

EE 4215

Security Technology

Lecture 6: Hashing, Digital Signature & PKI
2018/19 Semester B

Course Structure

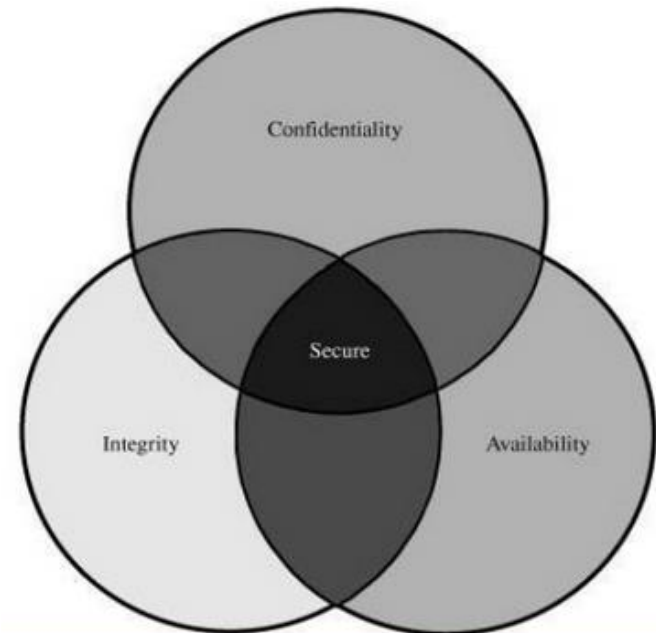
- Week 1: Security Fundamental
- Week 2: Classical Cryptography
- Week 3: Number Theory &
- Week 4: DES
- Week 5: Secret-key Cryptography
- Week 6: Public-key Cryptography
- Week 7: Quiz 1
- Week 8: Hashing, Digital Signature, PKI
- Week 9: SSL/TLS, Web Security
- Week 10: Wi-Fi & P2P Security
- Week 11: Cloud Computing Security
- Week 11: Quiz 2
- Week 12: Smart-card Security
- Week 13: Blockchain Security, Secure-code, Trusted Hardware, Review

Lecture 6: Hashing and Digital Signature

- Hashing
 - SHA
 - Collision
 - Message Authentication
 - Digital Signature

CIA Model

- Some people consider “security” from the perspective of confidentiality, integrity and availability
 - Encryption (symmetric key and asymmetric key cryptosystems) provides tools for confidentiality
- How about integrity?

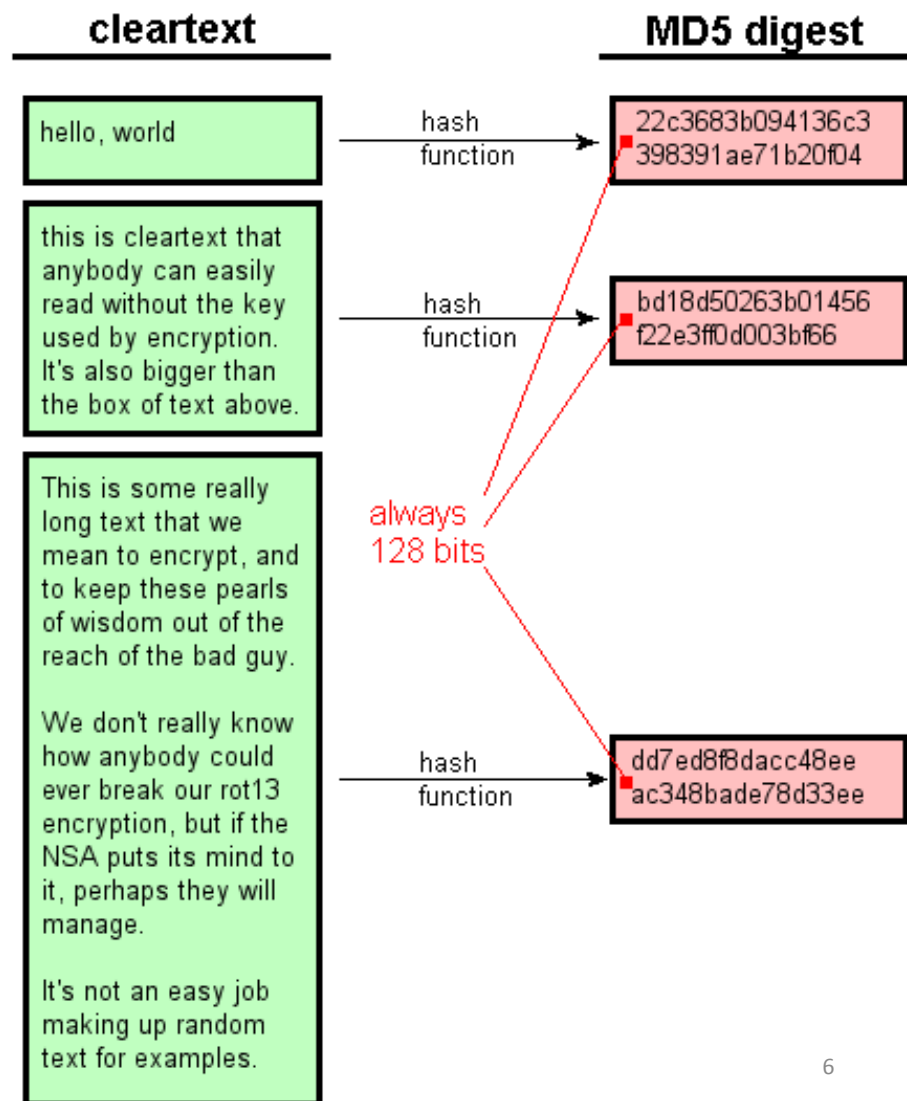


Why Integrity?

- Computer files could easily be modified
 - Anyone with Microsoft Office could modify your Office documents
 - Even PDF file could be modified
- How could we verify that a document (e.g. with 100 pages) was not modified unexpectedly?
 - Eye-ball checking could be error prone and time consuming
 - Use data integrity tool (e.g. hashing)

Characteristics of Hashing

- Hash value serves as a fingerprint for a file
 - Under a given algorithm, the **length of the hash value is constant** (but different hash algorithms might have different lengths)
- No matter whether the file is large or small



Hash Algorithms

- Common Hash Algorithms
 - MD5 (128 bits)
 - RIPEMD-160 (160 bits)
 - SHA-1 (160 bits)
 - SHA-256, SHA-384, and SHA-512 (longer versions of SHA-1, with slightly different designs)
 - ...
- In Linux, tools such as md5sum, sha1sum, etc are available for use
- In Windows, can use the tool HashCalc
 - <http://www.slavasoft.com/hashcalc/>
- Online hash generator
 - <http://quickhash.com/>

Some Practical Uses

- Some vendors might give hash values for the files that can be downloaded

32-bit Platforms

The following definition sets are suitable for 32-bit operating systems only.

File Name	Creation Date	Release Date	File Size	MD5 all
20131009-004-v5i32.exe FTP	10/09/2013	10/09/2013	221.27 MB	16584B2D0EA3B7BEA1D4B8F82118C840

Supports the following versions of Symantec antivirus software:

- Norton AntiVirus / Norton Internet Security (later than 2012)
- Norton AntiVirus / Norton Internet Security 2008/2009/2010/2011/2012

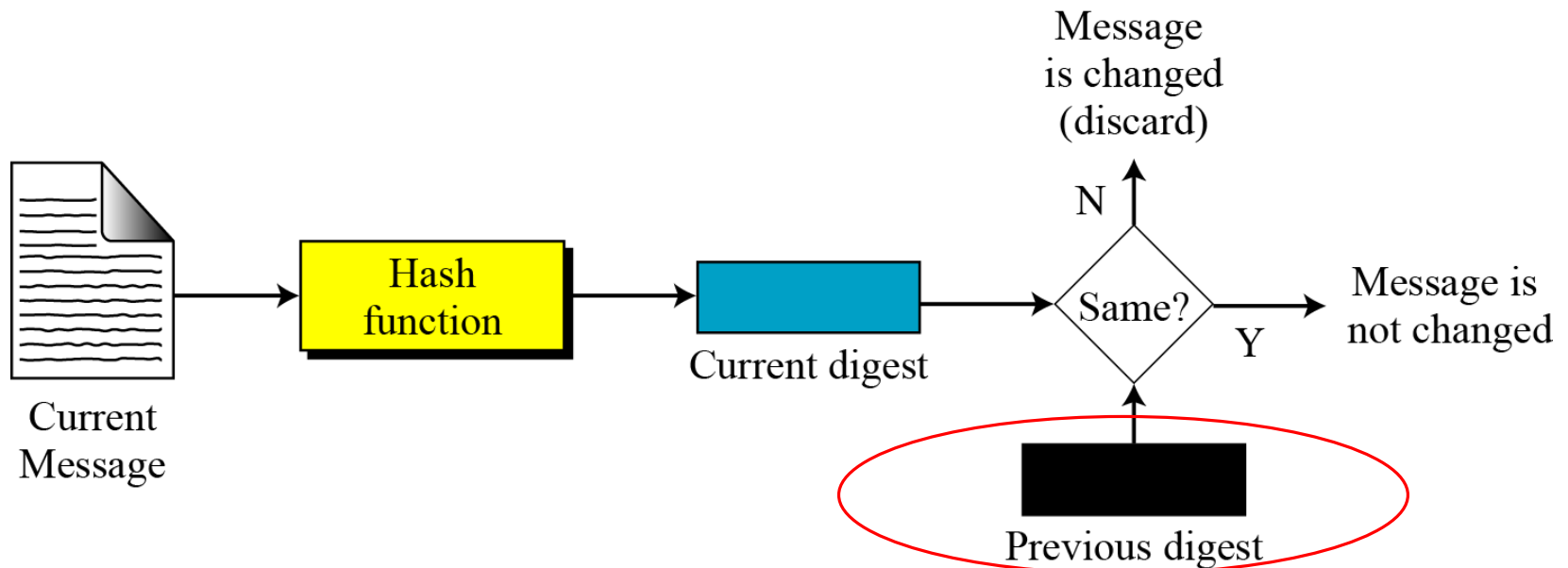
Some Practical Uses

- Some developers might store the hash value of a password in the database instead of storing the plaintext value
 - Drawbacks
 - If user A and user B chose the same password, the hash value will be identical and the administrators can infer that the passwords of user A and user B are the same
 - Solution: use “salt”
- c.f. what are the problems of using encryption?

Checking Integrity

- By comparison

- File verification: calculate the hash of your download file and compare the value with vendor's given hash value
- Password checking: comparison the hash of your input password with the value stored in the database



Requirements of Hash One-way Functions

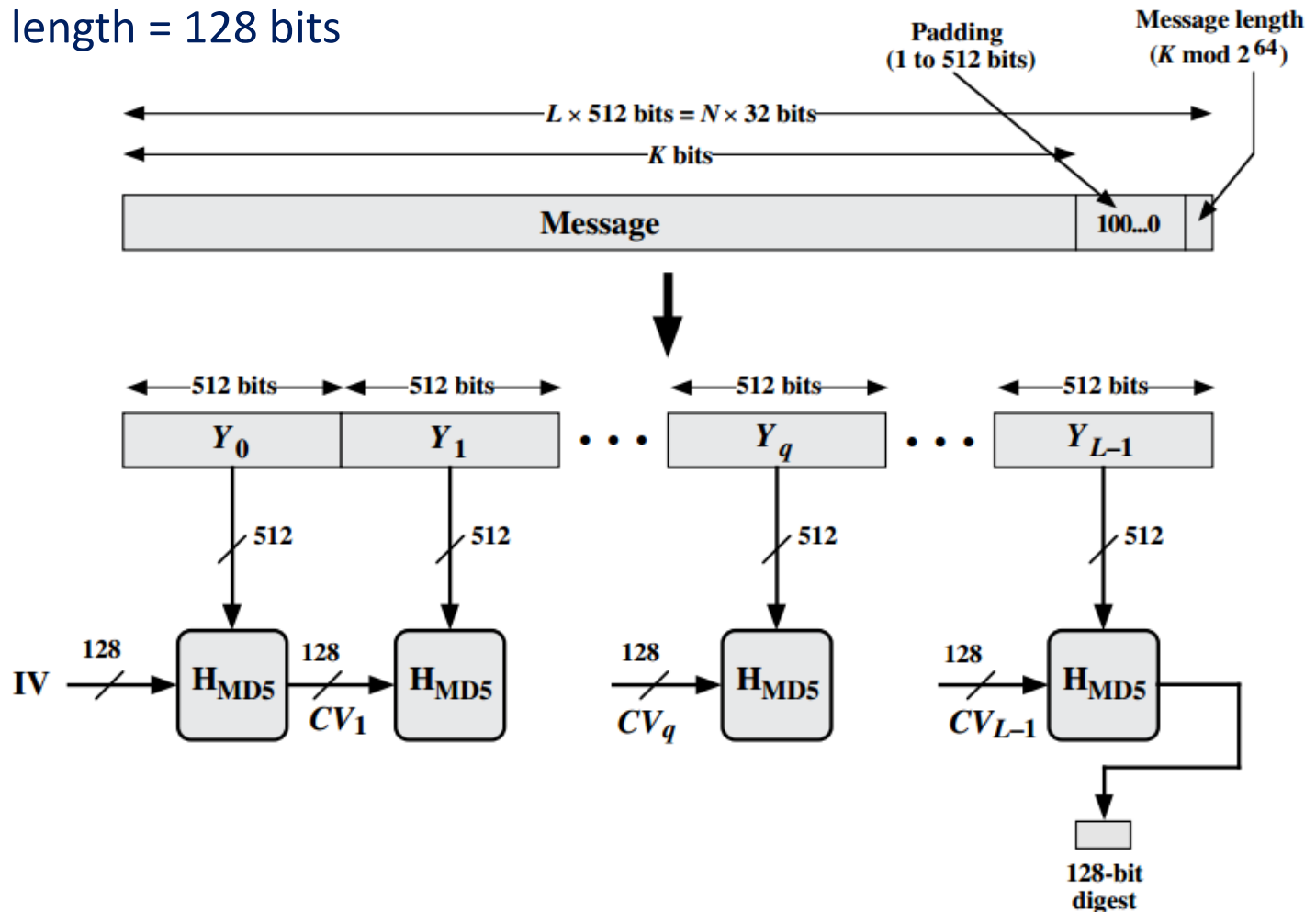
Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given x , making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value h , it is computationally infeasible to find y such that $H(y) = h$.
Second preimage resistant (weak collision resistant)	For any given block x , it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$.
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

Hash Functions

V · T · E		Cryptographic hash functions & message authentication codes
		List · Comparison · Known attacks
Common functions		MD5 · SHA-1 · SHA-2 · SHA-3 · BLAKE2
SHA-3 finalists		BLAKE · Grøstl · JH · Skein · Keccak (winner)
Other functions		CubeHash · ECOH · FSB · GOST · HAS-160 · HAVAL · Kupyua · LM hash · MD2 · MD4 · MD6 · MDC-2 · N-Hash · RIPEMD · RadioGatún · WHIRLPOOL
Key derivation functions		bcrypt · crypt · PBKDF2 · scrypt · Argon2 · Lyra2
MAC functions		DAA · CBC-MAC · HMAC · OMAC/CMAC · PMAC · VMAC · UMAC · Poly1305 · SipHash
Authenticated encryption modes		CCM · CWC · EAX · GCM · IAPM · OCB
Attacks		Collision attack · Preimage attack · Birthday attack · Brute-force attack · Rainbow table · Side-channel attack · Length extension attack
Design		Avalanche effect · Hash collision · Merkle–Damgård construction · Sponge function · HAIFA construction · Unique Block Iteration
Standardization		CRYPTREC · NESSIE · NIST hash function competition
Utilization		Hash-based cryptography · Key stretching · Merkle tree · Message authentication · Proof of work · Salt

