Cryptographic Hash Functions and Message Authentication Codes

Tecnologie Internet

a.a. 2022/2023

Summary

- Hash Function
- Application: Message Authentication
- Security Requirements
- General Structure of Secure Hash Function
- Secure Hash Algorithm (SHA)
- Message Authentication Code (MAC)

Hash Function

A **hash function** H accepts a variable-length block of data *M* as input and produces a fixed-size hash value $h = H(M)$.

A **cryptographic hash function** is an algorithm for which it is computationally infeasible to find either

- a data object that maps to a pre-specified hash result (**one-way property**)

or

- two data objects that map to the same hash result (**collision-free property**).

Because of these characteristics, hash functions are often used to determine whether or not data has changed.

Message Authentication with Hash Functions

Message authentication is a mechanism used to verify the **integrity** of a message. Message authentication assures that data received are exactly as sent (i.e., there is no modification, insertion, deletion, or replay).

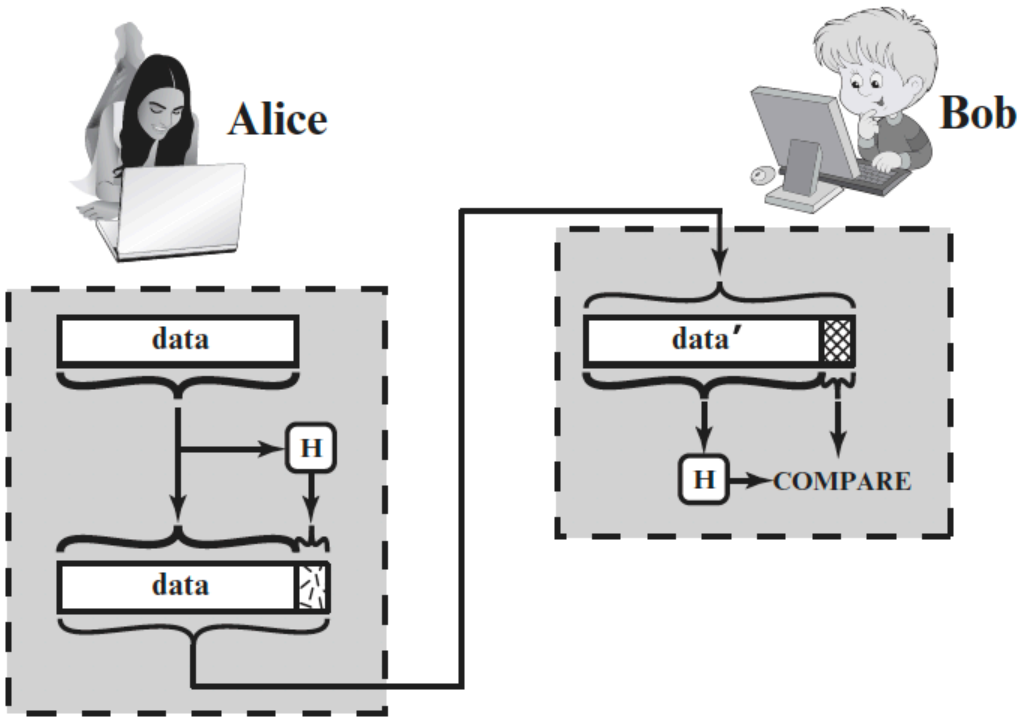
When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest**.

If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered.

Note: authentication techniques can be applied to a file in storage to assure that the file is not tampered with.

Message Authentication with Hash Functions

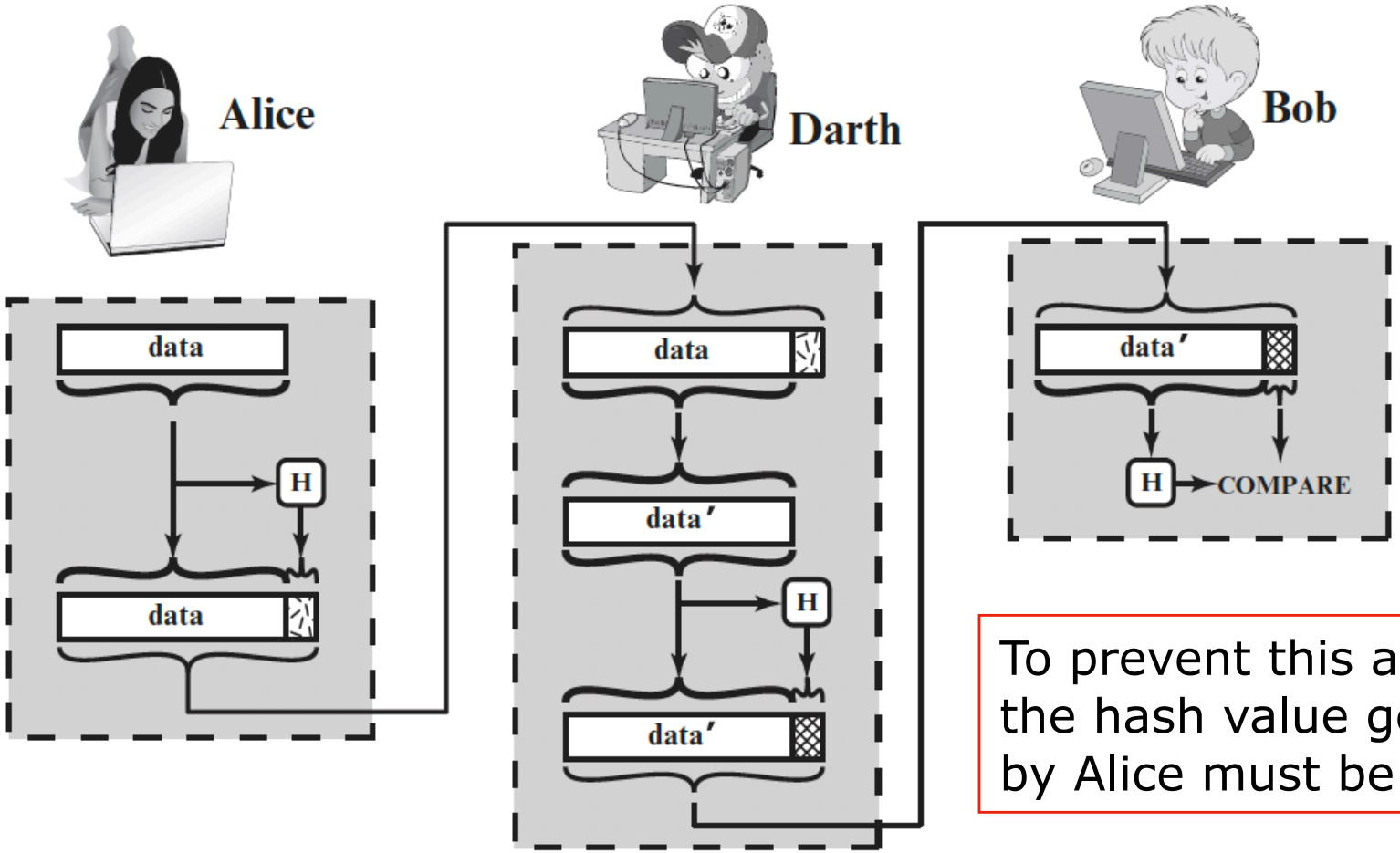
Ideally:



