NT219- Cryptography

Week 09: Hash Function and Message Authentication Codes

PhD. Ngoc-Tu Nguyen

tunn@uit.edu.vn

Security goals

Goals

- Confidentiality
- Privacy

Cipher systems
- Symmetric (DES, AES)
- Asymmetric (RSA, ECC, CRYSTALS-KYBER)

- Integrity
- Authentication
- Non-repudiation (Accountability)

Hash functions
Message authentication code (MAC)
Digital signature (digital certificate)

- Access control

RBAC, ABAC, PBAC

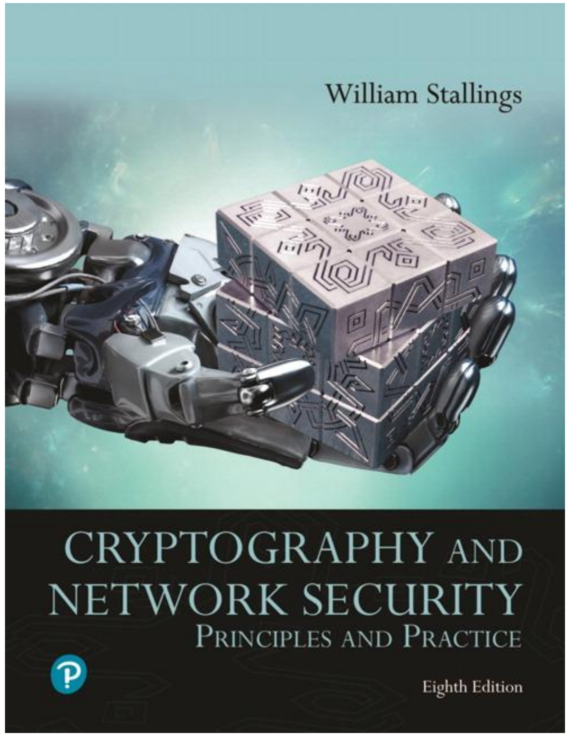
- Availability

Outline

- Motivations
- Hash function
  - CRC
- Cryptographic Hash function
  - SHA2, SHA3
- Message authentication code

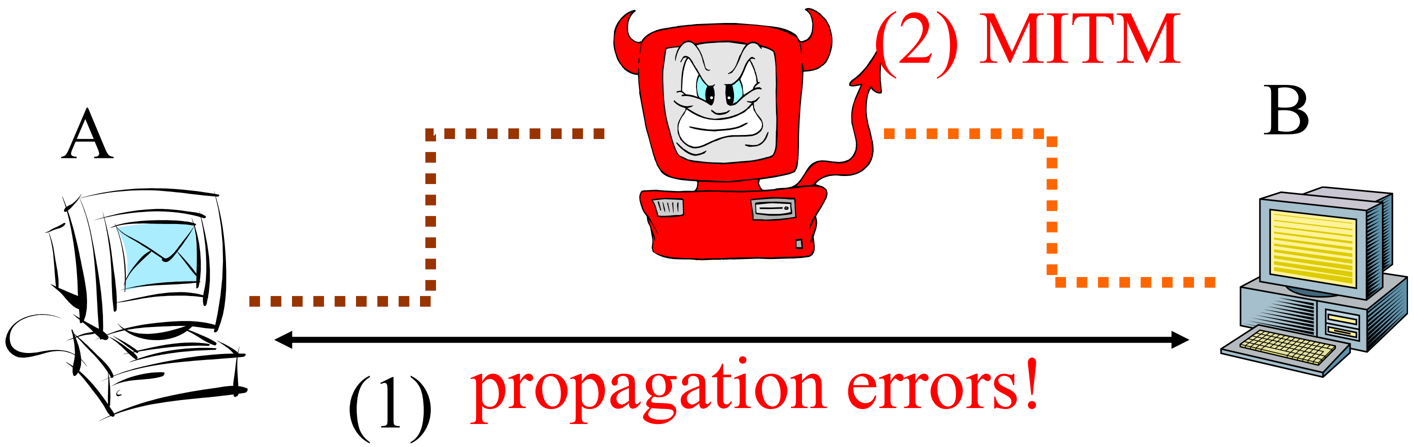
Textbooks and References

- Text books



[1] Chapter 11,12

Motivations



- Who are we communication with?
- Does the destination data original form?

