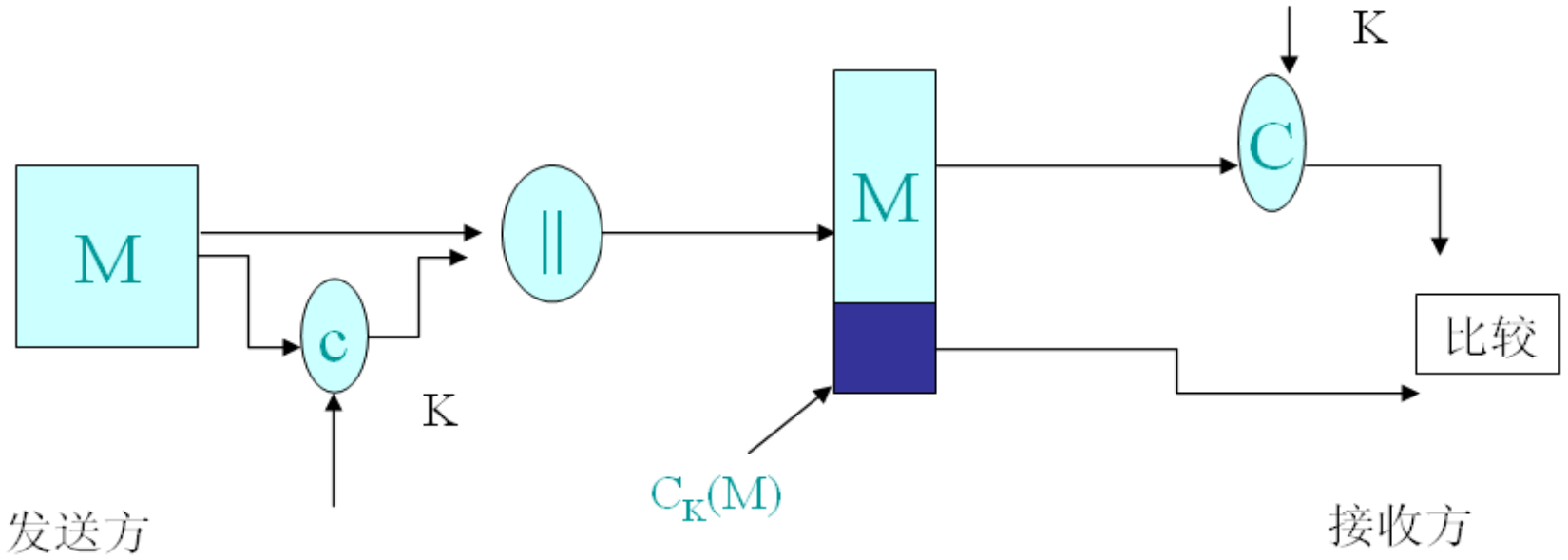
删除TCP Data的后面数据

加密算法不能抵抗消息篡改或伪造

消息认证码概述

消息认证码（Message Authentication Code，MAC）

消息认证码是密码学的一个重要研究方向，是保证消息完整性的基本算法，利用密钥计算消息的认证信息。



消息认证码算法

消息算法包括三个子算法：

1. 密钥生成子算法(Gen)

   算法输入：安全参数n；算法输出：满足特定分布的密钥k

2. 消息认证码子算法(Mac)

   算法输入：密钥k和消息m；

   算法输出：Mac标签t，$t \leftarrow Mac(k, m)$

3. 验证子算法(Vrfy)

   算法输入：密钥k，消息m，标签t；

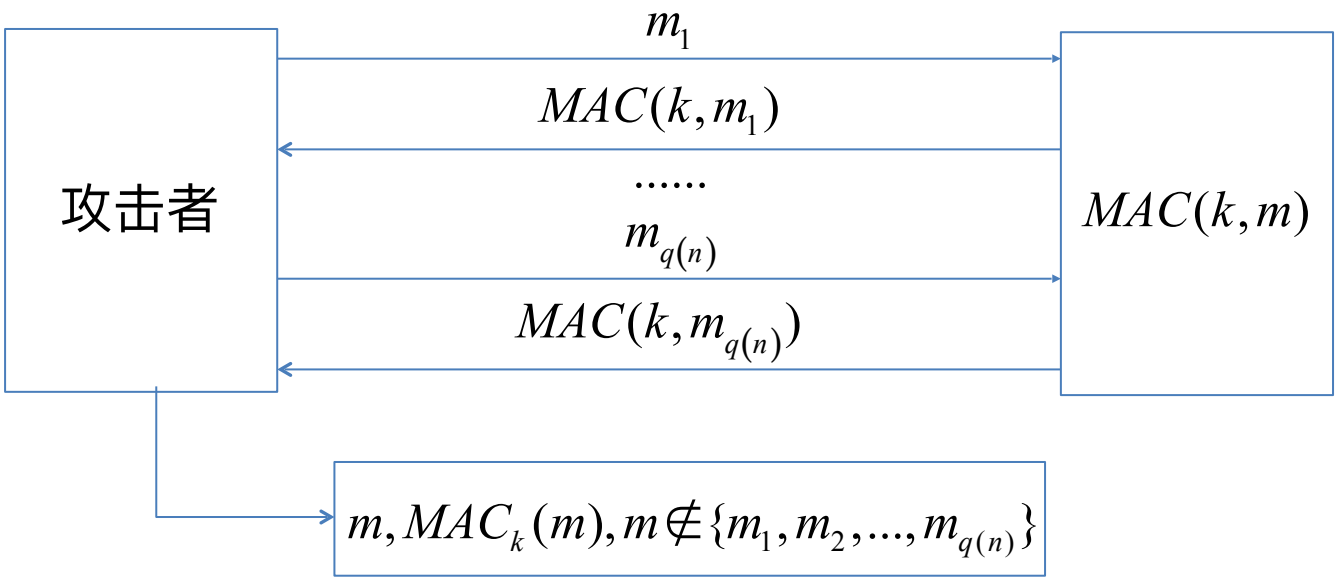
   算法输出：b=Vrfy(k, m, t) 验证通过为1，验证失败为0

Vrfy(k, m, Mac(k, m)) = 1

消息认证码概述

抵抗选择消息攻击的消息认证码 （Mac-forge 实验流程）

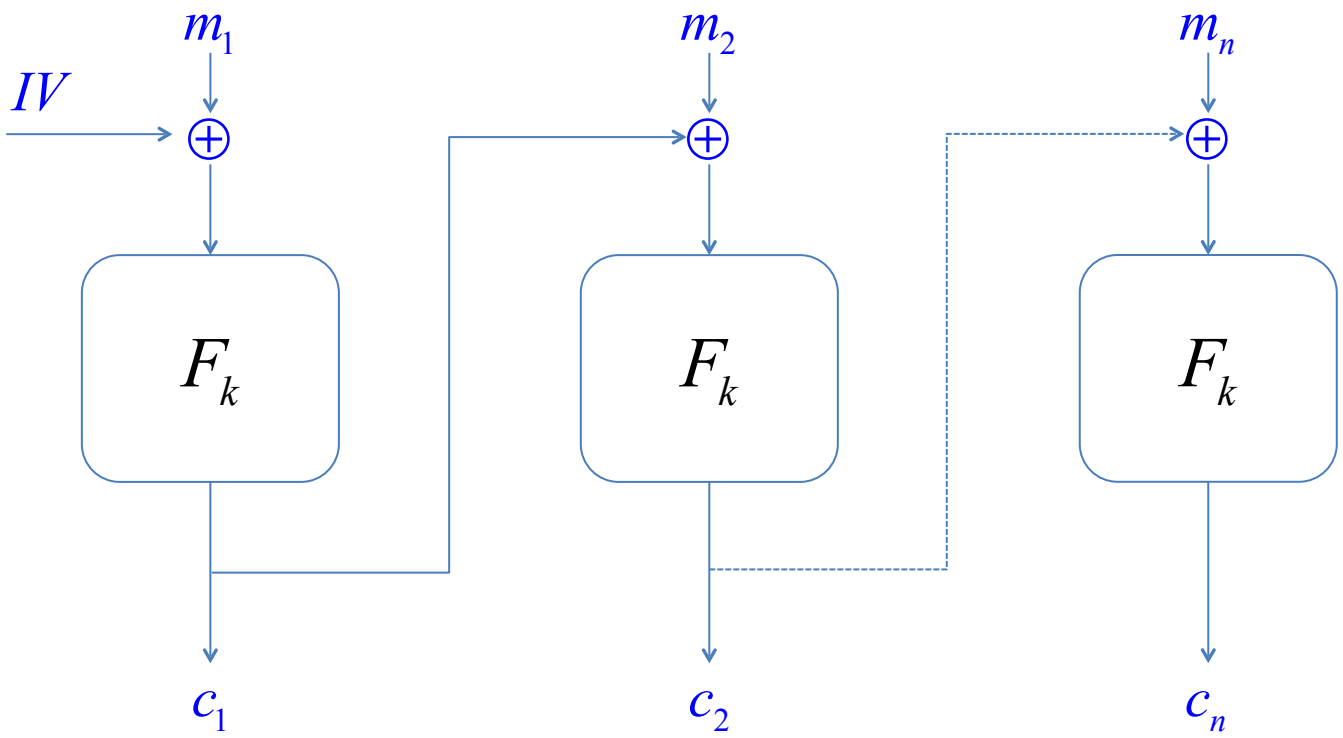
攻击者通过查询获得一些消息的消息认证码，不能构造出没有查询过的消息的消息认证码。



