Vietnam National University Ho Chi Minh City, University of Science
Department of Information Technology

Topic 9:

Hash function & MAC

Assoc. Prof. Trần Minh Triết
PhD. Trương Toàn Thịnh

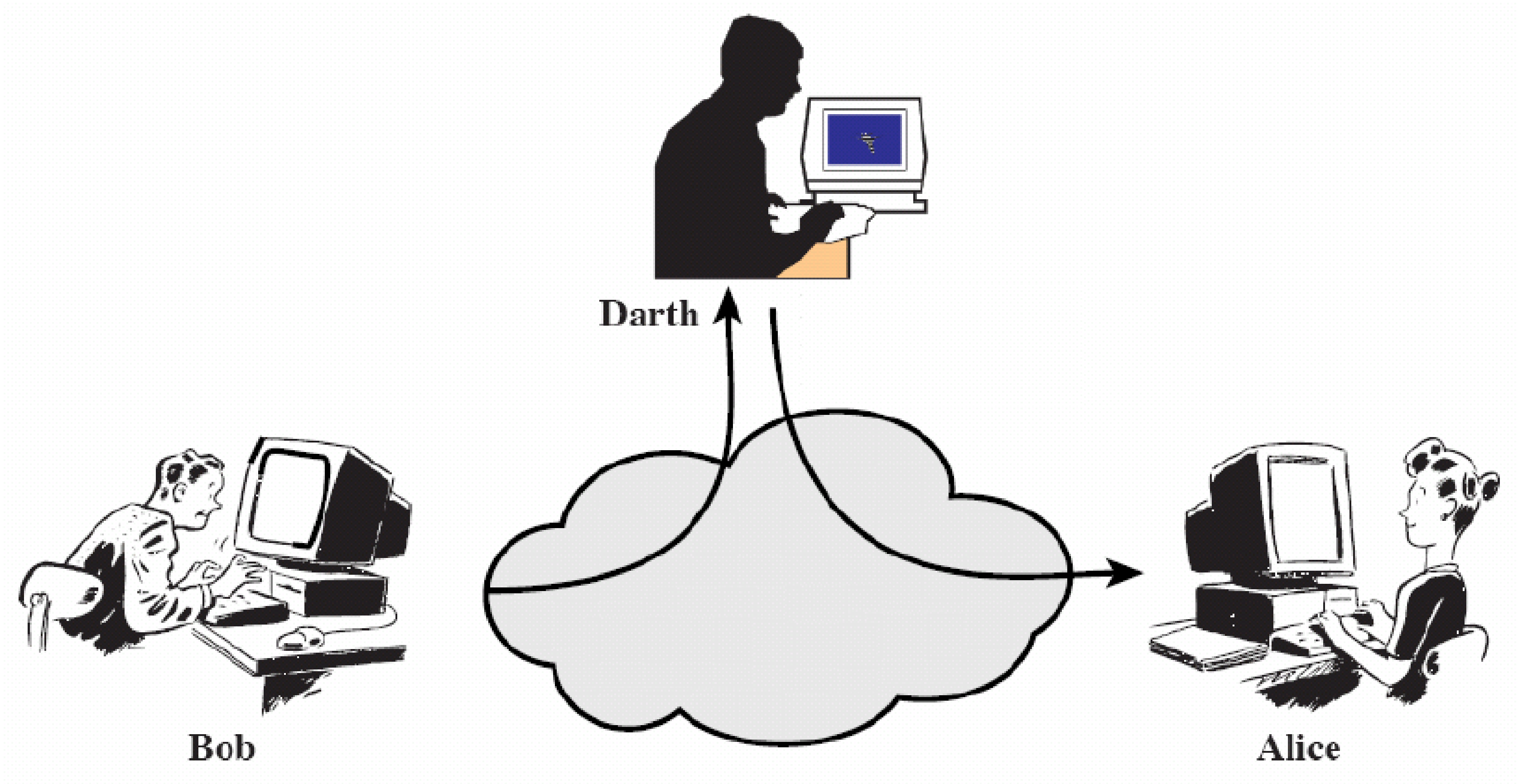
fit@hcmus

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Contents

- Introduction
- Properties of hash function
- Classification of cryptographic hash function
- Some popular hash function architectures
- MD5 hash function
- SHA hash functions
- MAC and HMAC

Introduction

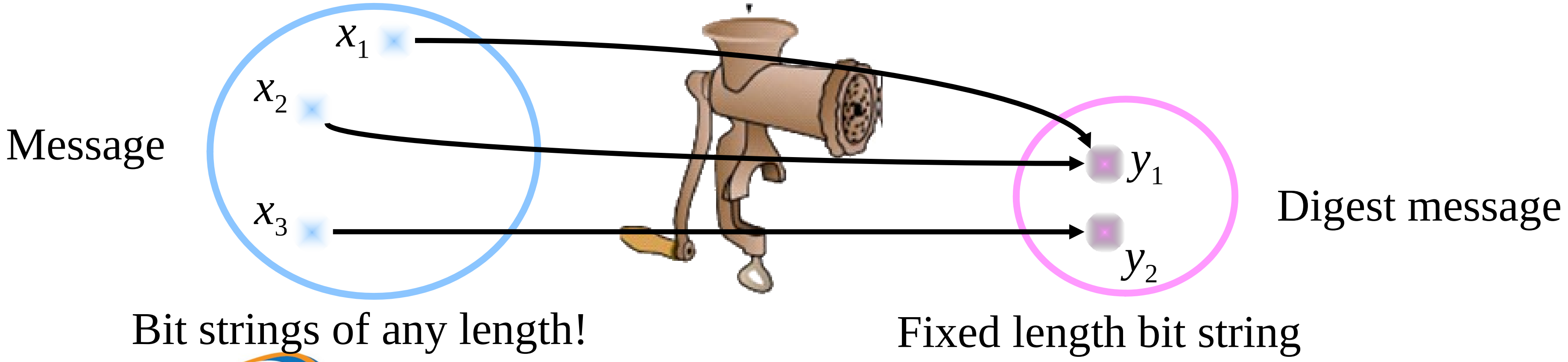


Integrity and secrecy

- Integrity: the attacker cannot intervene to edit the message content
- Encryption is only intended to ensure confidentiality, not to help ensure information integrity
- An attacker can modify the encrypted message without knowing the actual content of the message
- Example:
  - In an online auction, it is possible to change a competitor’s bid without knowing the actual content of the bid

Main ideas of hash function

- □ *H* is a lossy compression function
- □ Collision: $H(x) = H(x') \text{ for } x \neq x'$
- □ *H* can apply on data of almost any size
  - □ Result of *H* is a *n*-bit string (fixed *n*) “looks random”
  - □ Easy to compute $H(x)$ for any x
  - □ *H* is one-way function and secure against to “collision”