(a) Use of hash function to check data integrity

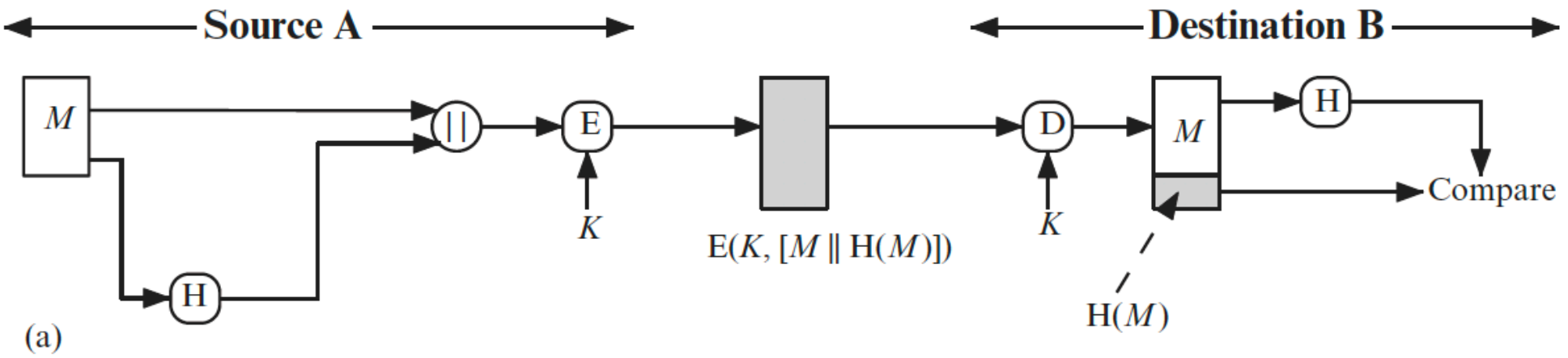
Message Authentication with Hash Functions

Man-in-the-middle attack:

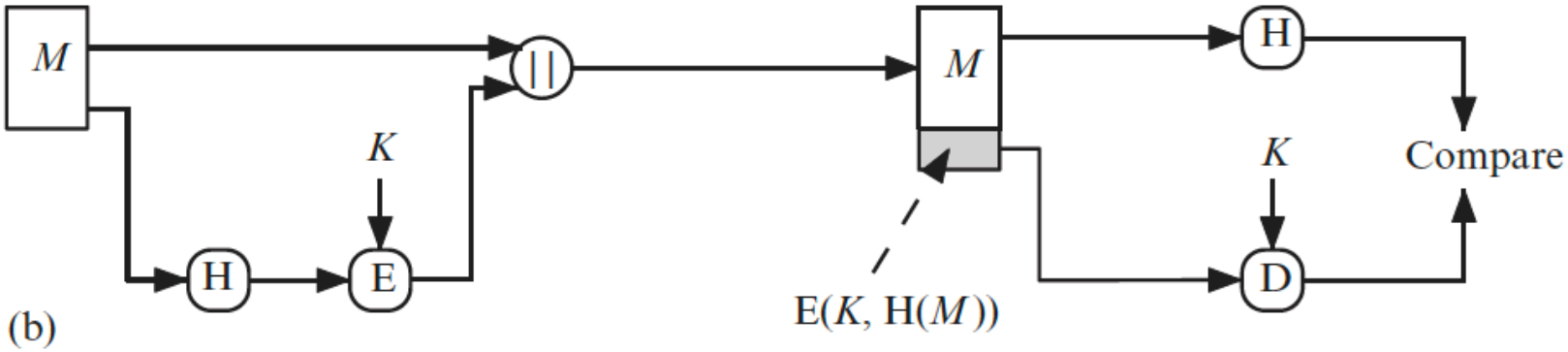


To prevent this attack, the hash value generated by Alice must be protected.

