CS577: Introduction to Blockchain and Cryptocurrency

Introduction to Cryptography

Dr. Raju Halder

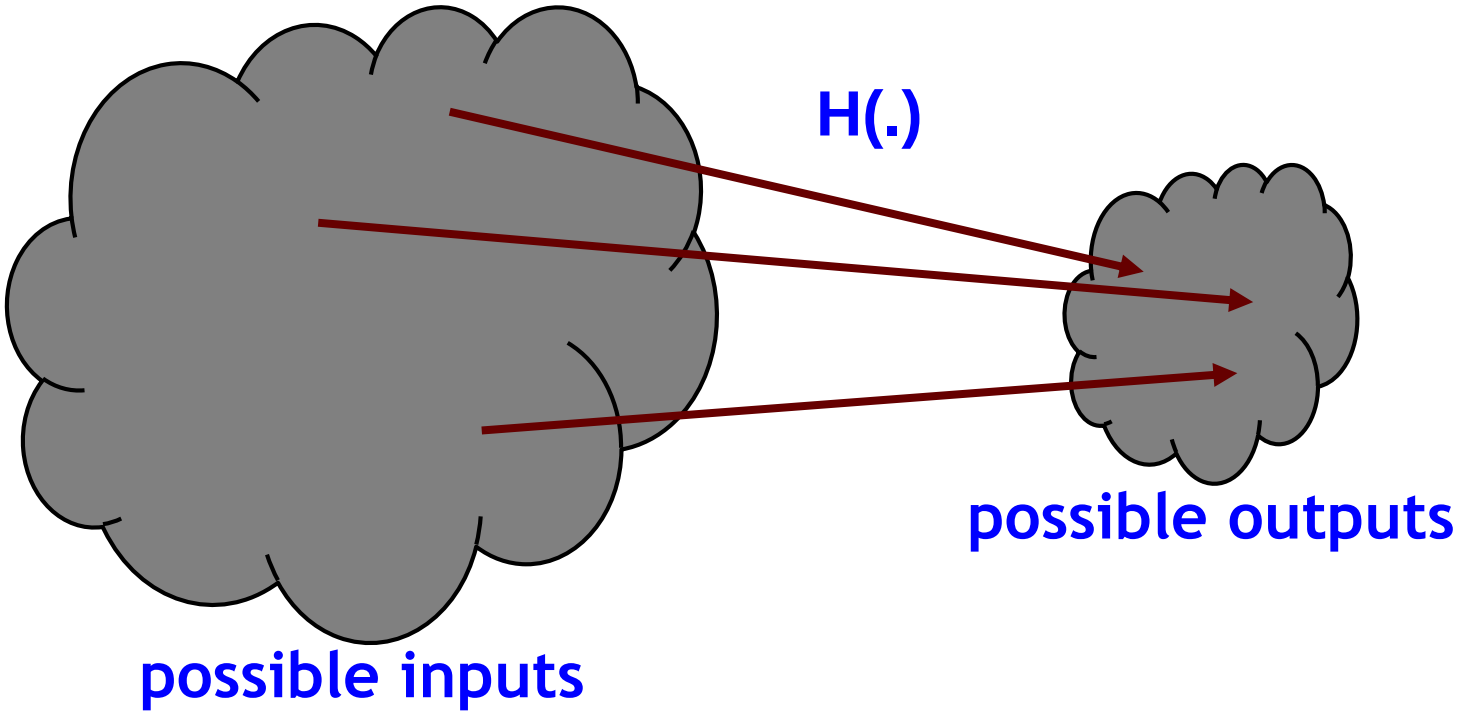
Hash function:

mathematical function

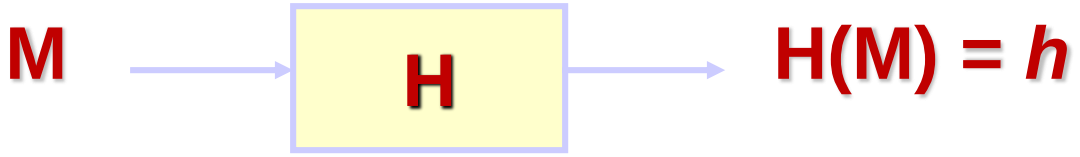
takes any string as input

fixed-size output (we'll use 256 bits)

efficiently computable (say, O(n))



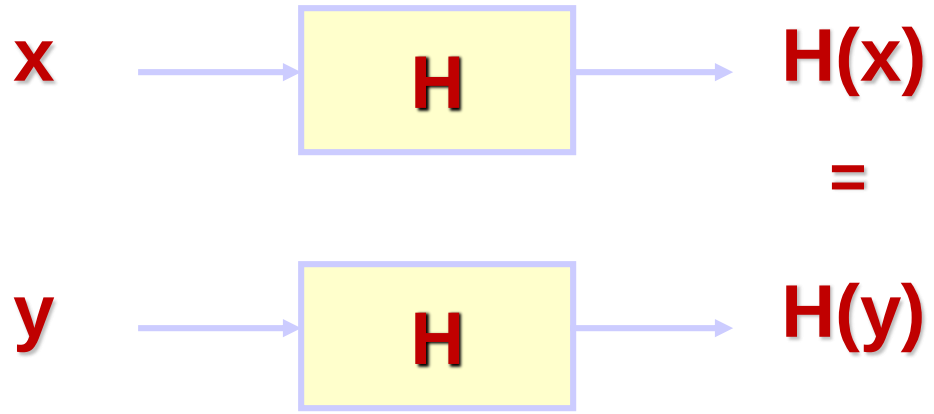
One-Way Hash Functions



Example

- M = “Elvis”
- H(M) = (“E” + “L” + “V” + “I” + “S”) mod 26
- H(M) = (5 + 12 + 22 + 9 + 19) mod 26
- H(M) = 67 mod 26
- H(M) = 15

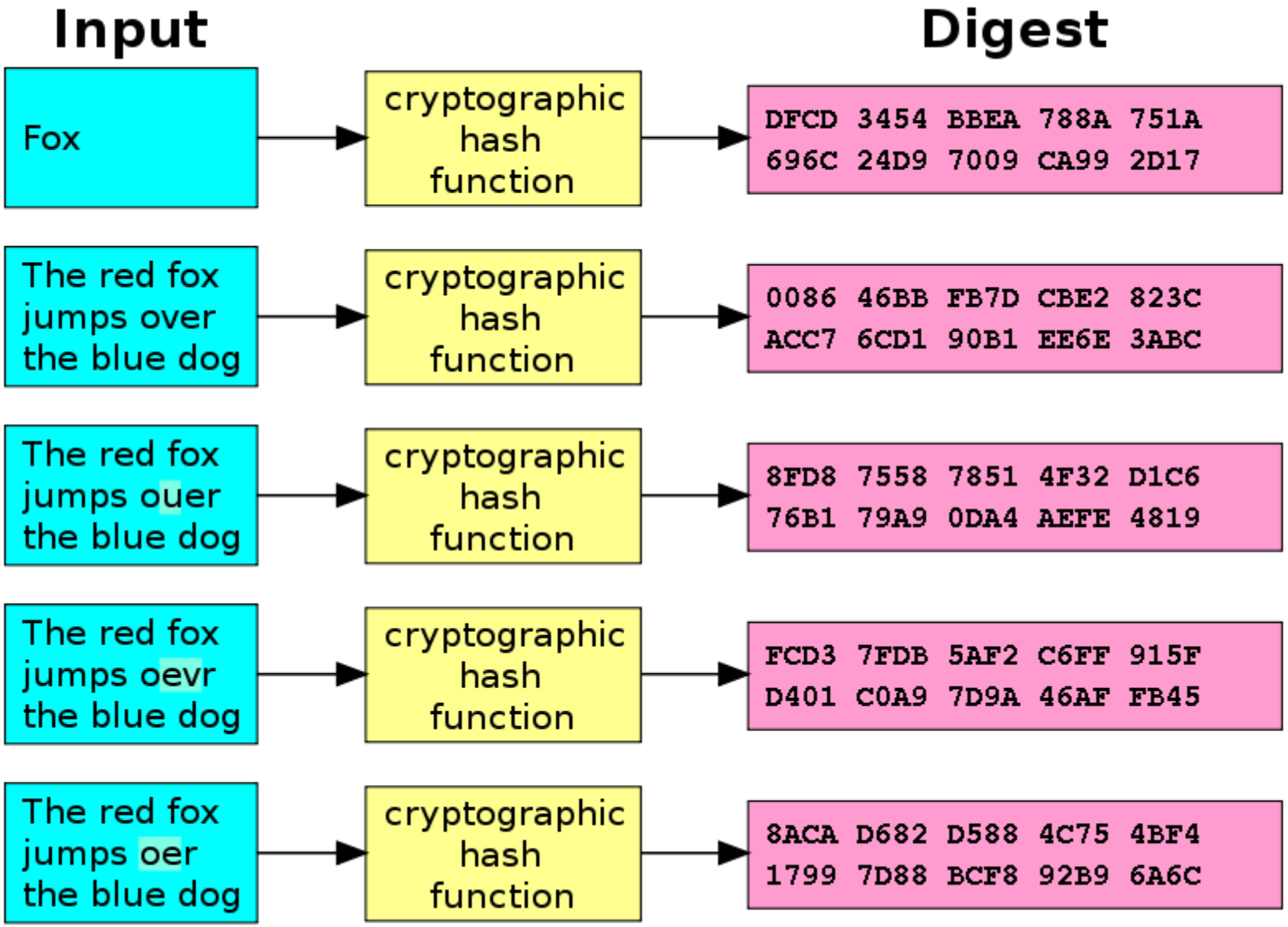
Collision



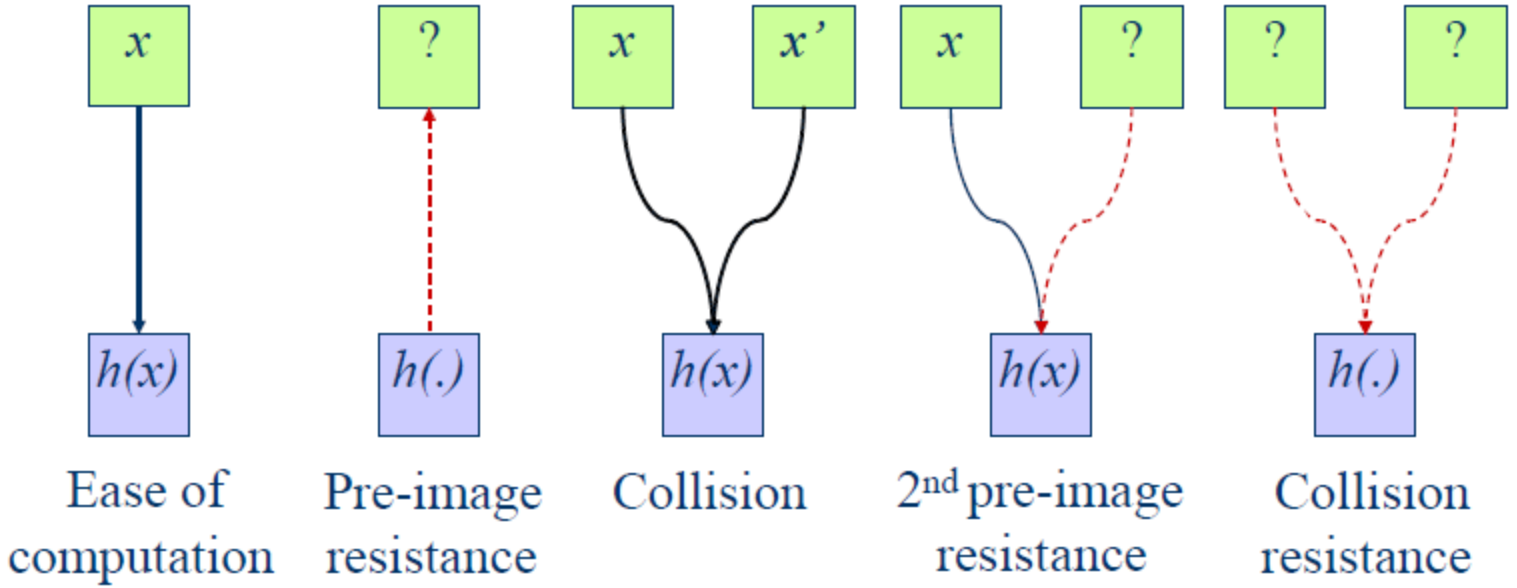
Example

- x = “Viva”
- Y = “Vegas”
- H(x) = H(y) = 2

avalanche effect



HASH PROPERTIES



- Collision resistance implies 2nd pre-image resistance
- Collision resistance does not imply pre-image resistance

Authentication

Basics

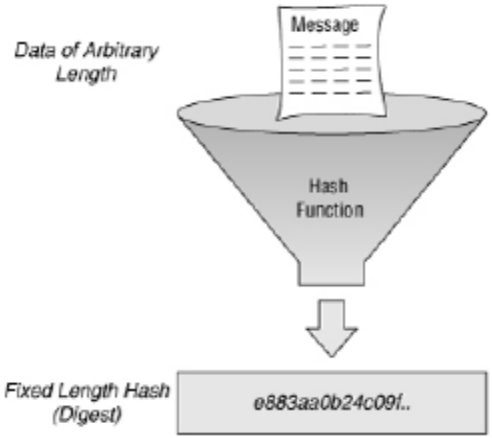
- Authentication is the process of validating the **identity** of a user or the **integrity** of a piece of data.