One-wayness

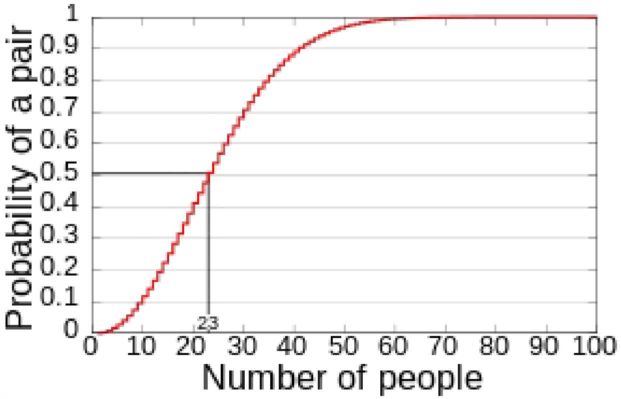
- The function H is difficult to reverse transform
  - Given random bit string $y \in \{0, 1\}^n$, hard to find bit string x such that $H(x) = y$
- Example: brute-force for each value x, check if $H(x) = y$ for SHA-1 producing a 160-bit string
  - Assume the hardware allows it to be done 2^{34} computations/s
  - Can perform 2^{59} computations/year
  - Need 2^{101} ($\sim 10^{30}$) years to reverse transform SHA-1 with given random value y

Safety against collisions

- Hard to find x and x' such that $H(x) = H(x')$
- Search collision by Brute-force just $O(2^{n/2})$, not $O(2^n)$
- Birthday paradox
  - We have t values of x_i and corresponding values $y_i = h(x_i)$, $1 \leq i \leq t$
  - For x_i, x_j, probability of collision is $1/2^n$
  - Total number of pairs $C_t^2 = t \times (t-1)/2 \sim O(t^2)$
  - If $t \approx 2^{n/2} \Rightarrow$ there are $\approx 2^n$ cặp (x_i, x_j)
  - For each pair, the probability of a collision is $1/2^n$, then probability of finding a pair of values that collide ≈ 1

Birthday Paradox

- Let $p(n)$ be the probability of finding 2 people with the same birthday in a group of n people?
- Let $\bar{p}(n)$ be the probability that any 2 people in a group of n people have different birthdays: $p(n) + \bar{p}(n) = 1$
- For $n \leq 365$ people, we have
  - $\bar{p}(n) = 1\left(1 - \frac{1}{365}\right)\left(1 - \frac{2}{365}\right)\ldots\left(1 - \frac{n-1}{365}\right) = \frac{365!}{365^n(365-n)!}$
  - $p(n) = 1 -$
  - $p(n) = 1 - \frac{365!}{365^n(365-n)!}$



https://en.wikipedia.org/wiki/Birthday_problem

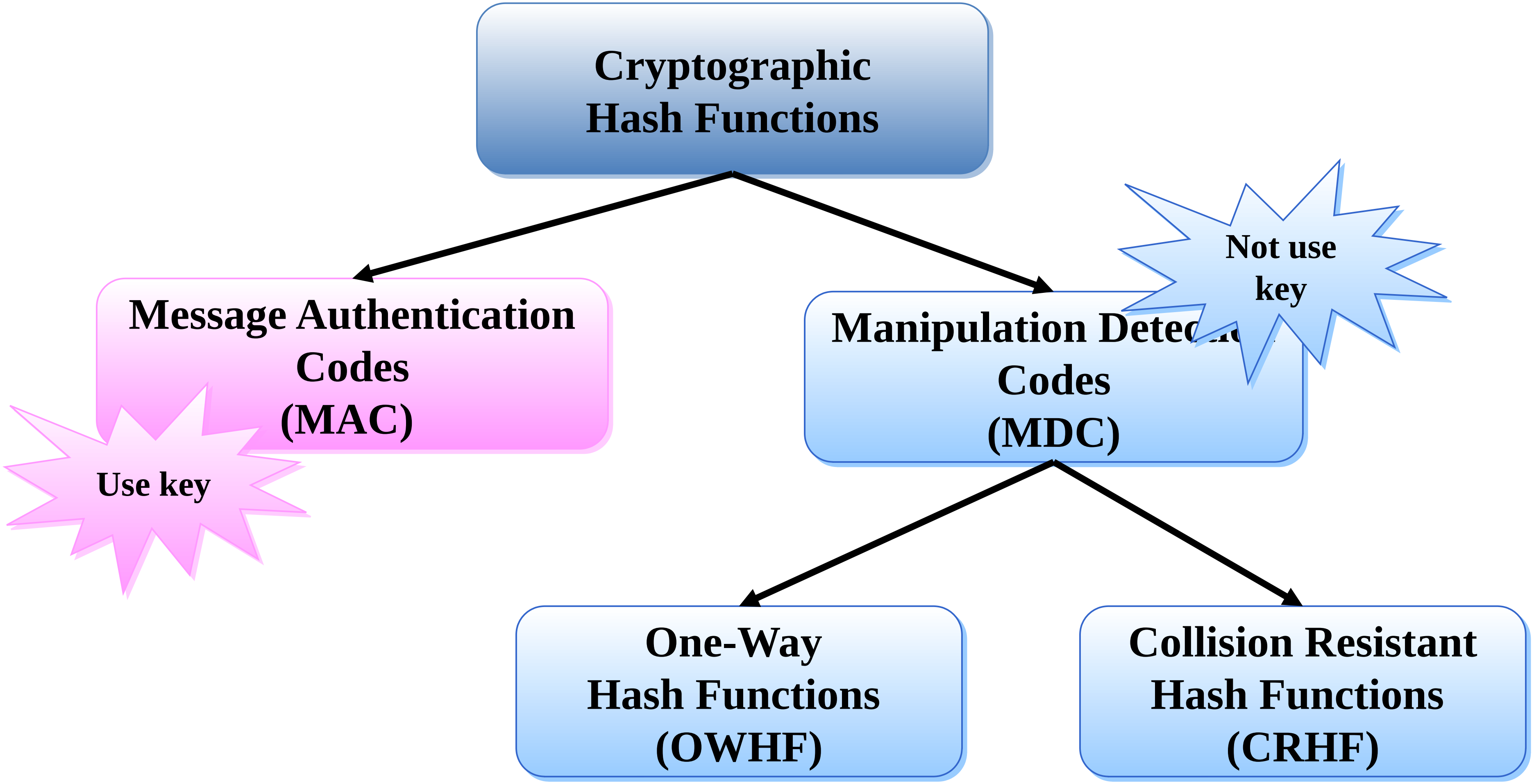
Safe from "weak" collisions

- **Weak Collision Resistance**
- Given a randomly chosen bit string x, hard to find x' such that $H(x) = H(x')$
- The attacker must find a value that collides with a given x value. This is harder to find a pair of x and x' colliding each other.
- Brute-force attack: $O(2^n)$
- **Comment:** safety against “weak” collisions does not guarantee safety against collisions