$$\Pr\left[\text{Mac-forge}_{\mathcal{A},\Pi}(n)=1\right] \le negl(n)$$

CBC-MAC

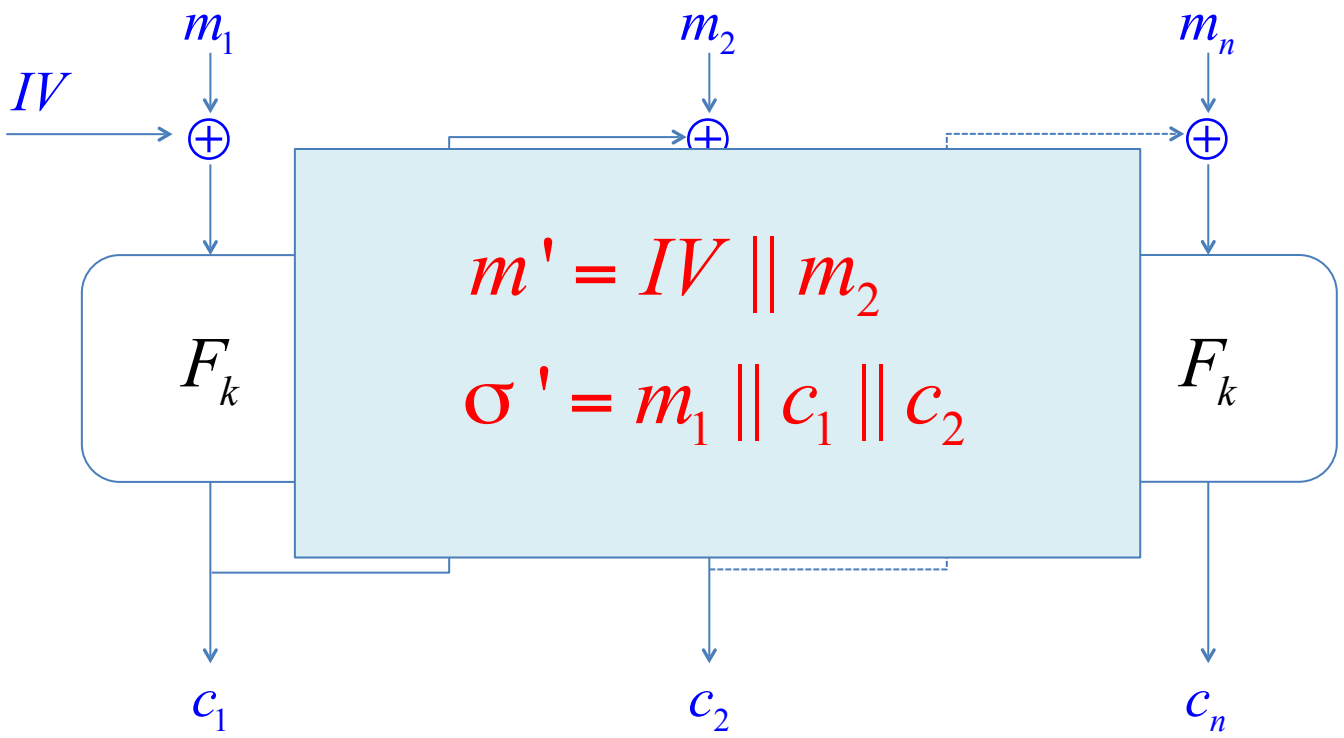
分组加密算法



$$\sigma = IV \| c_1 \| c_2 \| \cdots \| c_n$$

CBC-MAC

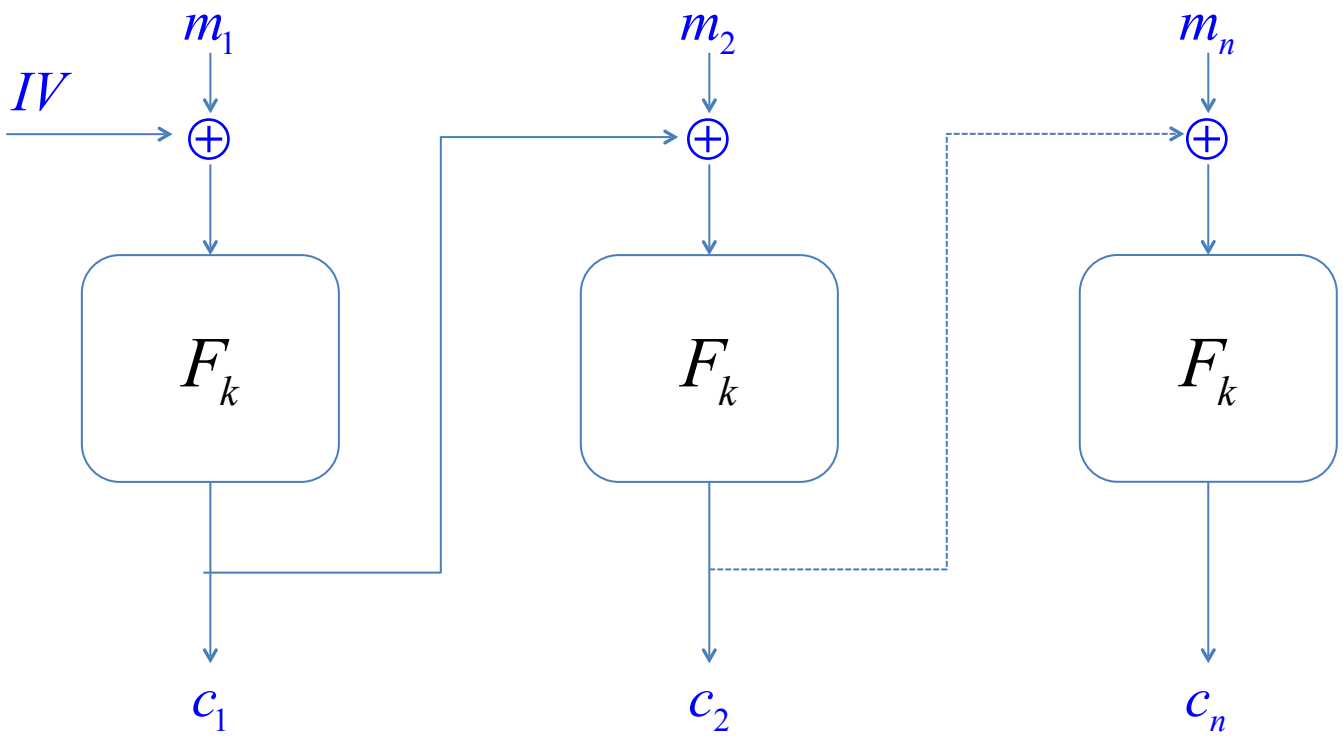
分组加密算法



$$\sigma = IV \| c_1 \| c_2 \| \cdots \| c_n$$

CBC-MAC

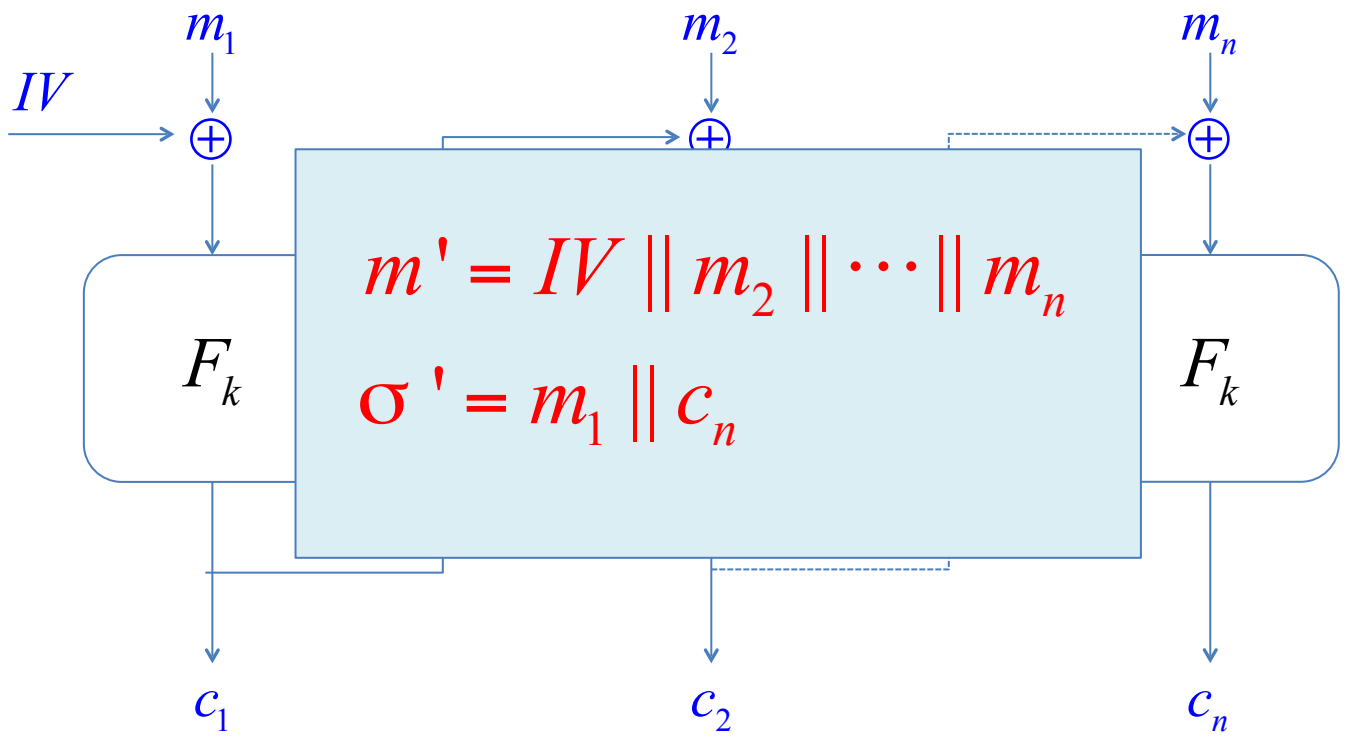
分组加密算法