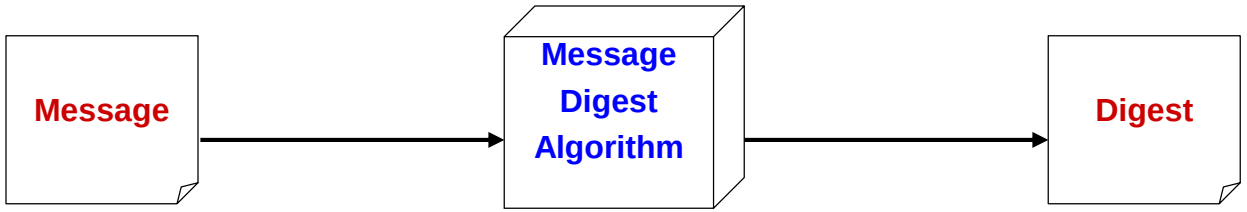
- Technologies that provide authentication
  - Message Digests (MD)
  - Message Authentication Codes (MAC)
  - Digital Signatures
  - Others….

Authentication

Message Digests (MD)

- A message digest is a fingerprint for a document.
- Purpose of the message digest is to provide proof that data has not altered - Integrity.
- Process of generating a message digest from data is called **hashing**

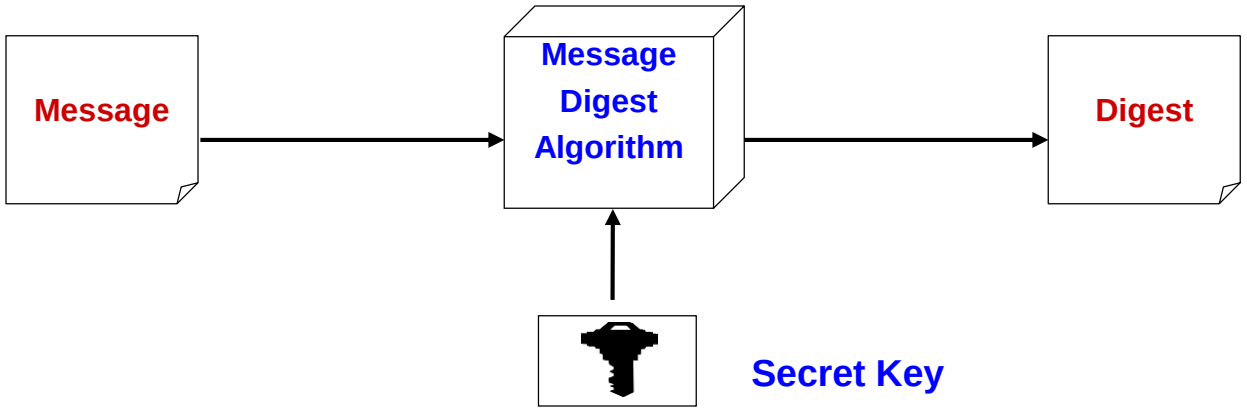




Authentication

Message Authentication Codes (MAC)

- A message digest created with a key
- Creates security by requiring a secret key to be possesses by both parties in order to retrieve the message
- A MAC is a short string used to verify the message integrity and authentication



COMMONLY USED HASH FUNCTIONS

- **MD** (Message Digest)
  - MD5
    - Max message < 2^{64}
    - Output: 128-bit
- **SHA** (Secure Hash Algorithm)
  - SHA-1
    - Max message < 2^{64}
    - Output: 160-bit
  - SHA-2
    - Max message < 2^{128}
    - Max output: 512-bit
  - SHA-3
    - Max message: Unlimited
    - Max output: 512-bit

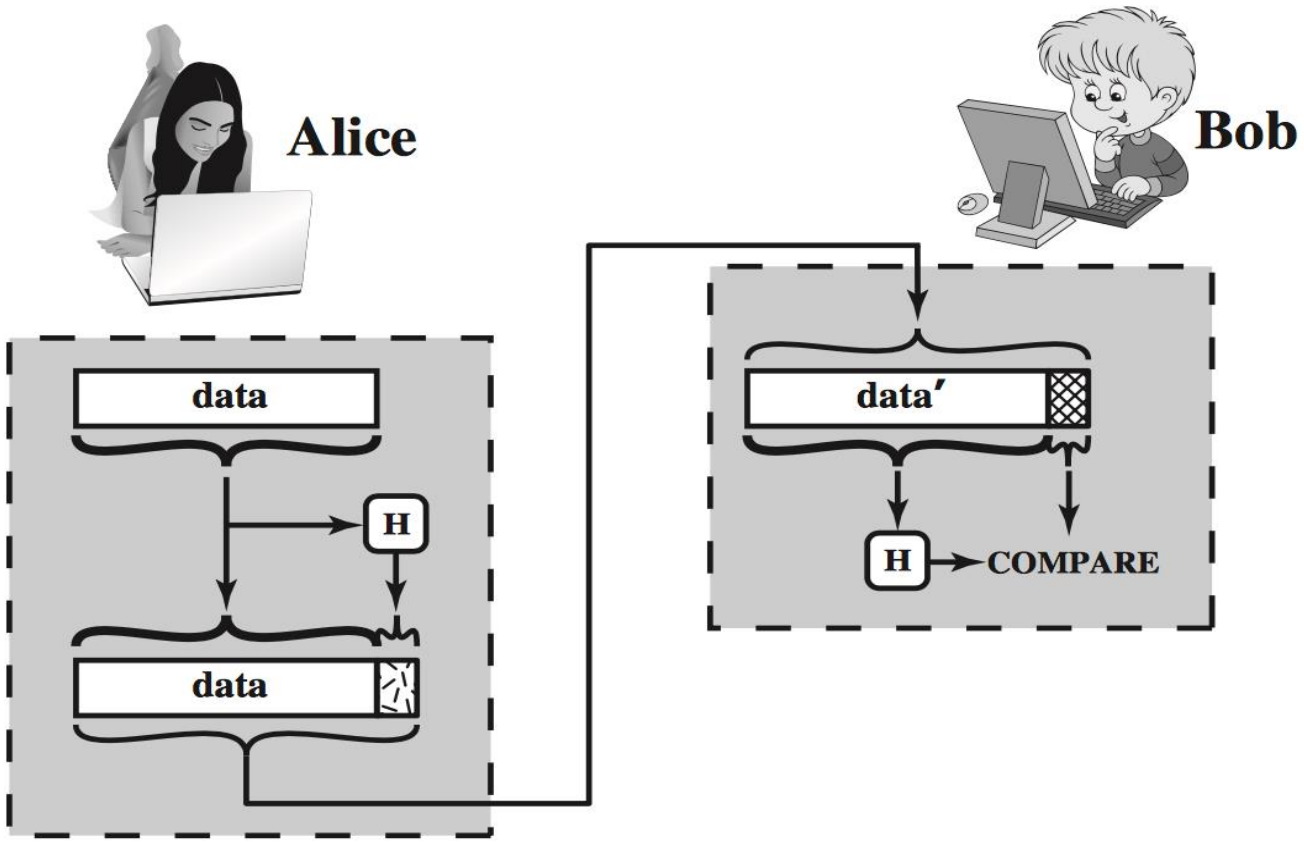
HASH VS. ENCRYPTION

- Hashing is a one-way
  - No unhashing
- Publicly known and there is no key used
- Efficient
- Deterministic (compared)
  - H(m) == H(m')
  - Of course, hashes with salts are not!
    - H(m || s1) and H(m || s2)

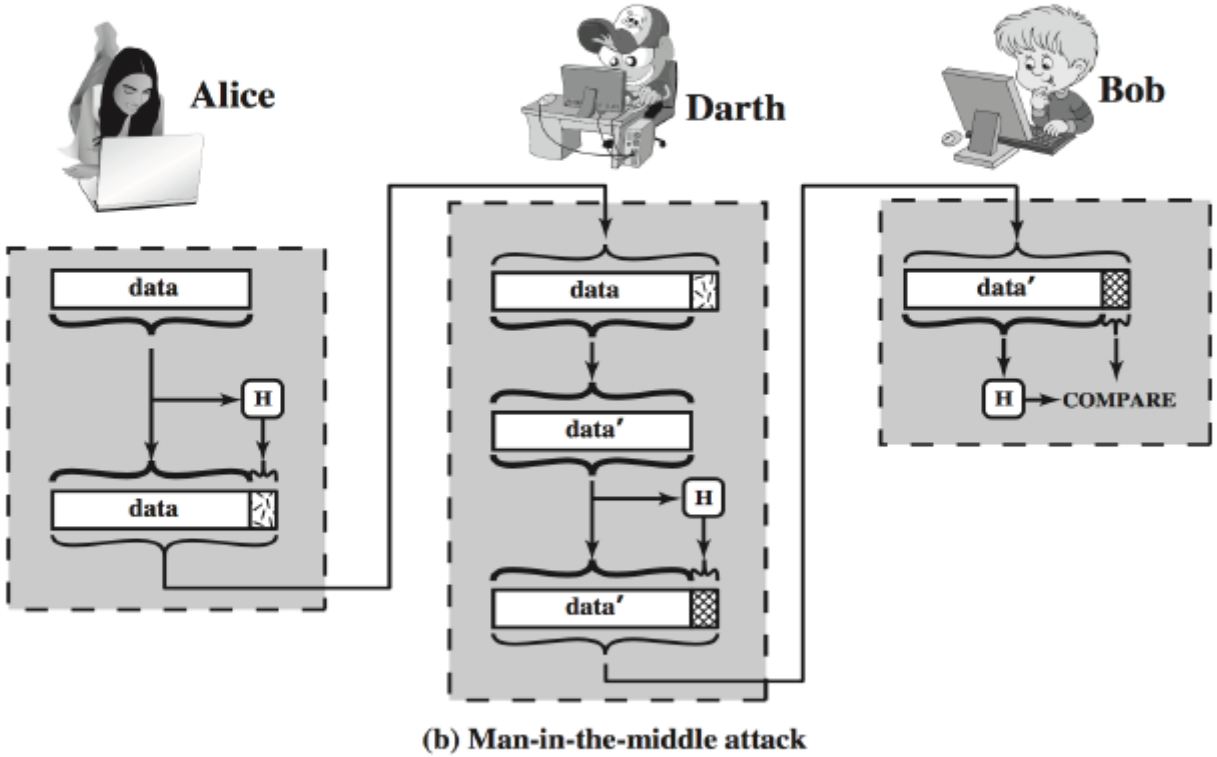
- Encryption is not one-way
  - Decryption renders the original message
- Publicly known algorithms but the key is kept secret
- Slower
- May or may not be deterministic (compared)
  - Randomised encryption
    - Enc(k, t1 || m) and Enc(k, t2 || m)