Properties of the hash function

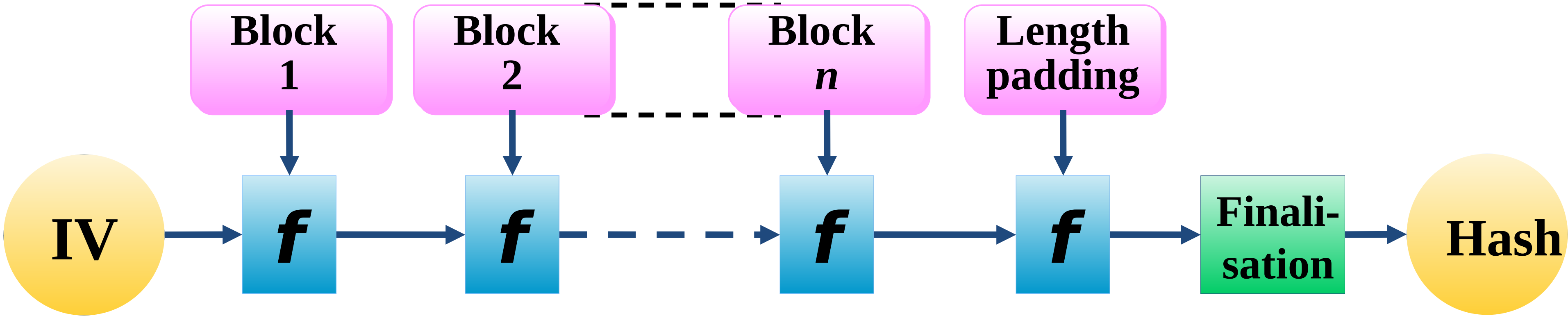
- Safe against “preimage” attacks
  - Preimage resistance or one-wayness
  - Given y, hard to find x such that $H(x) = y$
- Safety against collisions
  - Collision resistance
  - Hard to find 2 distinct values x and x' such that $H(x') = H(x)$
- Safe against “second preimage” attack
  - 2nd preimage resistance or weak collision resistance
  - Given x and $y = H(x)$, hard to find $x' \neq x$ such that $H(x') = y$

Classification of cryptographic hash function

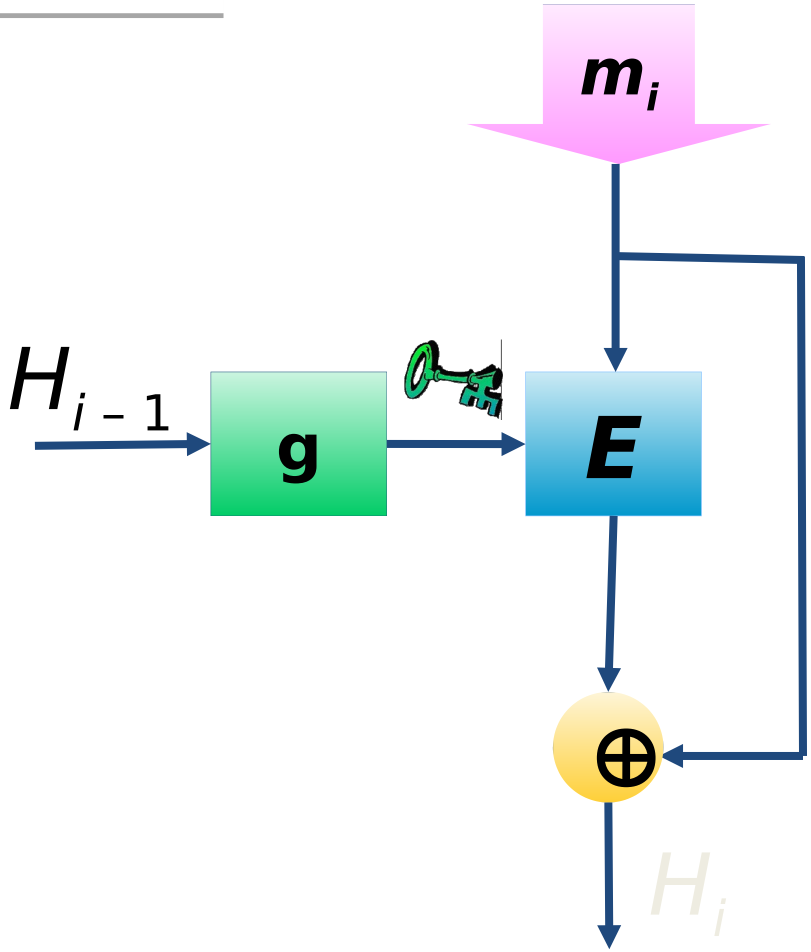


Merkle-Damgård architecture

- Authors: Ralph Merkle, Ivan Damgård
- Most hash functions use this structure
- Example: SHA-1, MD5



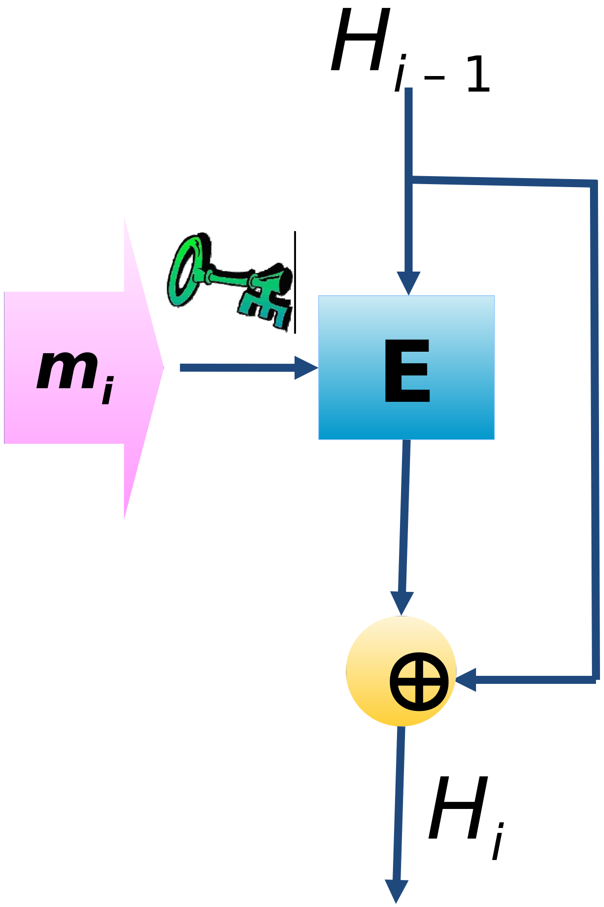
Matyas-Meyer-Oseas architecture



- □ Architecture "dual" with architecture **Davies-Mayer**
- □ At the 1st block (i = 1), need using initial value H_0
- □ If function E uses key and block with different sizes, function g need converting H_{i-1} to key suitable for function E