Needs:

- a way to ensure that data arrives at destination in its **original form** sent by the sender;
- it is coming from an **authenticated source** (user, server, mediate node).

Motivations

$$\boldsymbol{Q_A}$$

$$(a, Q_A = aG) \qquad (b, Q_B = bG)$$

$$\boldsymbol{Q_B}$$

Alice

Bob

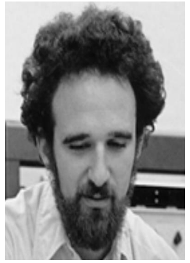
$$sk = aQ_B = bQ_A = abG$$

man-in-the-middle attack!

Need authentication entities

$$aG \qquad mG$$

$$mG \qquad bG$$



$$(a, Q_A = aG)$$

$$sk_1 = amG \qquad (m, Q_M = mG) \qquad sk_2 = bmG$$

$$(b, Q_B = bG)$$

Warmup: Man-in-the-Middle (MiTM) attacks

Need authentication entities

man-in-the-middle attack!

$\xrightarrow{aG}$

$\xleftarrow{mG}$

$(a, Q_A = aG)$

$sk_1 = amG$

$(m, Q_M = mG)$

$\xrightarrow{mG}$

$\xleftarrow{bG}$

$sk_2 = bmG$

$(b, Q_B = bG)$

https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/examples/sha256.pdf

Cryptographic Ciphers

Symmetric Cipher (DES, AES)
Asymmetric Cipher (RSA, ECC, ElGamal)

can provide which security services?

- Confidentiality ☑
- Privacy ☑

- Integrity ☒
- Authentication ☒
- Non-repudiation ☒
- Availability ☒

Hash function and MACs

- Hash function
- Message Authentication Codes (MAC)
- Digital signature
- Digital certificate