Cryptographic Hash Functions: Applications

- Integrity



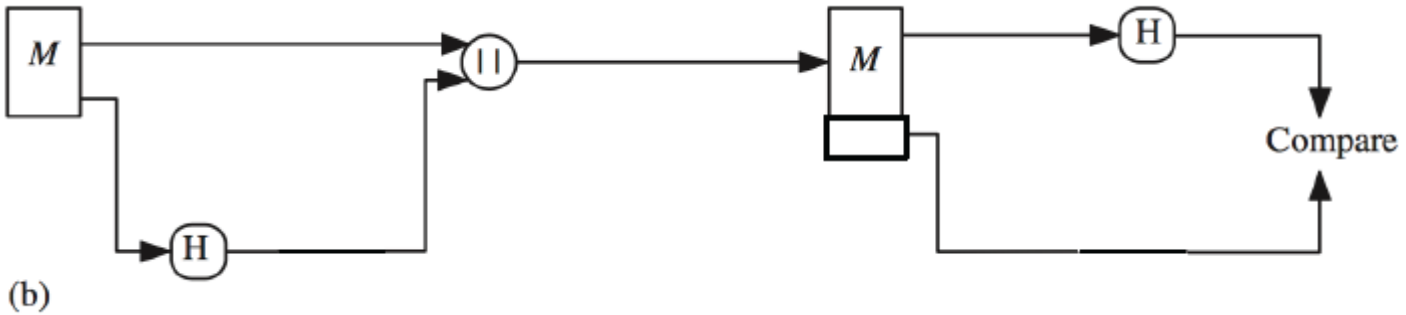
Attack Against Cryptographic Function

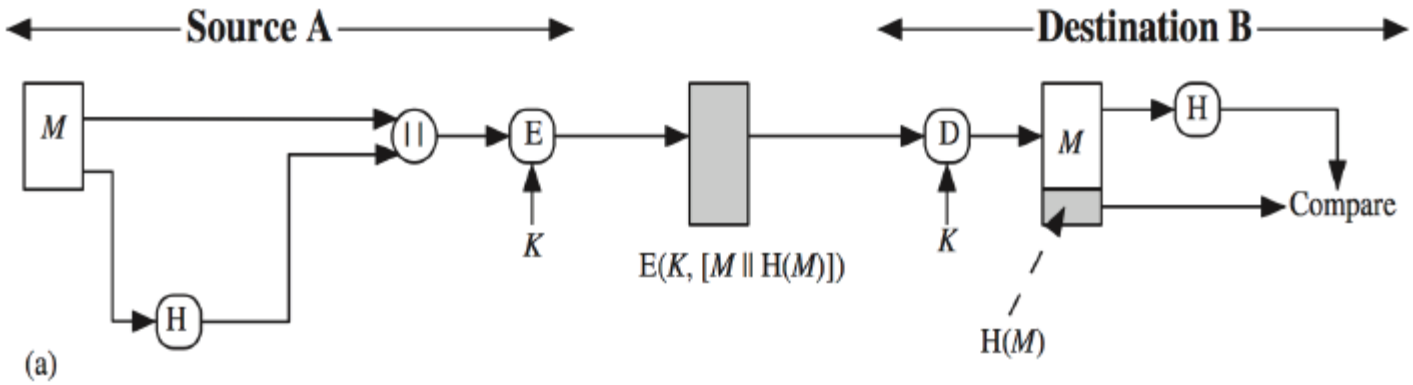


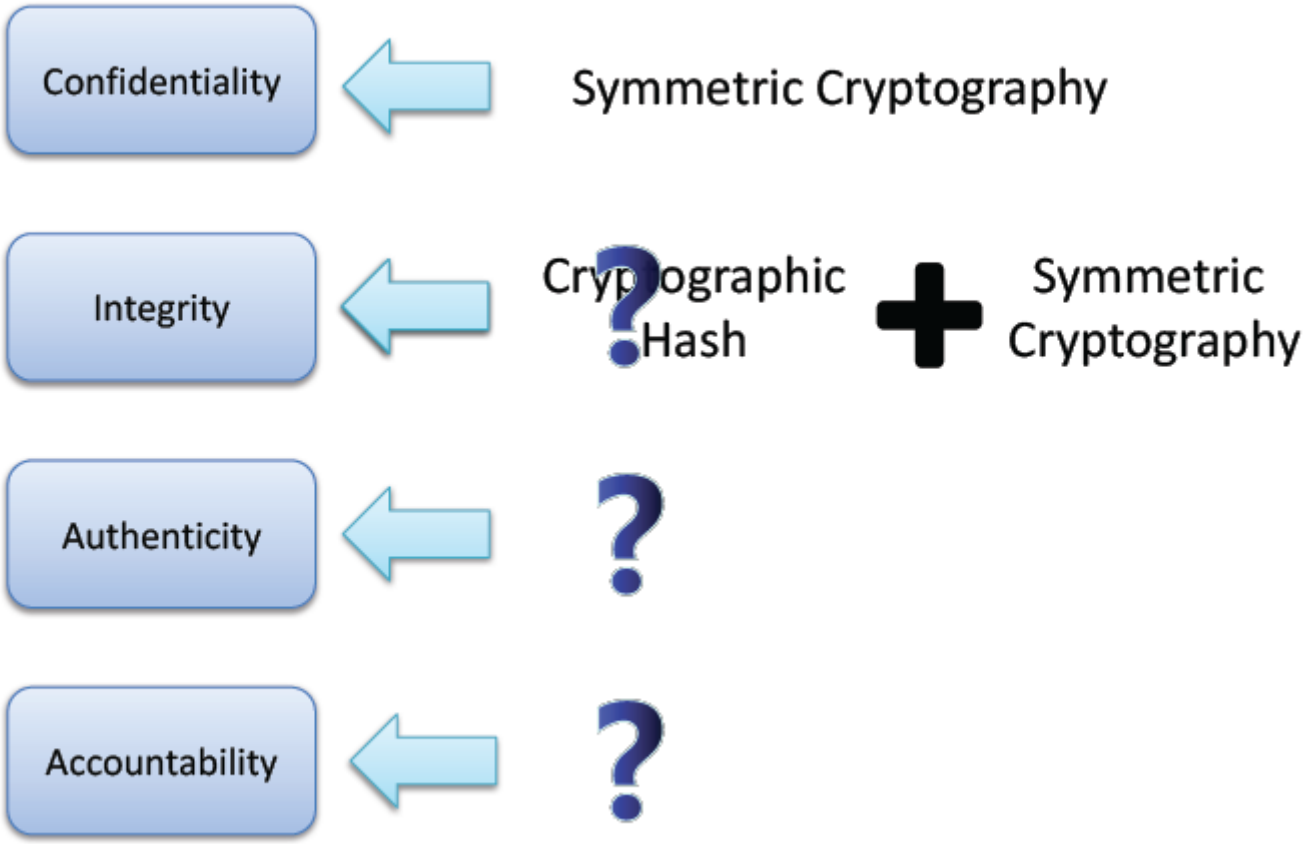
(b) Man-in-the-middle attack

The hash value must be **protected**.

Applying Symmetric Cryptography



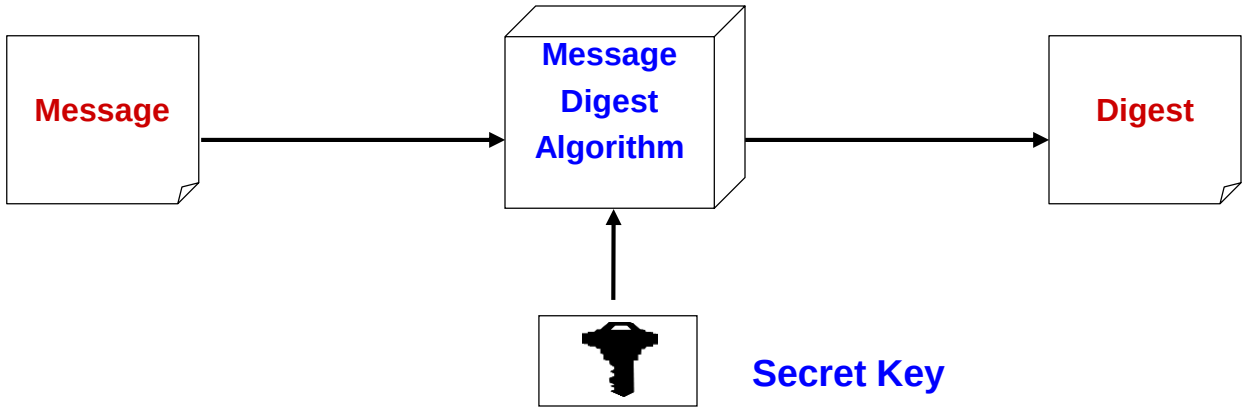



Confidentiality
Symmetric Cryptography
Integrity
Cryptographic Hash
Symmetric Cryptography
Authenticity
Accountability

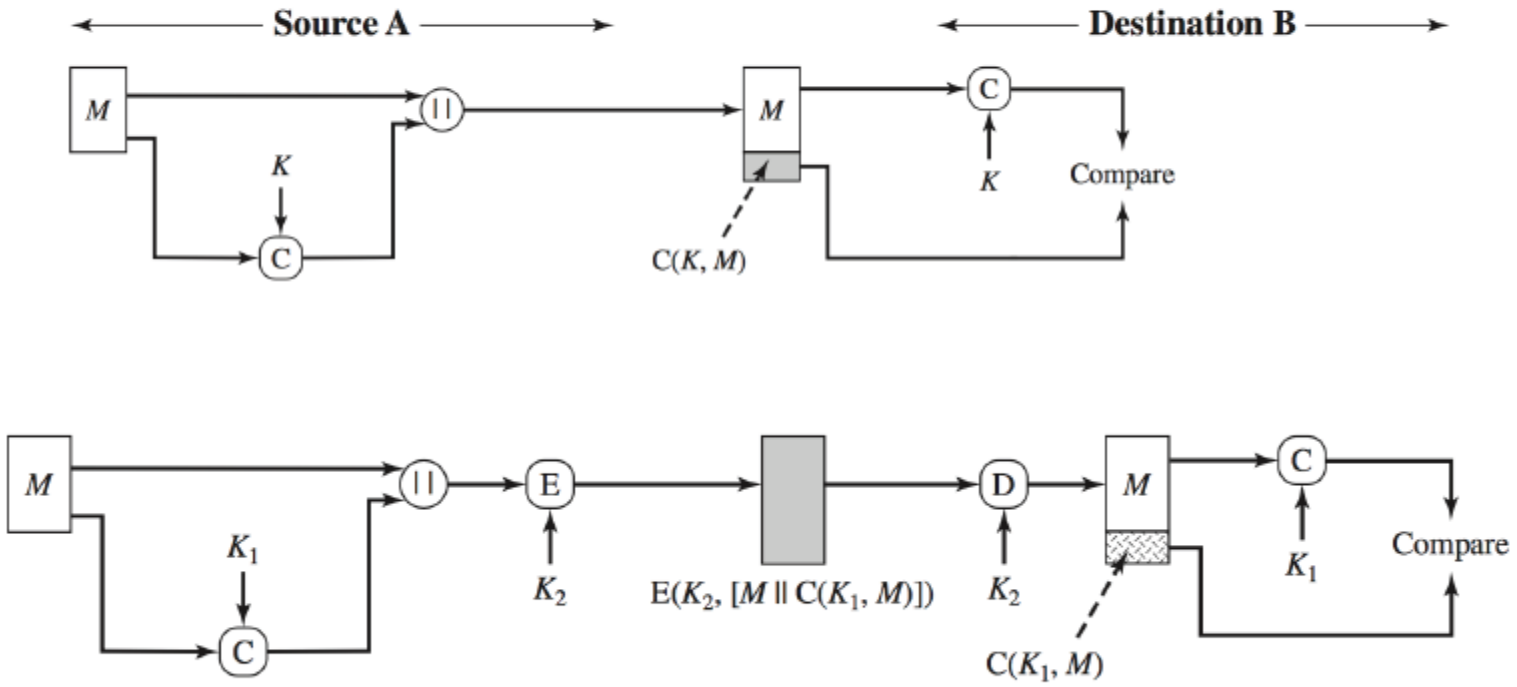
Authentication

Message Authentication Codes (MAC)

- The digest h = H(m) is generated without any key, such that anyone (including the adversary) can create it
- What if a key is used? Now the adversary cannot forge a keyed-hash value without the key
- So, a MAC is a short string used to verify the message integrity and authentication
- **A Popular MAC Algorithm: HMAC**