$$H_i = E_{g(H_{i-1})}(m_i) \oplus m_i$$

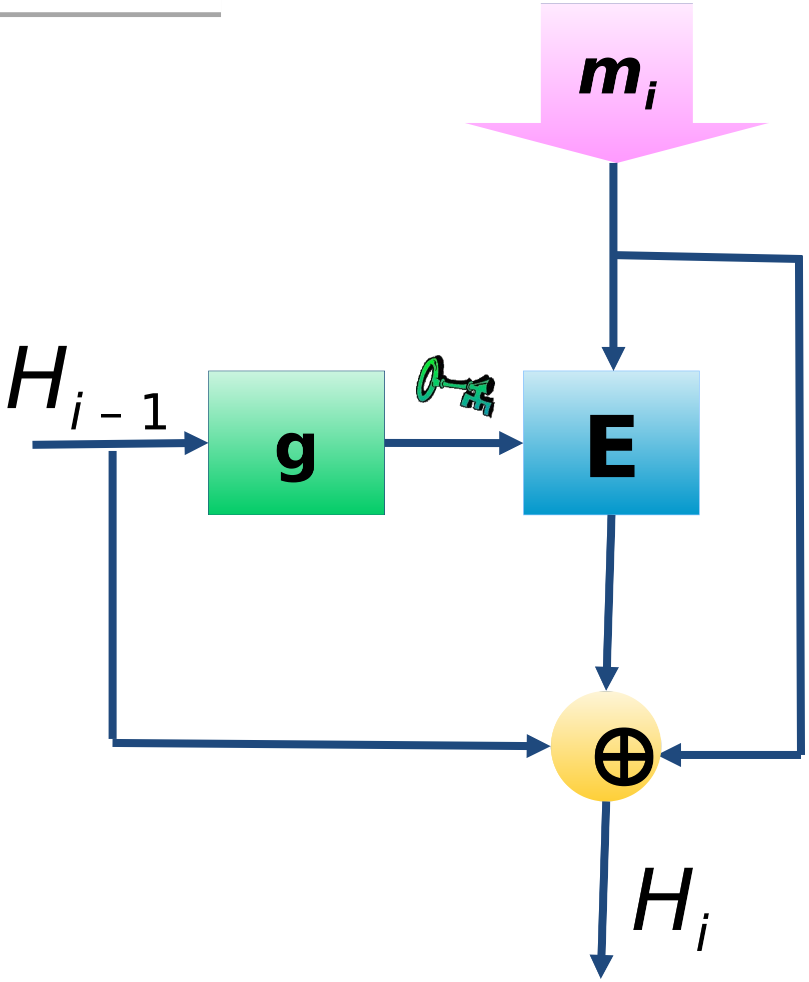
Davies-Meyer architecture



- □ Architecture "dual" with architecture **Matyas-Meyer-Oseas**
- □ At the 1st block (i = 1), need using initial value H_0
- □ If function E is not safe, then applying method of fixed-point attack to attack corresponding hash function

$$H_i = E_{m_i}(H_{i-1}) \oplus H_{i-1}$$

Miyaguchi-Preneel architecture

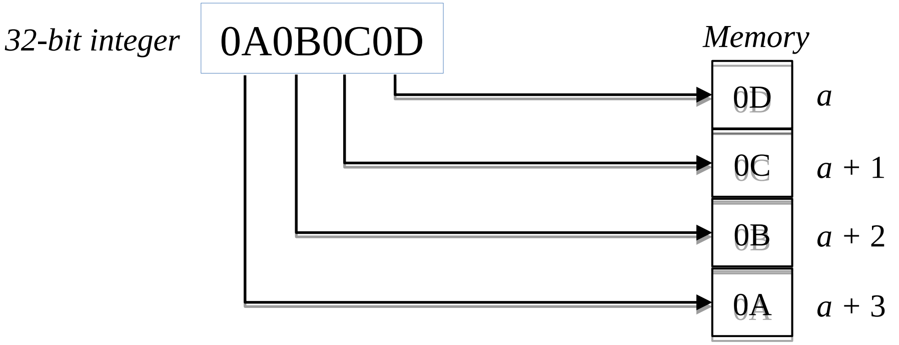


- □ Expansion of architecture **Matyas-Meyer-Oseas**
- □ At the 1st block (i = 1), need using initial value H_0
- □ If function *E* uses key and block with different sizes, then function *g* need converting H_{i-1} to key suitable for function *E*

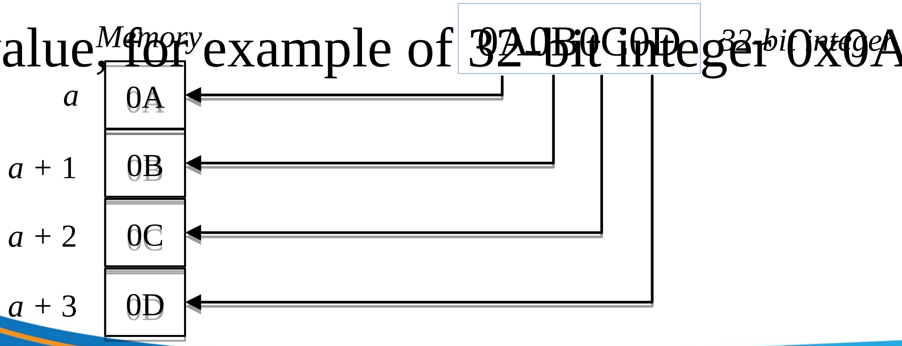
$$H_i = E_{g\,(H_{i-1})}(m_i) \oplus H_{i-1} \oplus m_i$$

Message-Digest algorithm (MD5)

- Message Digest 4 hash proposed by Rivest in 1990. In 1991, improved version called MD5 was proposed.
- Notes:
  - Little-endian value, for example of a 32-bit integer 0x0A0B0C0D