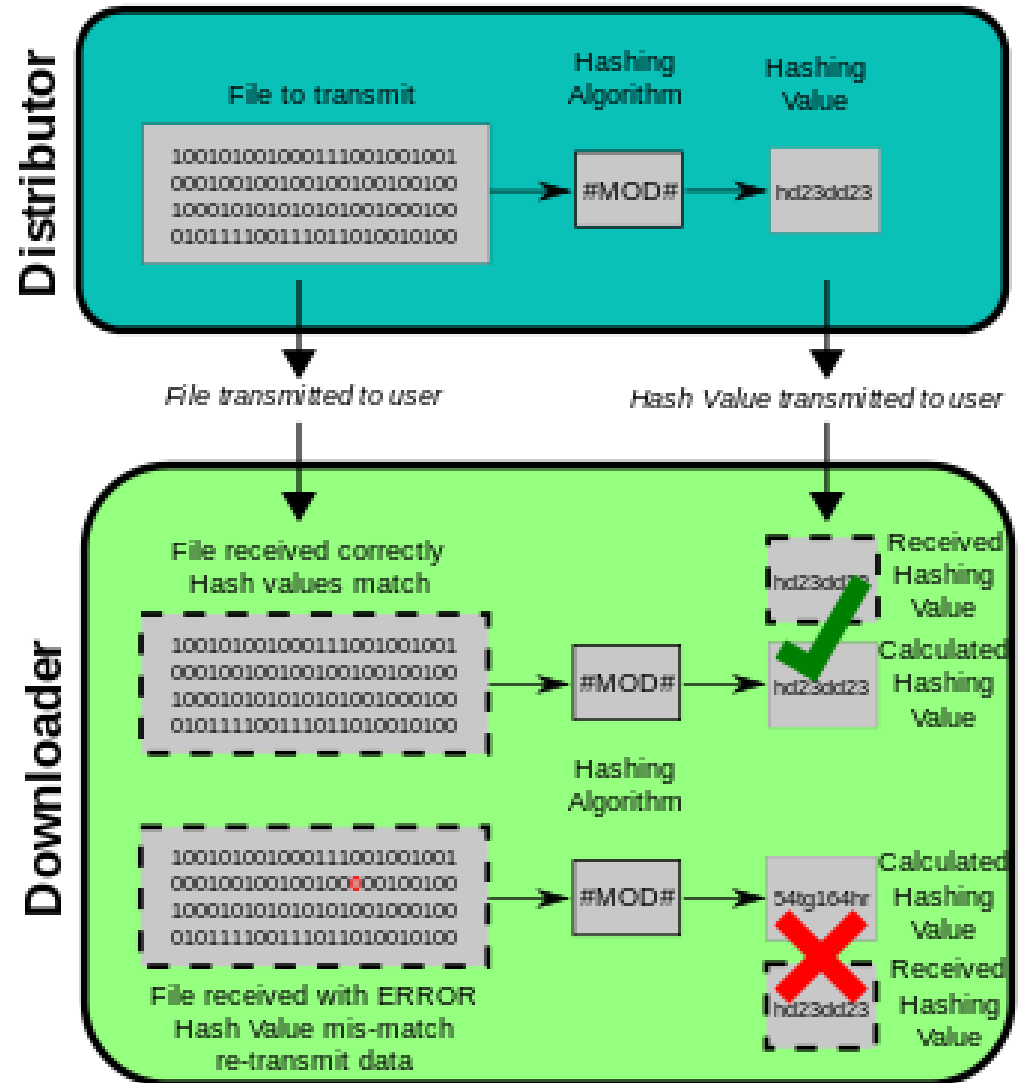
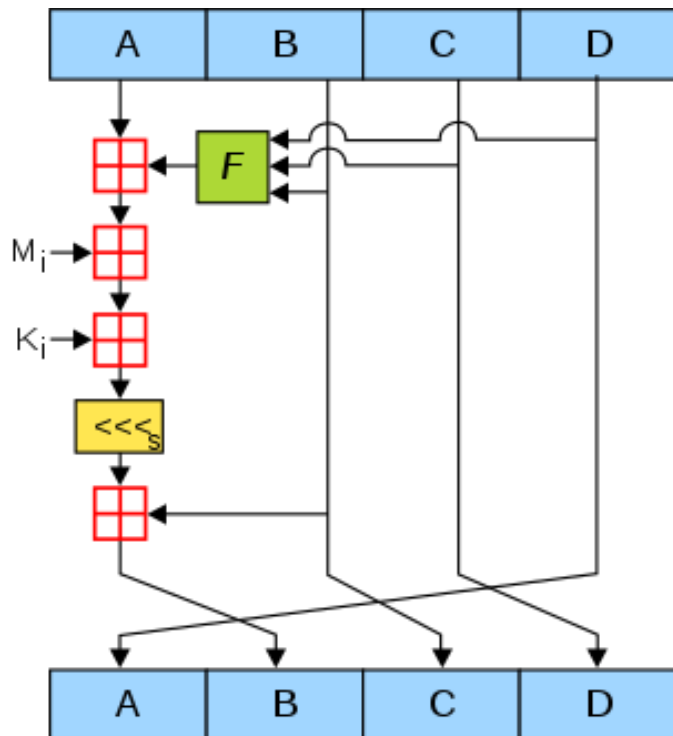
Message Digest 5 (MD5)

- The algorithm is described in RFC 1321 (considered broken)
- Output length = 128 bits



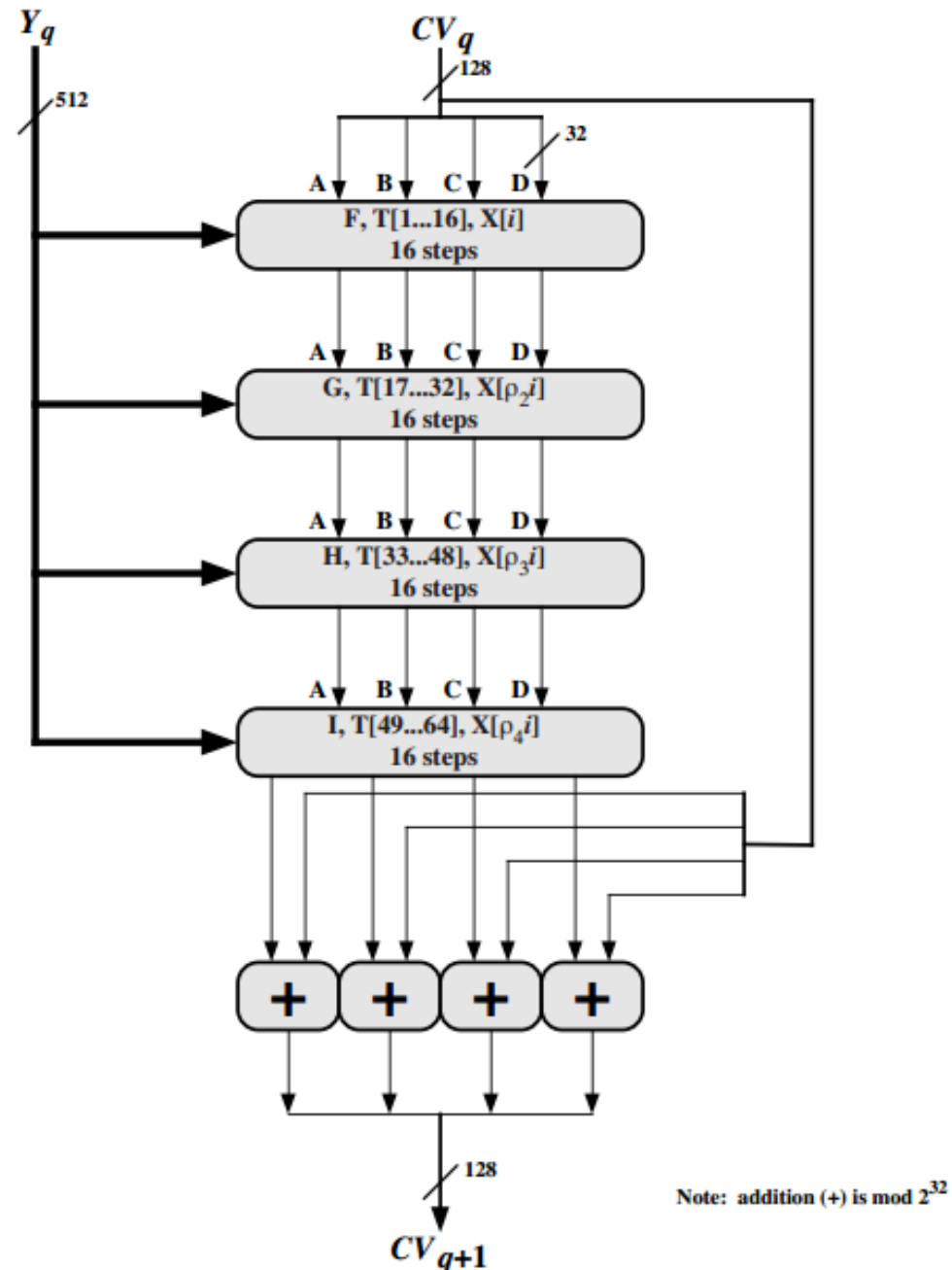
MD5 Example

- Only transmit the hash value
- Smaller size



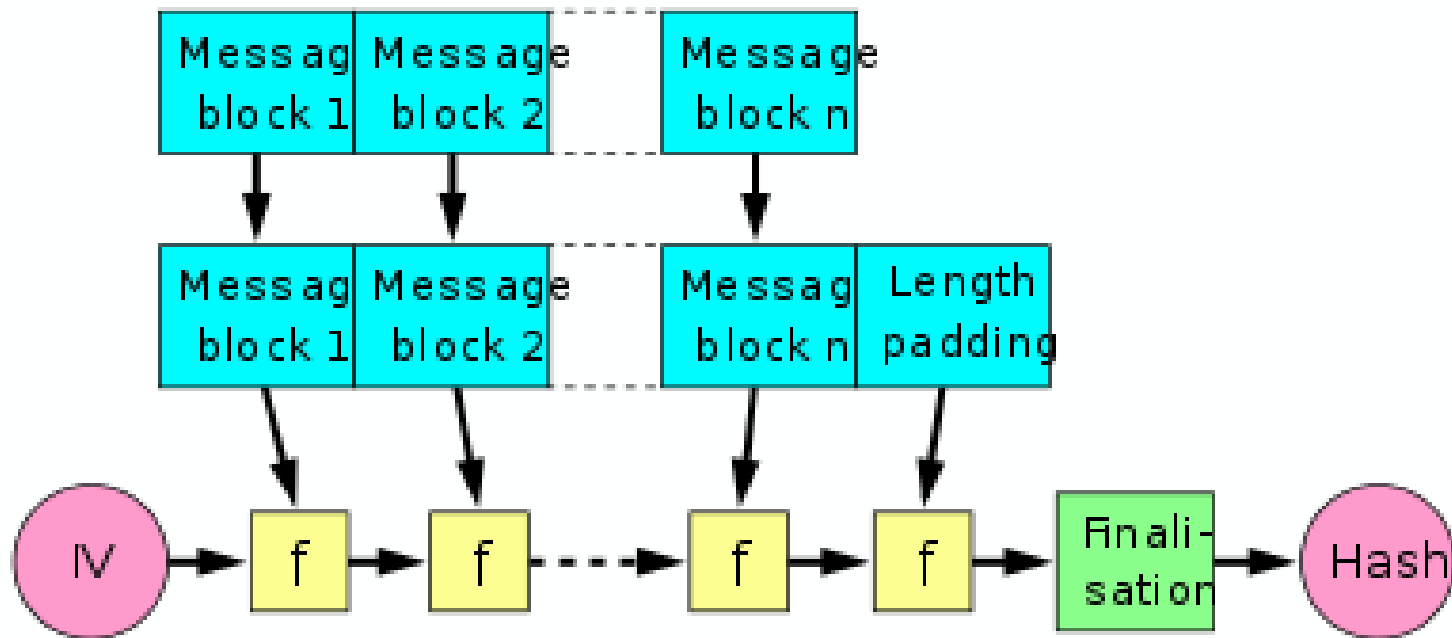
Operation for 512-bit Blocks

- Initial values:
 - A: 01 23 45 67
 - B: 89 ab cd ef
 - C: fe dc ba 98
 - D: 76 54 32 10
- T is a table constructed from sine function
- 4 functions
 - $F(X,Y,Z) = XY \vee \text{not}(X) Z$
 - $G(X,Y,Z) = XZ \vee Y \text{ not}(Z)$
 - $H(X,Y,Z) = X \text{ xor } Y \text{ xor } Z$
 - $I(X,Y,Z) = Y \text{ xor } (X \vee \text{not}(Z))$



MD - Merkle–Damgård construction in 1979

- a method of building [collision-resistant cryptographic hash functions](#) from collision-resistant [one-way compression functions](#). This construction was used in the design of many popular hash algorithms such as [MD5](#), [SHA1](#) and [SHA2](#).



Source Code

- Pseudo code for the step operation was given in RFC 1321
- Source code in C can be found at
 - <http://people.csail.mit.edu/rivest/Md5.c>

```
/* -- include the following line if the md5.h header file is separate .
/* #include "md5.h" */

/* forward declaration */
static void Transform ();

static unsigned char PADDING[64] = {
    0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

/* F, G and H are basic MD5 functions: selection, majority, parity */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4 */
/* Rotation is separate from addition to prevent recomputation */
#define FF(a, b, c, d, x, s, ac) \
    {(a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
     (a) = ROTATE_LEFT ((a), (s)); \
     (a) += (b); \
    }
```

MD5 source code

Pseudocode [\[edit\]](#)

The MD5 hash is calculated according to this algorithm. All values are in [little-endian](#).

```
//Note: All variables are unsigned 32 bit and wrap modulo 2^32 when calculating
var int[64] s, K
var int i

//s specifies the per-round shift amounts
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 }

//Use binary integer part of the sines of integers (Radians) as constants:
for i from 0 to 63
    K[i] := floor(232 × abs(sin(i + 1)))
end for
// (Or just use the following precomputed table):
K[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xclbdccee }
K[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
K[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
K[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
K[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
K[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
K[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
K[32..35] := { 0xffffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
K[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfb7c0 }
K[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
K[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
K[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
K[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
K[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
K[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

//Initialize variables:
var int a0 := 0x67452301 //A
var int b0 := 0xefcdab89 //B
var int c0 := 0x98badcfe //C
var int d0 := 0x10325476 //D
```

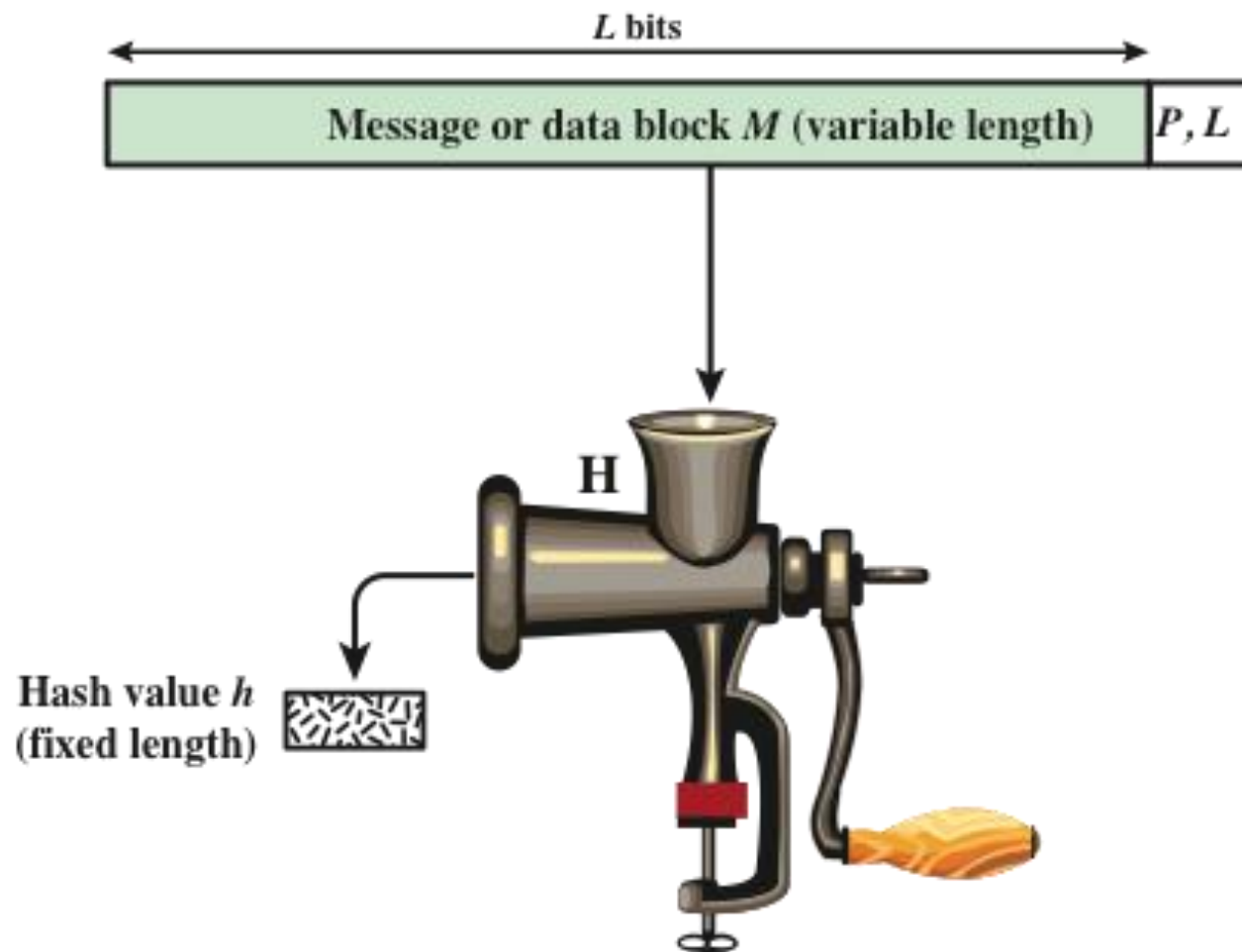
```
//Process the message in successive 512-bit chunks:
for each 512-bit chunk of padded message
    break chunk into sixteen 32-bit words M[j], 0 ≤ j ≤ 15
//Initialize hash value for this chunk:
var int A := a0
var int B := b0
var int C := c0
var int D := d0
//Main loop:
for i from 0 to 63
    var int F, g
    if 0 ≤ i ≤ 15 then
        F := (B and C) or ((not B) and D)
        g := i
    else if 16 ≤ i ≤ 31
        F := (D and B) or ((not D) and C)
        g := (5*i + 1) mod 16
    else if 32 ≤ i ≤ 47
        F := B xor C xor D
        g := (3*i + 5) mod 16
    else if 48 ≤ i ≤ 63
        F := C xor (B or (not D))
        g := (7*i) mod 16
    //Be wary of the below definitions of a,b,c,d
    F := F + A + K[i] + M[g]
    A := D
    D := C
    C := B
    B := B + leftrotate(F, s[i])
end for
//Add this chunk's hash to result so far:
a0 := a0 + A
b0 := b0 + B
c0 := c0 + C
d0 := d0 + D
end for

var char digest[16] := a0 append b0 append c0 append d0 // (Output is in little-endian)

//leftrotate function definition
leftrotate (x, c)
    return (x << c) binary or (x >> (32-c));
```