Schema of Applying MAC

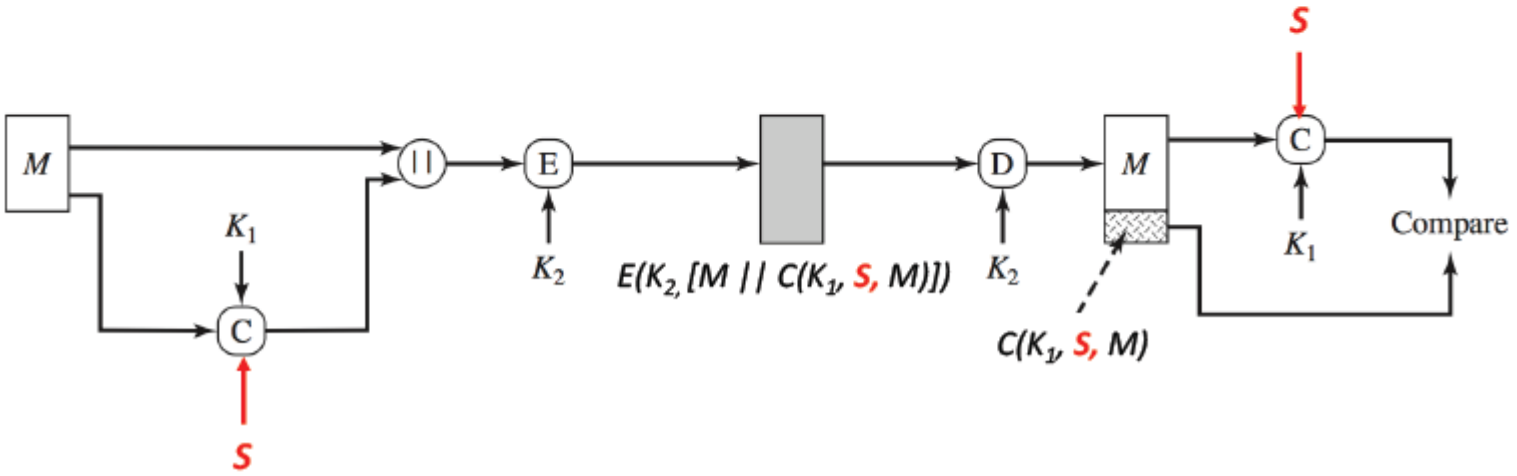


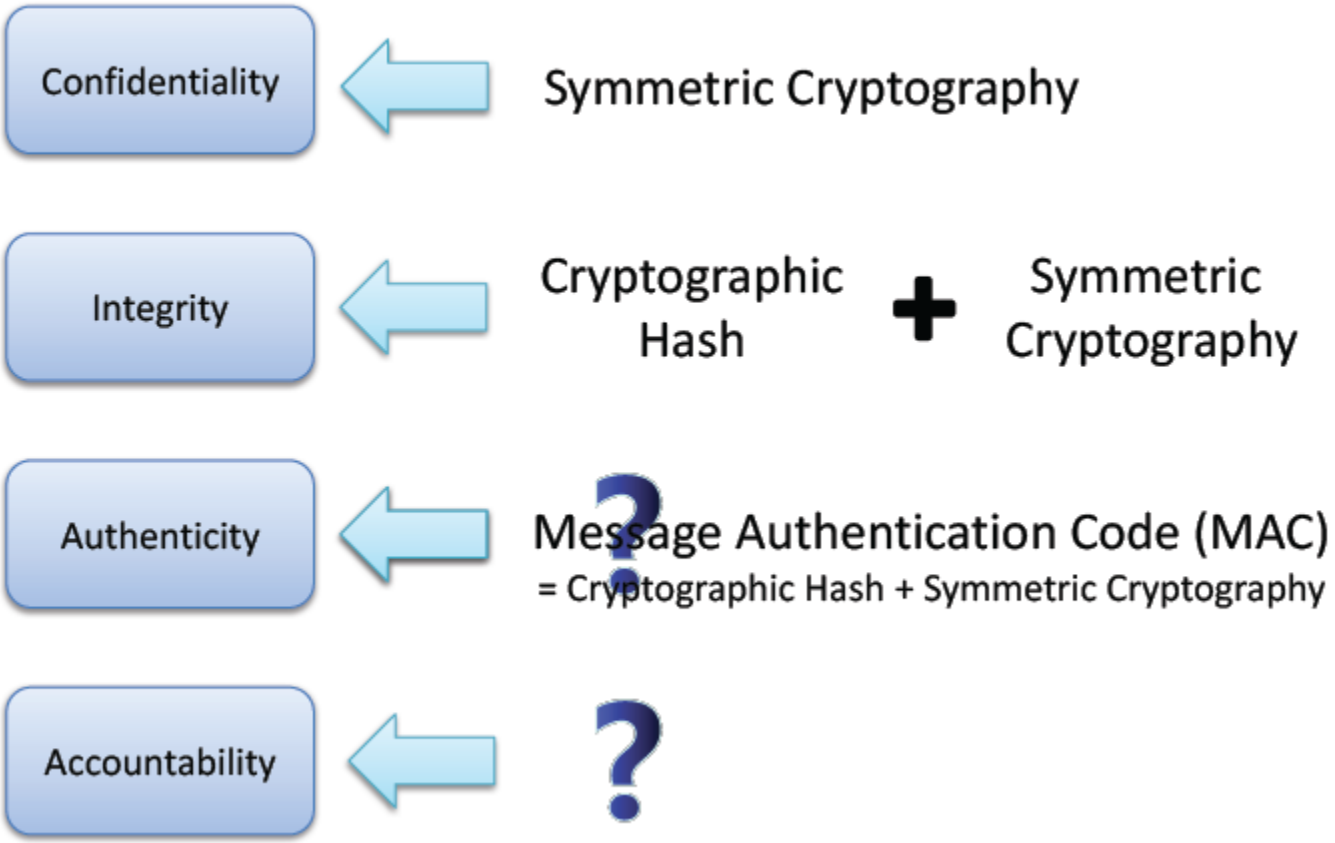
Vulnerable to Replay Attack

- mac = MAC(k, m)
- Adversary A can replay a message *m* that has been sent earlier by x and received by y

Adding timestamp, sequence number

- mac = MAC(k, t/seq, m)
  - To prevent the replay attack, the message should contain timestamp, or sequence number



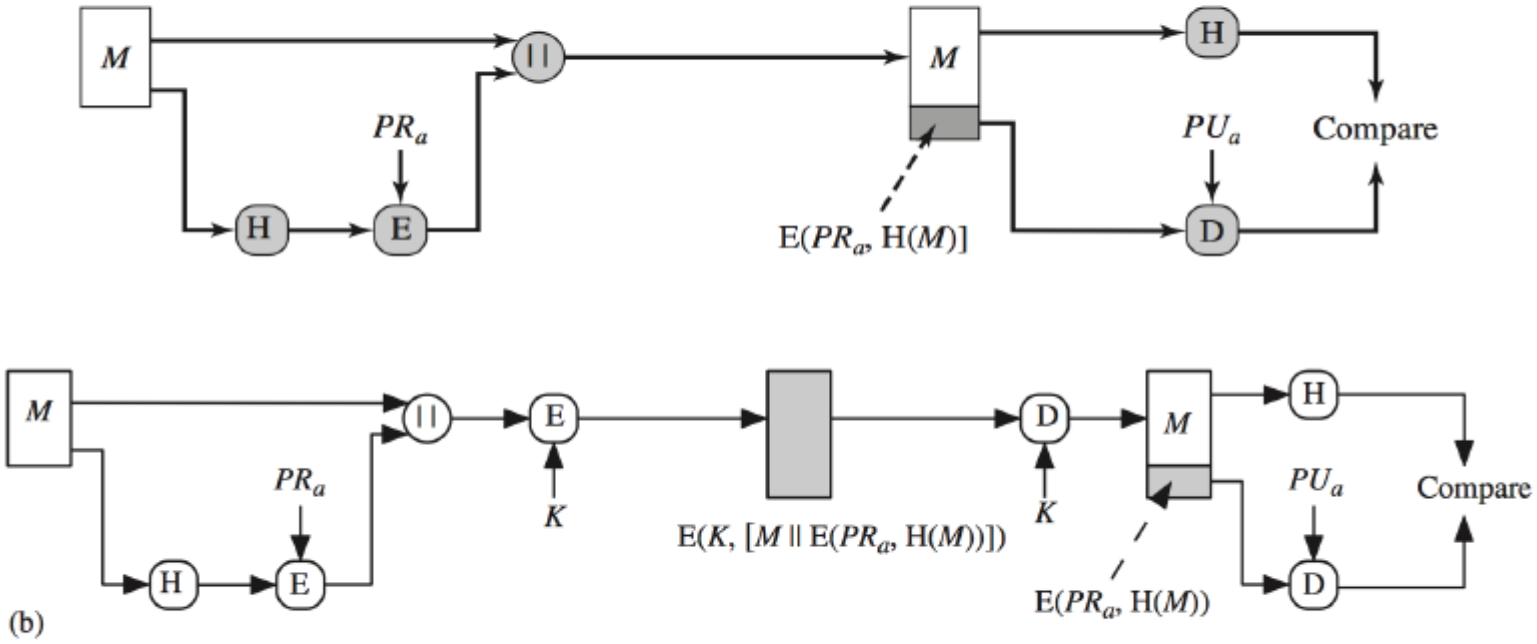

Confidentiality
Symmetric Cryptography
Integrity
Cryptographic Hash
+
Symmetric Cryptography
Authenticity
?
Message Authentication Code (MAC)
= Cryptographic Hash + Symmetric Cryptography
Accountability
?

Question

Can Message Authentication Code be used to verify data integrity, authentication, and achieve accountability?

MAC can be used to verify data integrity and authentication, but CANNOT achieve accountability (non-repudiation).

RSA's Digital Signature



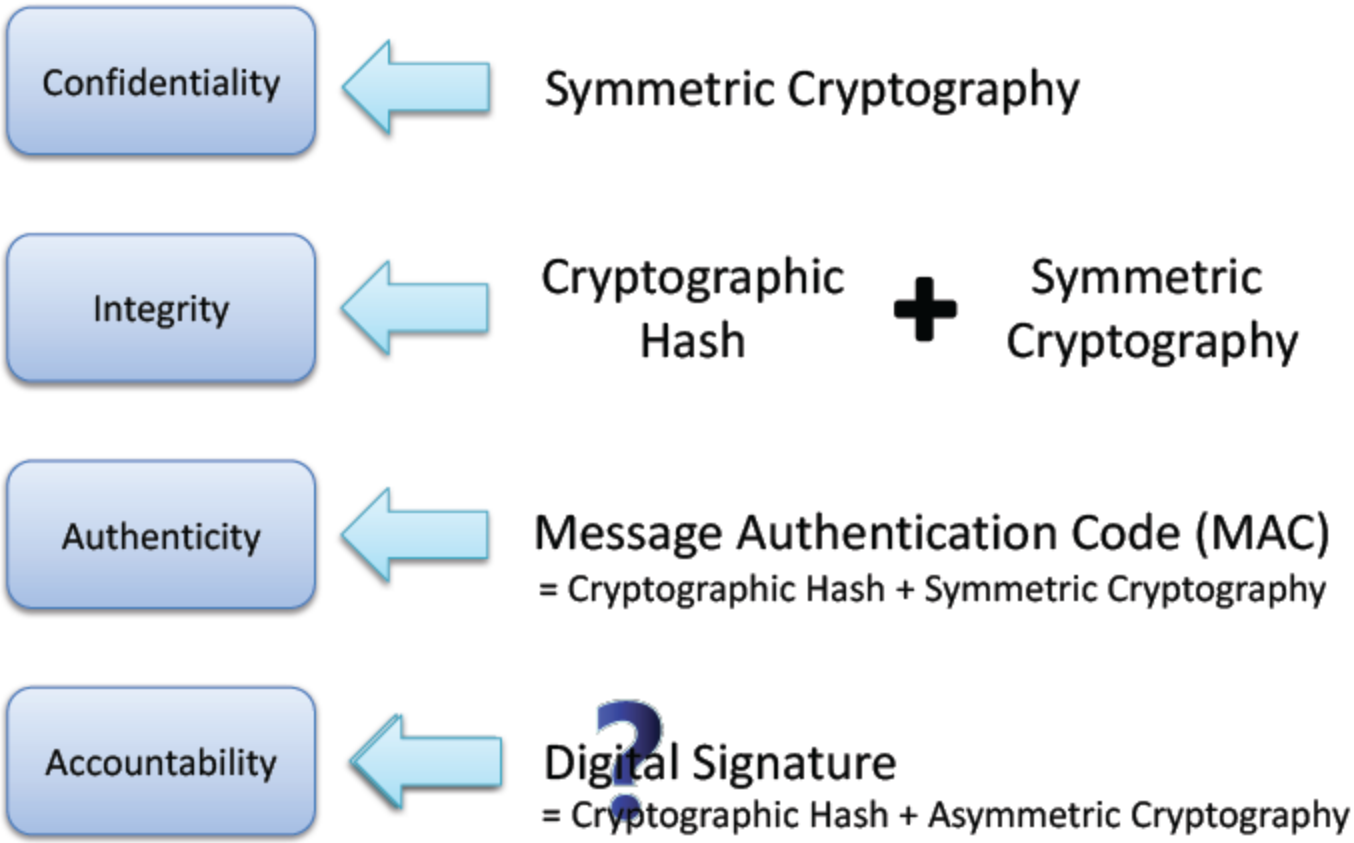
To prevent the replay attack, timestamp or sequence number should also be added

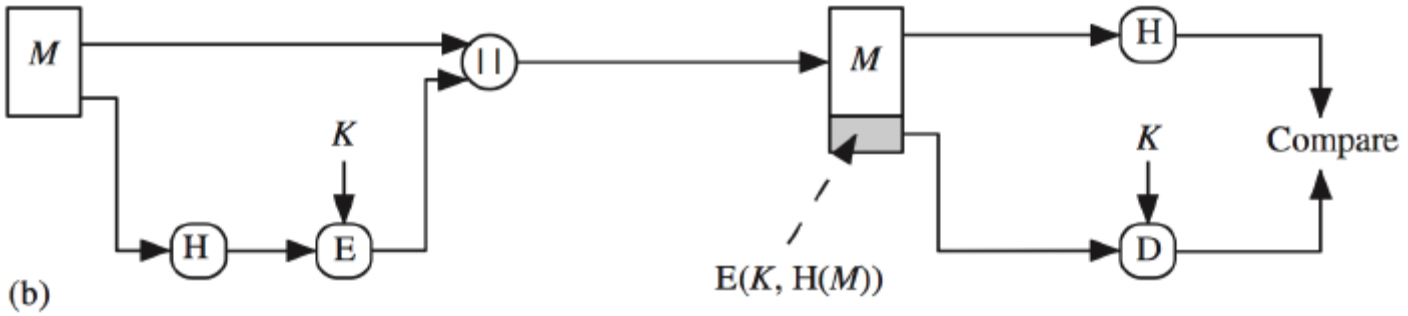
Question

Can Digital Signature be used to verify data integrity, authenticity, and achieve accountability?