  - Big-endian value, for example of 32-bit integer 0x0A0B0C0D



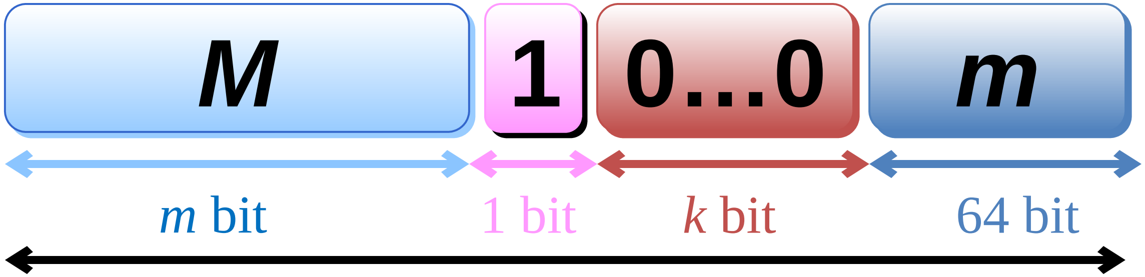
https://en.wikipedia.org/wiki/

Message-Digest algorithm (MD5)

- Steps in algorithm:
  - Declare: int i, $s[64]$, $K[64]$ //32-bit variables & mod 2^{32} when computing
  - Define values for left rotation coefficient $R[i]$ of each cycle:
    - $s[0..15]$ = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22}
    - $s[16..31]$ = {5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20}
    - $s[32..47]$ = {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23}
    - $s[48..63]$ = {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21}
  - Initialize variables:
    - a_0 = 0x67452301
    - b_0 = 0xEFCDAB89
    - c_0 = 0x98BADCFE
    - d_0 = 0x10325476

Message-Digest algorithm (MD5)

- Steps in algorithm:
  - Compute constants $K[i]$ using below loop:
    - **for** i **from** 0 **to** 63 { $K[i] = \text{floor}(abs(sin(i + 1)) \times 2^{32}\}^{*}$
  - Pre-processing:
    - Add bit 1 at the end of the message
    - Add k bit 0 such that length of message congruent 448 (mod 512)
    - Add 64 bits to represent the length of original message (little-endian stored value)



Multiples of 512

[*] *Can compute offline*

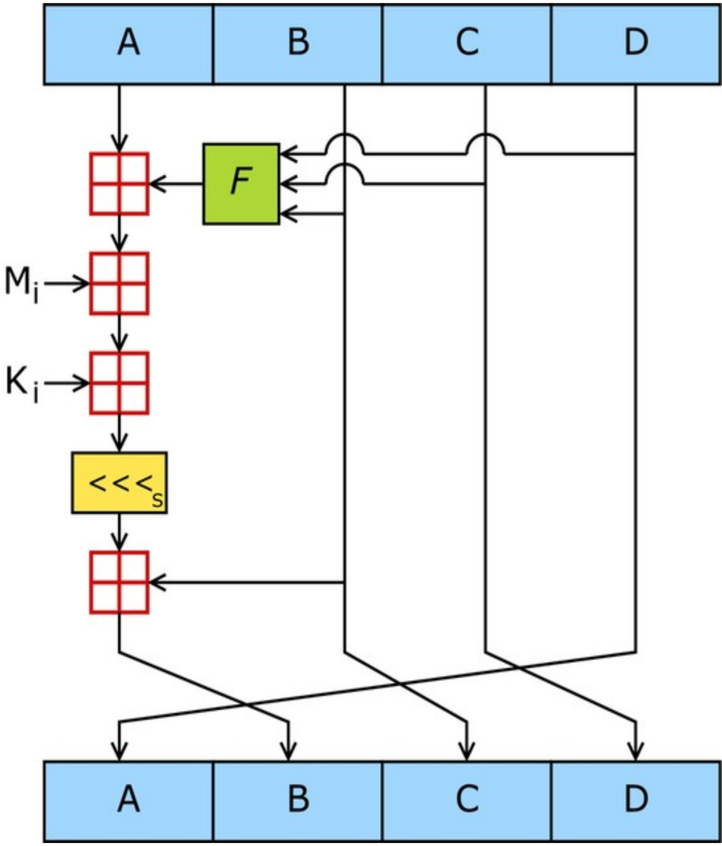
Message-Digest algorithm (MD5)

- Example of pre-processing step:
  - Assume input-data is a string *m* = “hello world”
  - Convert *m* to ASCII code:
    - 01101000 01100101 01101100 01101100 01101111 00100000 01110111 01101111 01110010 01101100 01100100
  - Due to |*m*| = 88 bits ⇒ need more 360 bits to satisfy 448 mod 512 ≅ 448 bits ⇒ add one bit ‘1’ and 359 bit ‘0’
  - Convert 88 bits to binary form:
    - 0000000000000000000000000000000000000000000000000000000001011000
  - So, we have *m* after padded: (|*m*| = 512 bits):

01101000	01100101	01101100	01101100	01101111	00100000	01110111	01101111	01110010	01101100	01100100
10000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	01011000		

Message-Digest algorithm (MD5)

- □ Steps in algorithm:
  - □ Divide message (padded *m*) into **512-bit blocks**
    - □ For each 512-bit block (ex: q^{th} block)
      - □ Divide into 16 words (little-endian 32-bit word) $w[0..15]$
      - □ Create 4 variables $A = a_0$, $B = b_0$, $C = c_0$, $D = d_0$
      - □ Start 64 cycles processing *A, B, C, D*
      - □ $a_0 \mathrel{+}= A$, $b_0 \mathrel{+}= B$, $c_0 \mathrel{+}= C$, $d_0 \mathrel{+}= D$
  - □ Final digest message: $a_0 \mid b_0 \mid c_0 \mid d_0$
  - □ Example of describing **one cycle** from 64 cycles
    - □ *A, B, C, D* are 4 words (32 bits) of a state
    - □ *F* is non-linear function (changed with i^{th} cycle)
    - □ $<<<_s$ is left-rotate *s* positions taken from *s*[64]
    - □ ⊞ add modulo 2^{32}.
    - □ K_i is a constant from *K*[64]