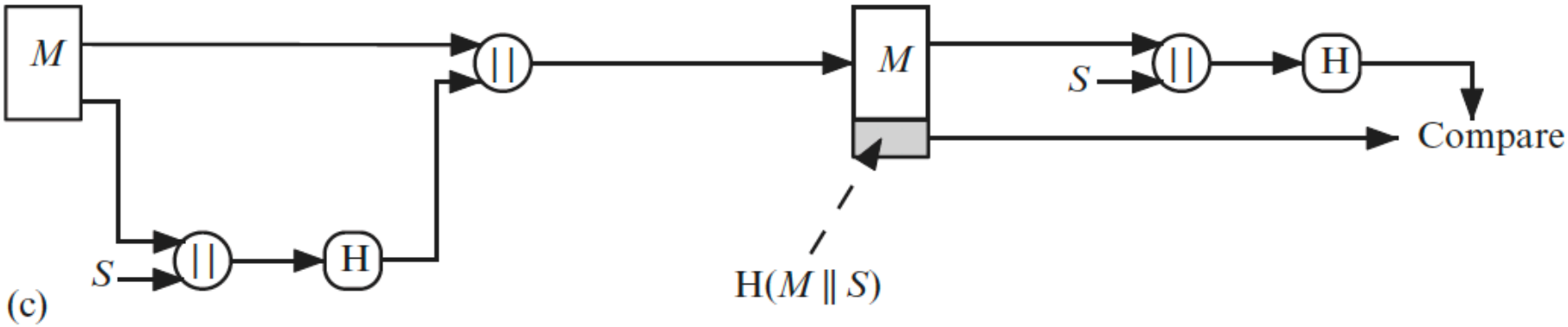
Message Authentication with Hash Functions



Message Authentication with Hash Functions

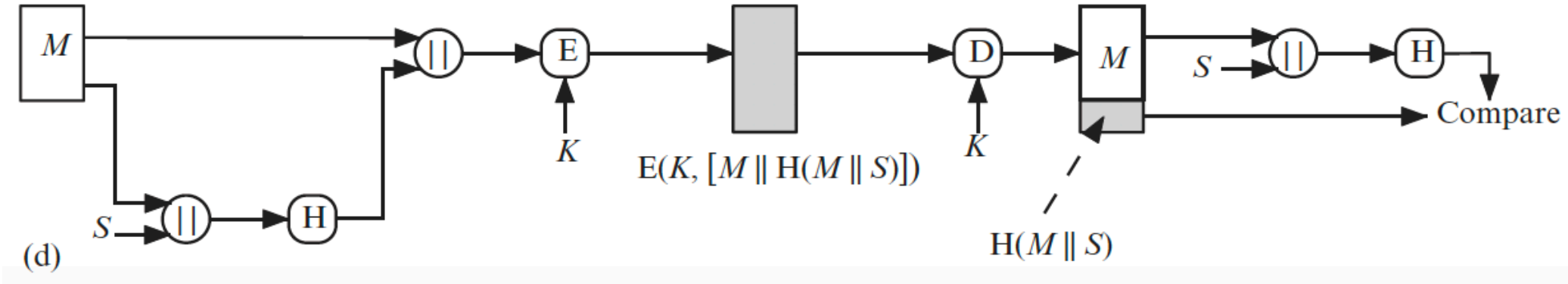


Message Authentication with Hash Functions



S = shared secret

Message Authentication with Hash Functions



S = shared secret

Security Requirements

For a hash value $h=H(x)$, we say that x is the **preimage** of h.

Because H is a many-to-one mapping, for any given hash value h, in general there will be multiple preimages.

A **collision** occurs if we have $x \neq y$ and $H(x)=H(y)$.

The length of the hash code is fixed, suppose n bits.

If we allow inputs of arbitrary length, not just a fixed length of some number of bits, then the number of preimages per hash value is arbitrarily large.

However, the security risks in the use of a hash function are not as severe as they might appear from this analysis.

Security Requirements

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) with $x \neq y$, such that $H(x) = H(y)$.
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

Security Requirements

Property 4 (**preimage resistant**) is important if the authentication technique involves the use of a secret value.
If the hash function is not one way, an attacker can easily discover the secret value.

If property 5 (**second preimage resistant**) were not true, an attacker would be capable of the following sequence:
1) observe or intercept a message plus its encrypted hash code;
2) generate an unencrypted hash code from the message;
3) generate an alternate message with the same hash code.

If properties 1 to 5 are satisfied: **weak hash function**.

If also property 6 (**collision resistant**) is satisfied: **strong hash function**.