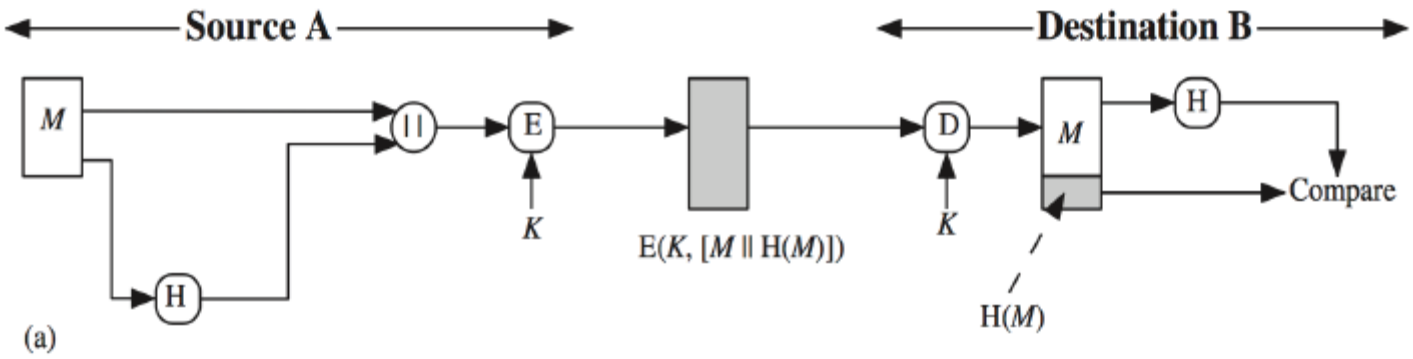
Yes.

- Data integrity and authenticity: the adversary may corrupt or replace the information being sent, but does not have the private key to sign the message digest
- Accountability: only the sender can generate the digital signature, since only the sender owns the private key. Thus, the sender cannot deny that the message was signed by her/him

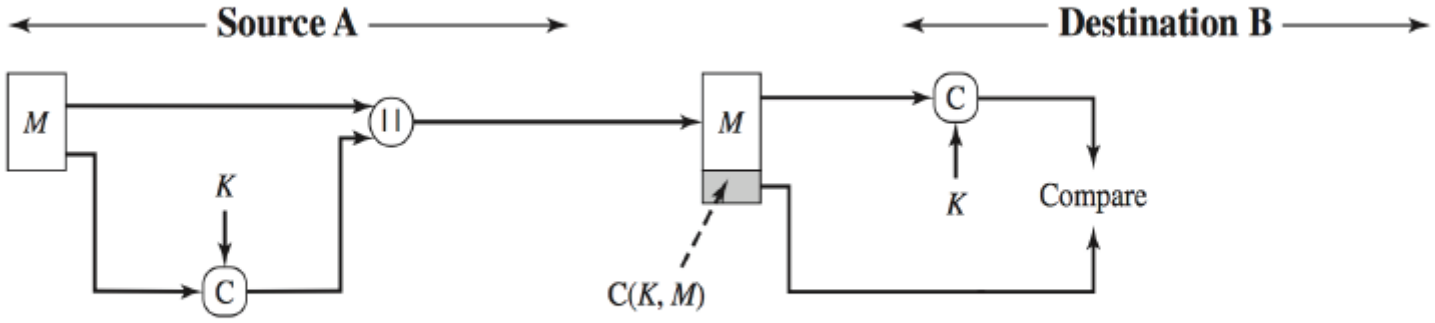

Confidentiality
Symmetric Cryptography
Integrity
Cryptographic Hash
+
Symmetric Cryptography
Authenticity
Message Authentication Code (MAC)
= Cryptographic Hash + Symmetric Cryptography
Accountability
?
Digital Signature
= Cryptographic Hash + Asymmetric Cryptography



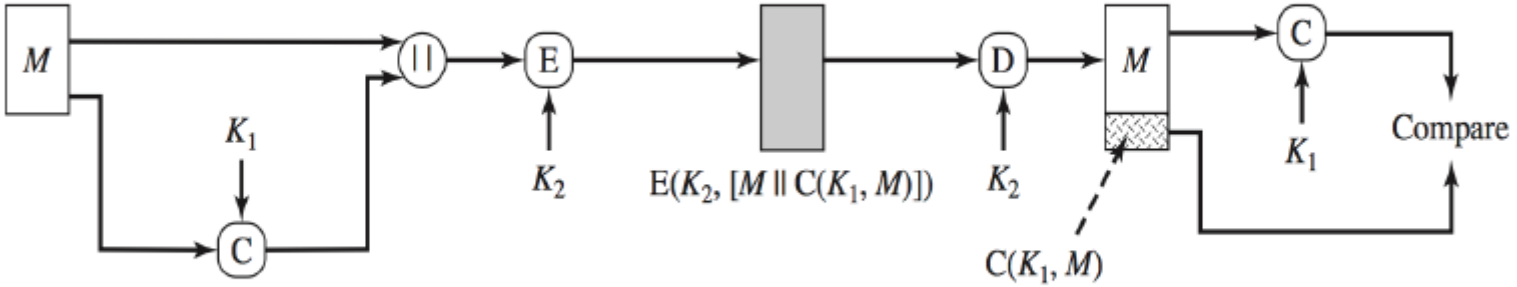
Integrity



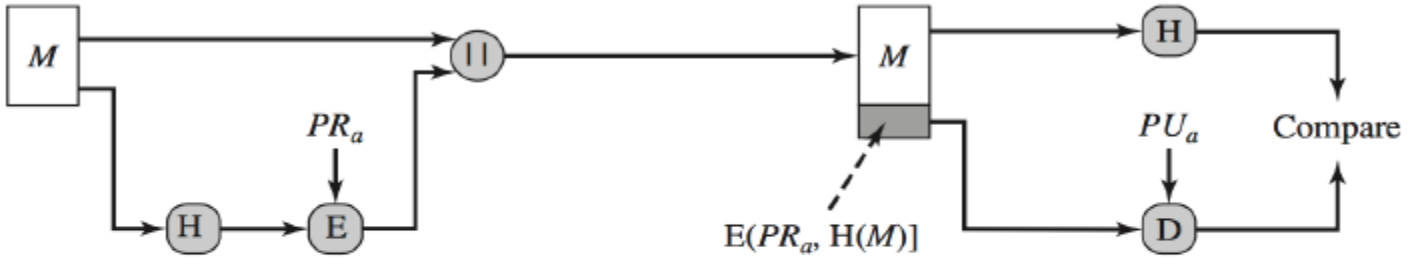
Confidentiality & Integrity



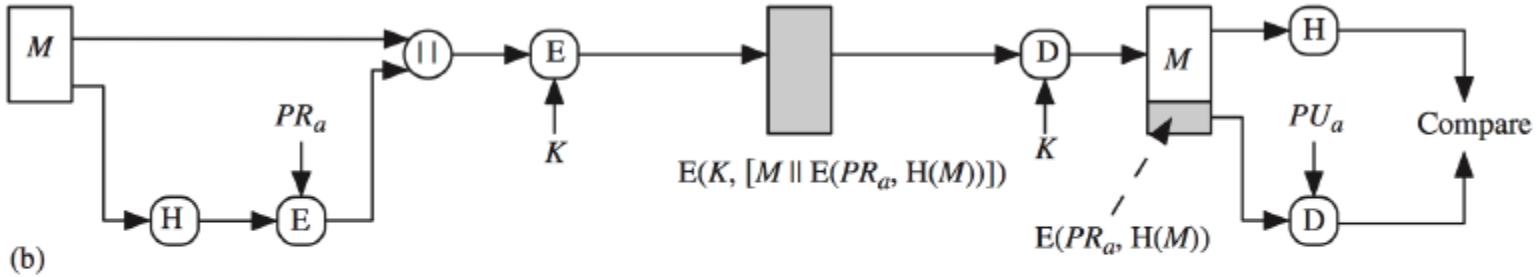
Integrity & Authenticity



Confidentiality & Integrity & Authenticity



Integrity & Authenticity & Accountability



Confidentiality & Integrity & Authenticity & Accountability

Authentication

Message Digests (MD)

- Standards:

  MD5 : 128 bit hashing algorithm by Ron Rivest of RSA

  - Broken since 2004 by Xiaoyun Wang
  - Collisions can be constructed in seconds on a laptop

  SHA & SHA-1 : 160 bit hashing algorithm developed by NIST

  - Considered insecure
  - Practically broken since 2005 by Xiaoyun Wang

Do not use them in your security products!

Authentication

Message Digests (MD)

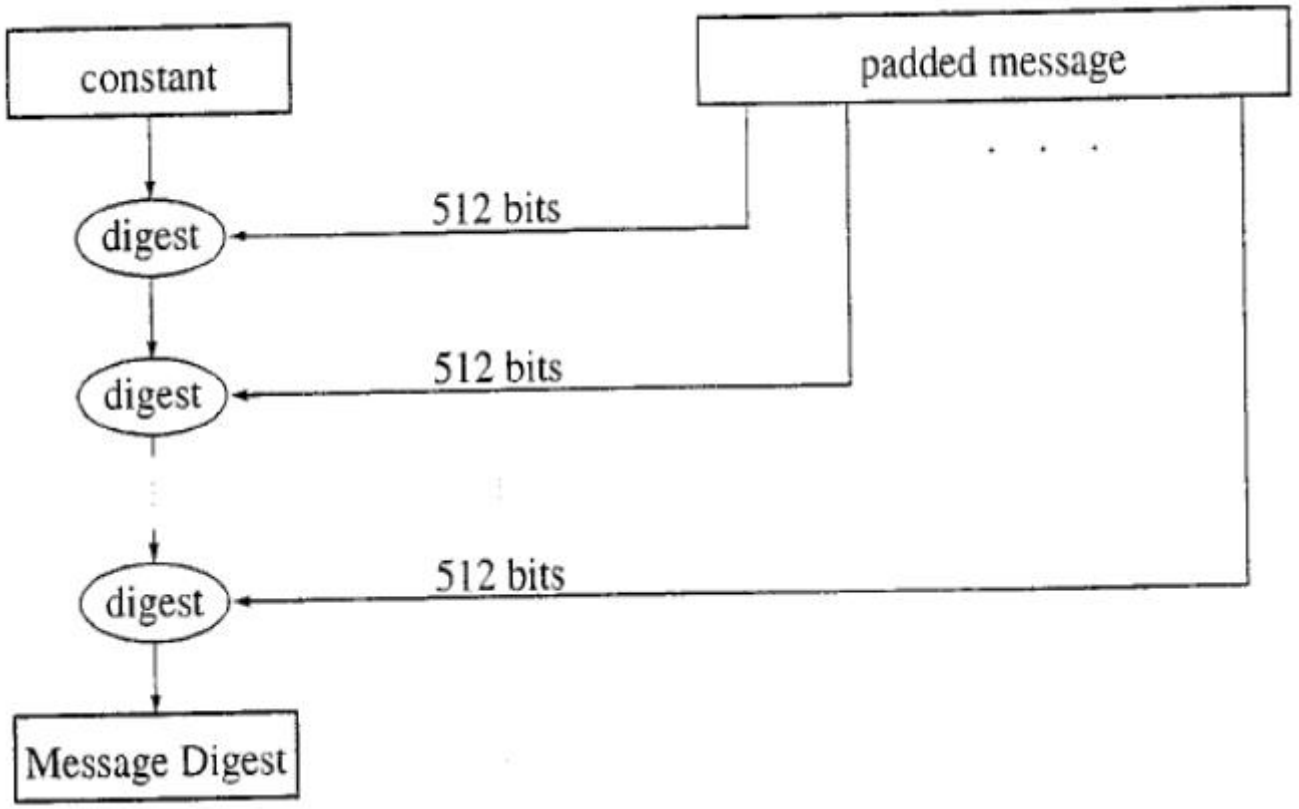
- SHA-2 family: 224, 256, 384 or 512 bits

- SHA-3 family output can be arbitrary size
  - NIST started a competition for SHA-3 in 2007.
  - NIST's original concern was that SHA-2 would soon be broken, although in fact SHA-2 is still fine.
- SHA-3 (Secure Hash Algorithm 3) is the latest member of the Secure Hash Algorithm family of standards, released by NIST on August 5, 2015.