*Describe **one cycle** in 64 cycles*

https://en.wikipedia.org/wiki/MD5

Processing cycle of MD5

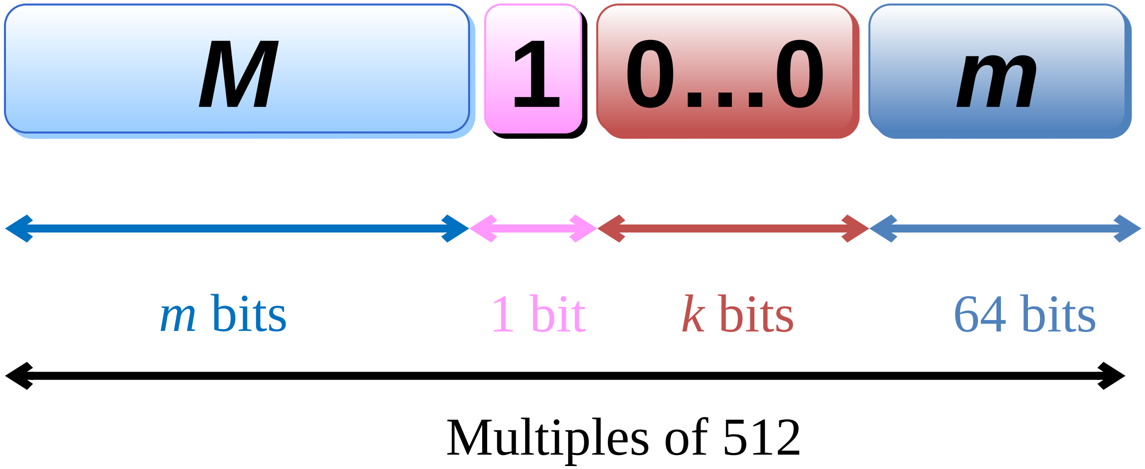
- Pseudo-code of 64 cycles
  - **for** i **from** 0 to 63
    - int f, g
    - if $0 \le i \le 15$ then { $f = (B \wedge C) \vee ((\neg B) \wedge D)$; $g = i$ }
    - if $16 \le i \le 31$ then { $f = (D \wedge B) \vee ((\neg D) \wedge C)$; $g = (5 \times i + 1) \bmod 16$ }
    - if $32 \le i \le 47$ then { $f = B \oplus C \oplus D$; $g = (3 \times i + 5) \bmod 16$ }
    - if $48 \le i \le 63$ then { $f = C \oplus (B \vee (\neg D))$; $g = (7 \times i) \bmod 16$ }
    - $f = f + A + K[\mathrm{i}] + M[g]$
    - $A = D$; $D = C$; $C = B$; $B = B + (f <<<_{s[\mathrm{i}]})$ // f left-rotates $s[i]$ positions
- Test-vector:
  - MD5(“”) = d41d8cd98f00b204e9800998ecf8427e
  - MD5(“fit.hcmus”) = 22227c3065cbf40733e9a11ffa07124a

Secure Hash Algorithm 1 (SHA-1)

- The Secure Hash Standard (SHS or SHA1) method developed by NIST and NSA was published in the Federal Register on January 31, 1992, and then officially became the standard method on May 13, 1993..
- Messages are processed in 512-bit blocks
- Digested message 160-bit length
- Steps in algorithm
  - Initialize variables:
    - h_0 = 0x67452301
    - h_1 = 0xEFCDAB89
    - h_2 = 0x98BADCFE
    - h_3 = 0x10325476
    - h_4 = 0xC3D2E1F0

Secure Hash Algorithm 1 (SHA-1)

- Steps in algorithm
  - Pre-processing data:
    - Add bit 1 at the end of the message
    - Add *k* bits ‘0’ such that the length of message ≅ 448 (mod 512)
    - Add 64 bits to represents the length of the original message (value stored in big-endian format)



Secure Hash Algorithm 1 (SHA-1)

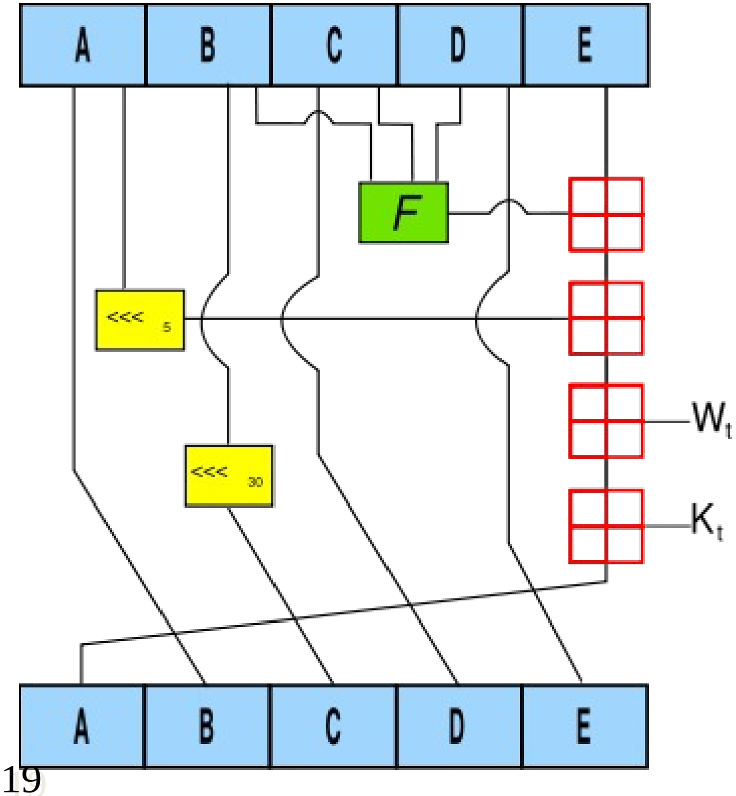
- Example of pre-processing step:
  - Assume input-data is a string *m* = “hello”
  - Convert *m* to ASCII code:
    - 01101000 01100101 01101100 01101100 01101111
  - Due to |*m*| = 40 bits ⇒ need more 408 bits to satisfy 448 mod 512 ≅ 448 bits ⇒ add one bit ‘1’ and 407 bits ‘0’
  - Convert 40 bits to binary form (stand for 64 bits = 2-word)
    - 0000000000000000000000000000000000000000000000000000000000101000
  - So, *m* after padded: (|*m*| = 512 bits, sixteen 32-bit words):

01101000	01100101	01101100	01101100	01101111	10000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00101000		

Secure Hash Algorithm 1 (SHA-1)

- □ Steps in algorithm
  - □ Divide message (padded *m*) into 512-bit blocks
    - □ For each 512-bit block:
      - □ Divide into 16 words (32 bits, big-endian) *w*[0..15]
      - □ for *i* from 16 to 79 // Extend 16 words (32 bits) to 80 words (32 bits)
        - □ $w[i] = (w[i-3] \oplus w[i-8] \oplus w[i-14] \oplus w[i-16]) \lll 1 \; (16 \le i < 80)$
      - □ $A = h_0$, $B = h_1$, $C = h_2$, $D = h_3$, $E = h_4$
      - □ **Start 80 cycles processing**
      - □ $h_0 \mathrel{+}= A$, $h_1 \mathrel{+}= B$, $h_2 \mathrel{+}= C$, $h_3 \mathrel{+}= D$, $h_4 \mathrel{+}= E$
  - □ Result = $h_0 \mid h_1 \mid h_2 \mid h_3 \mid h_4$
  - □ For example, describing **one cycle** in 80 cycles
    - □ *t* is an ordinal number of the cycle ($0 \le t \le 79$)
    - □ *A*, *B*, *C*, *D*, *E* are 5 words (32 bits) of a state
    - □ *F* is a non-linear function (changed with cycle)
    - □ <<< *n* is a left-rotate *n* positions
    - □ ⊞ add modulo 232.
    - □ K_t is a constant following ⇒

$$K_t = \begin{cases} 0x5a827999, & 0 \le t \le 19 \\ 0x6ed9eba1, & 20 \le t \le 39 \\ 0x8f1bbcdc, & 40 \le t \le 59 \\ 0xca62c1d6, & 60 \le t \le 79 \end{cases}$$