$$\sigma = IV \,||\, c_n$$

CBC-MAC

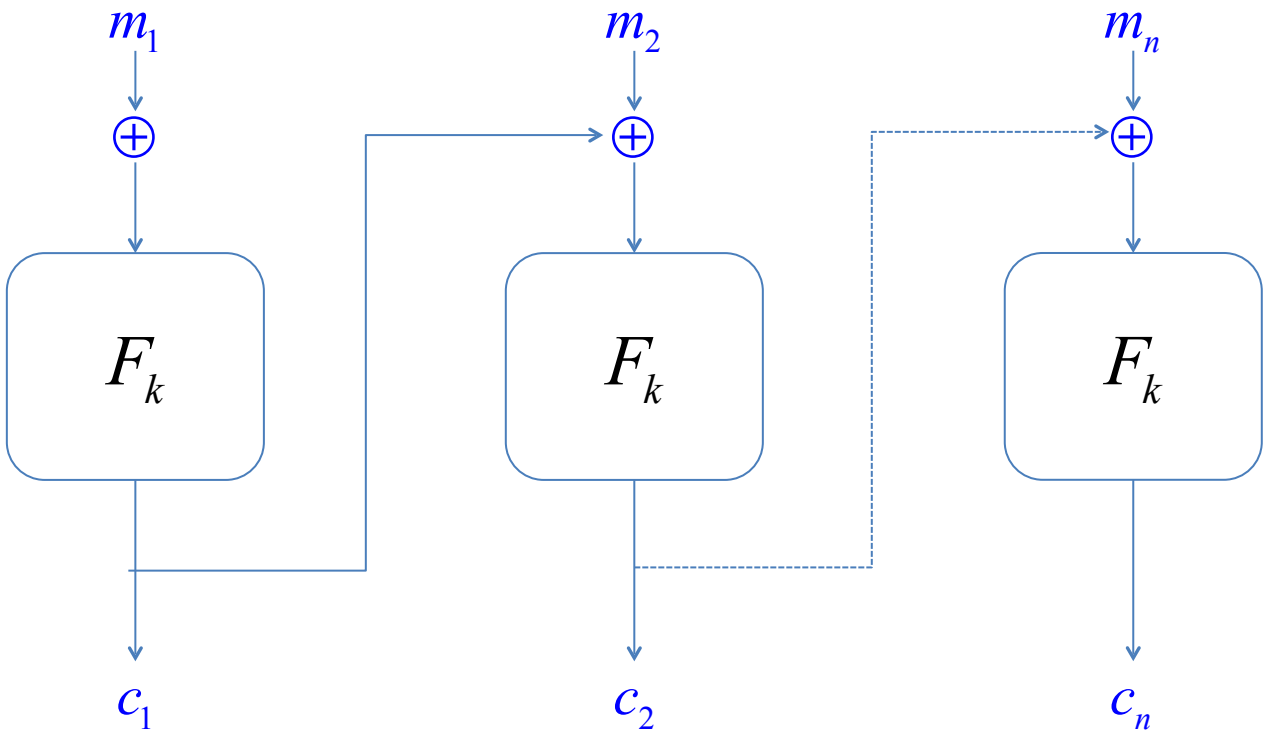
分组加密算法



$$\sigma = IV \| c_n$$

CBC-MAC

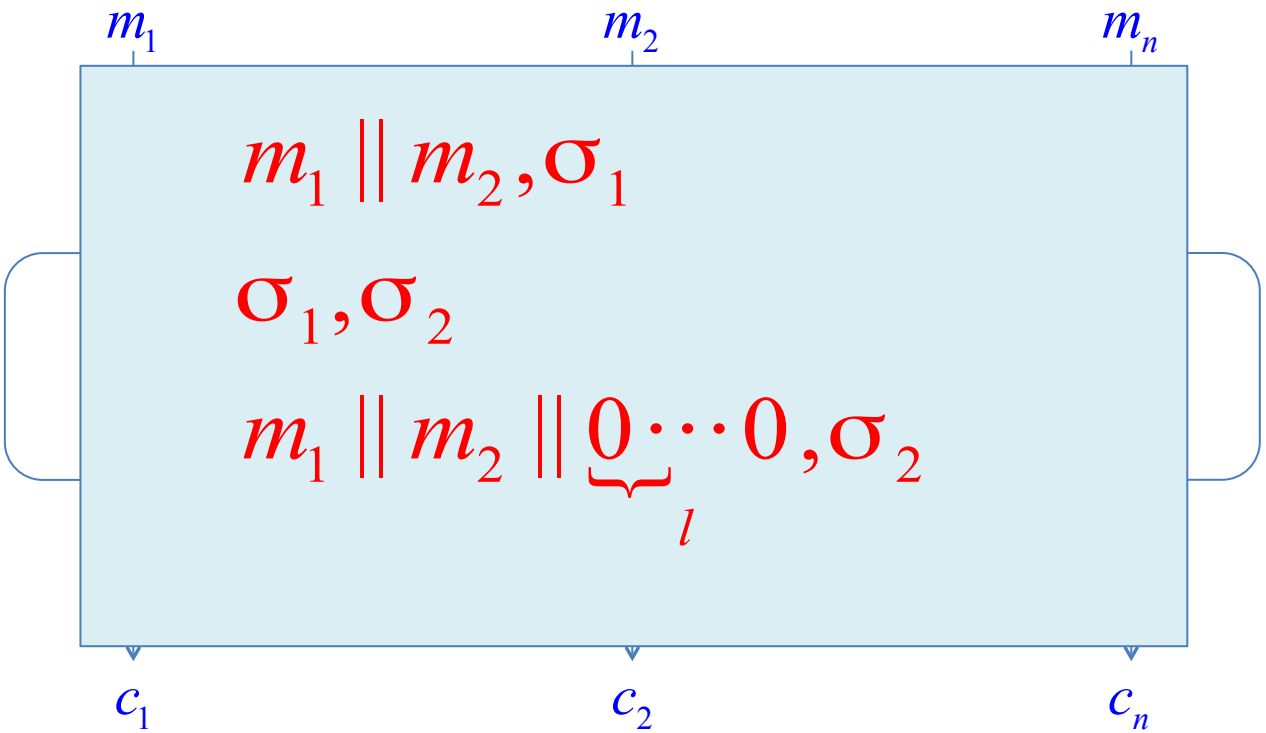
分组加密算法



$$\sigma = c_n$$

CBC-MAC

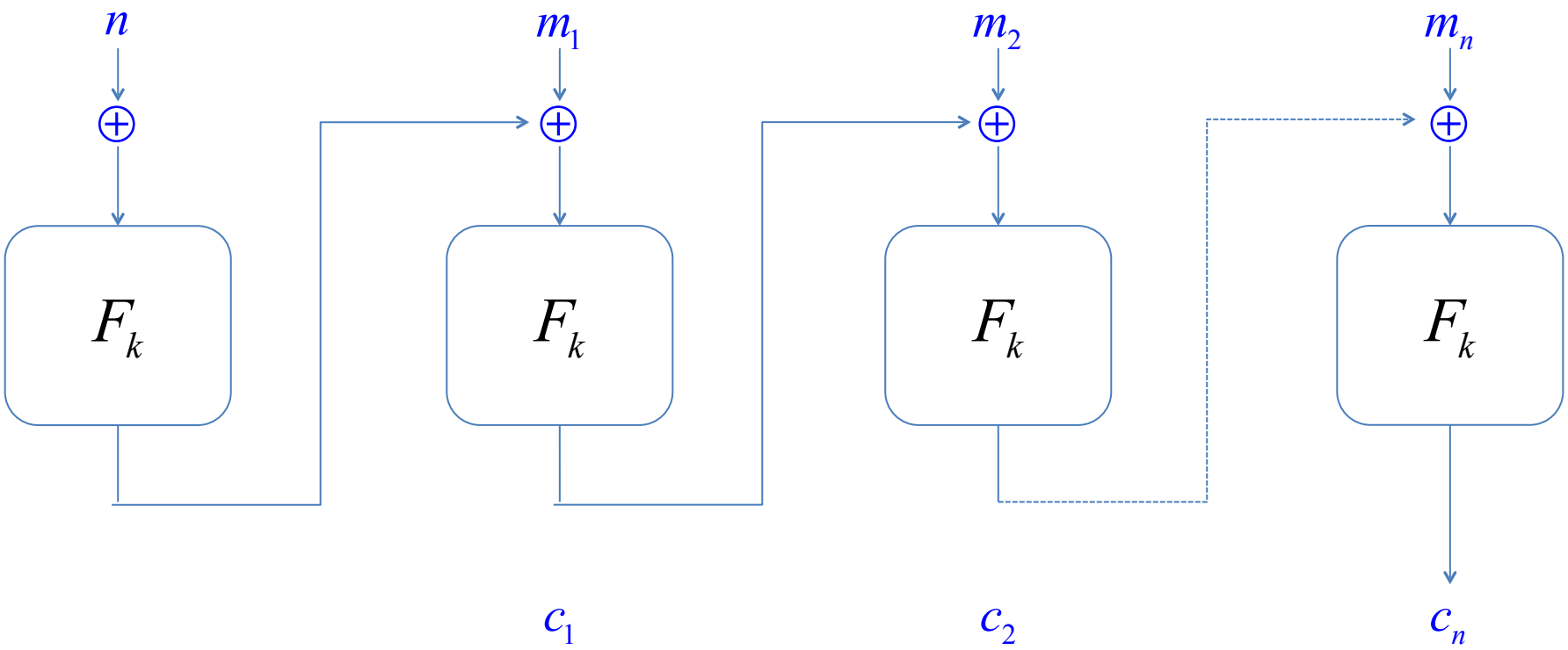
分组加密算法



$$\sigma = c_n$$

CBC-MAC

分组加密算法



$$\sigma = c_n$$

CBC-MAC

- CBC-MAC与CBC 加密模式的区别?
- CBC-MAC中消息长度n放到结尾是否安全?