MD5: Message Digest Version 5

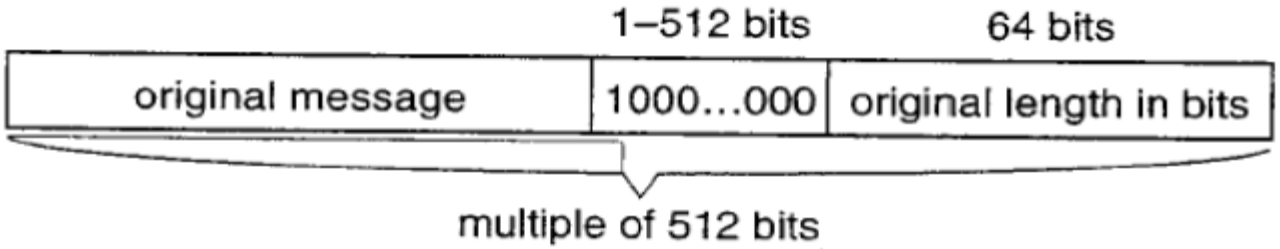
- Author: R. Rivest, 1992
- 128-bit hash
  - based on earlier, weaker MD4 (1990)
- **Collision resistance (B-day attack resistance)**
  - only 64-bit
- Output size not long enough today (due to various attacks)

MD5 Overview



Similar for MD4/MD5/SHA-1

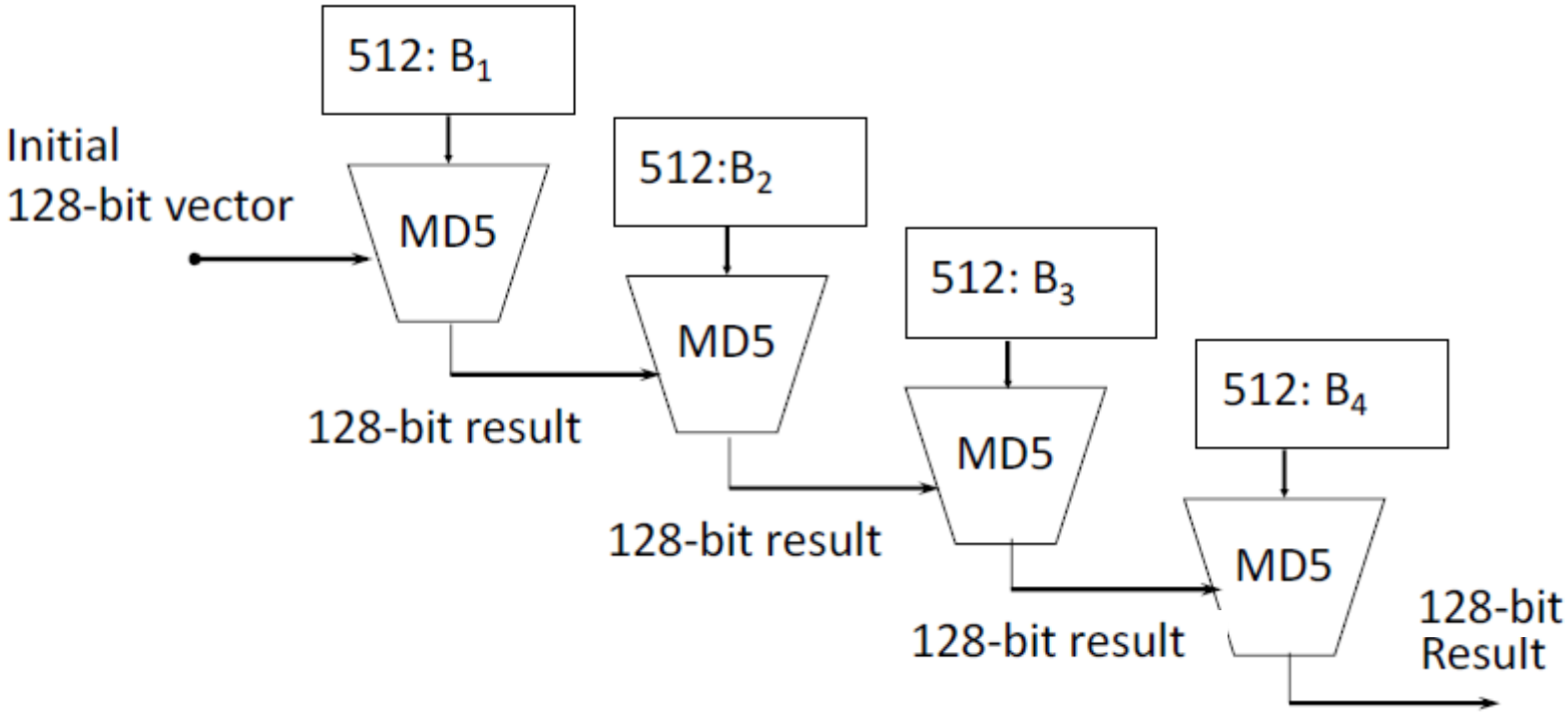
MD5: Padding



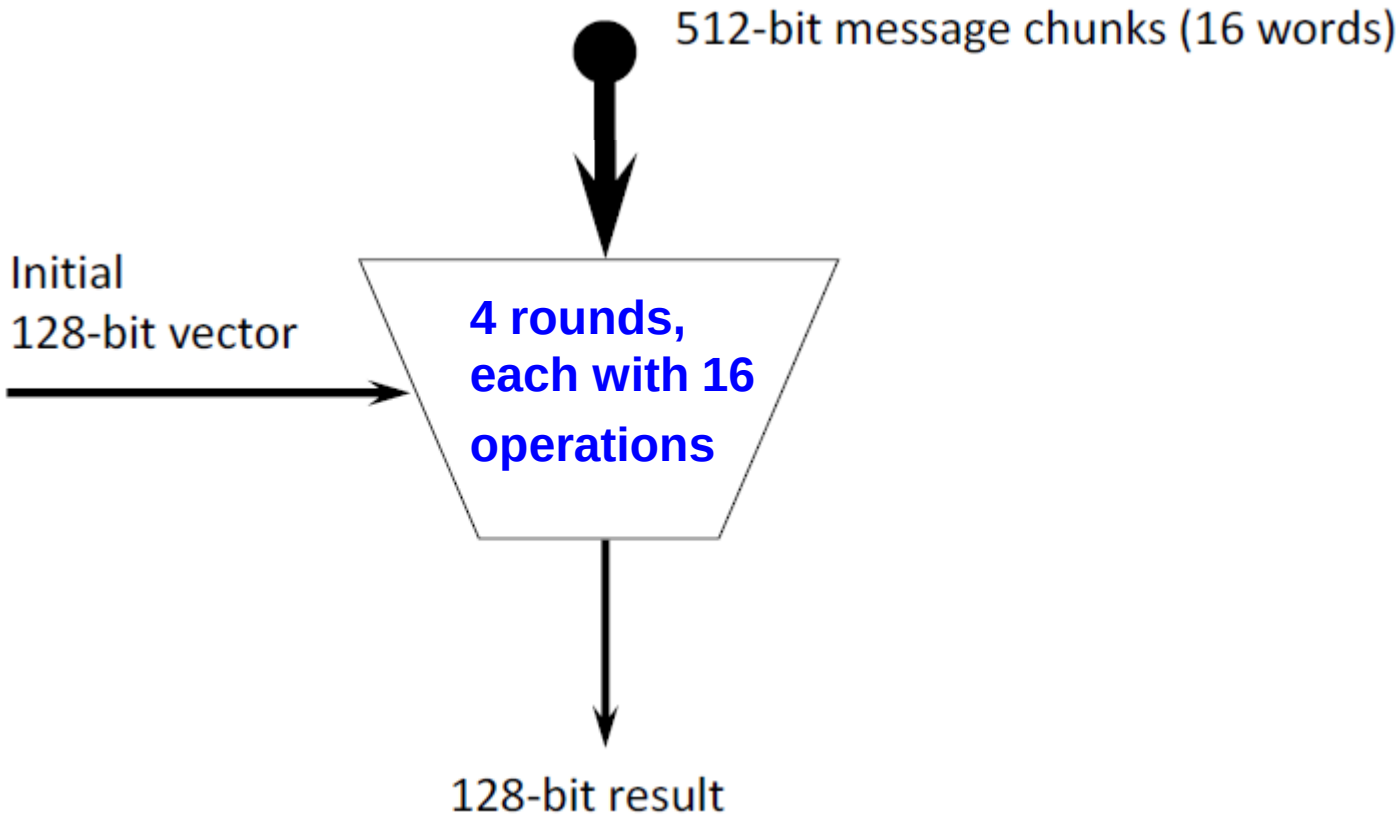
- Given original message M, add padding bits "100…" such that resulting length is 64 bits less than a multiple of 512 bits.
- Append *original length in bits* to the padded message
- Final message chopped into 512-bit blocks

MD5: Blocks

- As many stages as the number of 512-bit blocks in the final padded message



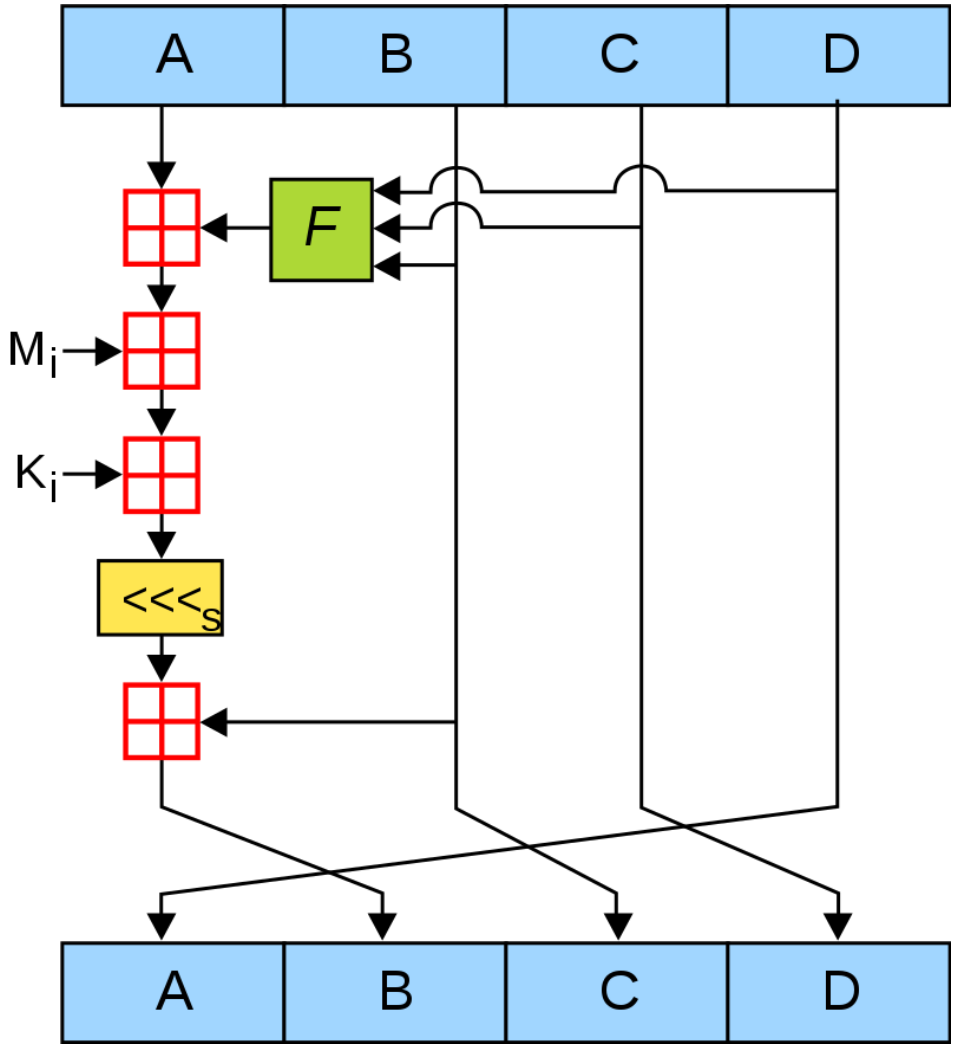
MD5: Single Block Process



MD5: Single Block Process

- Every message block contains 16 32-bit words: $m_0 | m_1 | m_2 \ldots | m_{15}$
- Digest MD_0 initialized to: A=01234567,B=89abcdef,C=fedcba98, D=76543210
- Every stage consists of four *rounds*, each involves 16 basic *operations*..