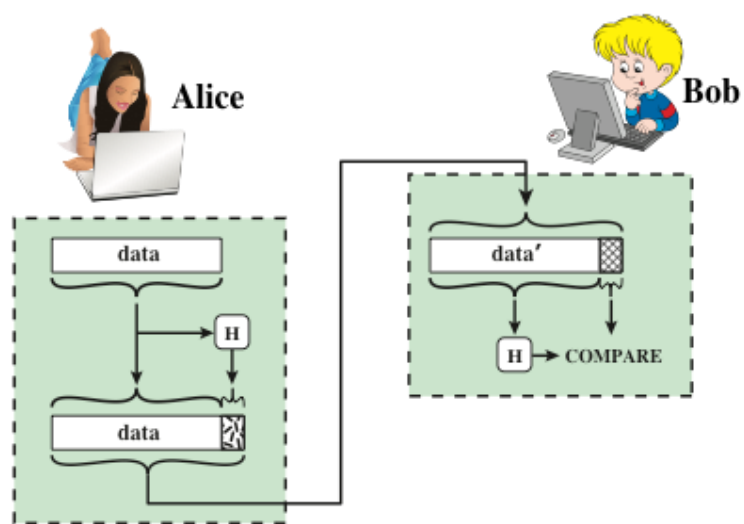
Hash Functions

- A hash function H accepts a variable-length block of data M as input and produces a fixed-size hash value
 - $h = H(M)$
 - Principal object is data integrity
- Cryptographic hash function
 - An algorithm for which it is computationally infeasible to find either:
 - (a) a data object that maps to a pre-specified hash result (the one-way property)
 - (b) two data objects that map to the same hash result (the collision-free property)

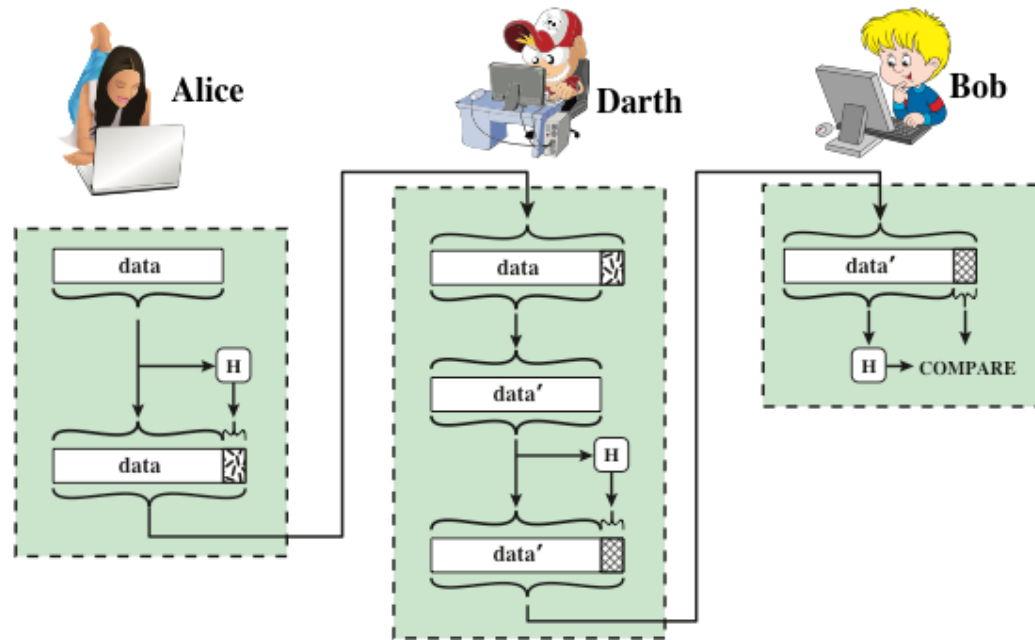


$P, L =$ padding plus length field

Figure 11.1 Cryptographic Hash Function; $h = H(M)$



(a) Use of hash function to check data integrity



(b) Man-in-the-middle attack

Figure 11.2 Attack Against Hash Function

Other Hash Function Uses

Commonly used to create a one-way password file

When a user enters a password, the hash of that password is compared to the stored hash value for verification

This approach to password protection is used by most operating systems

Can be used for intrusion and virus detection

Store $H(F)$ for each file on a system and secure the hash values

One can later determine if a file has been modified by recomputing $H(F)$

An intruder would need to change F without changing $H(F)$

Can be used to construct a pseudorandom function (PRF) or a pseudorandom number generator (PRNG)

A common application for a hash-based PRF is for the generation of symmetric keys

Two Simple Hash Functions

- Consider two simple insecure hash functions that operate using the following general principles:
 - The input is viewed as a sequence of n -bit blocks
 - The input is processed one block at a time in an iterative fashion to produce an n -bit hash function
- Bit-by-bit exclusive-OR (XOR) of every block
 - $C_i = b_{i1} \text{ xor } b_{i2} \text{ xor } \dots \text{ xor } b_{im}$
 - Produces a simple parity for each bit position and is known as a longitudinal redundancy check
 - Reasonably effective for random data as a data integrity check
- Perform a one-bit circular shift on the hash value after each block is processed
 - Has the effect of randomizing the input more completely and overcoming any regularities that appear in the input

Two Simple Hash Functions

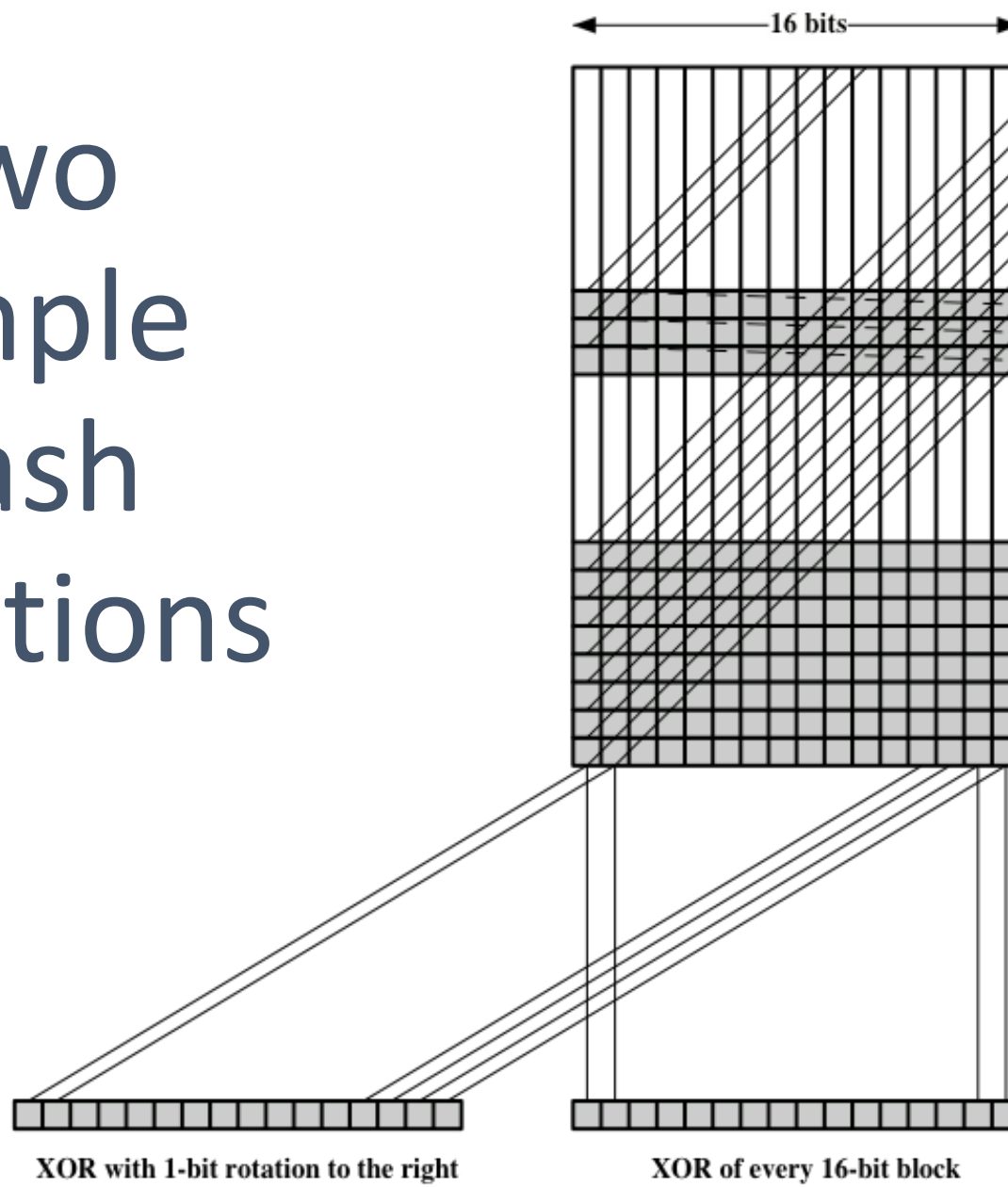


Figure 11.5 Two Simple Hash Functions

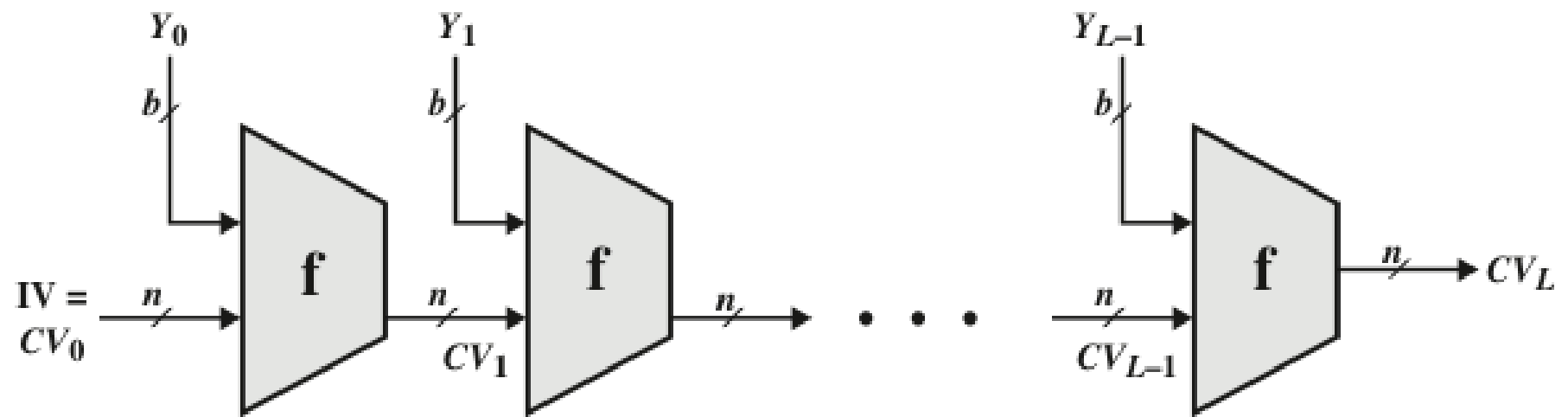
Attacks on Hash Functions

Brute-Force Attacks

- Does not depend on the specific algorithm, only depends on bit length
- In the case of a hash function, attack depends only on the bit length of the hash value
- Method is to pick values at random and try each one until a collision occurs

Cryptanalysis

- An attack based on weaknesses in a particular cryptographic algorithm
- Seek to exploit some property of the algorithm to perform some attack other than an exhaustive search



IV = Initial value
 CV_i = chaining variable
 Y_i = i th input block
 f = compression algorithm

L = number of input blocks
 n = length of hash code
 b = length of input block

Figure 11.8 General Structure of Secure Hash Code

Lecture 6: Hashing and Digital Signature

- Hashing
- SHA
- Collision
- Message Authentication
- Digital Signature

Secure Hash Algorithm (SHA)

- SHA was originally designed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993
- Was revised in 1995 as SHA-1
- Based on the hash function MD4 and its design closely models MD4
- Produces 160-bit hash values
- In 2002 NIST produced a revised version of the standard that defined three new versions of SHA with hash value lengths of 256, 384, and 512
 - Collectively known as SHA-2

SHA family

- A family of cryptographic hash functions published by NIST
 - SHA-1: 160-bit hash function
 - designed by NSA
 - weaknesses were discovered but not yet considered broken
 - no longer approved for most cryptographic uses after 2010 in US government
 - SHA-2
 - Designed by NSA
 - SHA-256 : uses 32-bit words
 - A variant of truncated version known as SHA-224
 - SHA-512 : uses 64-bit words
 - A variant of truncated version known as SHA-384
 - SHA-3
 - Same hash length as SHA-2 but not designed by NSA

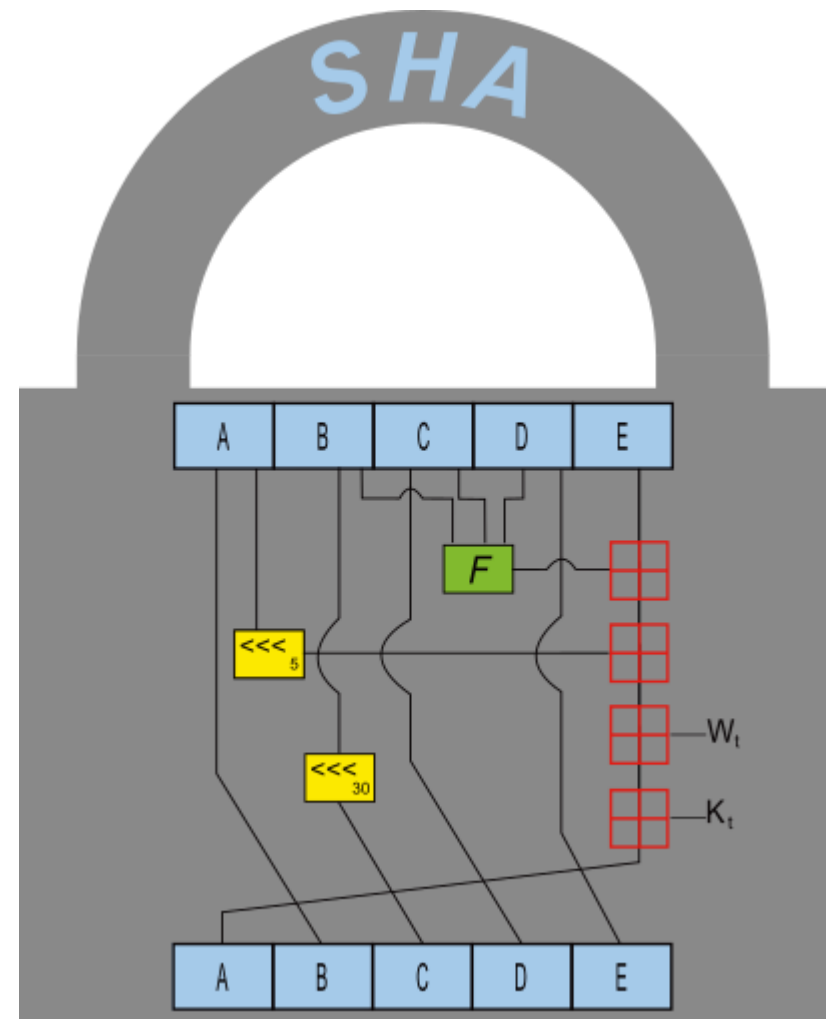
Various parameters for SHA

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80

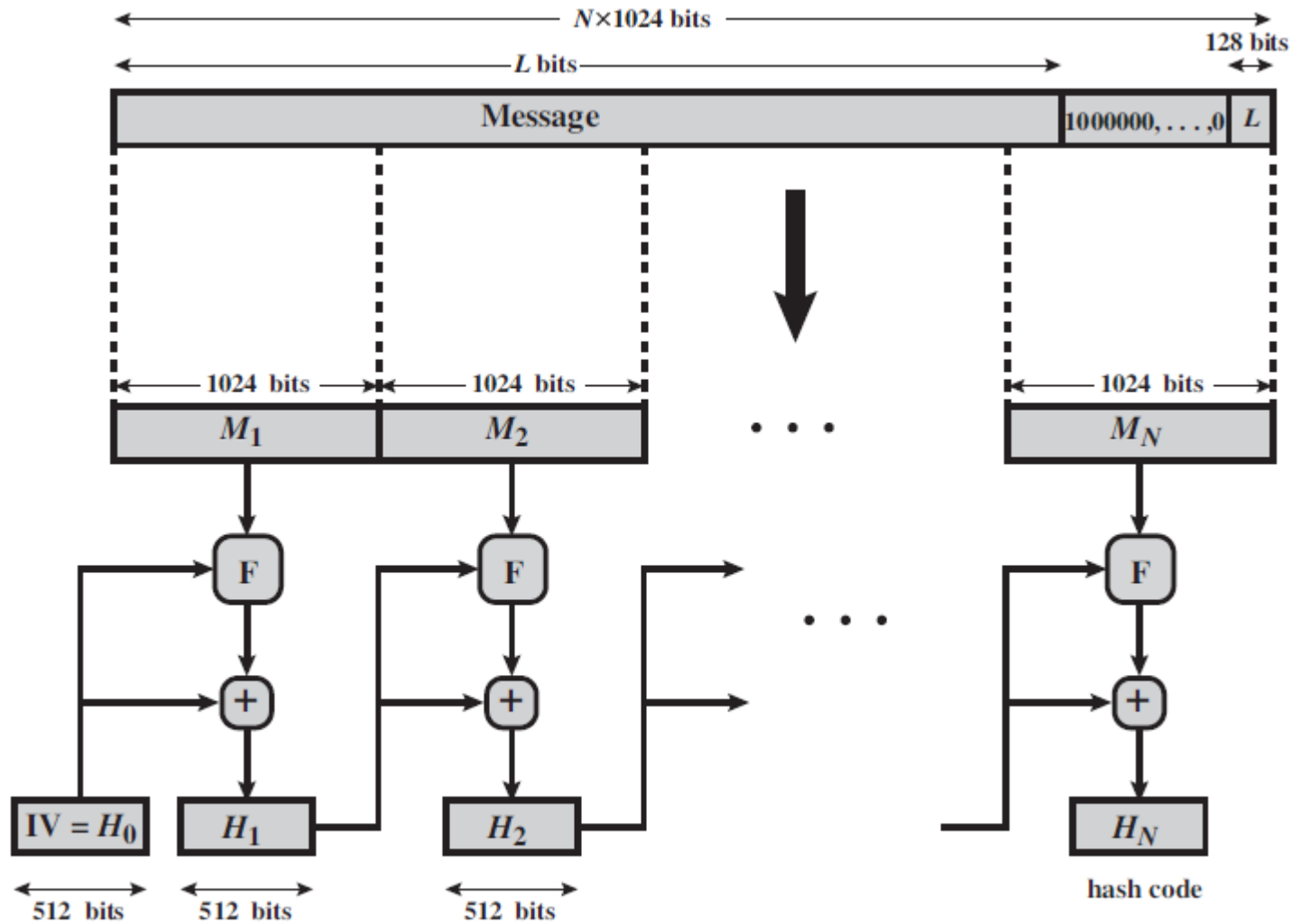
Note: All sizes are measured in bits.