散列函数（HASH函数）

单向函数（One-way Function）

- 已知x，计算y=f(x)容易。
- 已知y，计算x使得y=f(x)困难。

散列函数（Cryptographic HASH Function）

- 已知x，计算y=f(x)容易。
- 已知y，计算x使得y=f(x)困难。
- y是定长的。

散列函数（HASH函数）

HASH函数

- 消息M可以是任意长度的数据。
- HASH函数产生定长的输出。
- 给定消息M，计算它的Hash值h=H(M) 是很容易的。
- 任意给定h，则很难找到M使得h=H(M)，即给出Hash值，要求输入M在计算上是不可行的，即运算过程是不可逆的，这种性质称为函数的单向性。
- 给定消息M和其Hash值H(M)，要找到另一个M’，且M≠M’，使得H(M)=H(M’)在计算上是不可行的，这条性质被称为抗弱碰撞性。
- 对于任意两个不同的消息M≠M’，它们的摘要值不可能相同，这条性质被称为抗强碰撞性。

散列函数（HASH函数）

HASH函数

- 弱抗碰撞性保证对于一个消息M及其Hash值，无法找到一个替代消息M’，使它的Hash值与给定的Hash值相同。这条性质可用于防止伪造。
- 强抗碰撞性对于消息Hash函数的安全性要求更高，这条性质保证了对生日攻击的防御能力。

散列函数（HASH函数）

“生日攻击”问题

多少个人的生日会出现重合的概率会大于0.5?

- $Q(365,k)$表示 k 个人中没有出现相同生日的概率。
- $P(365,k)$ 表示 k 个人中出现相同生日的概率。

$$Q(365,k)=\frac{365\times(365-1)\times\cdots\times(365-k+1)}{365^k}=\frac{365!}{(365-k)!365^k}$$

$$P(365,k)=1-Q(365,k)=1-\frac{365!}{(365-k)!365^k}$$

- 如何找到k的值?

散列函数（HASH函数）

“生日攻击”问题

q个元素 $y_1, y_2, \ldots, y_q$ 在 $[1, N]$ 内随机取值，则出现不同 i, j 满足 $y_i = y_j$ 的概率 $coll(q, N)$：

$$coll(q,N) = \Pr[\vee_{i \neq j} coll_{i,j}]$$

$$= 1 - \frac{N(N-1)\ldots(N-q+1)}{N^q} = 1 - \prod_{i=1}^{q-1}(1 - \frac{i}{N})$$

散列函数（HASH函数）

$$coll(q,N) = \Pr[\vee_{i\neq j} coll_{i,j}]$$

$$= 1 - \frac{N(N-1)...(N-q+1)}{N^q} = 1 - \prod_{i=1}^{q-1}(1-\frac{i}{N})$$

- $n = 30$
  - $coll(q,N)$
  - $= 1 - \frac{365}{365} \times \frac{364}{365} \times \ldots \times \frac{365-30+1}{365}$
  - $= 1 - 1 \times (1 - \frac{1}{365}) \times (1 - \frac{2}{365}) \ldots \times (1 - \frac{30-1}{365})$
  - $\approx 60\% - 70\%$

散列函数（HASH函数）

令 $\Pr[\vee_{i\neq j} coll_{i,j}]=\varepsilon$

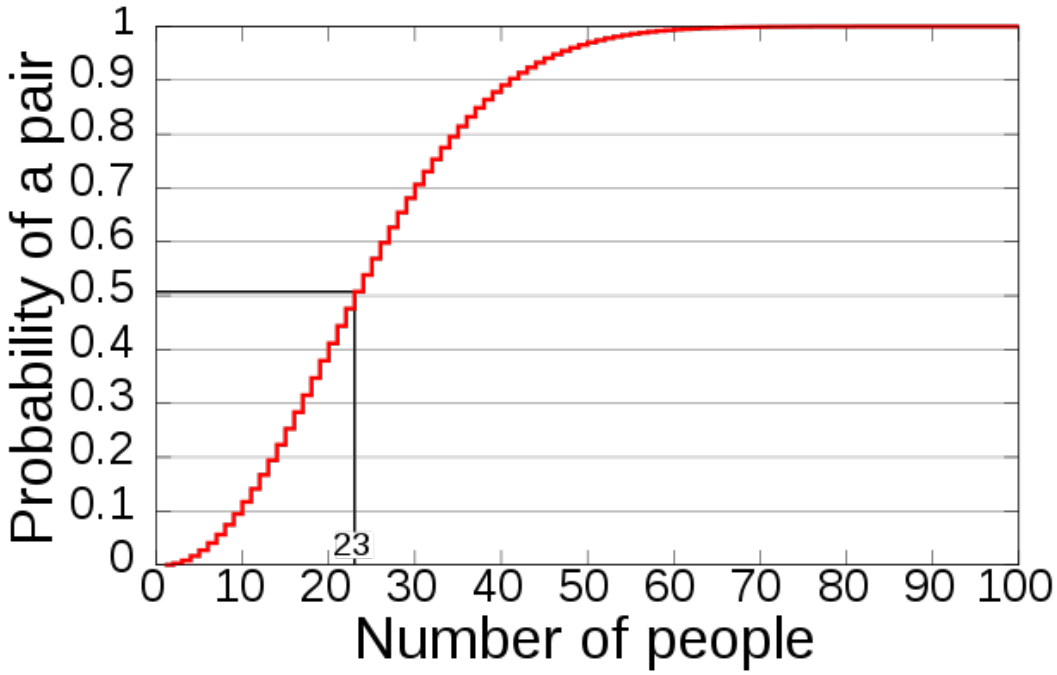
$$coll(q,N)\approx 1-e^{-\frac{q(q-1)}{2N}}=\varepsilon$$

$$e^{-\frac{q(q-1)}{2N}}\approx 1-\varepsilon$$

$$q^2-q\approx q^2\approx 2N\ln(1-\varepsilon)^{-1}$$

$$q\approx\sqrt{2N\ln(1-\varepsilon)^{-1}}$$

$N=365,\varepsilon=0.5,q\approx 23$



- https://en.wikipedia.org/wiki/File:Birthday_Paradox.svg

- $H(x)$ 如果具有 N 位输出, 攻击者计算 $2^{N/2}$ 个哈希值，出现碰撞的概率是多少?

- 当N足够大时，出现碰撞的概率如下:

$$1 - \frac{2^N}{2^N} \times \frac{2^N - 1}{2^N} \times \frac{2^N - 2}{2^N} \times \ldots \times \frac{2^N - 2^{N/2} + 1}{2^N}$$
$$= 1 - 1 \times (1 - \frac{1}{2^N}) \times (1 - \frac{2}{2^N}) \times \ldots \times (1 - \frac{2^{N/2} - 1}{2^N})$$

- 由于$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \ldots \approx 1 - x,$ 当 x 趋近于0.

- $$\approx 1 - e^{-\frac{1}{2^N}} \times e^{-\frac{2}{2^N}} \times \ldots \times e^{-\frac{2^{N/2}-1}{2^N}} = 1 - \prod_{i=1}^{2^{N/2}-1} e^{-\frac{i}{2^N}}$$
$$= 1 - \prod e^{-\sum_{i=1}^{2^{N/2}-1} \frac{i}{2^N}} = 1 - e^{-\frac{2^{N/2} \times (2^{N/2} - 1)}{2^N}} \approx 1 - e^{-1} \approx 63.21\,\%$$

散列函数（HASH函数）

- $H(x)$ 如果具有 N 位输出, C 表示一个常数，则：
- 攻击者每计算 $2^{N/2}$ 次消息, 无碰撞的概率为 $\Pr[lose] = 1 - (1 - e^{-1}) = e^{-1}$
- 攻击者发现碰撞的概率为 $\Pr[Win] = 1 - (e^{-1})^C \approx 1$, 当 C 足够大时。
- 对于 256 位输出, 攻击者计算 2^{128} 次哈希，假设每次计算哈希用时 10^{-6} 秒，
- 则找到碰撞需要 $2^{128} \times 10^{-6}/(365 \times 24 \times 3600) \approx 1.079 * 10^{31}$ 年.

散列函数（HASH函数）

It was the best of times,
it was the worst of times,

it was the age of wisdom,
it was the age of foolishness,

it was the epoch of belief,
it was the epoch of incredulity,

it was the season of light,
it was the season of darkness,

it was the spring of hope,
it was the winter of despair,

we had everything before us,
we had nothing before us,
……
……

There were a king with a large jaw
and a queen with a plain face,
on the throne of England;
There were a king with a large jaw
and a queen with a fair face,
on the throne of France.