*Describe **one cycle** in 80 cycles*

https://en.wikipedia.org/wiki/SHA-1

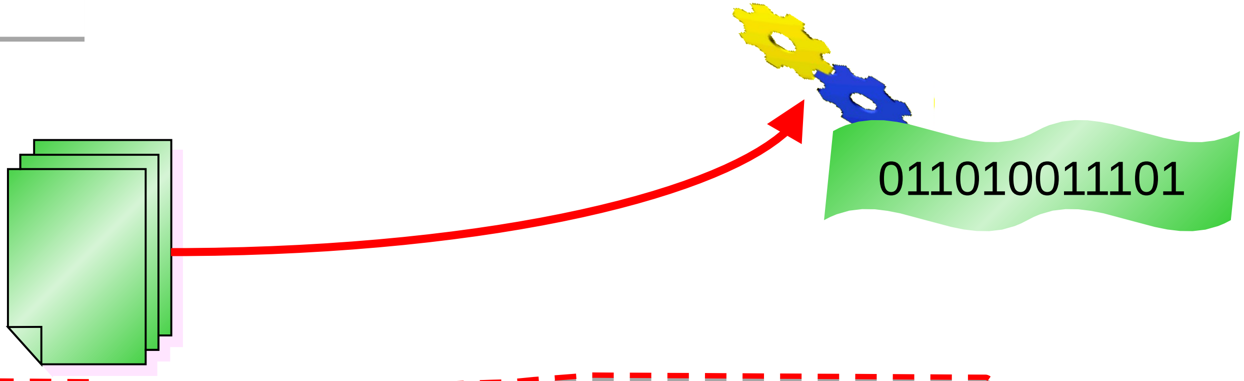
Processing cycle of SHA-1

- Pseudo-code of 80 cycles

```
for $i$ from 0 to 79
    if $0 \le i \le 19$ then { $f = (B \wedge C) \vee ((\neg B) \wedge D)$; $K$ = 0x5A827999 }
    if $20 \le i \le 39$ then { $f = B \oplus C \oplus D$; $K$ = 0x6ED9EBA1 }
    if $40 \le i \le 59$ then { $f = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$; $K$ = 0x8F1BBCDC }
    if $60 \le i \le 79$ then { $f = B \oplus C \oplus D$; $K$ = 0xCA62C1D6 }
    temp = $(A <<< 5) + f + E + K + w$[i]
    $E = D$; $D$ = C; $C = B <<< 30$
    $B = A$
    $A$ = temp
```

- Test-vector:
  - SHA-1(“”) = da39a3ee5e6b4b0d3255bfef95601890afd80709
  - SHA-1(“fit.hcmus”) = 86cb71d2190be898de94356d59e5f0138f8d8496

A group of SHA hash functions



SHA-256 2001

SHA-1 1994

SHA-0 1993

SHA-384 2001

SHA-224 2004

SHA-512 2001

SHA3-224 2015

SHA-512/256 2012

SHA-512/224 2012

SHA3-256 2015

SHAKE256 2015

SHAKE128 2015

SHA3-384 2015

SHA3-512 2015

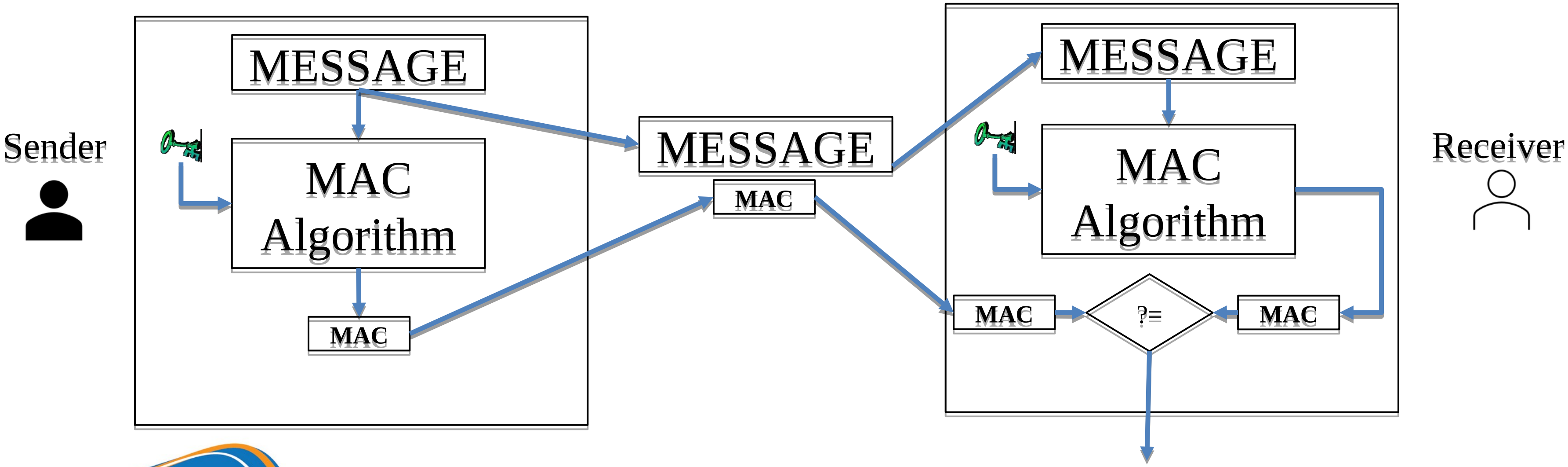
Secure Hash Standard

SHA algorithms

Algorithm	Result (bit)	State (bit)	Block (bit)	Maximum message (bit)	Cycle	Operation	Collision
SHA-0	160	160	512	$2^{64} - 1$	80	+, and, or, xor, rotl	Yes
SHA-1							2^{63} operations
SHA-256/224	256/224	256			64	+, and, or, xor, shr, rotr	No
SHA-512/384	512/384	512	1024	$2^{128} - 1$	80		
SHA3-224/256/384/512	224/256/384/512	1600	1152/1088/832/576	No limit	24	and, xor, rot, not	
SHAKE-128/256	Tùy ý		1344/1088				