MD5: One Operation (4 rounds, each with 16 operations)



For 4 rounds:

$F(x,y,z) = (x \wedge y) \vee (\sim x \wedge z)$
$G(x,y,z) = (x \wedge z) \vee (y \wedge \sim z)$
$H(x,y,z) = x \oplus y \oplus z$
$I(x,y,z) = y \oplus (x \wedge \sim z)$

x↵y: x left rotate y bits

K_i = abs(sin(i + 1)) * 2^{32}

M_i denotes a 32-bit block of the message input

Secure Hash Algorithm (SHA)

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174

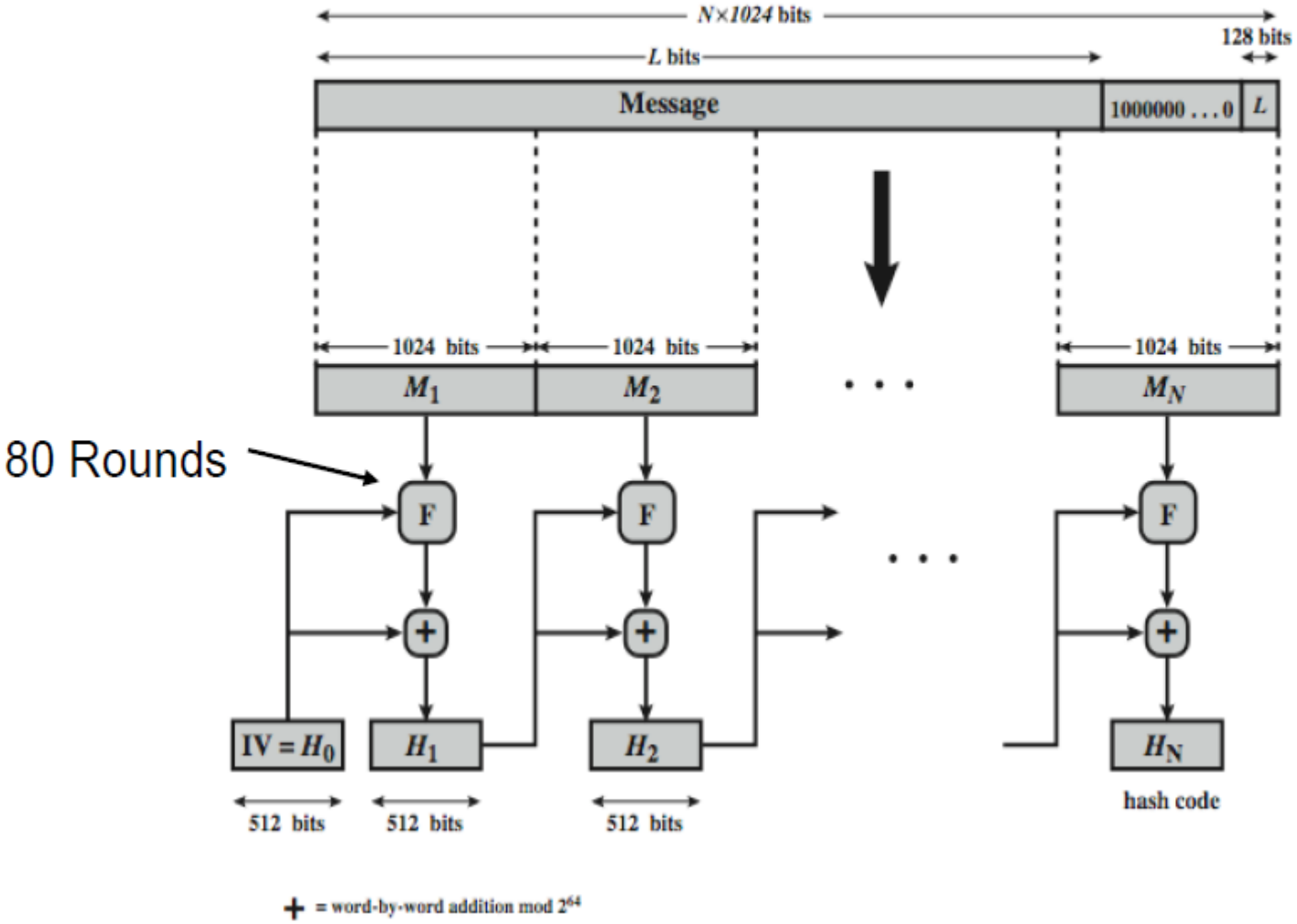
- based on design of MD4 with key differences
- produces 160-bit hash values
- 2005 results on security of SHA-1 raised concerns on its use in future applications

Revised Secure Hash Standard

- NIST issued revision FIPS 180-2 in 2002
- adds 4 additional versions of SHA
  - SHA-224, SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

SHA-512 Overview

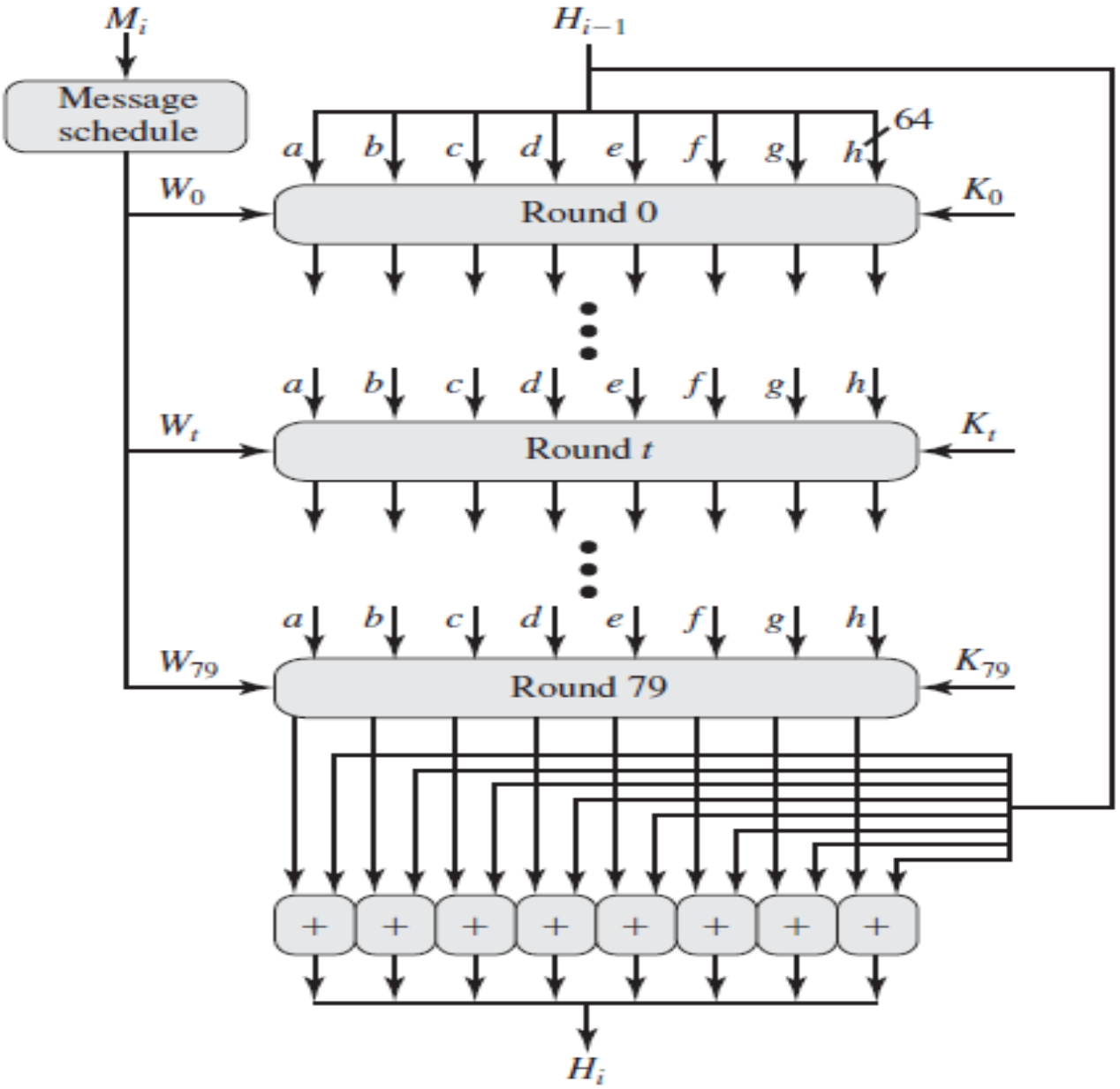
- 1. Append padding bits
- 2. Append length



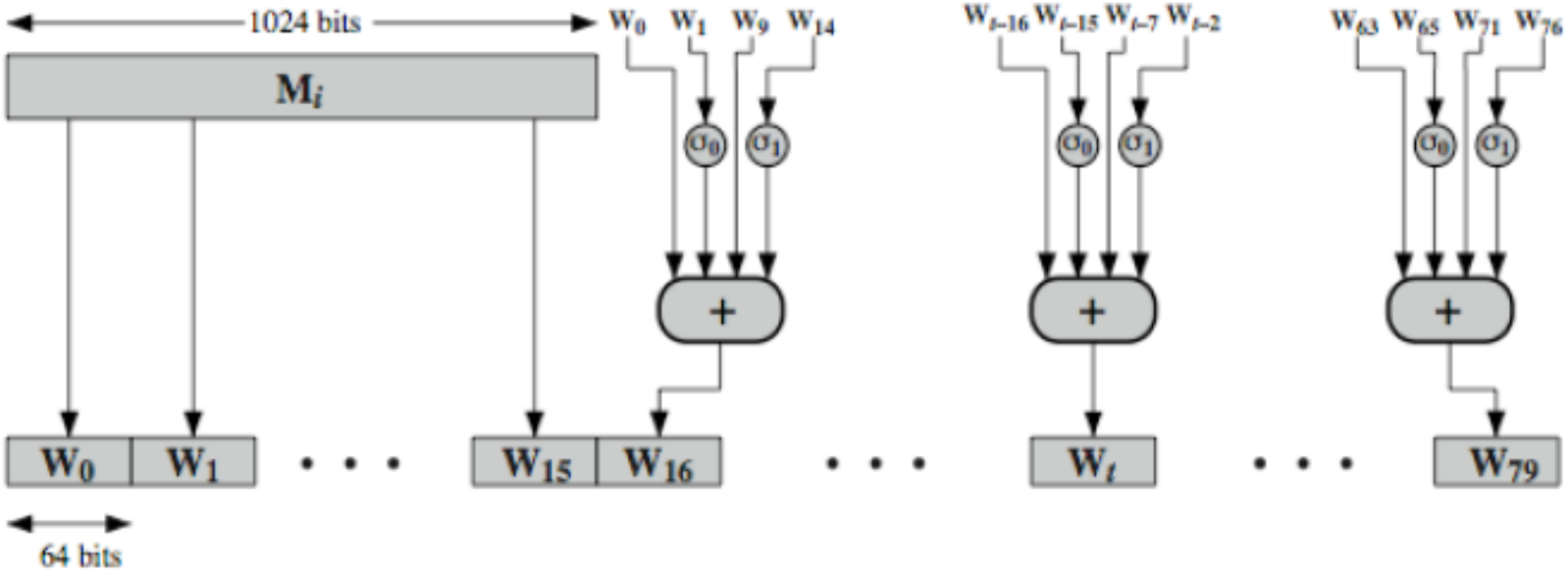
SHA-512 Compression Function

- heart of the algorithm
- processing message in 1024-bit blocks
- consists of 80 rounds
  - updating a 512-bit buffer
  - using a 64-bit value w_t derived from the current message block
  - and a round constant based on cube root of first 80 prime numbers

SHA-512 Compression Function

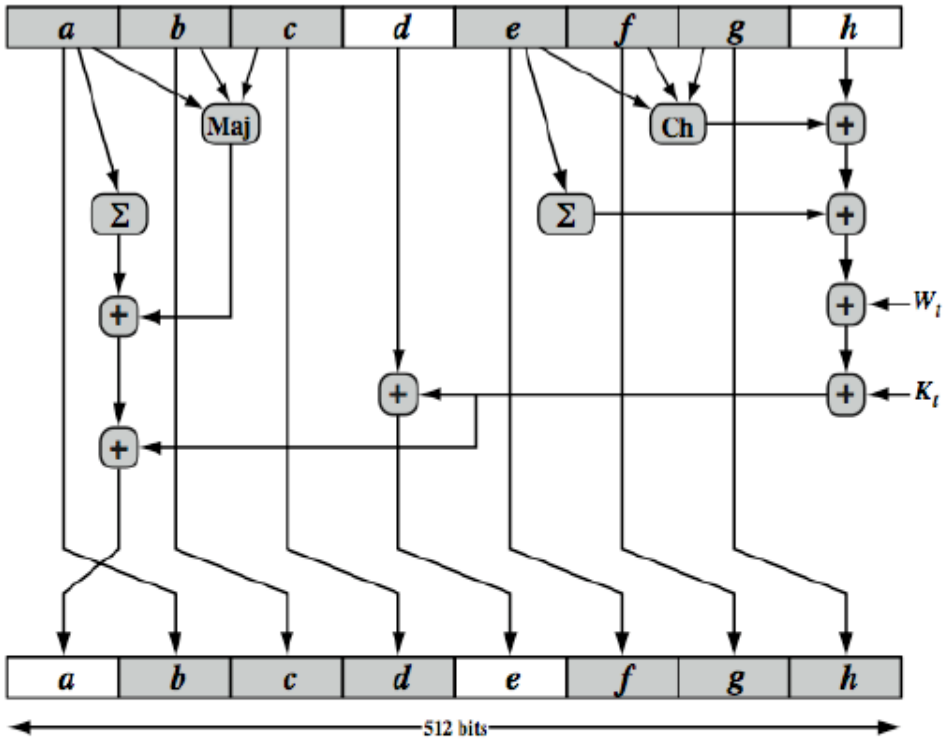


SHA-512 Creation of 80 Words



- $W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}$
- $\sigma_0(x) = ROTR^1(x) + ROTR^8(x) + SHR^7(x)$
- $\sigma_1(x) = ROTR^{19}(x) + ROTR^{61}(x) + SHR^6(x)$
- $ROTR^n(x)$=rotate right by n bits
- $SHR^n(x)$=Left shift n bits with padding by 0's on the right
- + = Addition modulo 2^{64}

SHA-512 Round Function



Input sequence word

Constant (see table)

- Conditional fn Ch(e,f,g): if e then f else g
  = (e AND f) ⊕(Not e and g)
- Majority Fn Maj(a, b, c): True if 2 of 3 args are true
  =(a AND b) ⊕(a AND c) ⊕(b AND c)

SHA-512 Round Function

$\text{Maj}(a, b, c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$
the function is true only of the majority (two or three) of the arguments are true

$\left(\sum_0^{512} a\right) = \text{ROTR}^{28}(a) \oplus \text{ROTR}^{34}(a) \oplus \text{ROTR}^{39}(a)$

$\left(\sum_1^{512} e\right) = \text{ROTR}^{14}(e) \oplus \text{ROTR}^{18}(e) \oplus \text{ROTR}^{41}(e)$

$\text{ROTR}^n(x) =$ circular right shift (rotation) of the 64-bit argument x by n bits

$W_t =$ a 64-bit word derived from the current 512-bit input block

$K_t =$ a 64-bit additive constant

$+ =$ addition modulo 2^{64}

The padded message consists blocks $M_1, M_2, \ldots, M_N$. Each message block M_i consists of 16 64-bit words $M_{i,0}, M_{i,1}, \ldots, M_{i,15}$. All addition is performed modulo 2^{64}.