SHA-2

- **SHA-2 (Secure Hash Algorithm 2)** is a set of [cryptographic hash functions](#) designed by the United States [National Security Agency](#) (NSA).



Block Diagram of SHA-512 (SHA-2)



$+$ = word-by-word addition mod 2^{64}

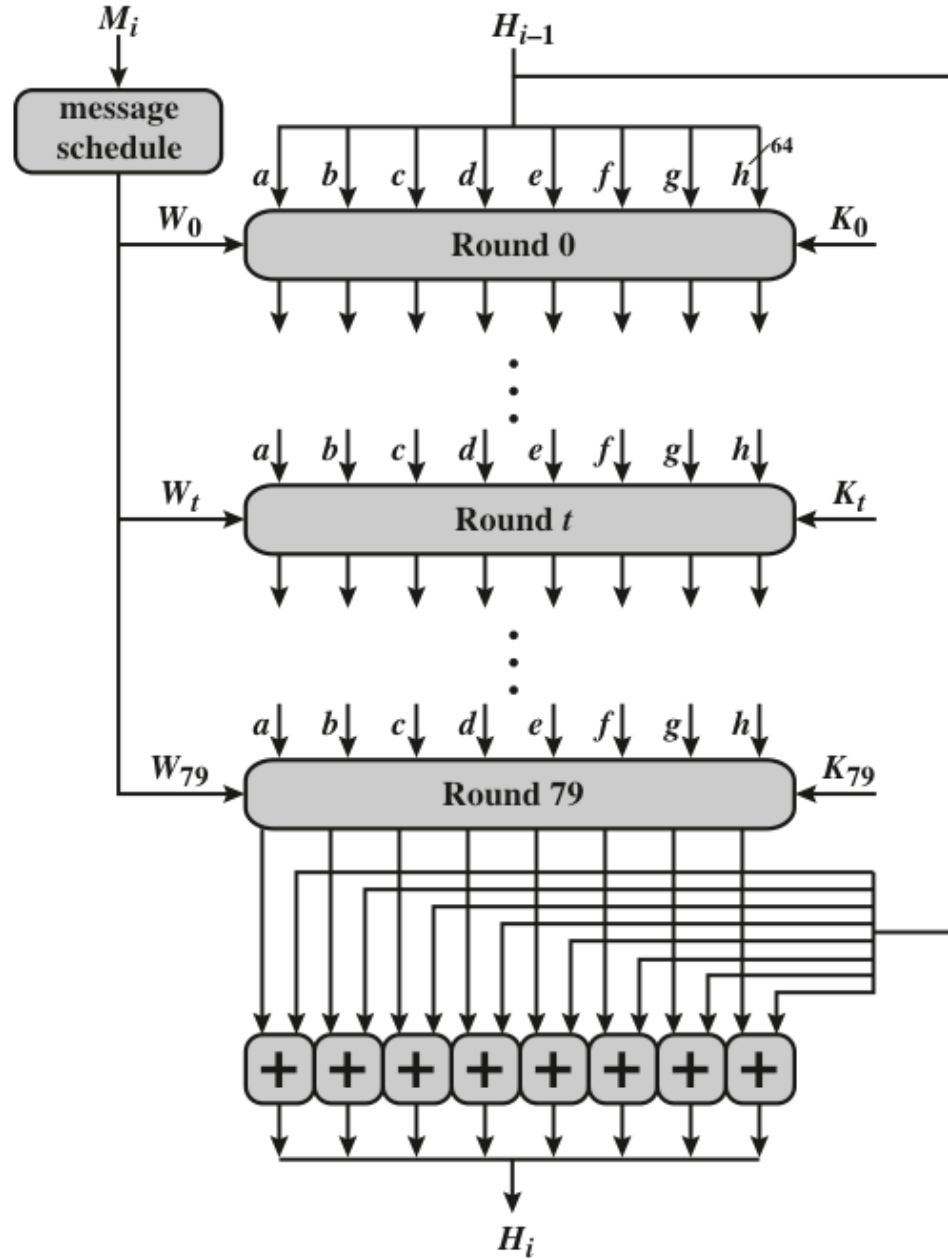


Figure 11.10 SHA-512 Processing of a Single 1024-Bit Block

Hash Buffers

- A 512-bit buffer is used to hold intermediate and final results of the hash function
- The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h)
- Message in 1024-bit (128-word) blocks will be processed in 80 rounds
 - Each round takes as input the 512-bit buffer value, $abcdefgh$, and updates the contents of the buffer

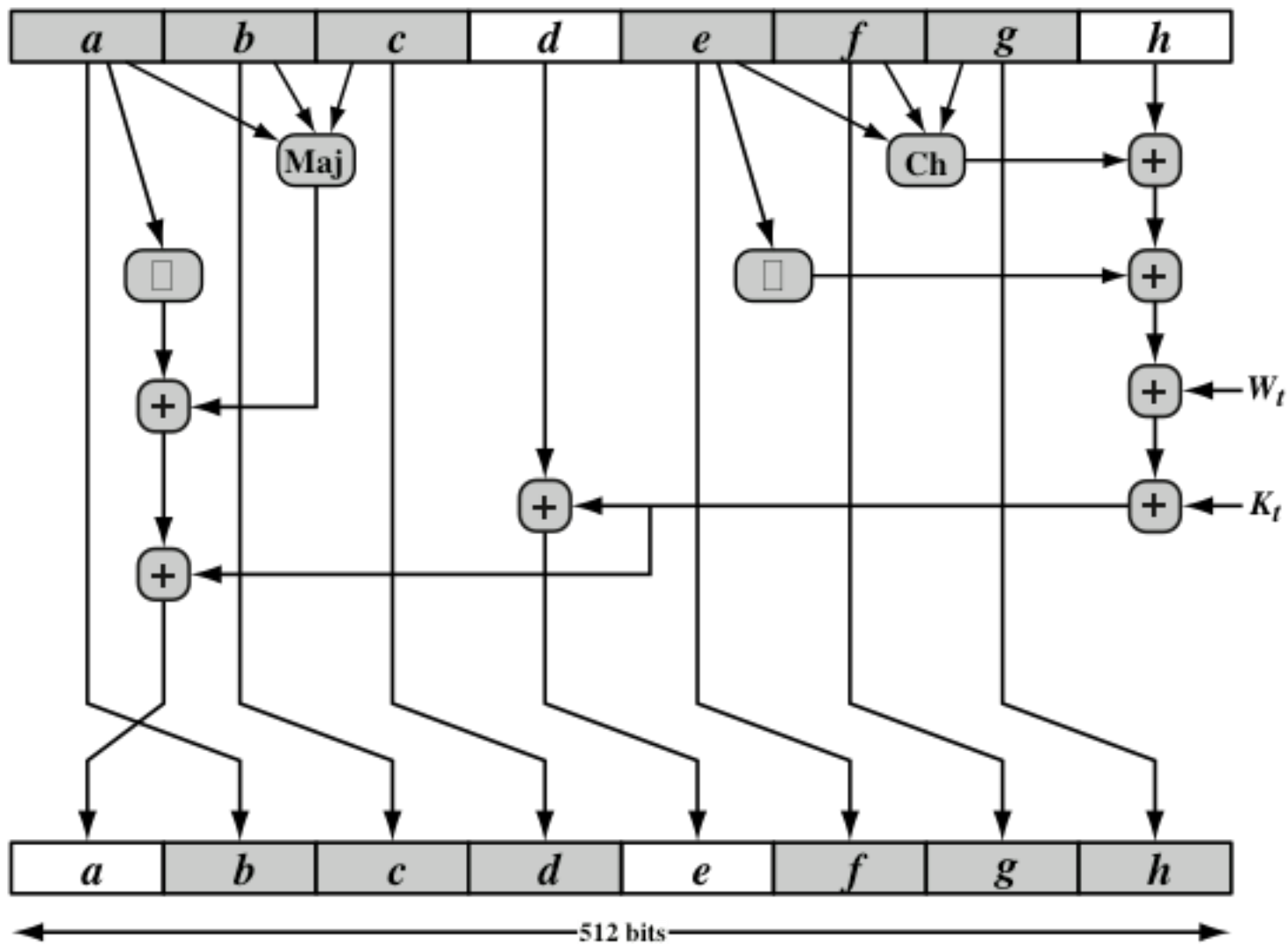


Figure 11.11 Elementary SHA-512 Operation (single round)

SHA-2 Compression

One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$$\text{Ch}(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$$

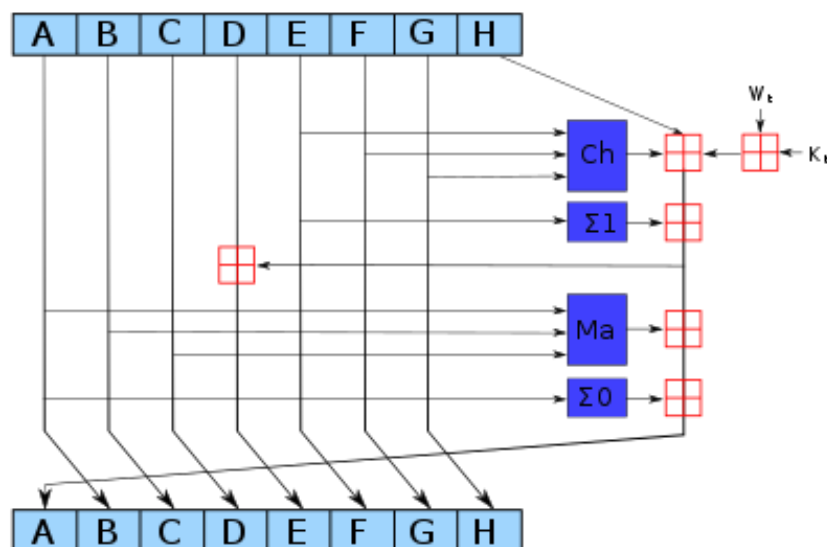
$$\text{Ma}(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256.

The red $\boxplus$ is addition modulo 2^{32} for SHA-256, or 2^{64} for SHA-512.



Default Test Vectors

- For example, hash value of empty strings, and other default values for testing.

```
SHA224("")
```

```
0x d14a028c2a3a2bc9476102bb288234c415a2b01f828ea62ac5b3e42f
```

```
SHA256("")
```

```
0x e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

```
SHA384("")
```

```
0x 38b060a751ac96384cd9327eb1b1e36a21fdb71114be07434c0cc7bf63f6e1da274edebfe76f
```

```
SHA512("")
```

```
0x cf83e1357eeeb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85:
```

```
SHA512/224("")
```

```
0x 6ed0dd02806fa89e25de060c19d3ac86cabb87d6a0ddd05c333b84f4
```

```
SHA512/256("")
```

```
0x c672b8d1ef56ed28ab87c3622c5114069bdd3ad7b8f9737498d0c01ecef0967a
```

Source code of SHA-256

- The C source code for SHA-256 and SHA-512 can be found at
 - <http://www.opensource.apple.com/source/OpenSSL098/OpenSSL098-35/src/crypto/sha/sha256.c>
 - <http://www.opensource.apple.com/source/OpenSSL098/OpenSSL098-35/src/crypto/sha/sha512.c>

```
int SHA384_Init (SHA512_CTX *c)
{
#ifdef OPENSSL_FIPS
    FIPS_selftest_check();
#endif
    c->h[0]=U64(0xcbbb9d5dc1059ed8);
    c->h[1]=U64(0x629a292a367cd507);
    c->h[2]=U64(0x9159015a3070dd17);
    c->h[3]=U64(0x152fec8f70e5939);
    c->h[4]=U64(0x67332667ffc00b31);
    c->h[5]=U64(0x8eb44a8768581511);
    c->h[6]=U64(0xdb0c2e0d64f98fa7);
    c->h[7]=U64(0x47b5481dbefa4fa4);
    c->Nl=0;          c->Nh=0;
    c->num=0;         c->md_len=SHA384_DIGEST_LENGTH;
    return 1;
}