symmetric

asymmetric

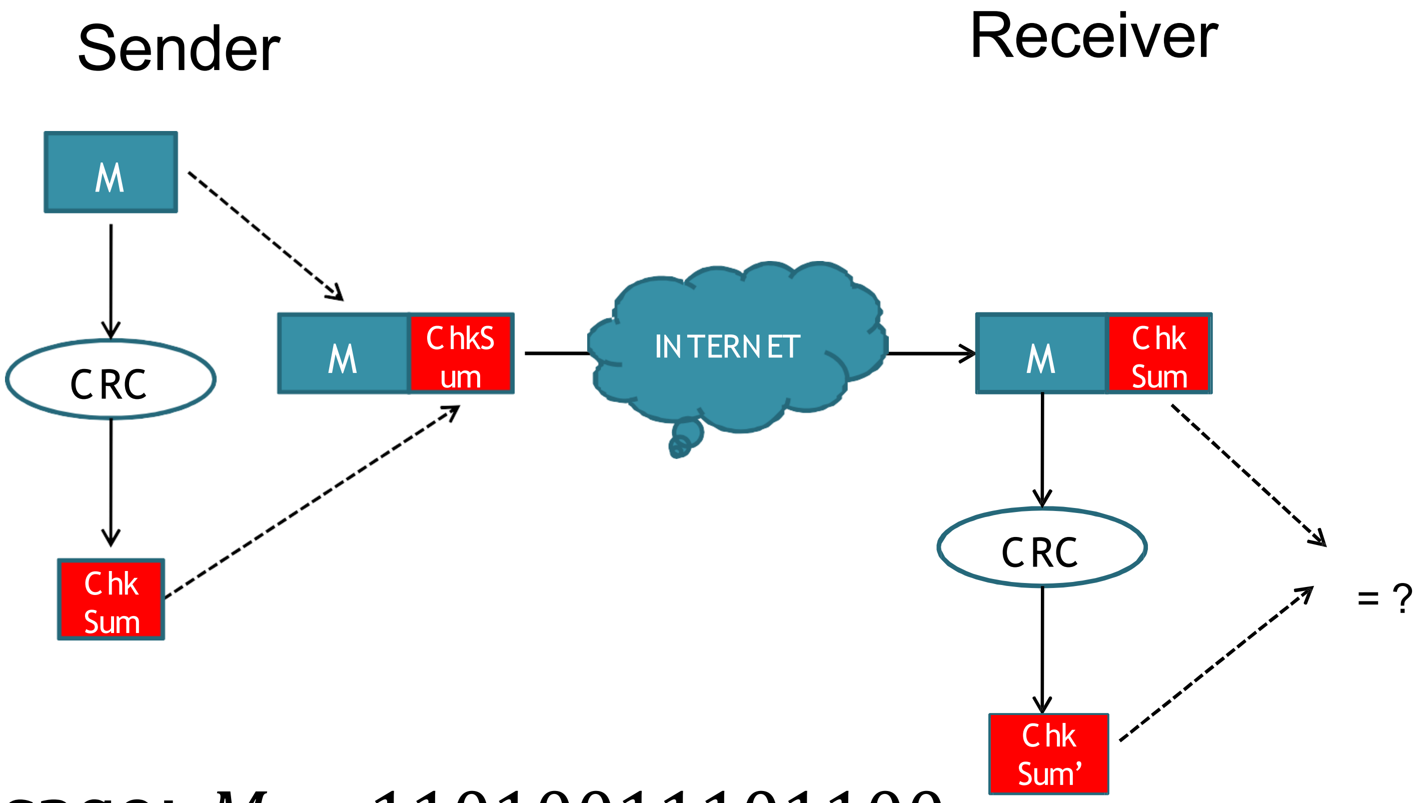
can provide

- Integrity ☑
- Authentication ☑
- Availability ☑
- Non-repudiation ☑

- Further reading (Wikipedia)
  - Cryptographic Hash Functions
  - Message Authentication Code

Hash function: An example

CRC Checksum in Networking



Message: $M = 11010011101100$

Checksum: $M(x)/x^3 + x + 1$

Hash function: An example

- CRC Checksum in Networking H: $\{0,1\}^* \rightarrow \{0,1\}^3$

Message: $M = 11010011101100$ (14 bits)

$M' = 11010011101100\ 000$

$$M'(x) = x^{16} + x^{15} + x^{13} + x^{10} + x^9 + x^8 + x^6 + x^5$$

Checksum: $\frac{M'(x)}{x^3+x+1}$

Quotient: $Q(x) = x^{13} + x^{12} + x^{11} + x^{10} + x^6 + x^5 + x^4 + x^3 + x^2$

Remider: $R(x) = x^2$

$$R(x) \; -> \; RCR \; = 100$$

Sender

(M||CRC)

11010011101100 100

Receiver

11010011101100 100

(M'||CRC)

Check: $\frac{M'(x)}{x^3+x+1}$=CRC?