$H_{0,0} = 6A09E667F3BCC908$ $H_{0,4} = 510E527FADE682D1$
$H_{0,1} = BB67AE8584CAA73B$ $H_{0,5} = 9B05688C2B3E6C1F$
$H_{0,2} = 3C6EF372FE94F82B$ $H_{0,6} = 1F83D9ABFB41BD6B$
$H_{0,3} = A54FF53A5F1D36F1$ $H_{0,7} = 5BE0CDI9137E2179$

for $i = 1$ **to** N

1. Prepare the message schedule W
   for $t = 0$ **to** 15
   $W_t = M_{i,t}$
   for t = 16 **to** 79
   $W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$
2. Initialize the working variables
   $a = H_{i-1,0}$ $e = H_{i-1,4}$
   $b = H_{i-1,1}$ $f = H_{i-1,5}$
   $c = H_{i-1,2}$ $g = H_{i-1,6}$
   $d = H_{i-1,3}$ $h = H_{i-1,7}$
3. Perform the main hash computation
   for $t = 0$ **to** 79
   $T_1 = h + \mathrm{Ch}(e,f,g) + \left(\sum_1^{512} e\right) + W_t + K_t$
   $T_2 = \left(\sum_0^{512} a\right) + \mathrm{Maj}(a,b,c)$
   $h = g$
   $g = f$
   $f = e$
   $e = d + T_1$
   $d = c$
   $c = b$
   $b = a$
   $a = T_1 + T_2$
4. Compute the inermediate hash value
   $H_{i,0} = a + H_{i-1,0}$ $H_{i,4} = a + H_{i-1,4}$
   $H_{i,1} = a + H_{i-1,1}$ $H_{i,5} = a + H_{i-1,5}$
   $H_{i,2} = a + H_{i-1,2}$ $H_{i,6} = a + H_{i-1,6}$
   $H_{i,3} = a + H_{i-1,3}$ $H_{i,7} = a + H_{i-1,7}$

return $\{H_{N,0} \| H_{N,1} \| H_{N,2} \| H_{N,3} \| H_{N,4} \| H_{N,5} \| H_{N,6} \| H_{N,7}\}$

SHA-256

Pad the message M and break into N 512-bit (sixteen 32 bits word) blocks
Initialize *H* (eight 32 bits word)
for i = 1 to N {
    Initialize intermediate variables a,b,c,d,e,f,g,h
    Prepare W_t (t=0…63)
    Initialize constants K_t (t=0…63)
    Pass through t=0…63 rounds
    update *H*
}
Output *H*

SHA-256: Initialize *H*

Initialization of chaining variable Eight 32-bit words

$$H_0^{(0)} = \texttt{6a09e667}$$
$$H_1^{(0)} = \texttt{bb67ae85}$$
$$H_2^{(0)} = \texttt{3c6ef372}$$
$$H_3^{(0)} = \texttt{a54ff53a}$$
$$H_4^{(0)} = \texttt{510e527f}$$
$$H_5^{(0)} = \texttt{9b05688c}$$
$$H_6^{(0)} = \texttt{1f83d9ab}$$
$$H_7^{(0)} = \texttt{5be0cd19}.$$

SHA-256: Initialize intermediate variables a,b,c,d,e,f,g,h

$$a = H_0^{(i-1)}$$

$$b = H_1^{(i-1)}$$

$$c = H_2^{(i-1)}$$

$$d = H_3^{(i-1)}$$

$$e = H_4^{(i-1)}$$

$$f = H_5^{(i-1)}$$

$$g = H_6^{(i-1)}$$

$$h = H_7^{(i-1)}$$

SHA-256: Prepare W_t (t=0…63)

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

SHA-256:
Initialize constants K_t (t=0…63)

Constants used in the rounds: 64 32-bit constants K_0, K_1 .. K_{63}

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2.
```

SHA-256: Pass through t=0…63 rounds

For t=0 to 63:

{

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t$$

$$T_2 = \sum_0^{\{256\}}(a) + Maj(a,b,c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = T_1 + T_2$$

}

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$

$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$

$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$

$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

SHA-256: update H and Output H

$$
\begin{aligned}
H_0^{(i)} &= a + H_0^{(i-1)} \\
H_1^{(i)} &= b + H_1^{(i-1)} \\
H_2^{(i)} &= c + H_2^{(i-1)} \\
H_3^{(i)} &= d + H_3^{(i-1)} \\
H_4^{(i)} &= e + H_4^{(i-1)} \\
H_5^{(i)} &= f + H_5^{(i-1)} \\
H_6^{(i)} &= g + H_6^{(i-1)} \\
H_7^{(i)} &= h + H_7^{(i-1)}
\end{aligned}
$$

After all N blocks processed, the output :

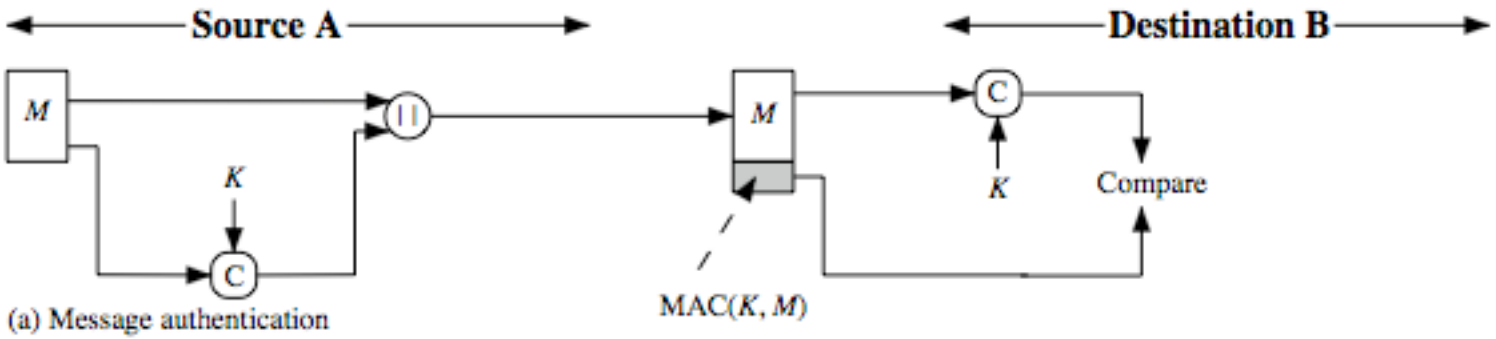
$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)} .$$

SHA

	Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Operations	Collisions found
SHA-0	**160**	**160**	**512**	**$2^{64} - 1$**	**32**	**80**	**+, and, or, xor, rot**	**Yes**
SHA-1	**160**	**160**	**512**	**$2^{64} - 1$**	**32**	**80**	**+, and, or, xor, rot**	**None (2^{52} attack)**
SHA-2	**256/224**	**256**	**512**	**$2^{64} - 1$**	**32**	**64**	**+, and, or, xor, shr, rot**	**None**
	512/384	**512**	**1024**	**$2^{128} - 1$**	**64**	**80**	**+, and, or, xor, shr, rot**	**None**

Message Authentication Code

- A small fixed-sized block of data
  - generated from message + secret key
  - MAC = C(K,M)
  - appended to message when sent



MAC Properties

- a MAC is a cryptographic checksum

  $MAC = C_K(M)$

  – condenses a variable-length message M
  – using a secret key K
  – to a fixed-sized authenticator
- is a many-to-one function
  – potentially many messages have same MAC
  – but finding these needs to be very difficult

Requirements for MACs

- taking into account the types of attacks
- need the MAC to satisfy the following:
  1. knowing a message and MAC, is infeasible to find another message with same MAC
  2. MACs should be uniformly distributed
  3. MAC should depend equally on all bits of the message

HMAC

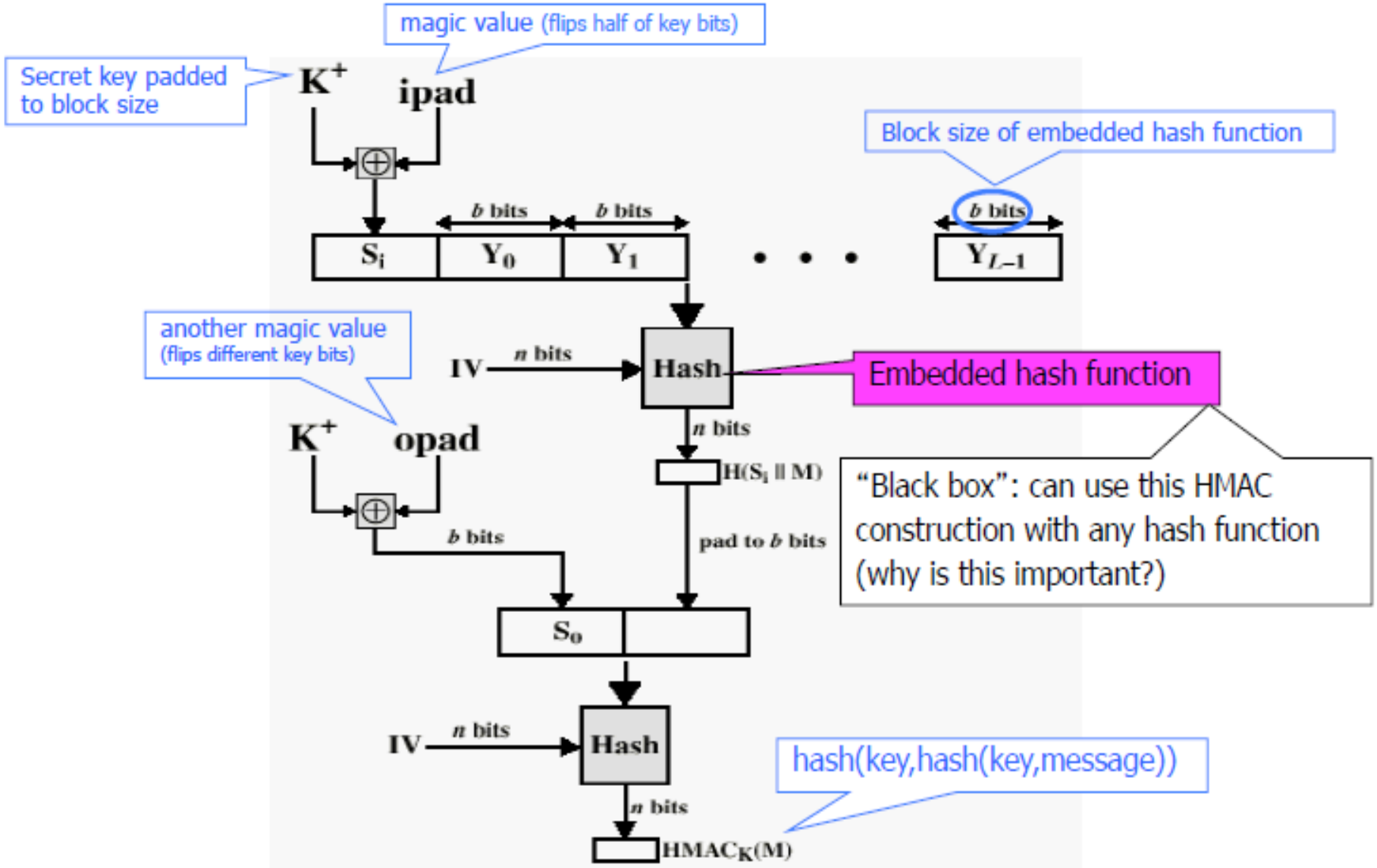
- Construct MAC from a cryptographic hash function
  - Invented by Bellare, Canetti, and Krawczyk (1996)
  - Used in SSL/TLS, mandatory for IPsec
- Why not encryption?
  - Hashing is faster than encryption
  - Library code for hash functions widely available
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption

Structure of HMAC

ipad: 00110110

Opad:01011100

[Repeated b/8 times]



HMAC Security

- proved security of HMAC relates to that of the underlying hash algorithm
- attacking HMAC requires either:
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)
- choose hash function used based on speed verses security constraints