散列函数（HASH函数）

压缩函数：定长输入，定长输出。

$$compress : \{0,1\}^{m+l} \rightarrow \{0,1\}^{m}$$

散列函数（HASH函数）

预处理

给定一个比特串 x，其中 $|x| \geq m+l+1$，用公开算法构造比特串 y，使得 $|y| = 0 (\mathrm{mod}\, l)$，记为

$$y = y_1 || y_2 || \ldots || y_r, |y| = rl$$

处理

设IV是长度为 rl 的初始比特串，则计算

$$z_0 \leftarrow IV$$

$$z_1 \leftarrow compress(z_0 \| y_1)$$

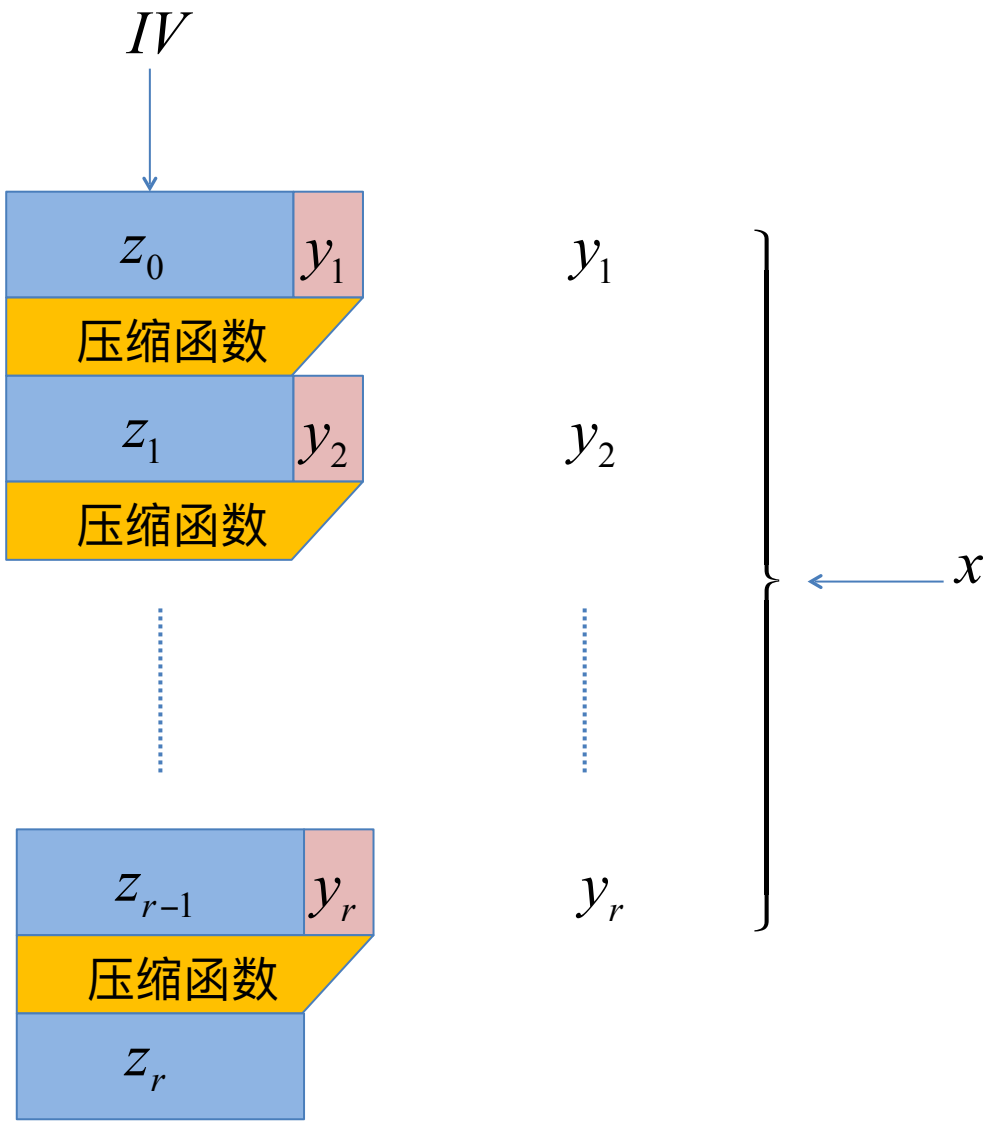
$$z_2 \leftarrow compress(z_1 \| y_2)$$

$$\ldots$$

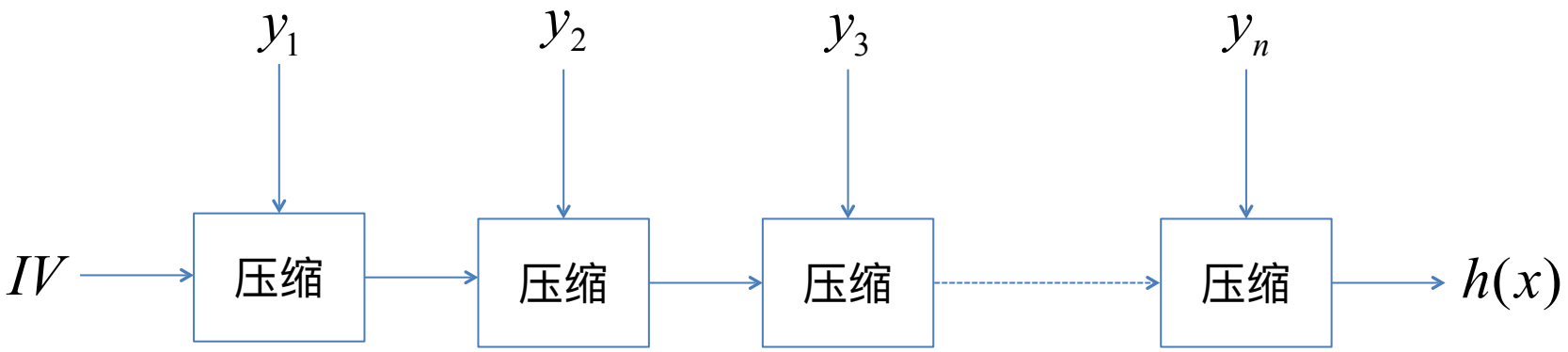
$$\ldots$$

$$z_r \leftarrow compress(z_{r-1} \| y_r)$$

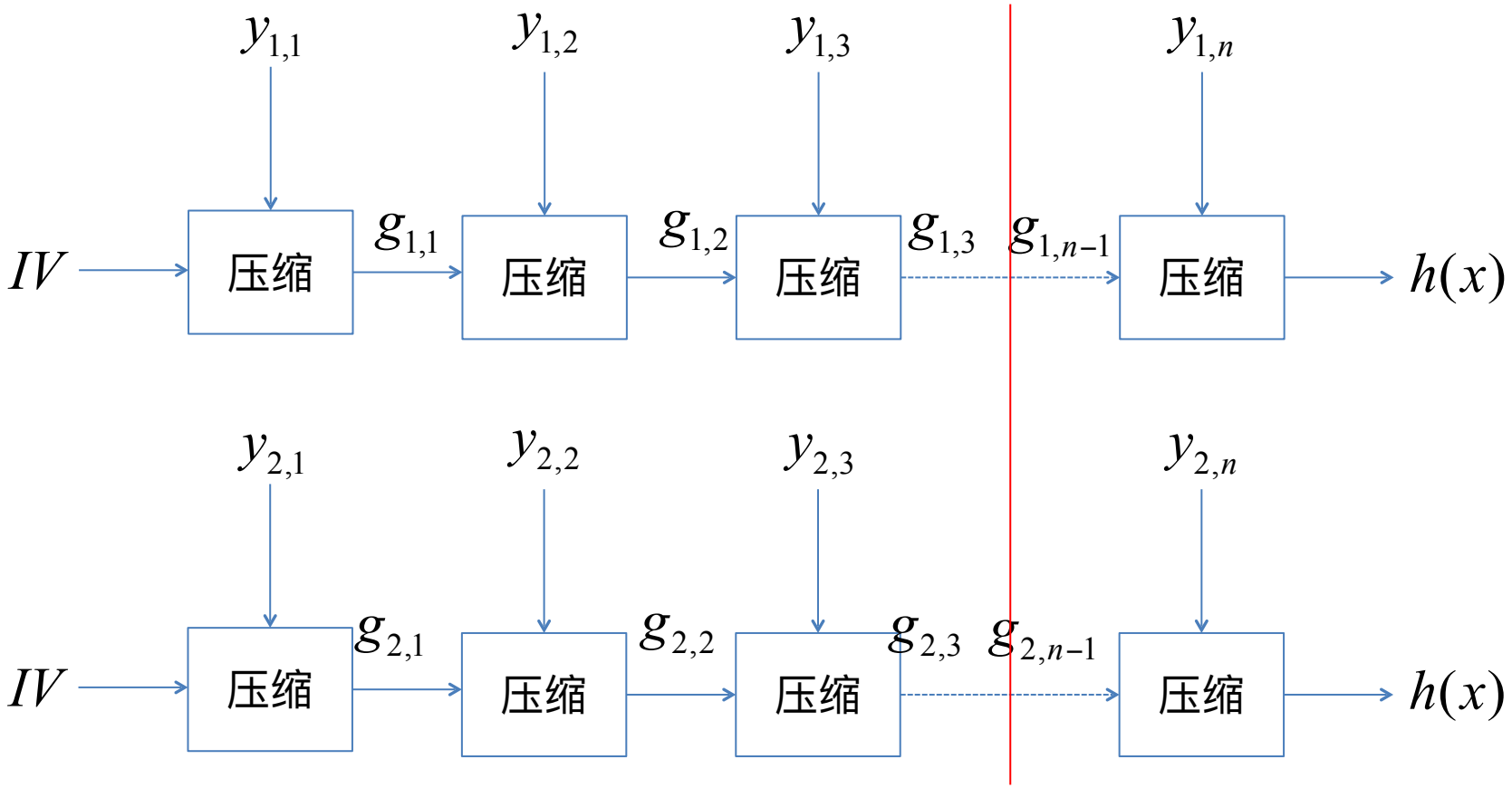
散列函数（HASH函数）



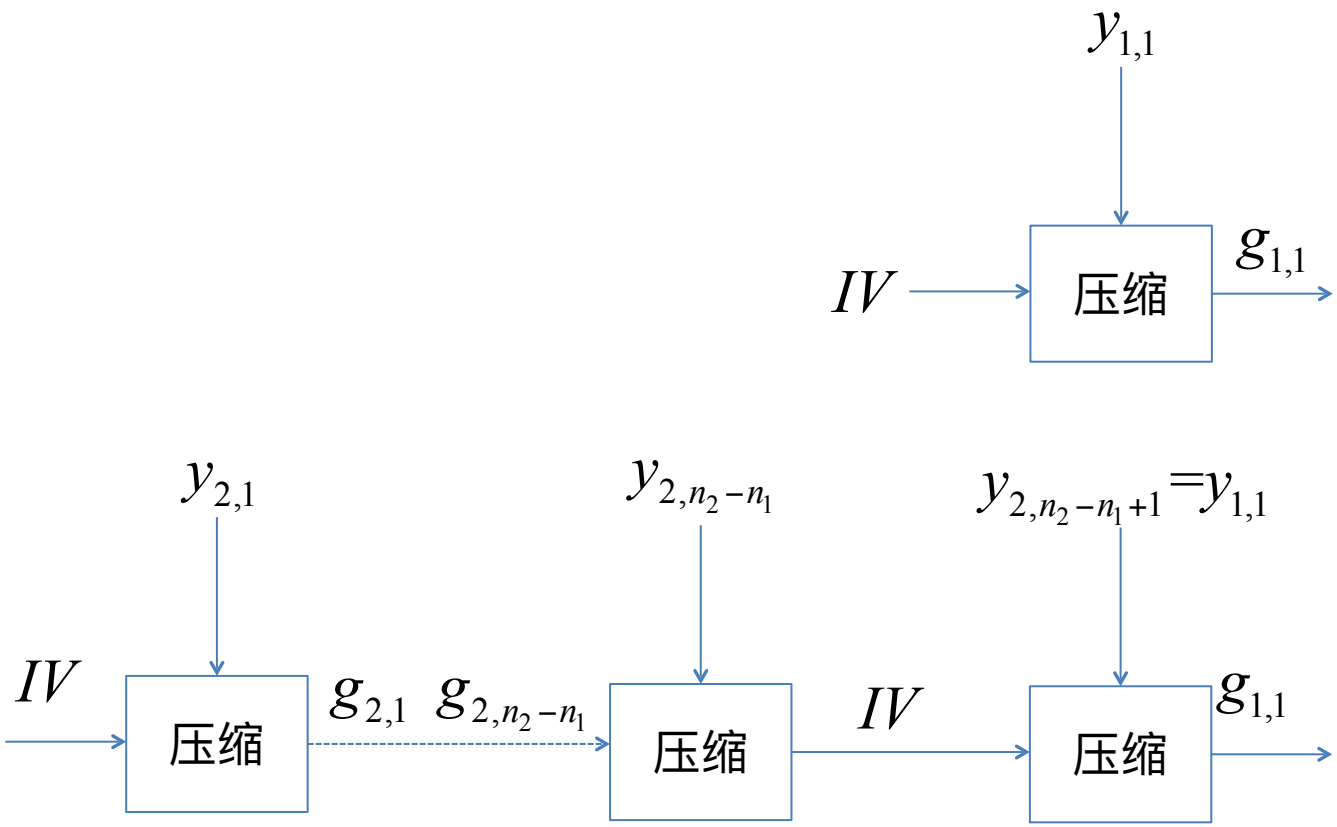