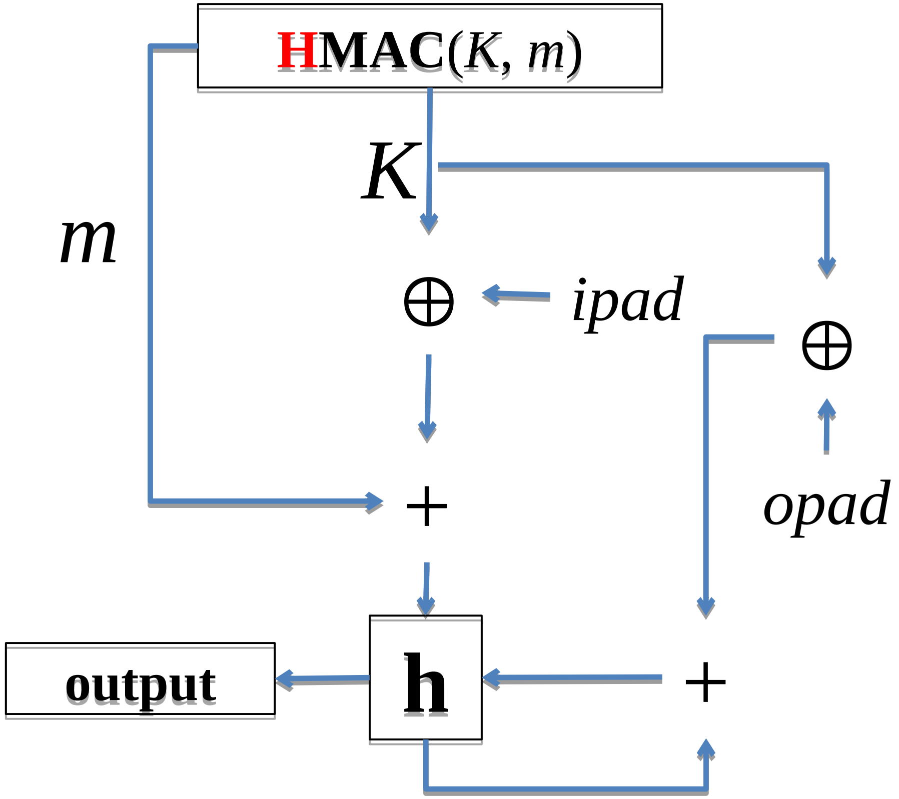
Message authentication code (MAC)

- Purpose: determine the origin of information (digital signature)
  - Generate MAC & check MAC shared secret key
  - The sender & receiver must agree on the secret key in advance
  - Does not support non-repudiation
  - The MAC can be generated from a cryptographic hash function (HMAC) or from a block cipher (OMAC, CBC-MAC, PMAC).



Decision: if same then authentic and integrity checked else something is wrong

Keyed-hash message authentication code



Pseudo-code:

```
function HMAC(K, m)
  opad = [0x5c × blocksize]
  ipad = [0x36 × blocksize]
  if (length(K) > blocksize) then
    K = hash(K)
  end if
  for i from 0 to length(K) step 1
    ipad[i] ^= K[i]
    opad[i] ^= K[i]
  end for
  return hash(opad || hash(ipad || m))
```

$$\mathbf{HMAC}_K(m) = h((K \oplus opad) \| h((K \oplus ipad) \| m))$$

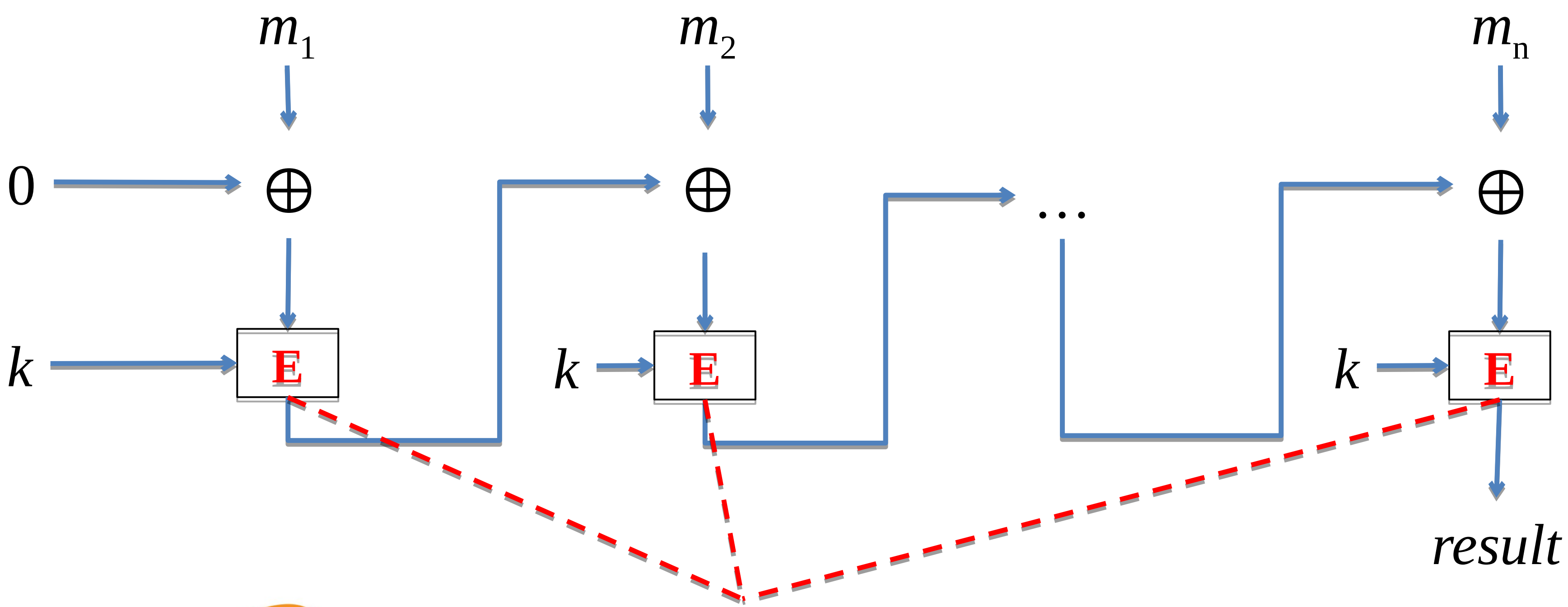
0x5c5c5c…5c5c

0x363636…3636

Mihir Bellare, Ran Canetti, Hugo Krawczyk (1996)

CBC-MAC

- See more:
  - How to attack?
  - Reference: CMAC



Example DES or AES