int SHA512_Init (SHA512_CTX *c)
{
#ifdef OPENSSL_FIPS
    FIPS_selftest_check();
#endif
    c->h[0]=U64(0x6a09e667f3bcc908);
    c->h[1]=U64(0xbb67ae8584caa73b);
    c->h[2]=U64(0x3c6ef372fe94f82b);
    c->h[3]=U64(0xa54ff53a5f1d36f1);
    c->h[4]=U64(0x510e527fade682d1);
    c->h[5]=U64(0x9b05688c2b3e6c1f);
    c->h[6]=U64(0x1f83d9abfb41bd6b);
    c->h[7]=U64(0x5be0cd19137e2179);
    c->Nl=0;          c->Nh=0;
    c->num=0;         c->md_len=SHA512_DIGEST_LENGTH;
    return 1;
}
```

SHA Comparison

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Rounds	Operations	Security bits (Info)	Capacity against length extension attacks	Performance on Skylake (median cpb) ^[40]		First Published
										long messages	8 bytes	
MD5 (as reference)		128	128 (4 × 32)	512	Unlimited ^[41]	64	And, Xor, Rot, Add (mod 2 ³²), Or	<64 (collisions found)	0	4.99	55.00	1992
SHA-0		160	160 (5 × 32)	512	2 ⁶⁴ – 1	80	And, Xor, Rot, Add (mod 2 ³²), Or	<34 (collisions found)	0	≈ SHA-1	≈ SHA-1	1993
SHA-1								<63 (collisions found ^[42])		3.47	52.00	1995
SHA-2	SHA-224	224	256 (8 × 32)	512	2 ⁶⁴ – 1	64	And, Xor, Rot, Add (mod 2 ³²), Or, Shr	112	32	7.62	84.50	2001
	SHA-256	256						128		7.63	85.25	
	SHA-384	384	512 (8 × 64)	1024	2 ¹²⁸ – 1	80	And, Xor, Rot, Add (mod 2 ⁶⁴), Or, Shr	192	128 (≤ 384)	5.12	135.75	
	SHA-512	512						256		5.06	135.50	
	SHA-512/224	224						112	288	≈ SHA-384	≈ SHA-384	
	SHA-512/256	256						128	256			
SHA-3	SHA3-224	224	1600 (5 × 5 × 64)	1152	Unlimited ^[43]	24 ^[44]	And, Xor, Rot, Not	112	448	8.12	154.25	2015
	SHA3-256	256		1088				128	512	8.59	155.50	
	SHA3-384	384		832				192	768	11.06	164.00	
	SHA3-512	512		576				256	1024	15.88	164.00	
	SHAKE128	d (arbitrary)	1344	min(d/2, 128)				256	7.08	155.25		
	SHAKE256	d (arbitrary)	1088	min(d/2, 256)				512	8.59	155.50		

Simplified Comparison of SHA Parameters

Algorithm	Message Size	Block Size	Word Size	Message Digest Size
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Note: All sizes are measured in bits.

SHA-3 - Keccak

Designers: Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.

SHA-1 has not yet been "broken"

- No one has demonstrated a technique for producing collisions in a practical amount of time
- Considered to be insecure and has been phased out for SHA-2



NIST announced in 2007 a competition for the SHA-3 next generation NIST hash function

- Winning design was announced by NIST in October 2012
- SHA-3 is a cryptographic hash function that is intended to complement SHA-2 as the approved standard for a wide range of applications

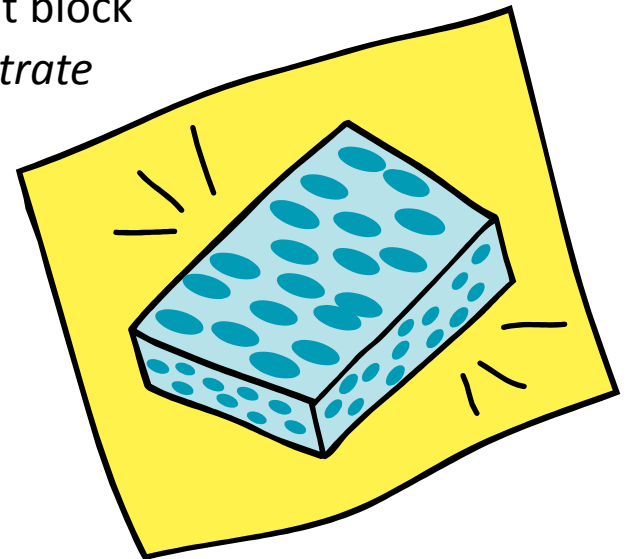
SHA-2 shares the same structure and mathematical operations as its predecessors so this is a cause for concern

- Because it will take years to find a suitable replacement for SHA-2 should it become vulnerable, NIST decided to begin the process of developing a new hash standard

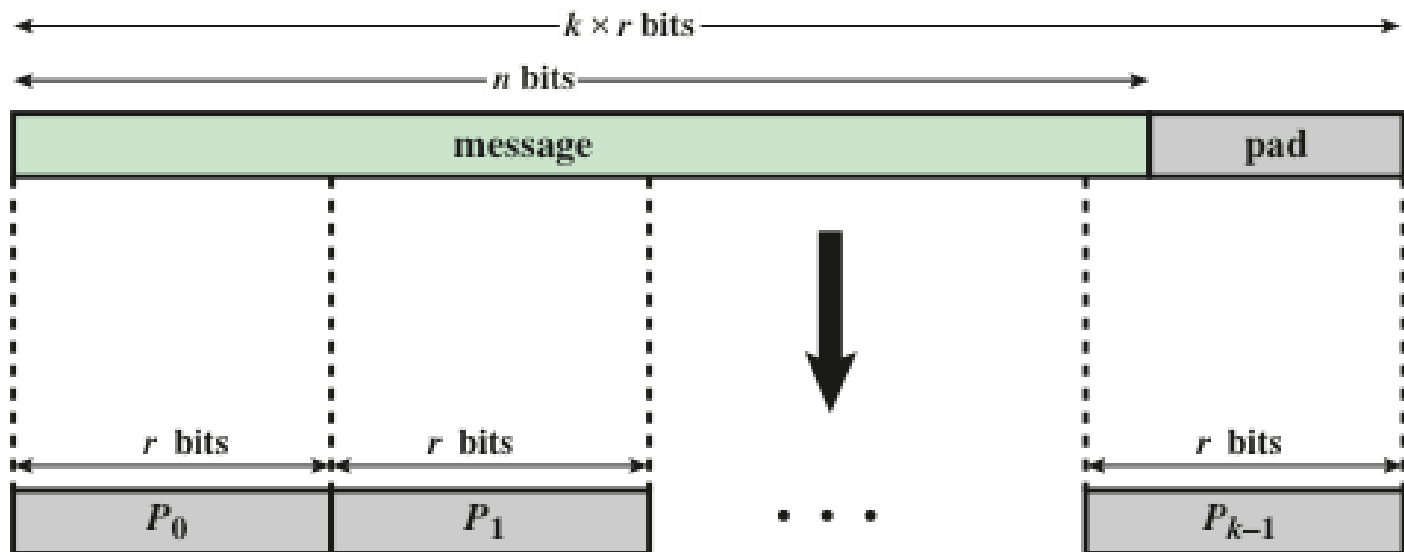


The Sponge Construction

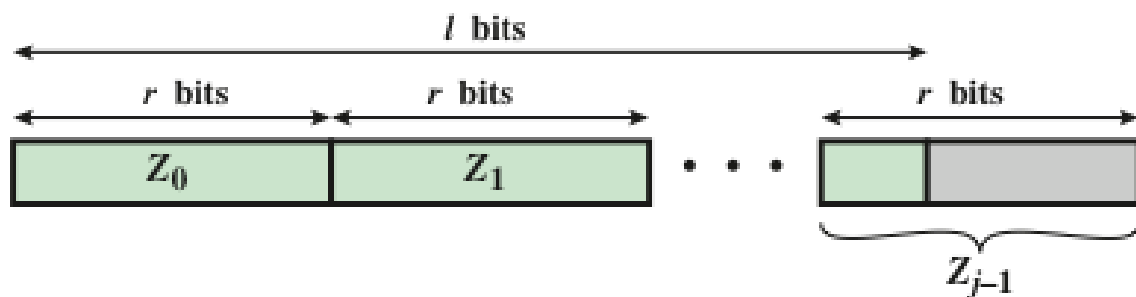
- Underlying structure of SHA-3 is a scheme referred to by its designers as a *sponge construction*
- Takes an input message and partitions it into fixed-size blocks
- Each block is processed in turn with the output of each iteration fed into the next iteration, finally producing an output block
- The sponge function is defined by three parameters:
 - f = the internal function used to process each input block
 - r = the size in bits of the input blocks, called the *bitrate*
 - pad = the padding algorithm



<https://en.wikipedia.org/wiki/SHA-3>



(a) Input



(b) Output

Figure 11.14 Sponge Function Input and Output

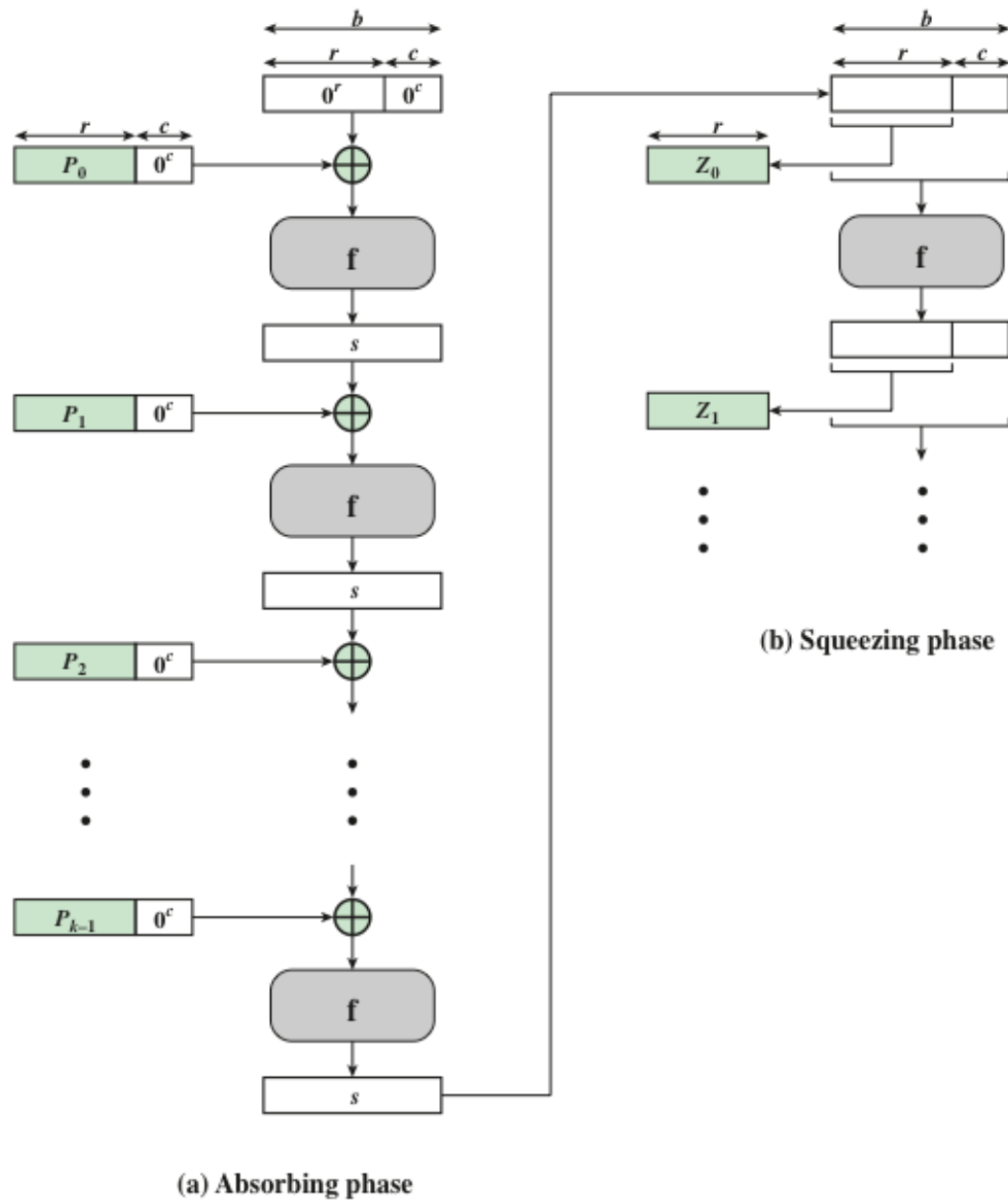


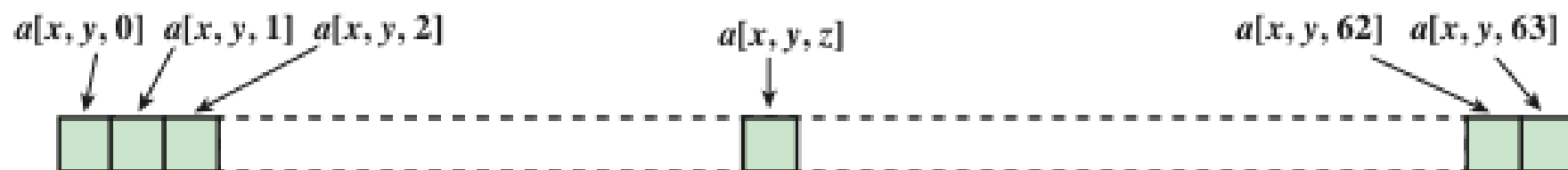
Figure 11.15 Sponge Construction

SHA-3 Parameters

Message Digest Size	224	256	384	512
Message Size	no maximum	no maximum	no maximum	no maximum
Block Size (bitrate r)	1152	1088	832	576
Word Size	64	64	64	64
Number of Rounds	24	24	24	24
Capacity c	448	512	768	1024
Collision resistance	2^{112}	2^{128}	2^{192}	2^{256}
Second preimage resistance	2^{224}	2^{256}	2^{384}	2^{512}