Merkle-Damgard结构



Merkle-Damgard结构



Merkle-Damgard结构



HASH算法的安全性

对于长度为n的理想HASH算法，找到原像、弱碰撞和强碰撞需要穷举的消息数分别为：

单　向　性	2^n
抗弱碰撞性	2^n
抗强碰撞性	$2^{n/2}$

但实际上任何具体HASH算法实现都不是理想的HASH算法，密码学家希望穷举尽可能少的消息数攻破HASH算法。

练习

问题：

假设HASH算法是抗强碰撞的，如何使用HASH算法构造抵抗选择消息攻击的MAC算法?

一种做法：$Mac(k,x) = Hash(k \,||\, x)$?

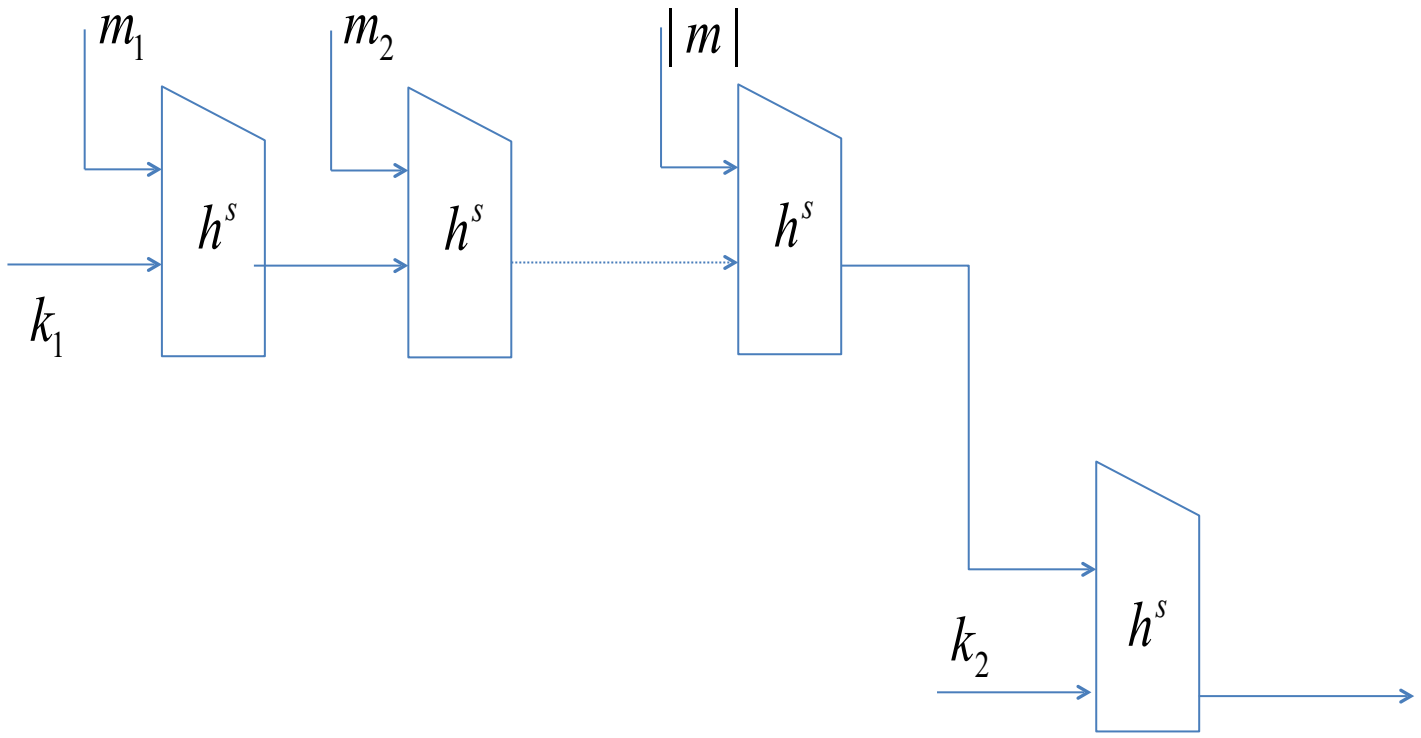
目前大多数HASH算法迭代运行压缩函数实现的，如果采用迭代结构的HASH算法则上面的MAC构造方案是不安全的。

NMAC

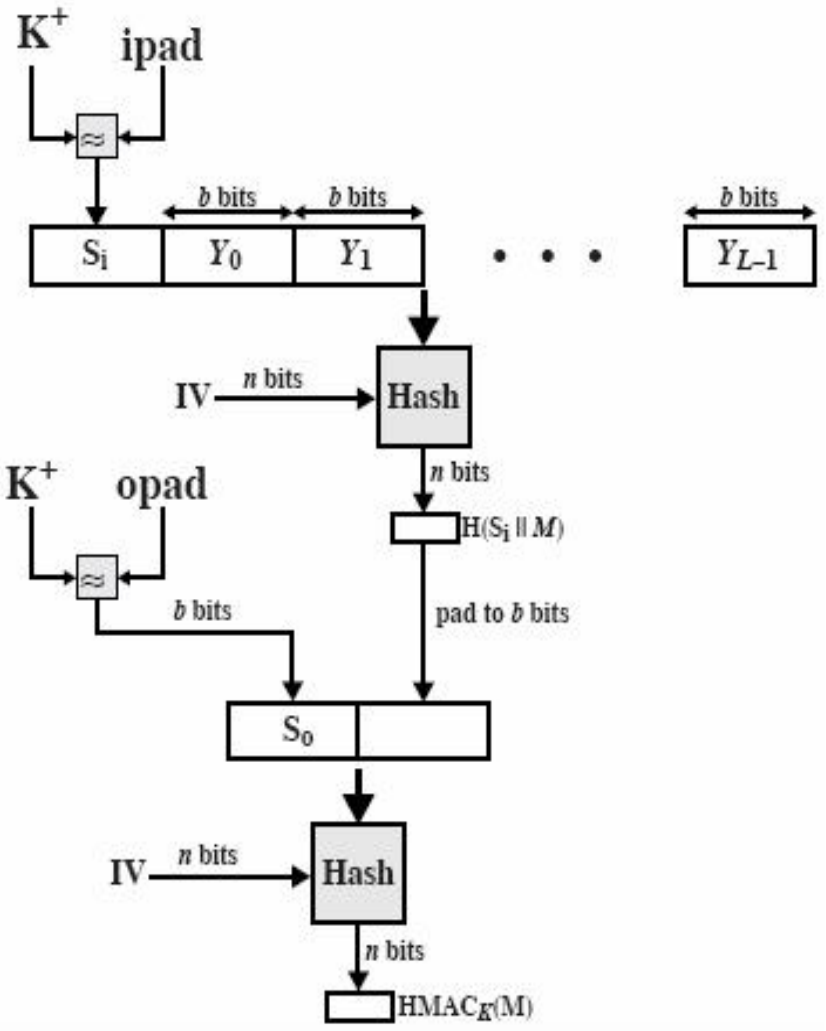
NMAC设计目标

- 无需修改地使用现有的散列函数。特别是，散列函数的软件实现执行很快，且程序代码是公开的和容易获得的。
- 当出现或获得更快的或更安全的散列函数时，对算法中嵌入的散列函数要能轻易地进行替换。
- 保持散列函数的原有性能不会导致算法性能的降低。
- 使用和处理密钥的方式很简单。
- 基于对嵌入散列函数合理的假设，对鉴别机制的强度有一个易懂的密码编码分析。