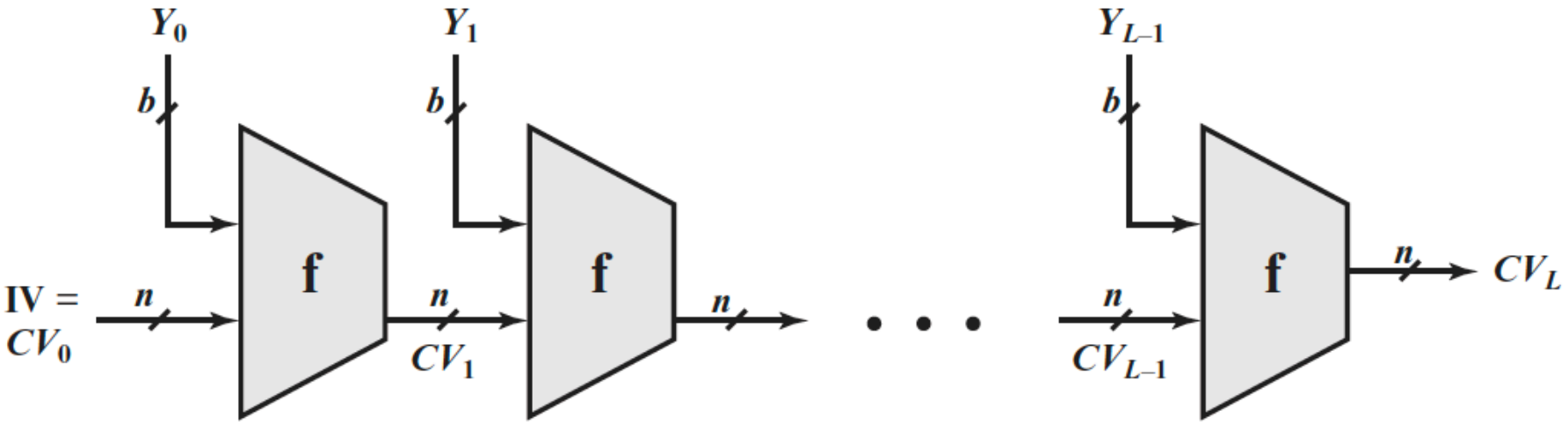
Security Requirements

Resistance properties required for various data integrity applications:

	Preimage Resistant	Second Preimage Resistant	Collision Resistant
Hash + digital signature	yes	yes	yes*
Intrusion detection and virus detection		yes	
Hash + symmetric encryption			
One-way password file	yes		
MAC	yes	yes	yes*

General Structure of Secure Hash Function

The structure, referred to as an iterated hash function, was proposed by Merkle.



IV = Initial value
CV_i = Chaining variable
Y_i = ith input block
f = Compression algorithm

L = Number of input blocks
n = Length of hash code
b = Length of input block

General Structure of Secure Hash Function

The hash algorithm involves repeated use of a **compression function**, *f*, that takes two inputs (an *n*-bit input from the previous step, called the chaining variable, and a *b*-bit block) and produces an *n*-bit output.

Often, ***b > n*** (hence the term compression).

The problem of designing a secure hash function reduces to that of designing a collision-resistant compression function that operates on inputs of some fixed size *b*.

Cryptanalysis of hash functions focuses on the internal structure of *f* and is based on attempts to find efficient techniques for producing collisions for a single execution of *f*.

Secure Hash Algorithm (SHA)

Published by NIST, SHA is the most widely used hash function.

1993: SHA-0 (has weaknesses)

1995: SHA-1 (considered insecure)

2002: SHA-256, SHA-384, SHA-512
2015: SHA-512/224, SHA-512/256 (SHA-2: secure, but.. until when?)

2015: **SHA-3** (different structure, highly secure)

SHA Standards

	Algorithm	Output size (bits) *n*	Internal state (bits)	Block size (bits) *b*	Word size (bits)	Rounds	Security $O(2^k)$
	SHA-1	160	160	512	32	80	$<63^{(*)}$
SHA-2	SHA-224	224	256	512	32	64	112
	SHA-256	256	256	512	32	64	128
	SHA-384	384	512	1024	64	80	192
	SHA-512	512	512	1024	64	80	256
SHA-3	SHA3-224	224	1600	1152	64	24	112
	SHA3-256	256	1600	1088	64	24	128
	SHA3-384	384	1600	832	64	24	192
	SHA3-512	512	1600	576	64	24	256
	SHAKE128	any	1600	1344	64	24	<128
	SHAKE256	any	1600	1088	64	24	<256

k=n/2

(*) SHA1 collision found (Feb, 2017)

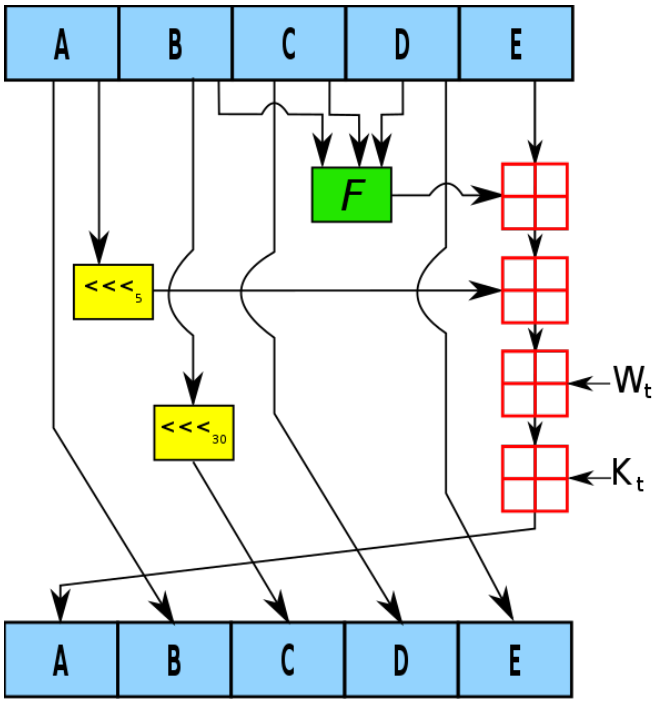
SHA-1

The (padded) input message is fragmented in 512-bit blocks.
Each 512-bit block has 16 words $X_0X_1..X_{15}$, each word being 32 bit long.