	$x = 0$	$x = 1$	$x = 2$	$x = 3$	$x = 4$
$y = 4$	$L[0, 4]$	$L[1, 4]$	$L[2, 4]$	$L[3, 4]$	$L[4, 4]$
$y = 3$	$L[0, 3]$	$L[1, 3]$	$L[2, 3]$	$L[3, 3]$	$L[4, 3]$
$y = 2$	$L[0, 2]$	$L[1, 2]$	$L[2, 2]$	$L[3, 2]$	$L[4, 2]$
$y = 1$	$L[0, 1]$	$L[1, 1]$	$L[2, 1]$	$L[3, 1]$	$L[4, 1]$
$y = 0$	$L[0, 0]$	$L[1, 0]$	$L[2, 0]$	$L[3, 0]$	$L[4, 0]$

(a) State variable as 5×5 matrix A of 64-bit words



(b) Bit labeling of 64-bit words

Figure 11.16 SHA-3 State Matrix

SHA-3

Iteration Function f

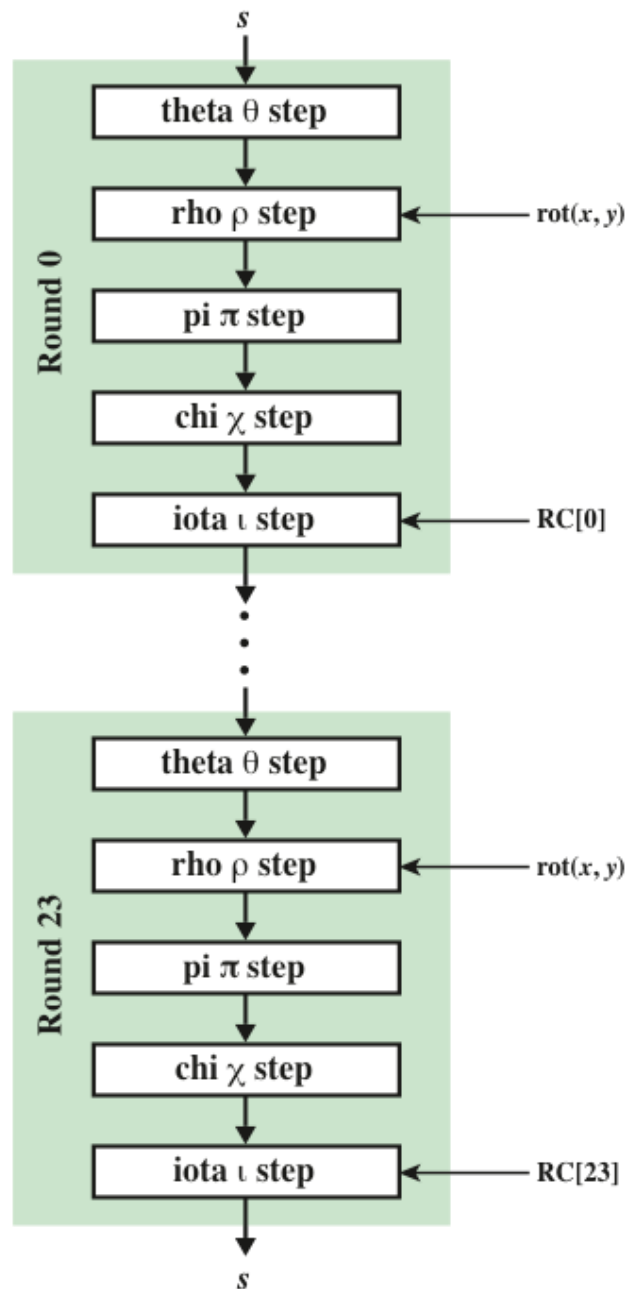


Figure 11.17 SHA-3 Iteration Function f

Step Functions in SHA-3

Function	Type	Description
θ	Substitution	New value of each bit in each word depends its current value and on one bit in each word of preceding column and one bit of each word in succeeding column.
ρ	Permutation	The bits of each word are permuted using a circular bit shift. $W[0, 0]$ is not affected.
π	Permutation	Words are permuted in the 5×5 matrix. $W[0, 0]$ is not affected.
χ	Substitution	New value of each bit in each word depends on its current value and on one bit in next word in the same row and one bit in the second next word in the same row.
ι	Substitution	$W[0, 0]$ is updated by XOR with a round constant.

Summary

- Applications of cryptographic hash functions
 - Message authentication
 - Digital signatures
 - Other applications
- Requirements and security
 - Security requirements for cryptographic hash functions
 - Brute-force attacks
 - Cryptanalysis



- Hash functions based on cipher block chaining
- Secure hash algorithm (SHA)
 - SHA-512 logic
 - SHA-512 round function
- SHA-3
 - The sponge construction

Lecture 6: Hashing and Digital Signature

- Hashing
- SHA
- Collision
- Message Authentication
- Digital Signature



SHA Collision

- <https://blog.pcisecuritystandards.org/how-the-sha-1-collision-impacts-security-of-payments>

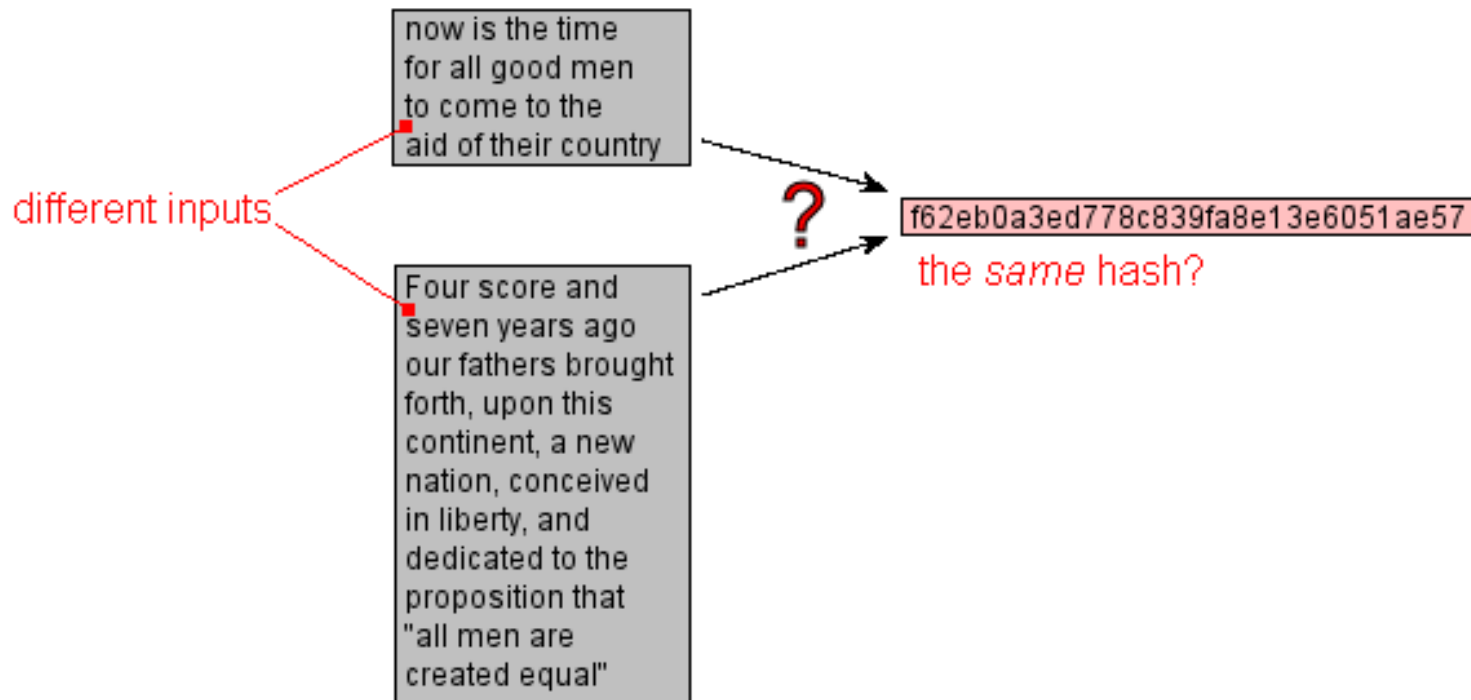


Collisions

- There is astonishingly large number of possible hashes available
 - a 128-bit hash can have 3.4×10^{38} possible values, which is 340,282,366,920,938,463,463,374,607,431,768,211,456 possible hashes
 - however large this number is, there are still a finite number of MD5 outputs (2^{128})
- But there are infinite number of possible inputs

Collisions

- When different chunks of data produce the same hash value, this is known as a collision



Collision Attacks

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

Published in	Year	Attack method	Attack	Variant	Rounds	Complexity
<i>New Collision Attacks Against Up To 24-step SHA-2</i> ^[34]	2008	Deterministic	Collision	SHA-256	24/64	$2^{28.5}$
				SHA-512	24/80	$2^{32.5}$
<i>Preimages for step-reduced SHA-2</i> ^[35]	2009	Meet-in-the-middle	Preimage	SHA-256	42/64	$2^{251.7}$
					43/64	$2^{254.9}$
				SHA-512	42/80	$2^{502.3}$
					46/80	$2^{511.5}$
<i>Advanced meet-in-the-middle preimage attacks</i> ^[36]	2010	Meet-in-the-middle	Preimage	SHA-256	42/64	$2^{248.4}$
				SHA-512	42/80	$2^{494.6}$
<i>Higher-Order Differential Attack on Reduced SHA-256</i> ^[2]	2011	Differential	Pseudo-collision	SHA-256	46/64	2^{178}
					33/64	2^{46}
<i>Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 family</i> ^[1]	2011	Biclique	Preimage	SHA-256	45/64	$2^{255.5}$
				SHA-512	50/80	$2^{511.5}$
			Pseudo-preimage	SHA-256	52/64	2^{255}
				SHA-512	57/80	2^{511}

MD5 is Considered Broken

- 2004
 - Researchers from China has reported that they have found ways to reliably generate collisions in four hash functions
 - Various groups of people have created demo showing that two different files with identical MD5 hash existed
- 2007
 - Two executable files (with different behaviors) with the same MD5 hash could be created
- Reference:
 - <http://www.mscs.dal.ca/~selinger/md5collision/>

Remarks

- Finding collisions does not mean that one can find the message, given its hash value
- In the web, resources exists for **trying** MD5 cracking / brute-forcing for the message, given its hash value
 - Search engine such as Google indexed random MD5 strings
 - “Rainbow tables” are also assisting the MD5 cracking process