NMAC

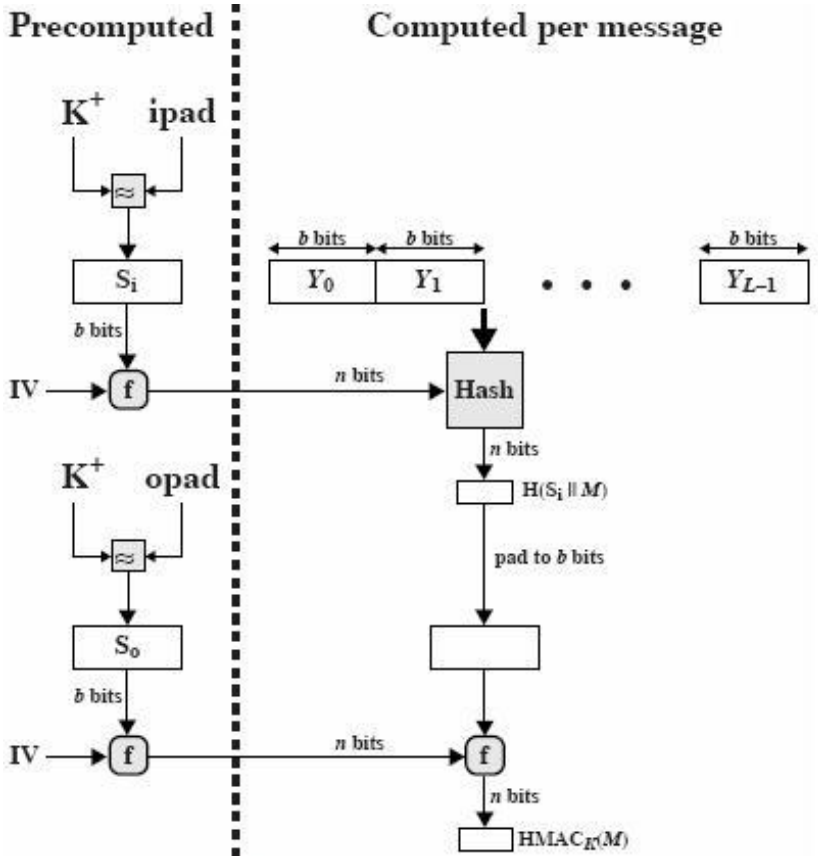


HMAC



$$\text{HMAC}(k, x) =$$

$$\text{H}(k \oplus \text{opad}) || \text{H}(k \oplus \text{ipad} \,||\, x))$$



Mac和Hash的关系

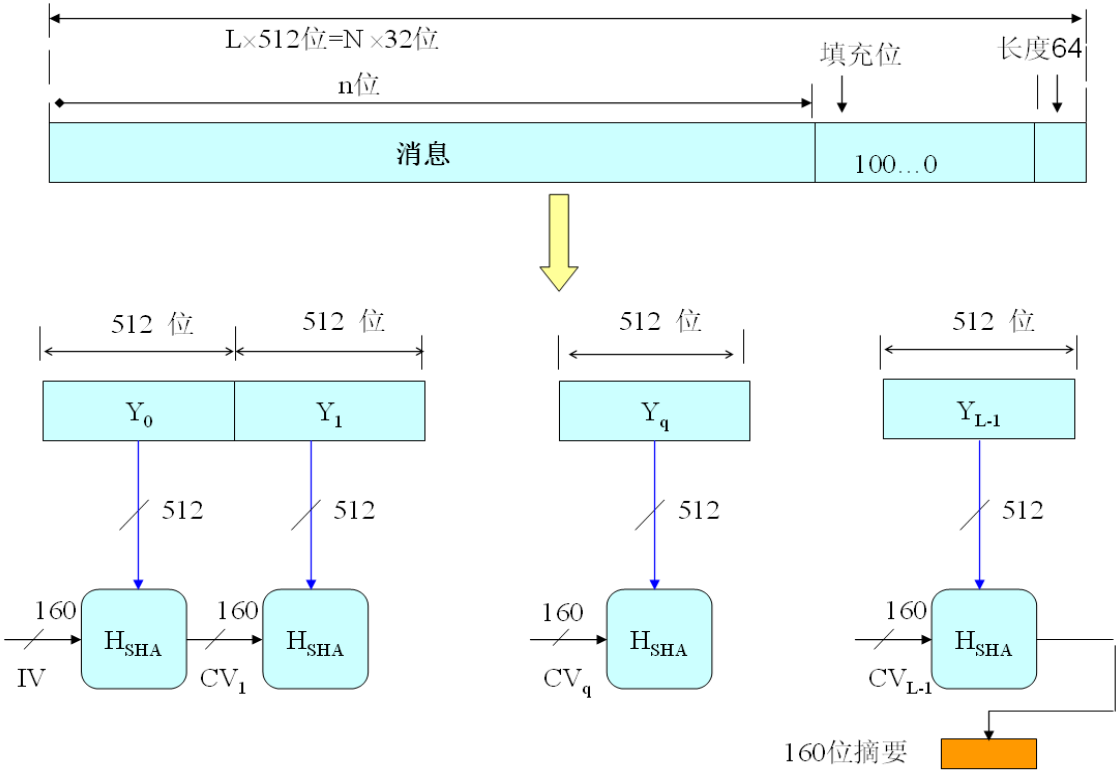
Mac可以有多种方式构造，其中基于Hash算法的Mac构造方法是一种主要方法。

Hash算法不仅可以用于Mac算法的构造，还可以用于设计加密与数字签名等其他密码学算法。

选择密文攻击安全的加密方案?

安全HASH算法（SHA)

- SHA（Secure Hash Algorithm，SHA）是由美国NIST开发的哈希函数，作为联邦信息处理标准于1993年发表，1995年修订成为SHA1，其流程大致如下图所示。SHA每个版本会增加IV向量的大小并修改数值，如SHA2中，IV可以是224位或256位。输出的位数与IV向量位数相同。
- 哈希算法在理论上，只要发现碰撞，即不在被认为是安全的。

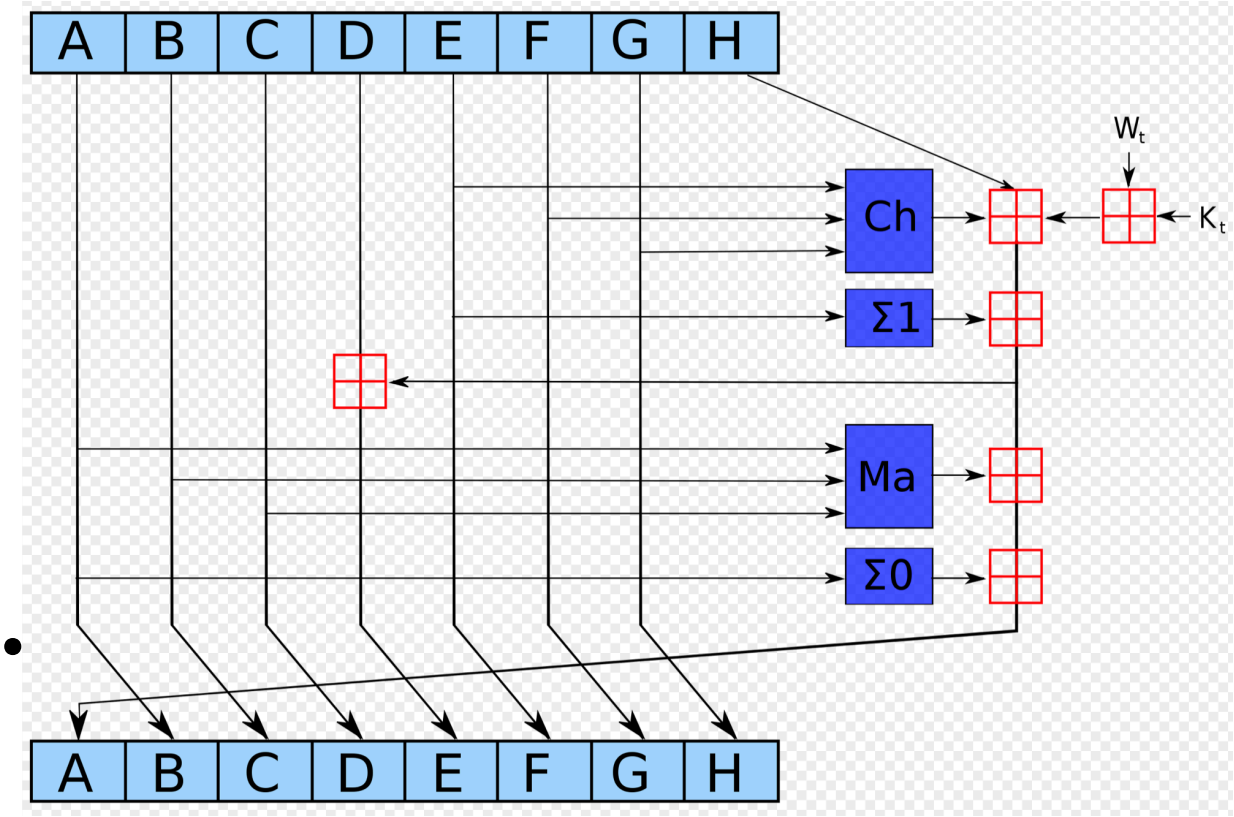


安全HASH算法（SHA)