The algorithm uses a **160-bit internal state** (5 blocks, 32 bits each: *ABCDE*). Its initial value for processing the first block is always the same.

After 80 rounds, the resulting internal state is used for processing next block.

When the processing of the last block has been completed, the internal state is the actual hash.



$T = ROTL_5(A)+F_t(B,C,D)+E+K_t+W_t$
$E = D$
$D = C$
$C = ROTL_{30}(B)$
$B = A$
$A = T$
$\boxplus$ denotes addition modulo 2^{32}
<<<s denotes shift left by s

SHA-1

In the *t*-th round:

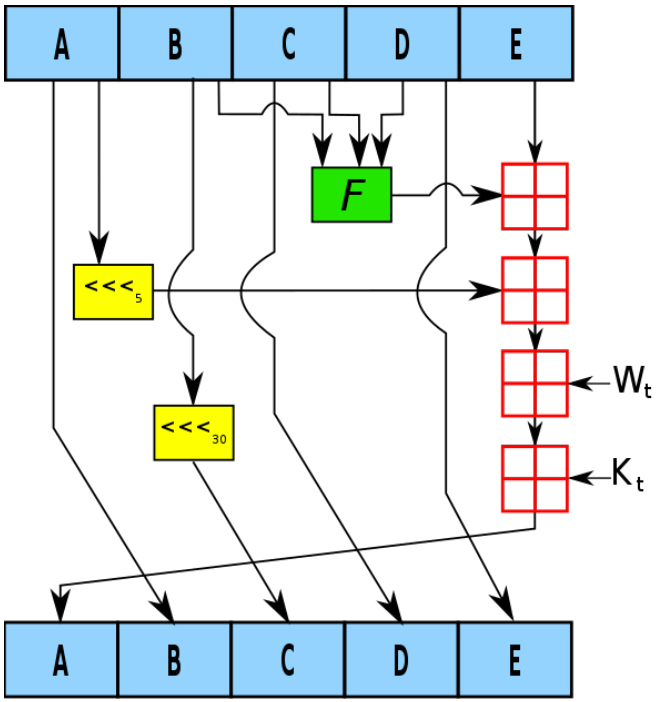
- derive W_t from $X_0X_1..X_{15}$
- use a different constant word K_t out of 4
- use a different function F_t out of 4:

– $Ch(x,y,z)=(x \wedge y) \oplus (!x \wedge z)$ $0 \leq t \leq 19$

– $Parity(x,y,z)=x \oplus y \oplus z$ $20 \leq t \leq 39$

– $Maj(x,y,z)=(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$ $40 \leq t \leq 59$

– $Parity(x,y,z)=x \oplus y \oplus z$ $60 \leq t \leq 79$



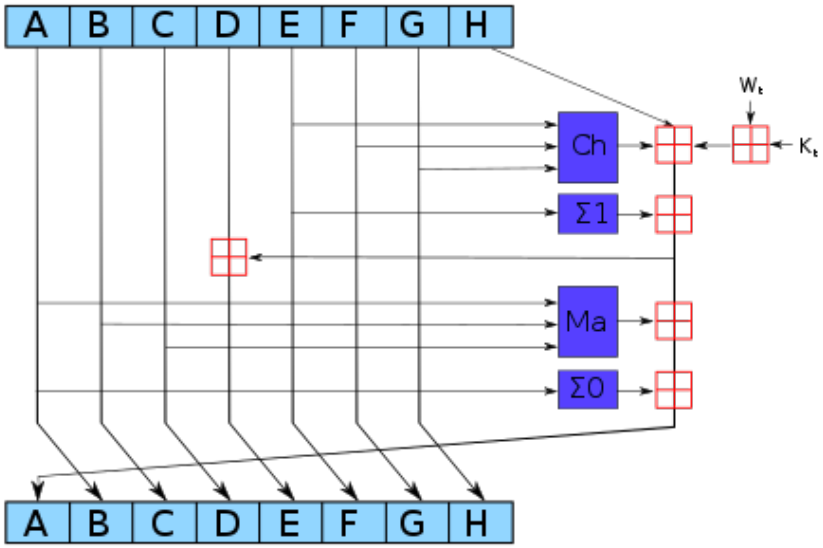
$T = ROTL_5(A)+F_t(B,C,D)+E+K_t+W_t$
$E = D$
$D = C$
$C = ROTL_{30}(B)$
$B = A$
$A = T$
$\boxplus$ denotes addition modulo 2^{32}
<<<s denotes shift left by s

SHA-2

SHA-256 and SHA-512 are computed with 32- and 64-bit words, respectively.

SHA-256 and SHA-512 perform 64 and 80 rounds, respectively.

SHA-224 and SHA-384 are simply truncated versions of the first two, computed with different initial values.



$T_1 = H + \Sigma1(E) + Ch(E,F,G) + K_t + W_t$
$T_2 = \Sigma0(A) + Maj(A,B,C)$
$H = G$
$G = F$
$E = D + T_1$
$D = C$
$C = B$
$B = A$
$A = T_1 + T_2$

$Ch(x,y,z) = (x \wedge y) \oplus (!x \wedge z)$
$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$
$\Sigma0(X) = ROTR_{s1}(X) \oplus ROTR_{s2}(X) \oplus ROTR_{s3}(X)$
$\Sigma1(X) = ROTR_{s4}(X) \oplus ROTR_{s5}(X) \oplus ROTR_{s6}(X)$

SHA-3

Hash function formerly called Keccak.

It supports the same hash lengths as SHA-2.