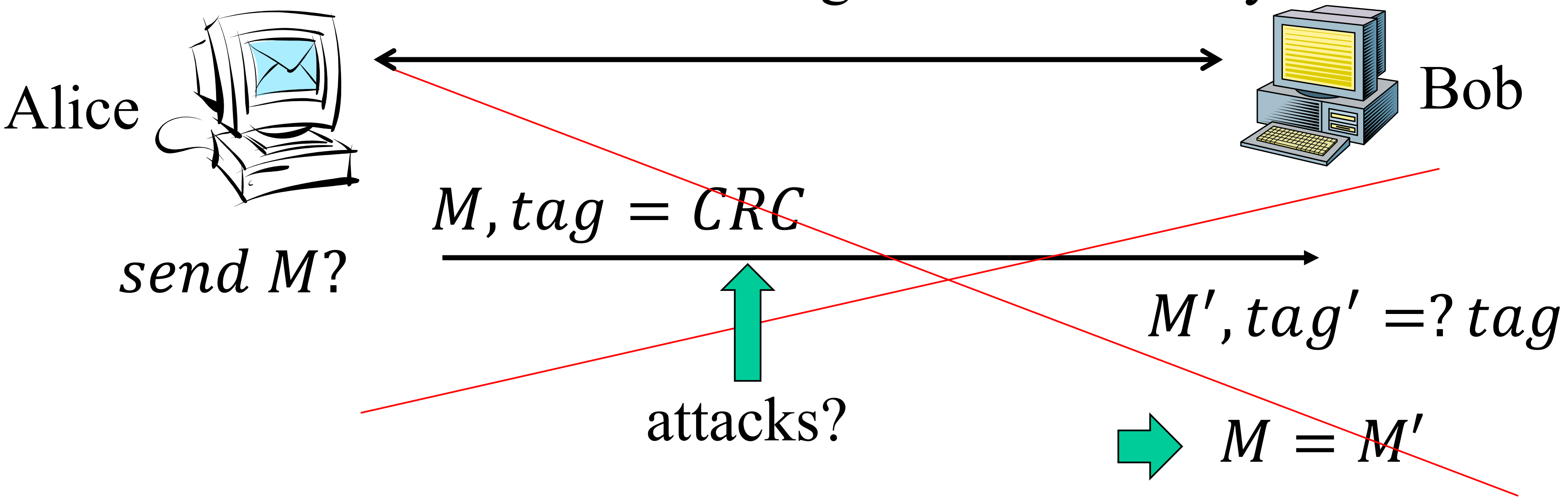
https://en.wikipedia.org/wiki/Cyclic_redundancy_check

Motivations

- How to ensure that the message is the original one?
    - Integrity?
- How to verify that a message comes from the claimed sender?
    - Authentication?

➢ Message Authentication Code (MAC)

If A and B can not agree a session key K?



Motivations

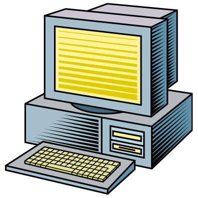
- How to ensure that the message is the original one?
    - Integrity?
- How to verify that a message comes from the claimed sender?
    - Authentication?

➢ MAC, HMAC

agree a session key K
(using Diffie–Hellman for ex.)

Elder

K $\longleftrightarrow$ K



Health care system server

$send\ M?$

$$M, tag = HMAC(K, M) = H(K, M)$$

$\longrightarrow$ M', tag

$$H(K, M') =? \, tag = H(K, M)$$

$$\Rightarrow M = M'$$

Hash Functions

- A hash function maps a message of an arbitrary length to a $l - bit$ output

$$H: \{0,1\}^* \rightarrow \{0,1\}^l$$

  - output known as the fingerprint or the message digest

- What is an example of hash functions?
  - Give a hash function that maps Strings to integers in $[0, 2^{32} - 1]$

- **Cryptographic hash functions** are hash functions with additional security requirements

Some terminology

- A short representation of *M* generated **without using secret** key is referred to as a *digital digest* or a *digital fingerprint*
- Digital fingerprint can be obtained using a *cryptographic hash function*, also called *one-way hash function*
- A short representation of M generated using a **secret key** is referred to as a *message authentication code* (*MAC*) or a *tag*
- MAC can be obtained using an *encrypted checksum algorithm*
- *Hash-based message authentication code* (*HMAC*) is the combination of cryptographic hash function and encrypted checksum algorithm

$$HMAC = H(K, M)$$

Cryptographic Hash Functions

Given a function $h: X \to Y$, then we say that h is:

- preimage resistant (one-way):

  if given $y \in Y$, it is **computationally infeasible** to find a value $x \in X$ s.t. $h(x) = y$

- 2-nd preimage resistant (weak collision resistant):

  if given $x \in X$, it is computationally infeasible to find a value $x' \in X$, s.t. $x' \neq x$ and $h(x') = h(x)$

- collision resistant (strong collision resistant):

  if it is computationally infeasible to find two distinct values $x', x \in X$, s.t. $h(x') = h(x)$

Usages of Cryptographic Hash Functions

- Software integrity

File: windows_10_enterprise_x64_dvd_9058303.iso / Ubuntu 20.04

SHA1:0629BF04AA2A61E125EE6EDDF917DB471DCB8535 UBUNTO 20 ISO

- Timestamping
  - How to prove that you have discovered a secret on an earlier date without disclosing it?

$$T, h(T, S)?$$

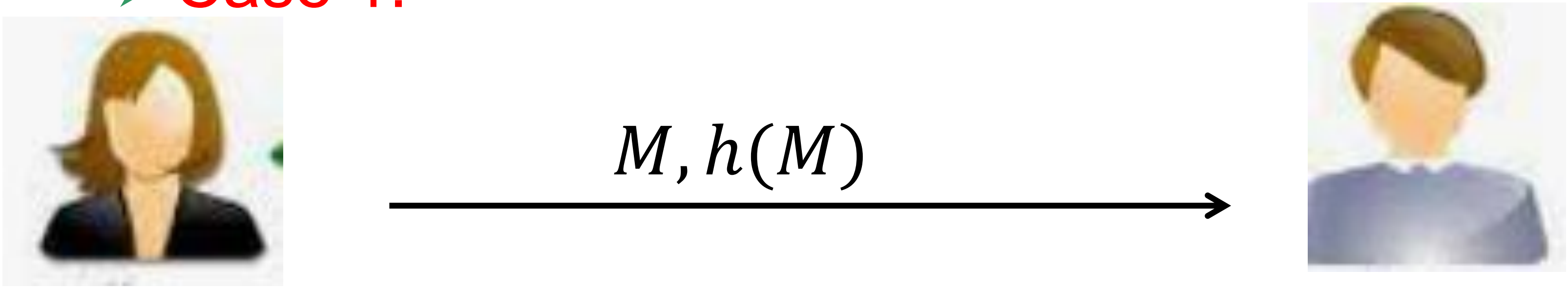
- **Covered later**
  - Message authentication
  - One-time passwords
  - Digital signature, Digital certificate

Further reading

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

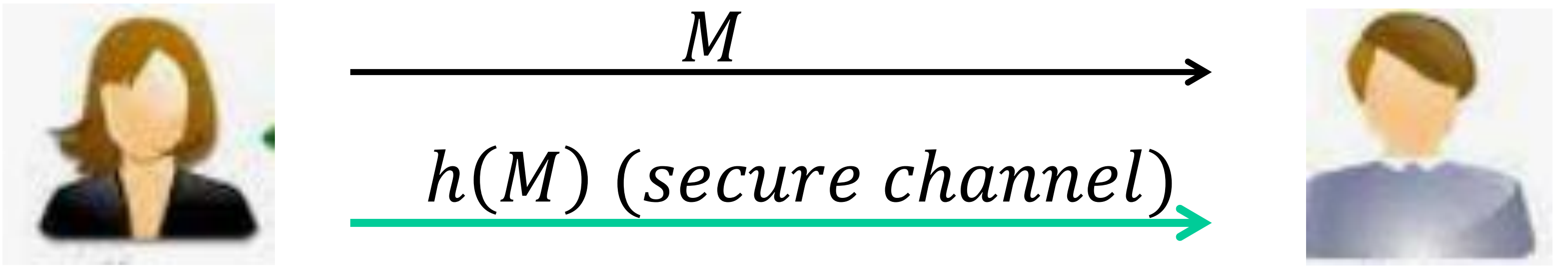
Using Hash Functions for Message Integrity