 - e.g. md5crack at *<http://www.md5crack.com/home>*
 - e.g. *<http://www.md5decrypter.co.uk/>*

Lecture 6: Hashing and Digital Signature

- Hashing
- SHA
- Collision
- Message Authentication
- Digital Signature

Problem

- A message digest can only help verify the data integrity, but does not authenticate the sender of the message
 - Attacker could change the data, re-compute the hash, and the recipient would not know for sure whether it originates from the sender
- To provide message authentication, the sender (say Alice) needs to provide proof that it is Alice sending the message but not an attacker

Possible Approaches to Message Authentication

- Authentication using conventional encryption
 - Only the sender and receiver should share a key
- Message Authentication Code
 - Also known as a keyed hash function
 - Calculate the MAC as a function of the message and the key
 - $MAC = F(K, M)$

Remarks: The digest created by a cryptographic hash function is sometimes called a modification detection code (MDC)

Message Authentication Code (MAC)

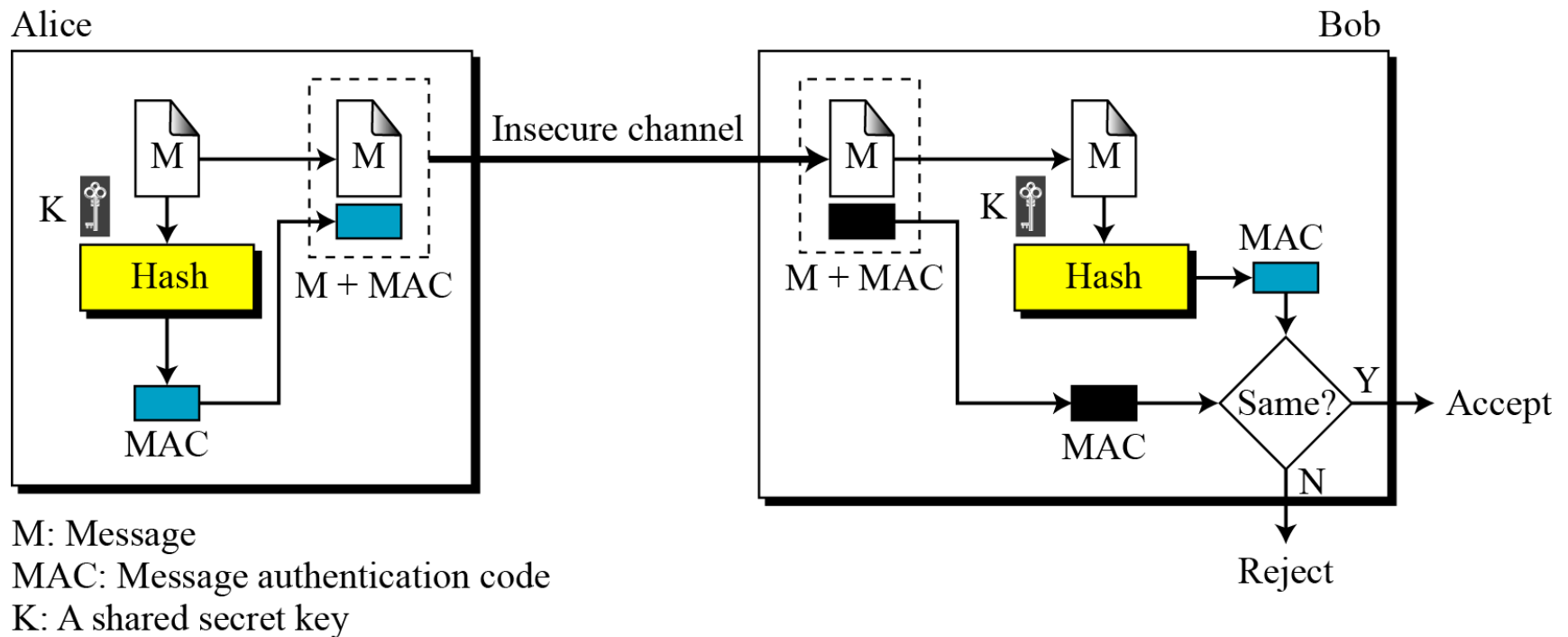
- Also known as a *keyed hash function*
- Typically used between two parties that share a secret key to authenticate information exchanged between those parties

Takes as input a secret key and a data block and produces a hash value (MAC) which is associated with the protected message

- If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value
- An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key

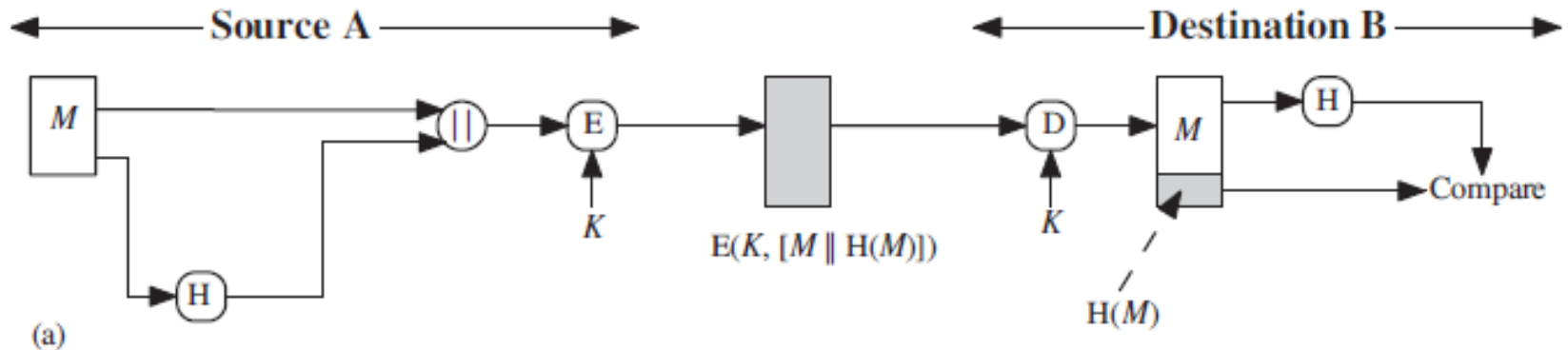
MAC

The security of a MAC depends on the security of the underlying hash algorithm



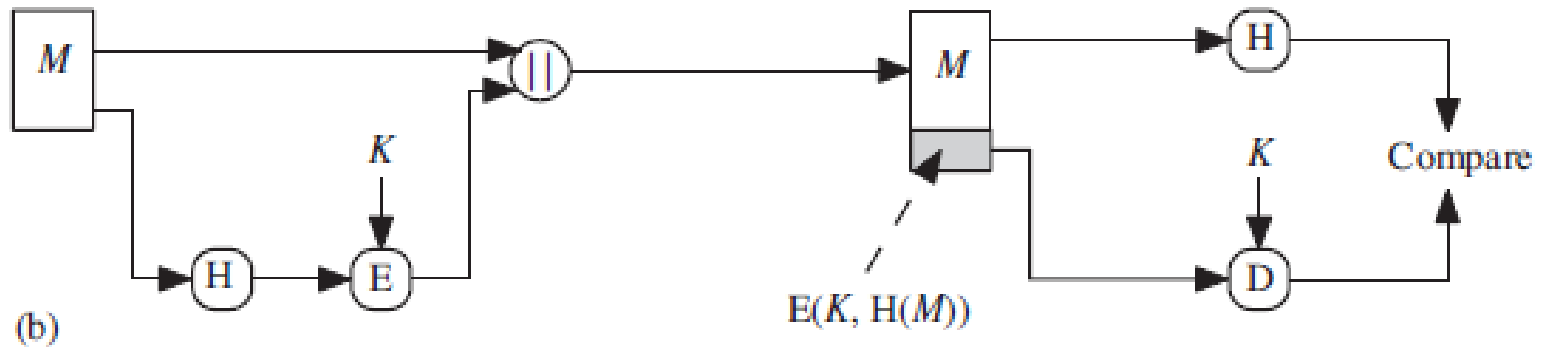
Example Use Case

- The message concatenated with its hash code is encrypted using symmetric encryption
 - Need computational time to encrypt & decrypt the message M + size of M 's hash



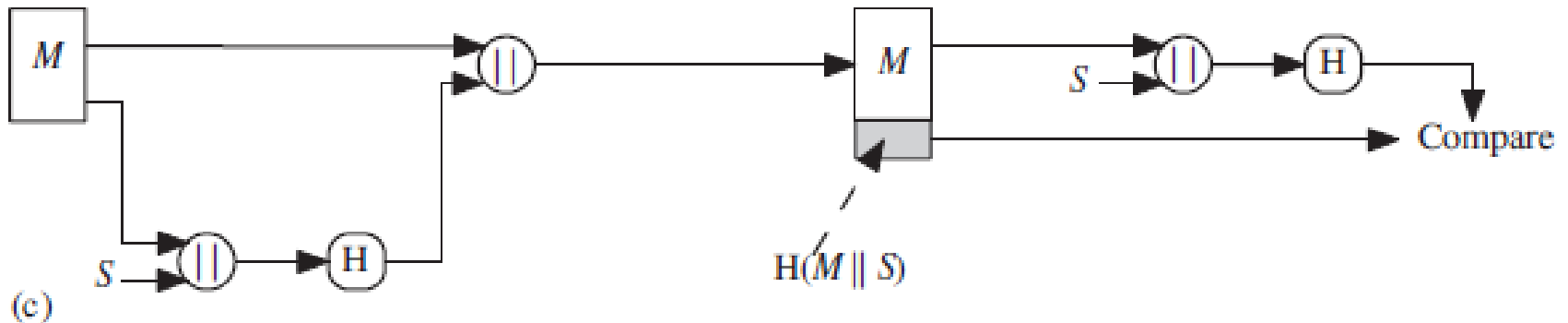
Example Use Case

- Only the hash code is encrypted using symmetric encryption
 - Only encrypting the length of the hash result
 - Message might be read during transmission



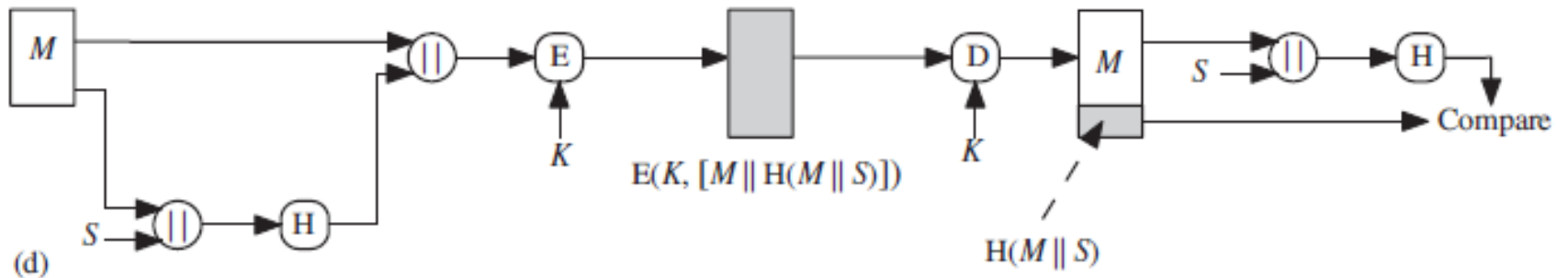
Example Use Case

- Assumes that the two communicating parties share a common secret value S
 - Need practical way to exchange the secret S
- Apply a hash function on message concatenated with S but no encryption



Example Use Case

- Extend the previous method by encrypting the entire message plus the hash code
 - Need time to do encryption / decryption

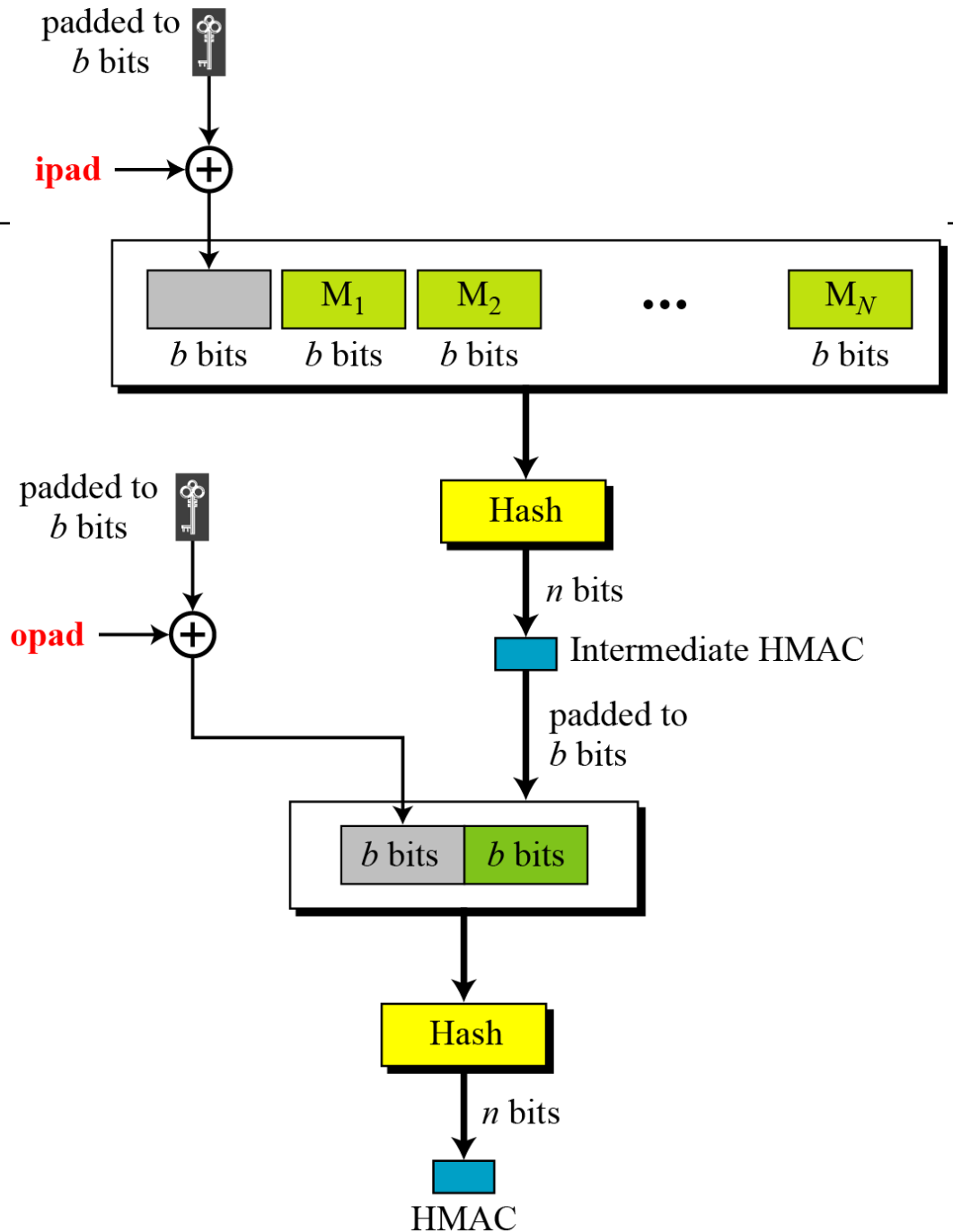


HMAC

- HMAC (keyed-Hashing for Message Authentication)
 - a generic term for an algorithm that uses a keyless hash function and a cryptographic key to produce a keyed hash function
 - The size of the output of HMAC is the same as that of the underlying hash function (128 bits for MD5 and 160 bits for SHA-1)
 - e.g. if the hash function is SHA-1, it is known as HMAC-SHA1
 - defined in RFC 2104
 - online HMAC generator:
 - <http://www.asecuritysite.com/encryption/hmac>
 - <http://www.freeformatter.com/hmac-generator.html>

HMAC Block Diagram

- B is the byte-length of the blocks
 - $B = 64$ when using MD5 / SHA-1
- $b = 8 \times B$
- ipad = the byte $0x36$ repeated B times
- opad = the byte $0x5C$ repeated B times.



HMAC Design Objectives

RFC 2104 lists the following objectives for HMAC:

To use, without modifications, available hash functions

To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required

To preserve the original performance of the hash function without incurring a significant degradation

To use and handle keys in a simple way

To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function

Security of HMAC

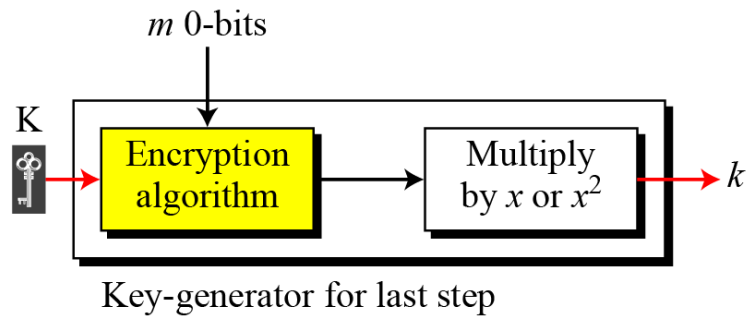
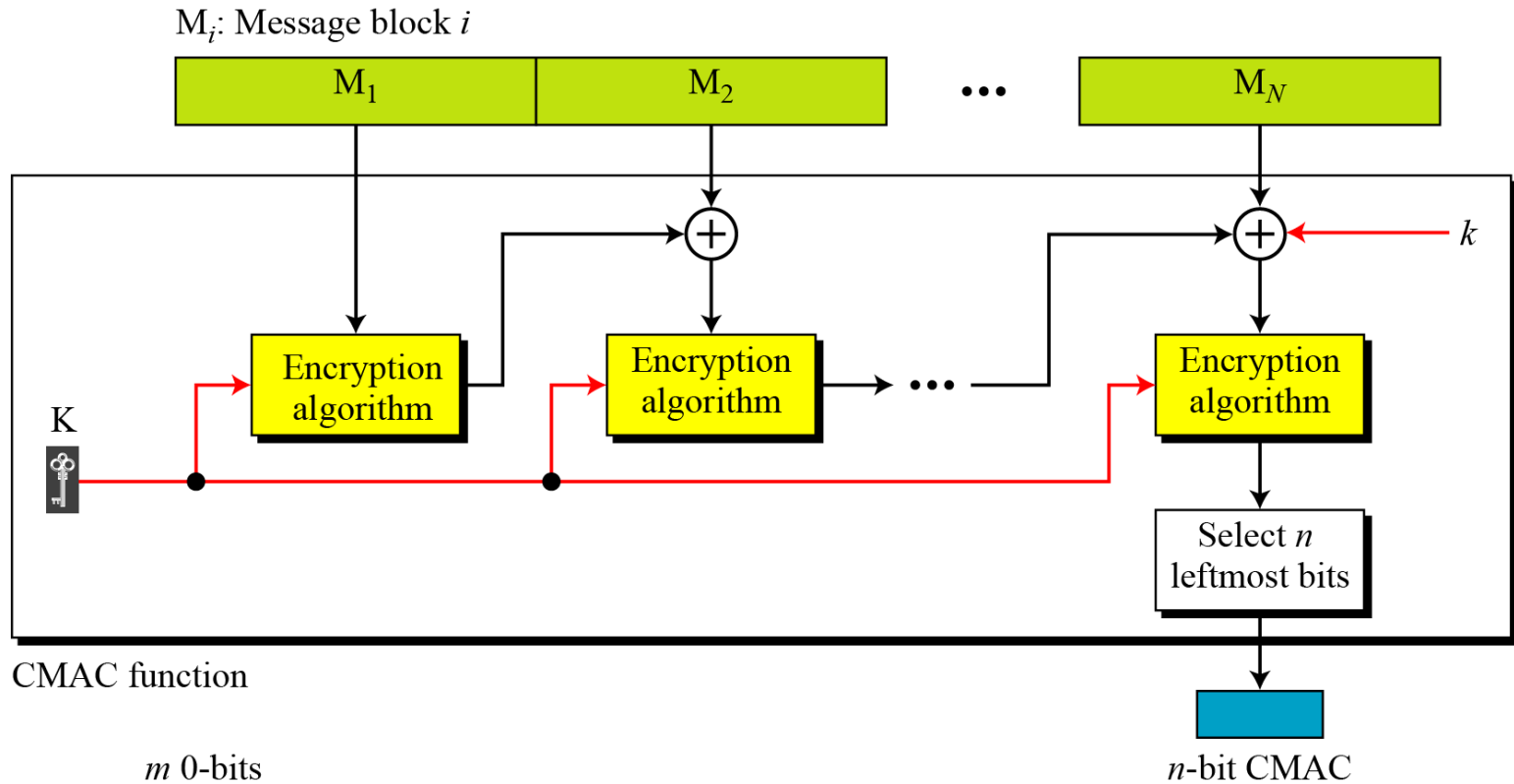


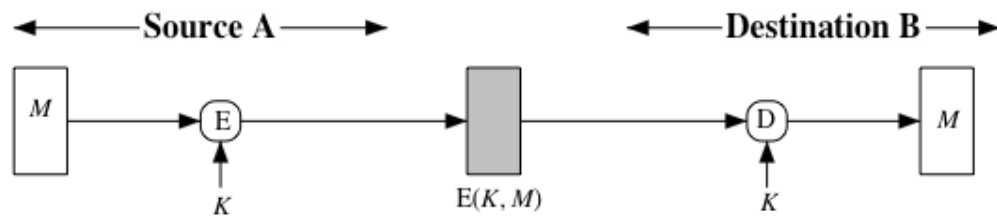
- Depends in some way on the cryptographic strength of the underlying hash function
- Appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC
- Generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message-tag pairs created with the same key

CMAC