Its internal structure differs significantly from the rest of the SHA family.

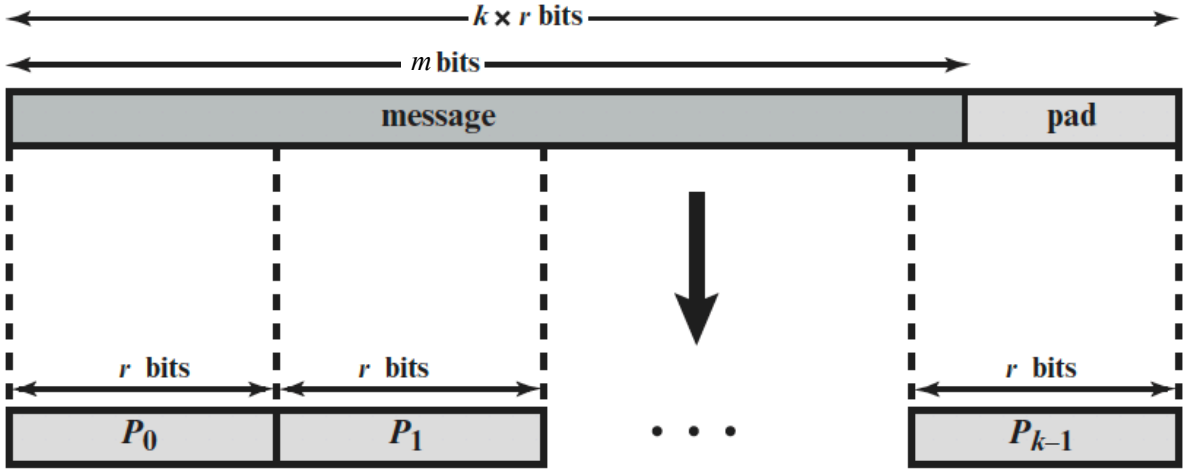
It is a **cryptographic sponge function**, i.e., a variable-length input variable-length output function based on a fixed length transformation or permutation *f* operating on a fixed number *b* of bits (the width).

f operates on a state of $b = r + c$ bits

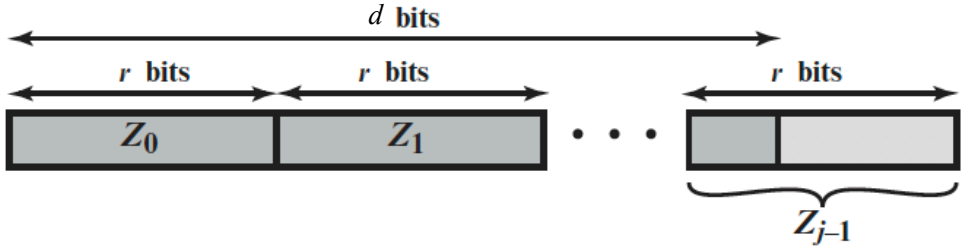
- the value *r* is called the **bitrate** and the value *c* the **capacity**
- default values for Keccak: $r = 576$ bits, $c = 1024$ bits --> $b=1600$

SHA-3

Sponge function input and output:



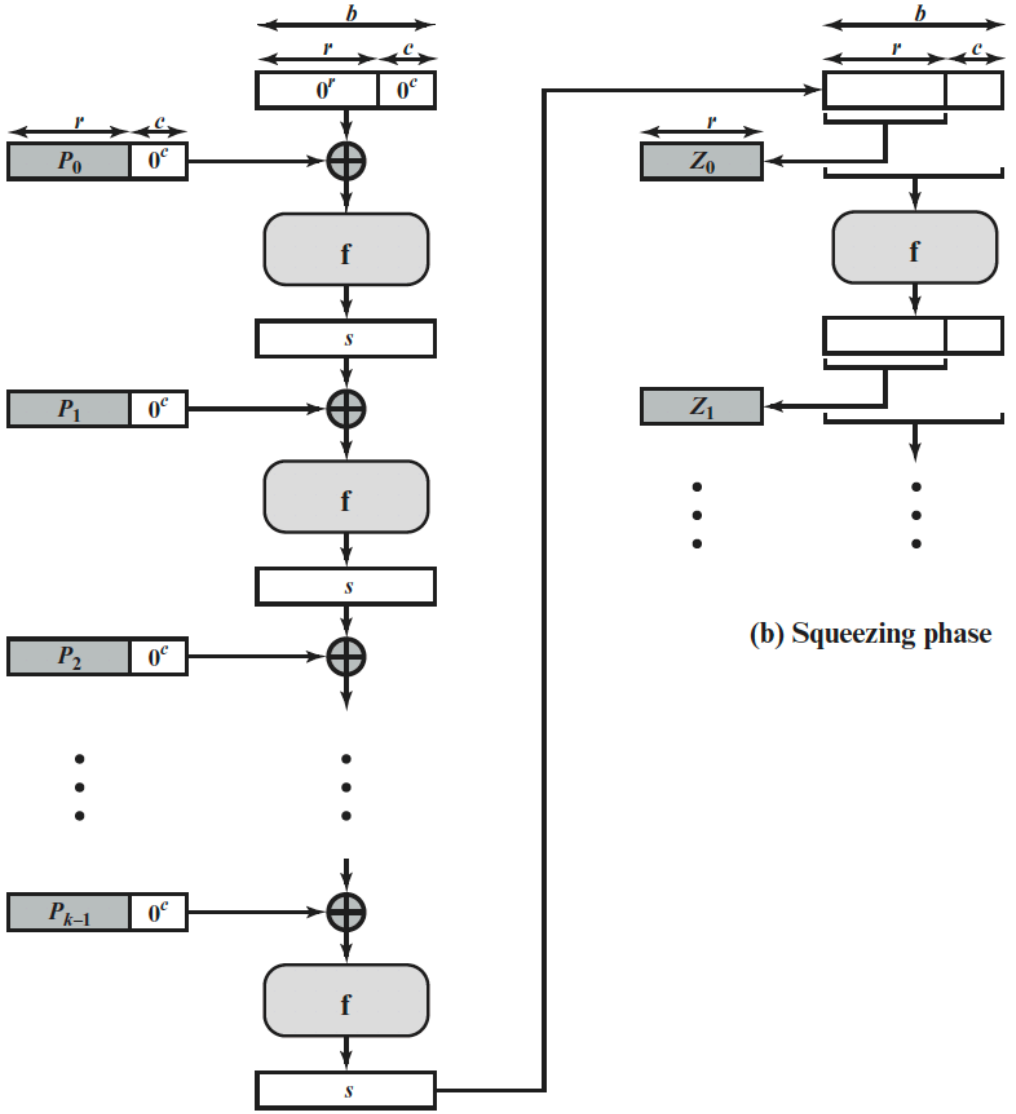
(a) Input



(b) Output

SHA-3

Sponge construction:

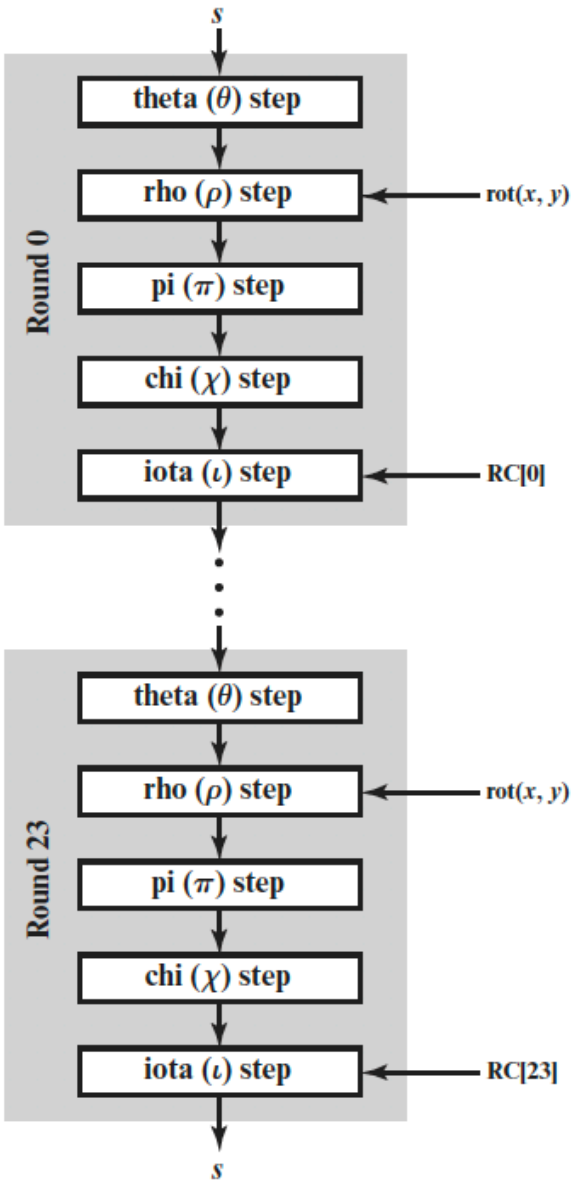


(a) Absorbing phase

(b) Squeezing phase

SHA-3

f in SHA-3:



Function	Type	Description
θ	Substitution	New value of each bit in each word depends on 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.

- *s* is organized as a 5 x 5 x 64 array *a*, according to the following mapping:

$$a[x,y,z] = s[64(5y+x)+z]$$

- RC[i] is a round constant

- rot*(x,y)* is the rotation value, which depends on the considered word

SHA-3

Sizes (in bits) and security levels:

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}

Note: $d < r$, thus squeezing is not necessary (just take the first d bits from s)

Message Authentication Code (MAC)

A MAC is a cryptographic checksum, generated by an algorithm that creates a small fixed-sized block, depending on both message and a secret key K:

$$MAC = C_K(M) = C(K,M)$$

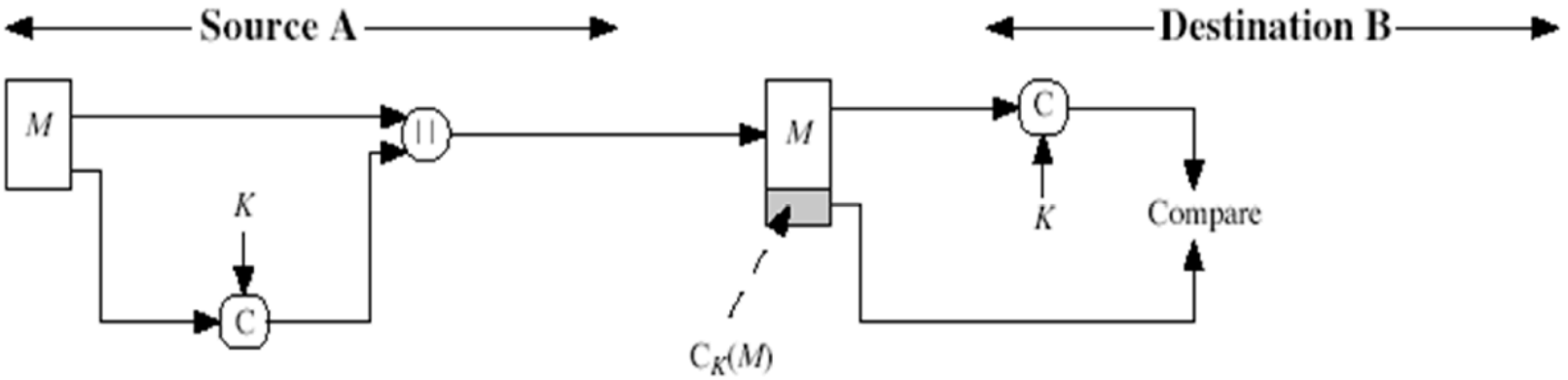
A MAC condenses a variable-length message M to a fixed-sized authenticator, which is appended to M as a **signature**.