- SHA256 https://en.wikipedia.org/wiki/SHA-2
  - The IV (Initial Vector) consists of eight 32-bit binary numbers.
  - A := 0x6a09e667 B := 0xbb67ae85
  - C := 0x3c6ef372 D := 0xa54ff53a
  - E := 0x510e527f F := 0x9b05688c
  - G := 0x1f83d9ab H := 0x5be0cd19
- first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19

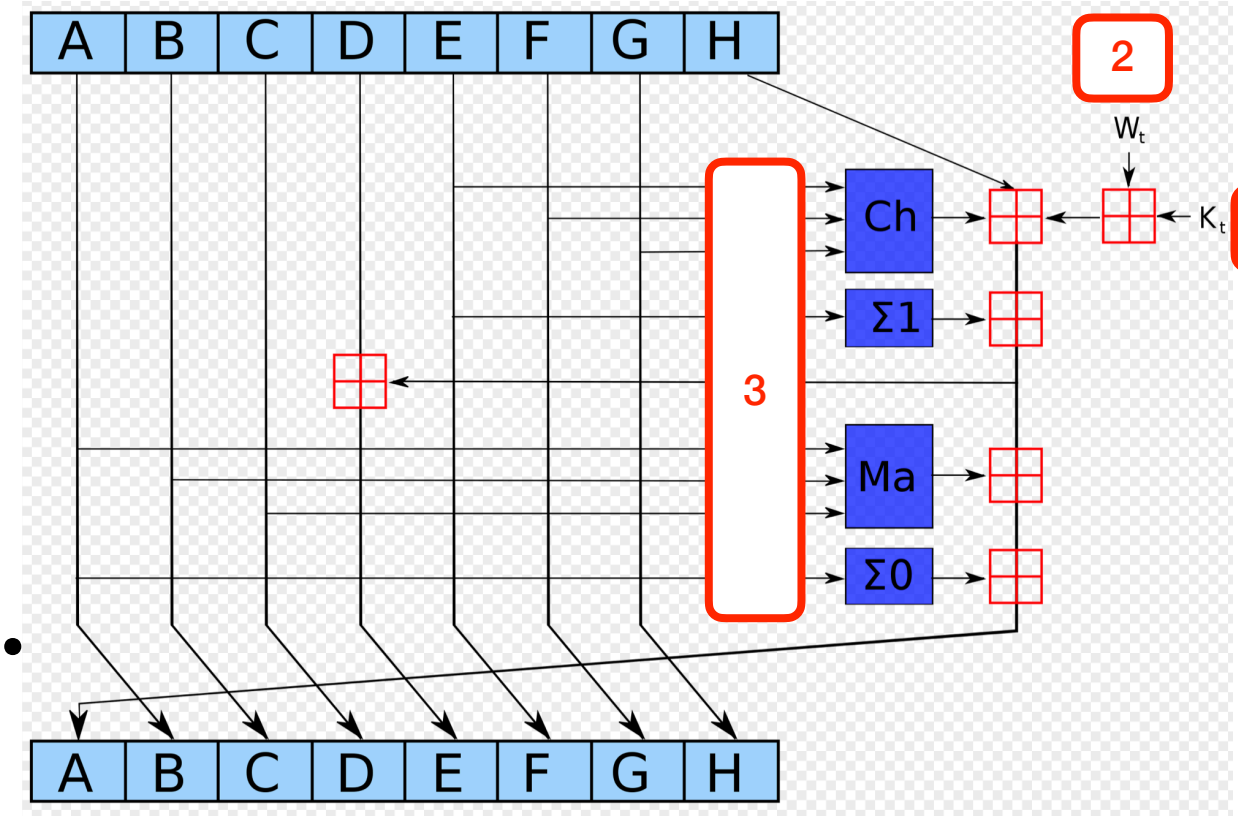
安全HASH算法（SHA)

- SHA256 https://en.wikipedia.org/wiki/SHA-2
  - The function f, where addition is calculated modulo 2^{32}.
  -



安全HASH算法（SHA)

- SHA256 https://en.wikipedia.org/wiki/SHA-2
  - The function f, where addition is calculated modulo 2^{32}.
  -



安全HASH算法（SHA)

- SHA256 https://en.wikipedia.org/wiki/SHA-2
  - K[0..63] :=
  - 0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
  - 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
  - 0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
  - 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
  - 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
  - 0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
  - 0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
  - 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
- first 32 bits of the fractional parts of the cube roots of the first 64 primes 2..311

安全HASH算法（SHA)

- SHA256 https://en.wikipedia.org/wiki/SHA-2
  - for each 512-bit trunk of the padding message {
    - Create a w[0..63] of 32-bit words. (Each word has 16 bits, so 16 * 32 = 512 bits)
    - Copy chunk into first 16 words w[0..15]
    - for i from 16 to 63 {
      - s0 := (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor (w[i-15] rightshift 3)
      - s1 := (w[i- 2] rightrotate 17) xor (w[i- 2] rightrotate 19) xor (w[i- 2] rightshift 10)
      - w[i] := w[i-16] + s0 + w[i-7] + s1}

安全HASH算法（SHA)

- SHA256 https://en.wikipedia.org/wiki/SHA-2
  - for i from 0 to 63{
    - Sigma1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
    - Ch := (e and f) xor ((not e) and g)
    - tmp1 := h + Sigma1 + Ch + k[i] + w[i]
    - Sigma0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
    - Ma := (a and b) xor (a and c) xor (b and c)
    - tmp2 := Sigma0 + Ma}
  - h := g; g := f; f := e; e := d + tmp1; d := c; c := b; b := a; a := tmp1 + tmp2

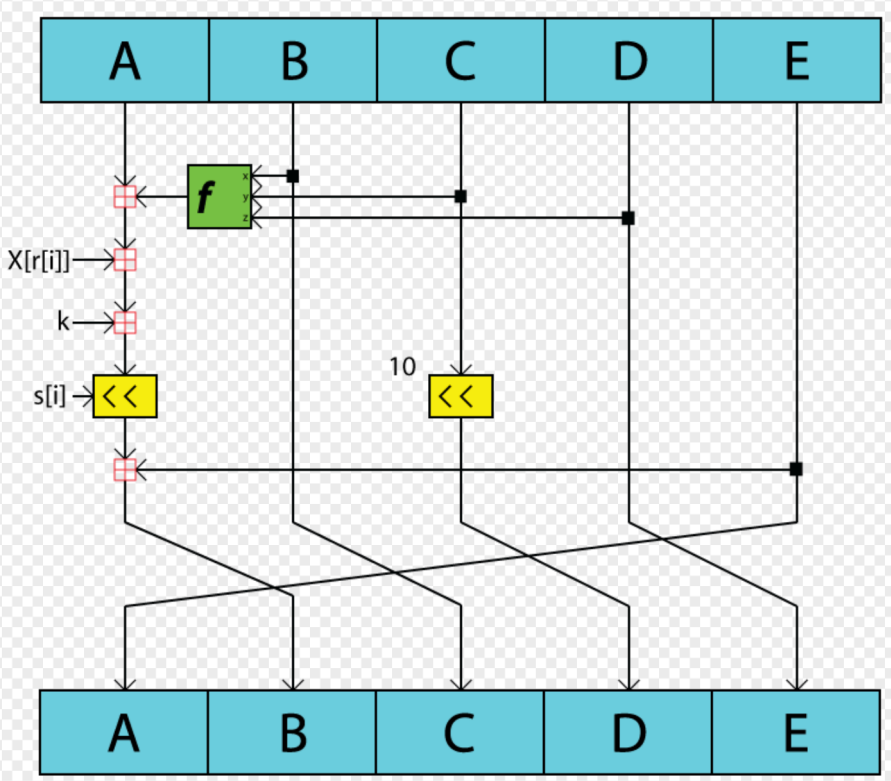
安全HASH算法（SHA)

https://en.wikipedia.org/wiki/SHA-2

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Rounds	Operations	Security (in bits) against collision attacks
MD5 (as reference)		128	128 (4 × 32)	512	64	And, Xor, Rot, Add (mod 2^{32}), Or	≤18 (collisions found)[38]
SHA-0		160	160 (5 × 32)	512	80	And, Xor, Rot, Add (mod 2^{32}), Or	<34 (collisions found)
SHA-1							<63 (collisions found)[39]
SHA-2	*SHA-224*	224	256 (8 × 32)	512	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112
	SHA-256	256					128
	SHA-384	384	512 (8 × 64)	1024	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192
	SHA-512	512					256
	SHA-512/224	224					112
	SHA-512/256	256					128
SHA-3	*SHA3-224*	224	1600 (5 × 5 × 64)	1152	24[40]	And, Xor, Rot, Not	112
	SHA3-256	256		1088			128
	SHA3-384	384		832			192
	SHA3-512	512		576			256
	SHAKE128	*d* (arbitrary)		1344			min(*d*/2, 128)
	SHAKE256	*d* (arbitrary)		1088			min(*d*/2, 256)

安全HASH算法（SHA)

- RIPEMD160
  - Pseudocode: https://rosettacode.org/wiki/RIPEMD-160



THANKS