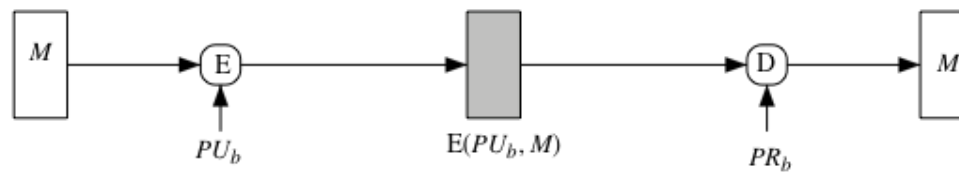
- Cipher-based Message Authentication Code (CMAC)
 - an authentication algorithm defined by the NIST
 - also called NIST-CMAC
 - it is a keyed hash function that is based on a symmetric key block cipher, such as AES [NIST-AES]

CMAC

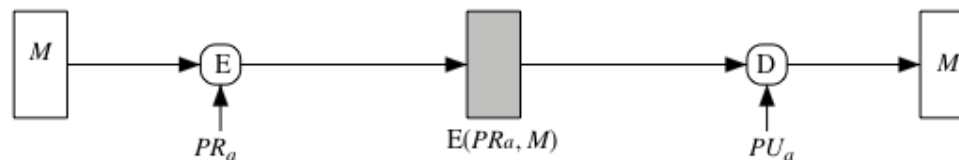




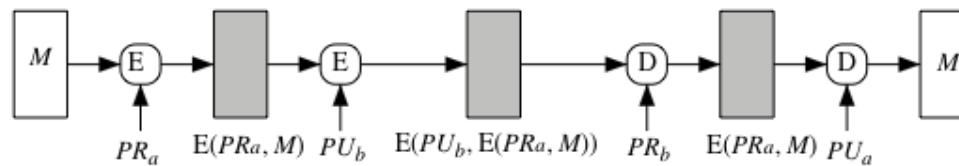
(a) Symmetric encryption: confidentiality and authentication



(b) Public-key encryption: confidentiality



(c) Public-key encryption: authentication and signature



(d) Public-key encryption: confidentiality, authentication, and signature

Figure 12.1 Basic Uses of Message Encryption

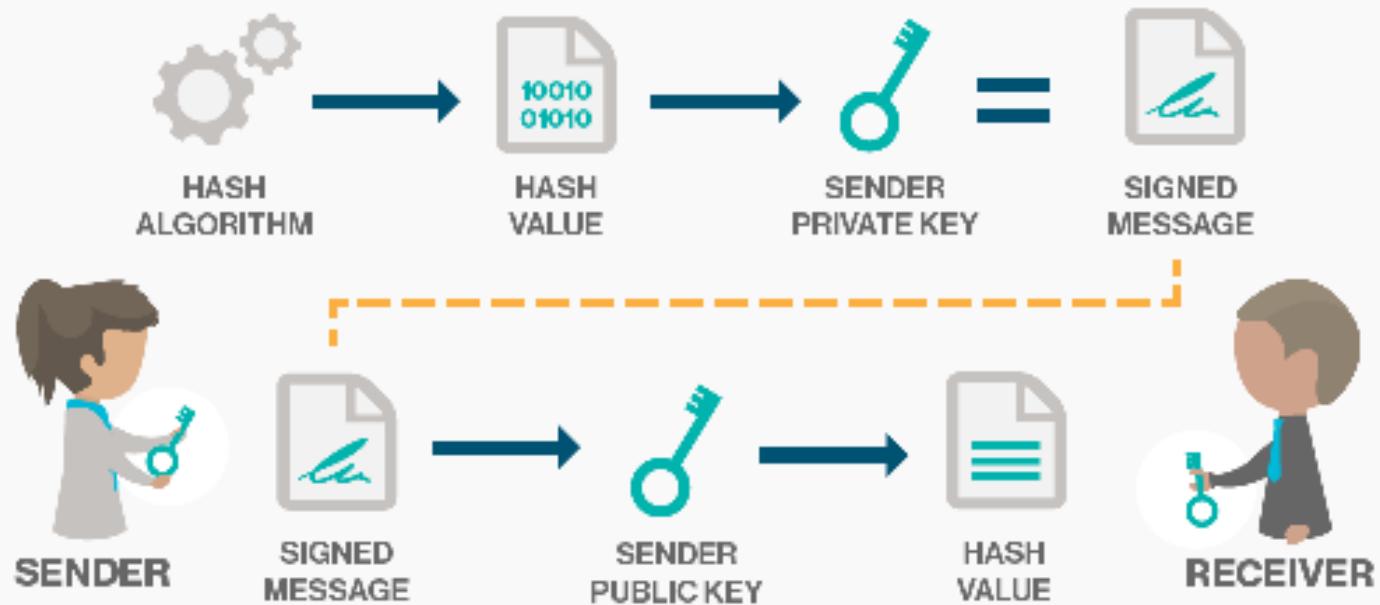
Lecture 6: Hashing and Digital Signature

- Hashing
- Collision
- Message Authentication
- Digital Signature

Digital Signature

- Why do we need it?
- <http://searchsecurity.techtarget.com/definition/digital-signature>

DEFINITION DIGITAL SIGNATURE



Digital Signature

- Digital signature plays a significant role in e-Commerce
- It provides a feature of integrity, authenticity and non-repudiation of a message by using public-key encryption
 - Remember, only you knows your own private key
 - c.f. MAC

Digital Signature

- Operation is similar to that of the MAC
- The hash value of a message is encrypted with a user's private key
- Anyone who knows the user's public key can verify the integrity of the message
- An attacker who wishes to alter the message would need to know the user's private key
- Implications of digital signatures go beyond just message authentication

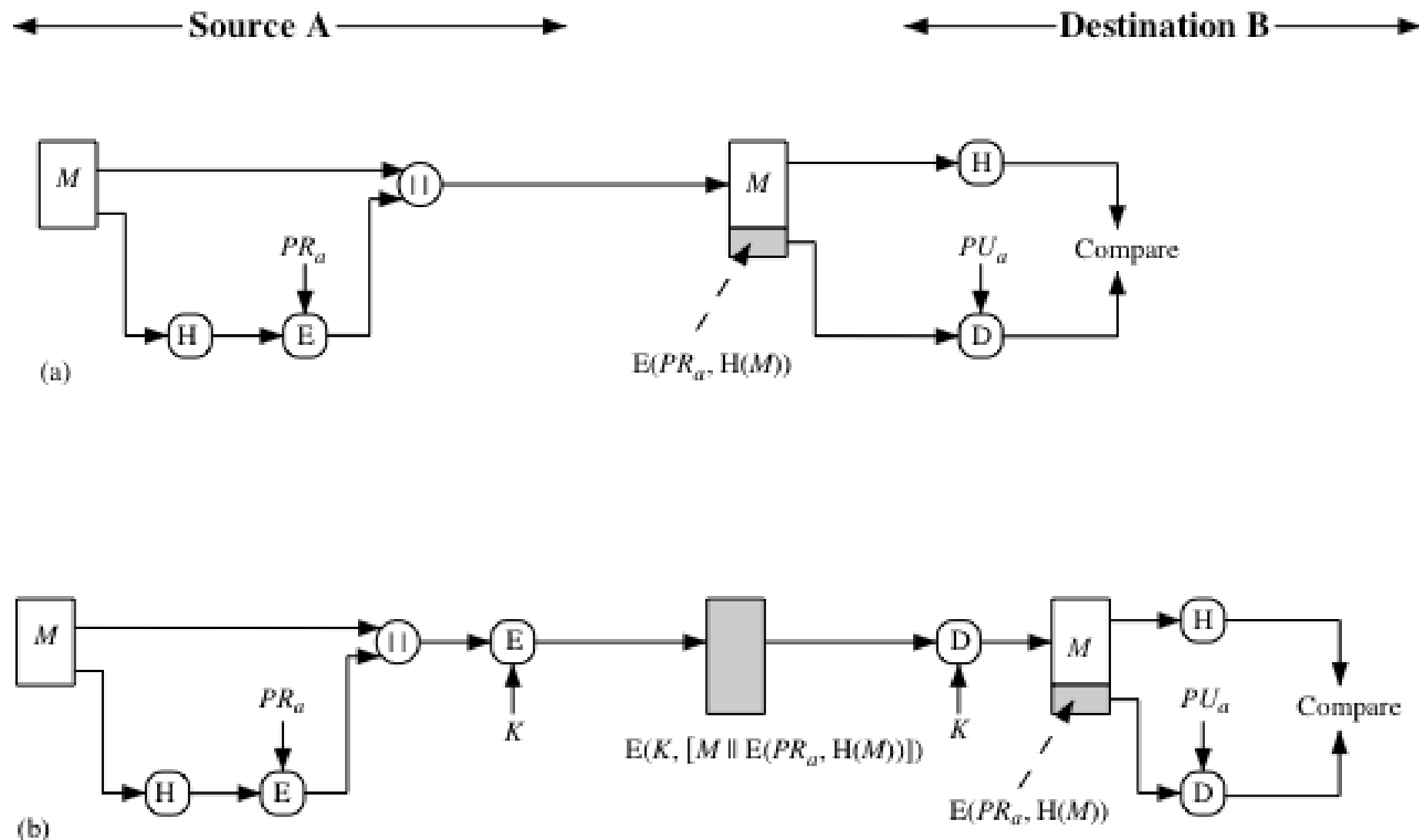
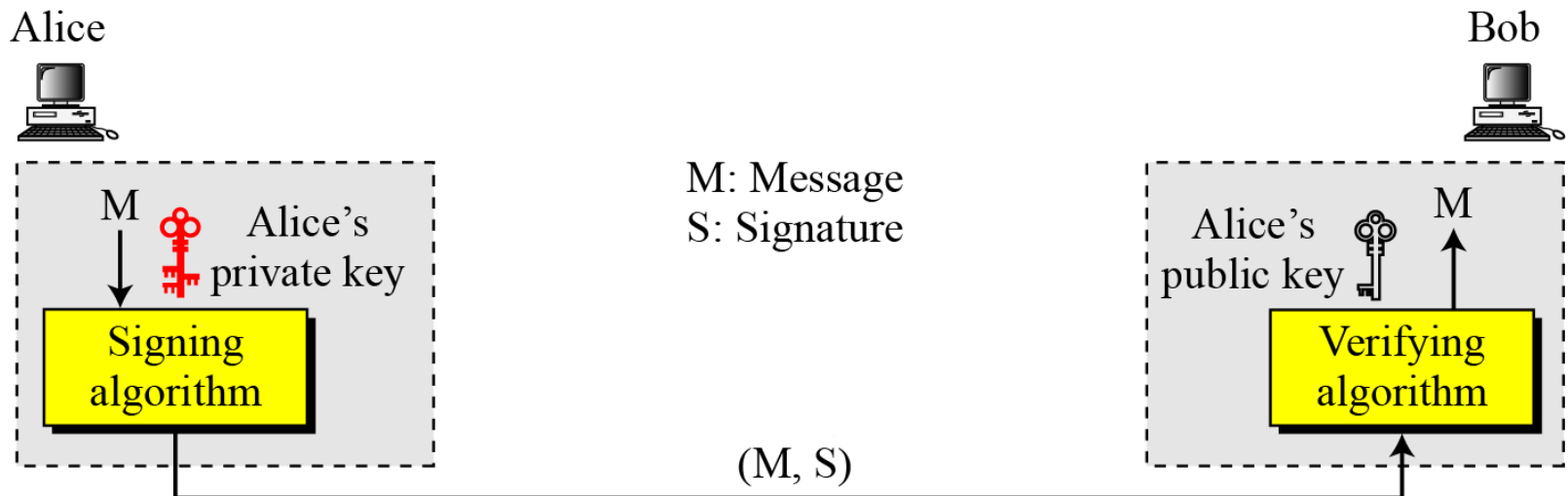


Figure 11.4 Simplified Examples of Digital Signatures

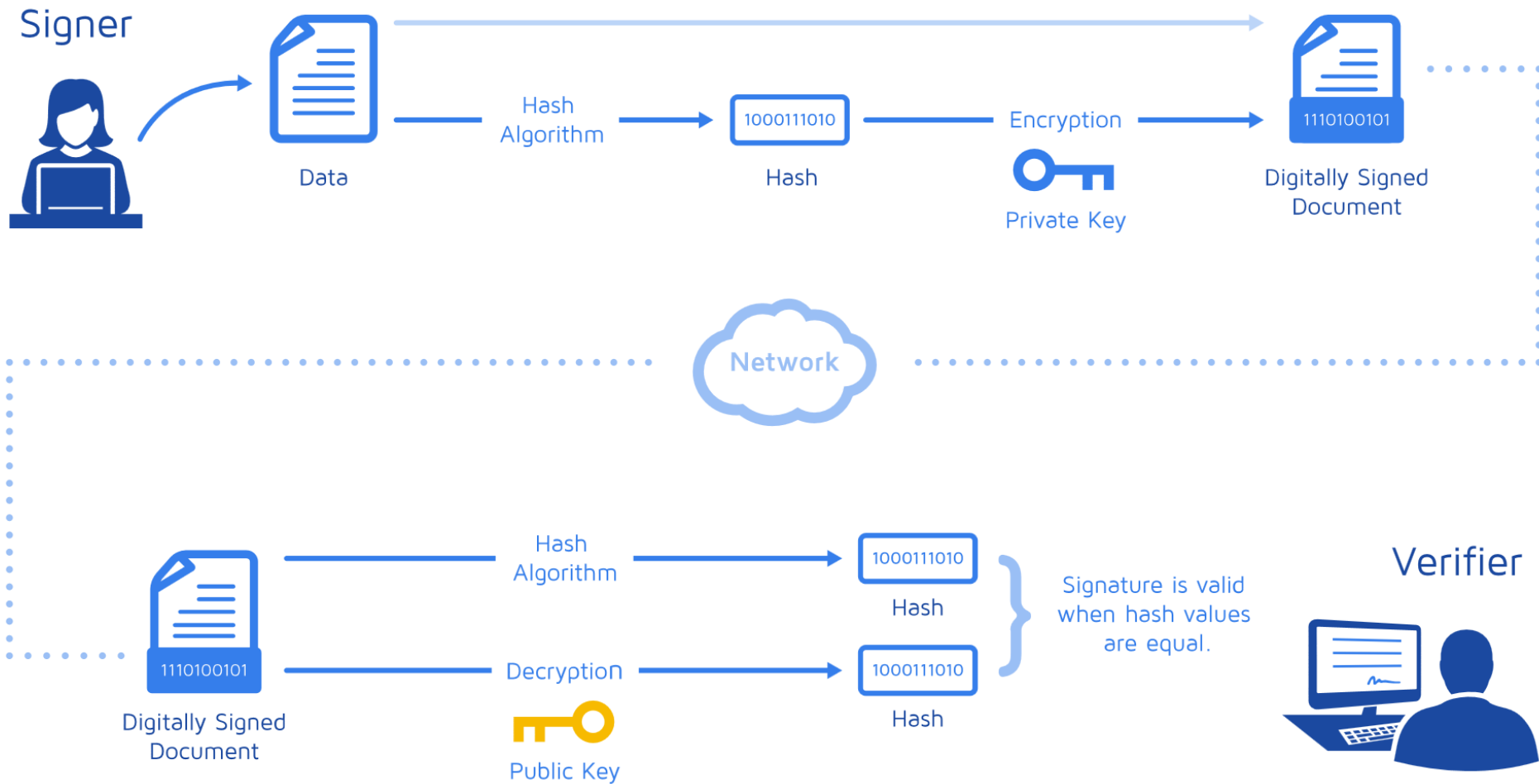
Digital Signature

- A digital signature needs a public-key system.
- The signer signs with her private key; the verifier verifies with the signer's public key.



Digital Signature – How to

- Hashing + Encryption → Over the network



Video Example

- https://www.youtube.com/watch?v=Rnn_HLtgJ2M

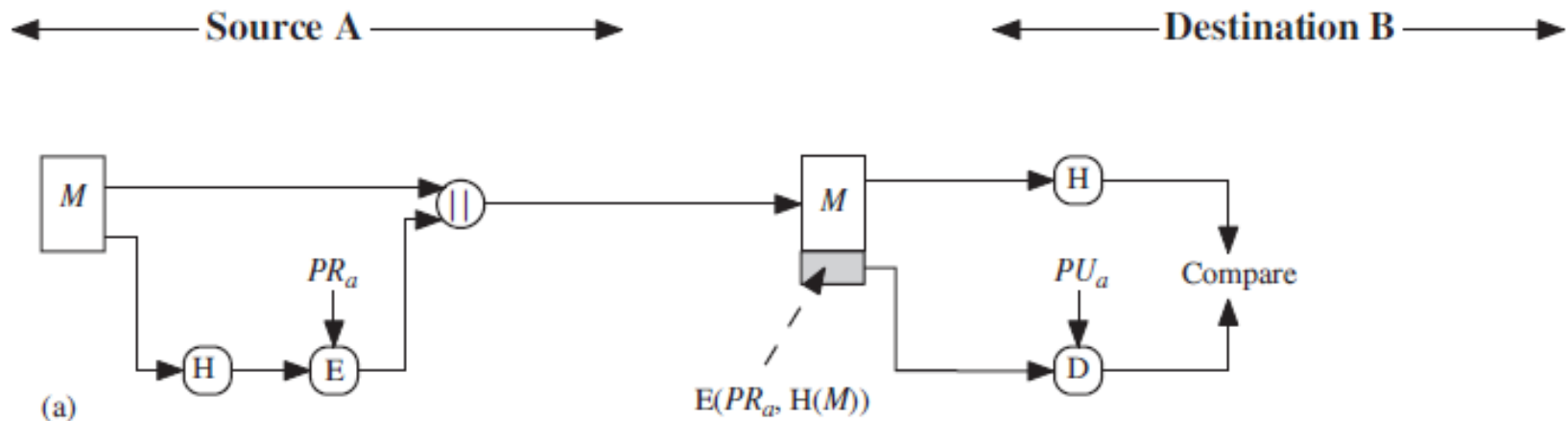
Digital signatures offer...

- Authentication
- Non-repudiation
- Integrity



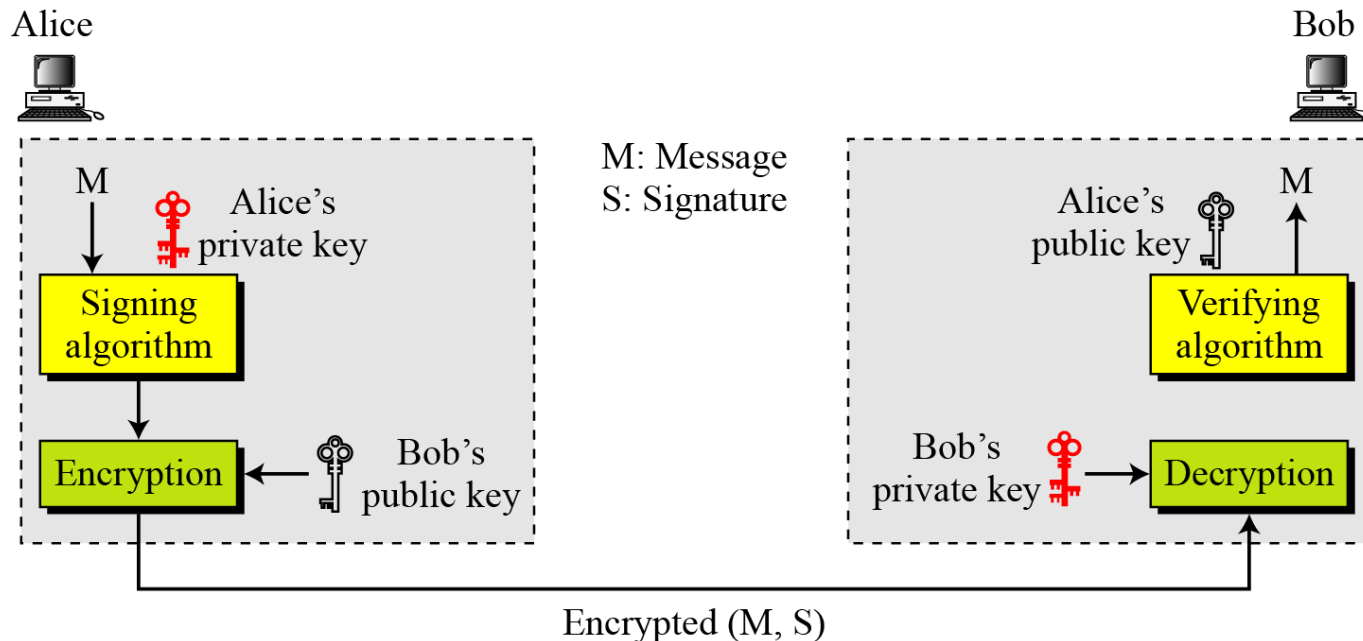
Example Use Example

- The hash code is encrypted using public-key encryption with the sender's private key
 - Only signing, no message confidentiality



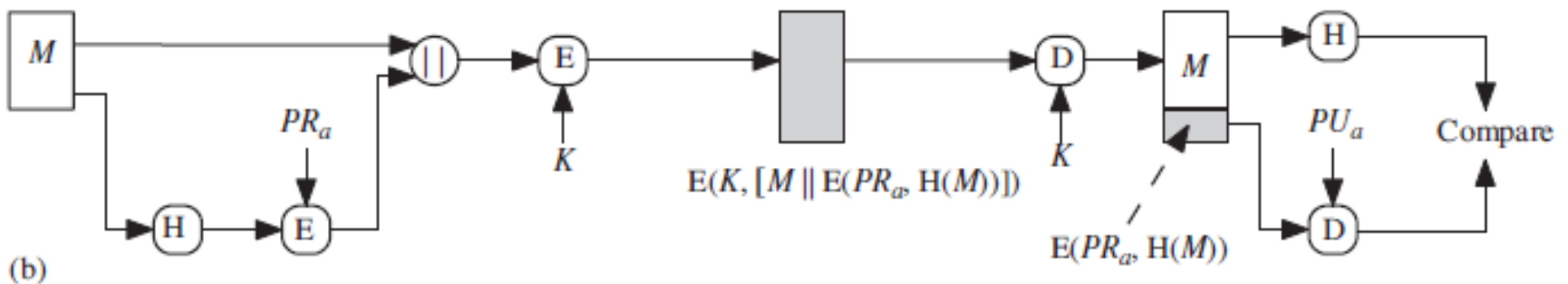
Working with Encryption

- A digital signature does not provide confidentiality
- If there is a need for confidentiality / privacy, another layer of encryption/decryption must be applied



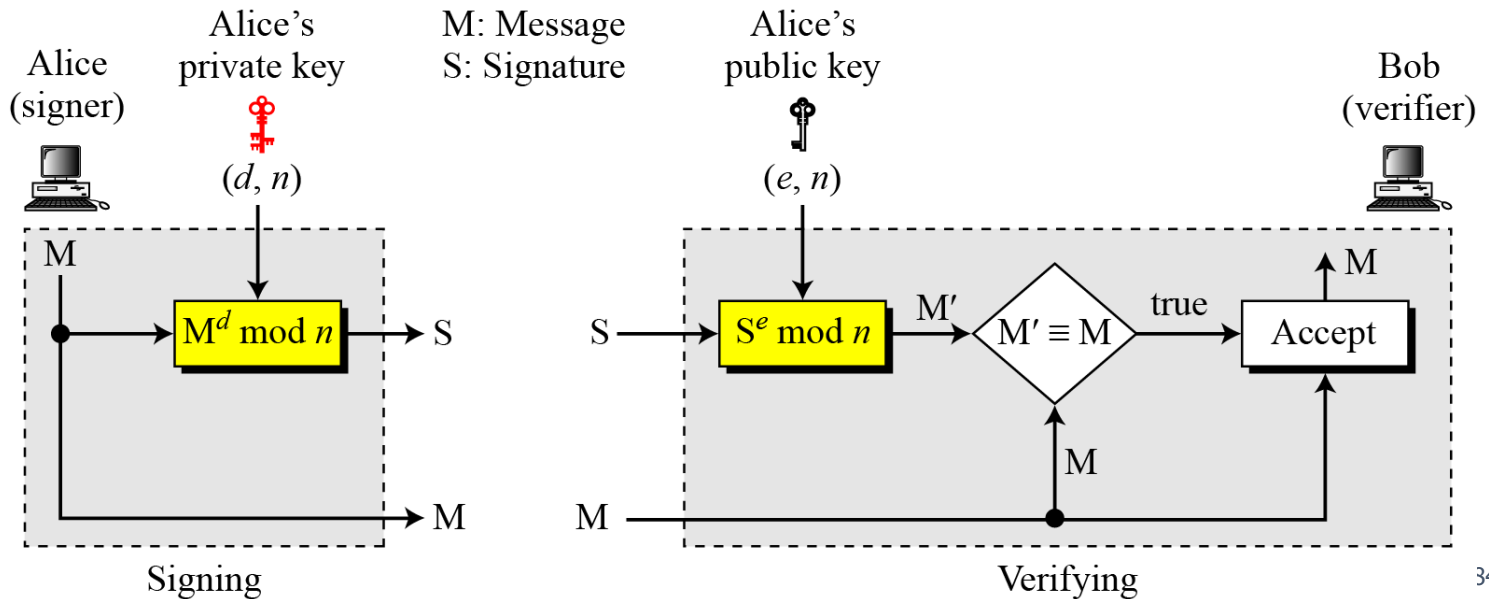
Example Use Example

- Sign and encrypt
 - Remember the hybrid encryption approach?
 - The encryption key can be a random session key to be encrypted with the receiver's public key
 - So that only the receiver can decrypt



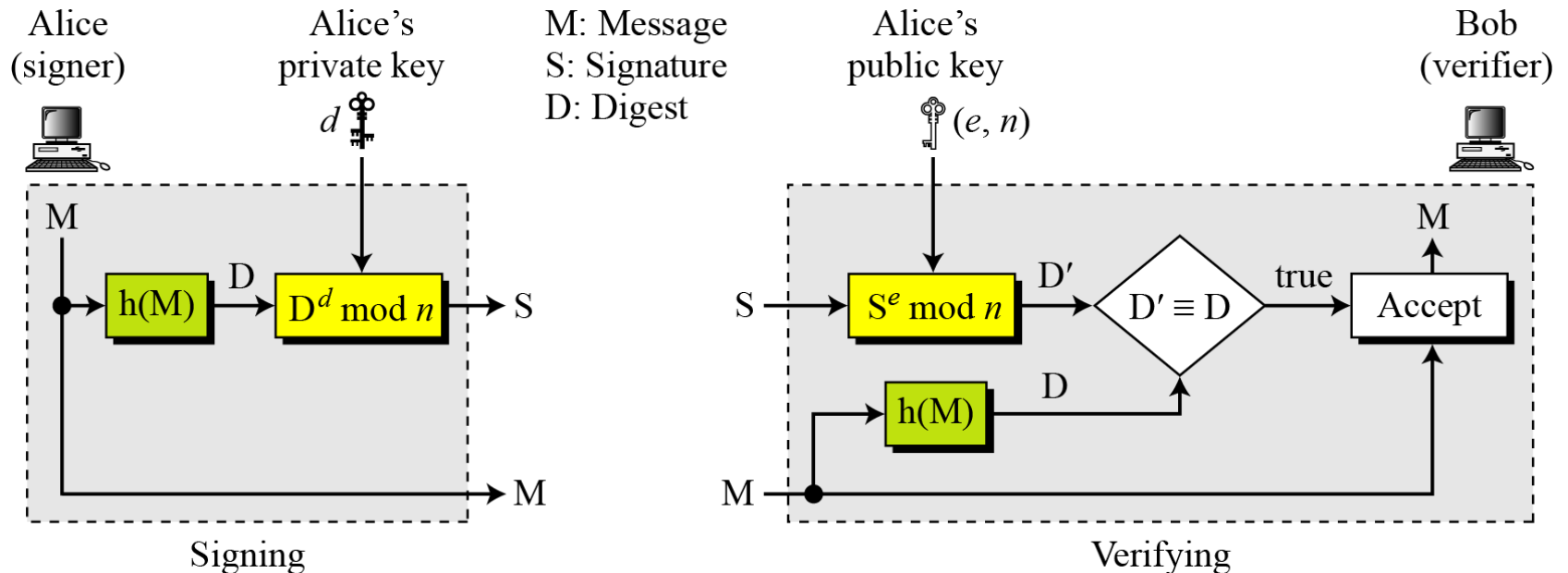
RSA Digital Signature

- RSA keys:
 - d is private; e and n are public
- Sign on the message directly
 - The signature is large in size



RSA Digital Signature

- Sign on the hash of the message to reduce the size of the signature
 - Security of the digital signature also depends on the strength of the chosen hash algorithm



Other Digital Signatures

- There are other digital signatures scheme
 - El Gamal digital signature
 - Similar to the Diffie-Hellman scheme and it relies on the difficulty of solving the discrete logarithm problem
 - Digital Signature Standard (DSS)
 - United States government standard for authentication of electronic documents
 - Use the algorithm called Digital Signature Algorithm (DSA) which is a variant of El Gamal
 - People dispute whether RSA digital signature is better than DSS or vice versa
 - comparing speed in key generation, signing and verification
 - Elliptic Curve Digital Signature Algorithm (ECDSA)
 - a variant of the Digital Signature Algorithm (DSA) which uses elliptic curve cryptography.

Your mission



- Understand the concept of hashing
- Understand the concept of MAC
- Understand the concept of digital signature