Thus, the recipient of the message can verify that the sender of the message has the shared secret key and the no-one who doesn't know the secret key could have sent or altered the message.

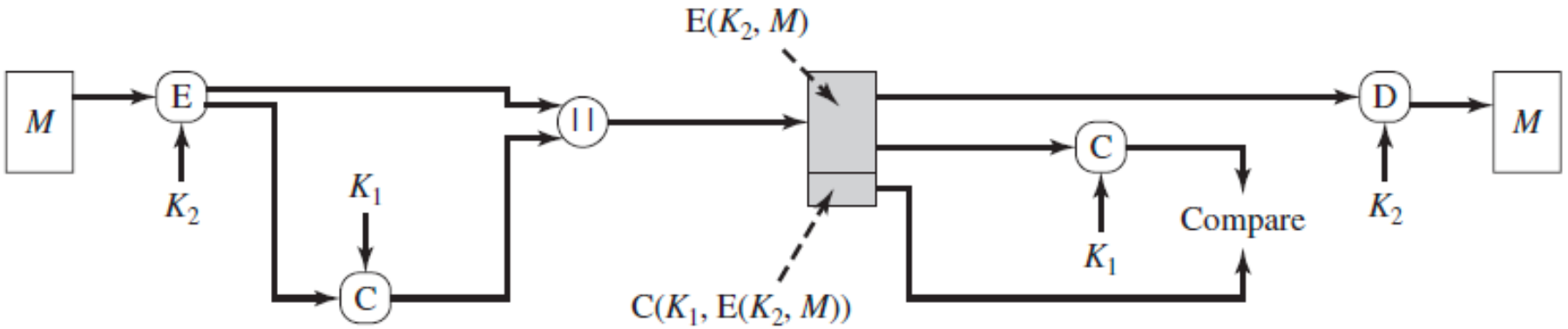
A MAC needs not be reversible and it is a many-to-one function: potentially many messages have same MAC, but finding these needs to be very difficult.

Message Authentication Code (MAC)

Authentication only:



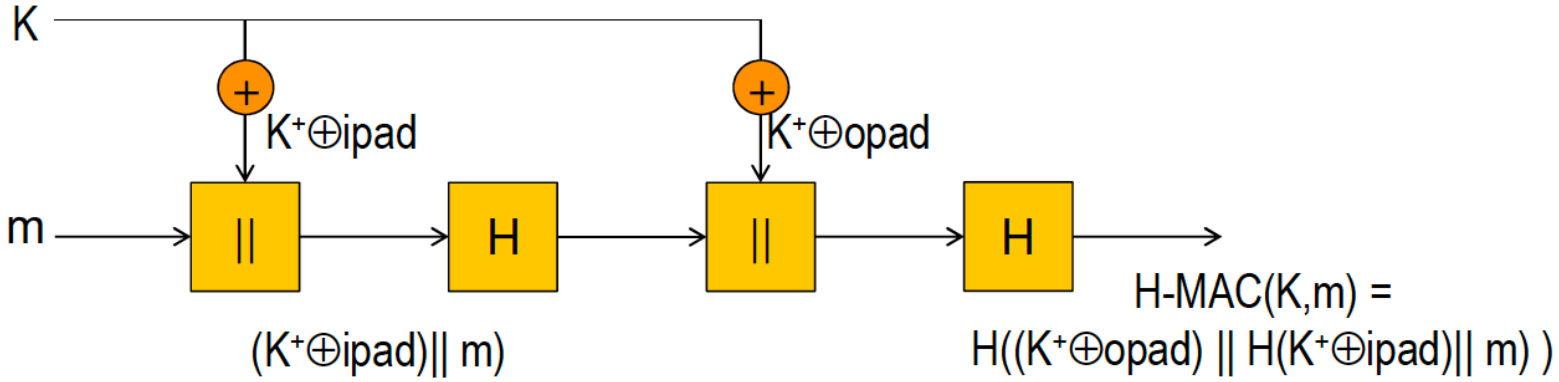
Authentication and confidentiality:



Message Authentication Code (MAC)

- Data Authentication Algorithm (**DAA**) is a MAC based on DES-CBC:
  - using IV=0 and zero-pad of final block
  - encrypt message using DES in CBC mode
  - and send just the final block as the MAC

- Hash Message Authetication CODE (**HMAC**):



K⁺ = the key, 0-padded out to size b
b = size of the processing block (512 bits in SHA)
ipad = the byte 0x36 repeated b/8 times
opad = the byte 0x5C repeated b/8 times

References

William Stallings, ***Cryptography and Network Security - Principles and Practice***, 7th edition, Pearson 2017