- Is this secure scheme (M cannot be modified)?
  - Case 1:



UBUNTO 20 ISO

  - Case 2:



  - Case 3:

Well Known Hash Functions

- MD5 (phased out)
  - output 128 bits
  - collision resistance completely broken
- SHA1 (phased out)
  - output 160 bits
  - collision attacks; length extension attacks
- SHA2 (SHA-224, SHA-256, SHA-384, SHA-512)
  - outputs 224, 256, 384, and 512 bits, respectively
  - collision attacks; length extension attacks
- SHA3

  https://csrc.nist.gov/publications/detail/fips/202/final
  https://en.wikipedia.org/wiki/Secure_Hash_Algorithm

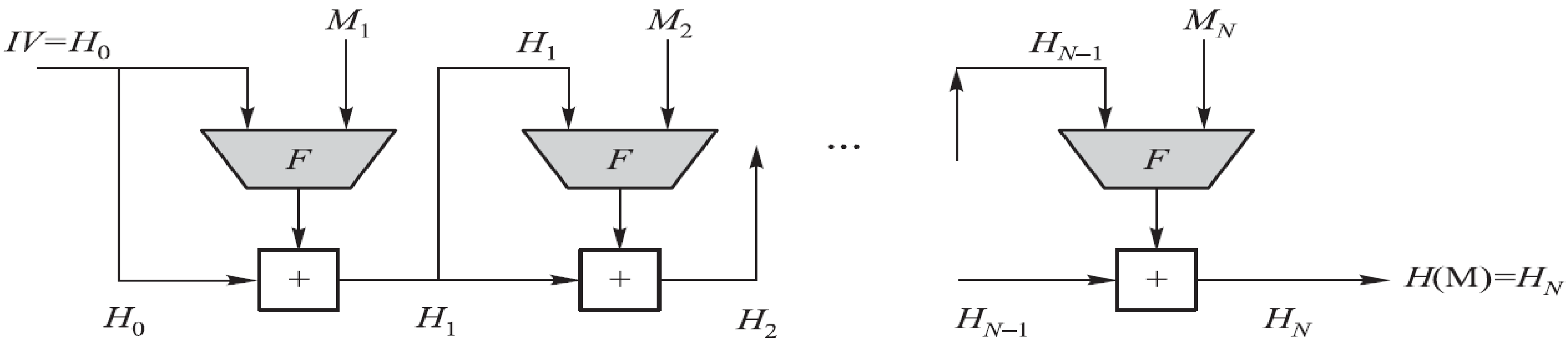
SHA-1, SHA-2, SHA-3

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Rounds	Operations	Security (in bits) against collision attacks	Capacity against length extension attacks
MD5 (as reference)		128	128 (4 × 32)	512	64	And, Xor, Rot, Add (mod 2^{32}), Or	≤18 (collisions found)[2]	0
SHA-0		160	160 (5 × 32)	512	80	And, Xor, Rot, Add (mod 2^{32}), Or	<34 (collisions found)	0
SHA-1							<63 (collisions found)[3]	
SHA-2	SHA-224 SHA-256	224 256	256 (8 × 32)	512	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112 128	32 0
	SHA-384 SHA-512	384 512	512 (8 × 64)	1024	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192 256	128 (≤ 384) 0
	SHA-512/224 SHA-512/256	224 256					112 128	288 256
SHA-3	SHA3-224 SHA3-256 SHA3-384 SHA3-512	224 256 384 512	1600 (5 × 5 × 64)	1152 1088 832 576	24[4]	And, Xor, Rot, Not	112 128 192 256	448 512 768 1024
	SHAKE128 SHAKE256	d (arbitrary) d (arbitrary)		1344 1088			min(d/2, 128) min(d/2, 256)	256 512

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

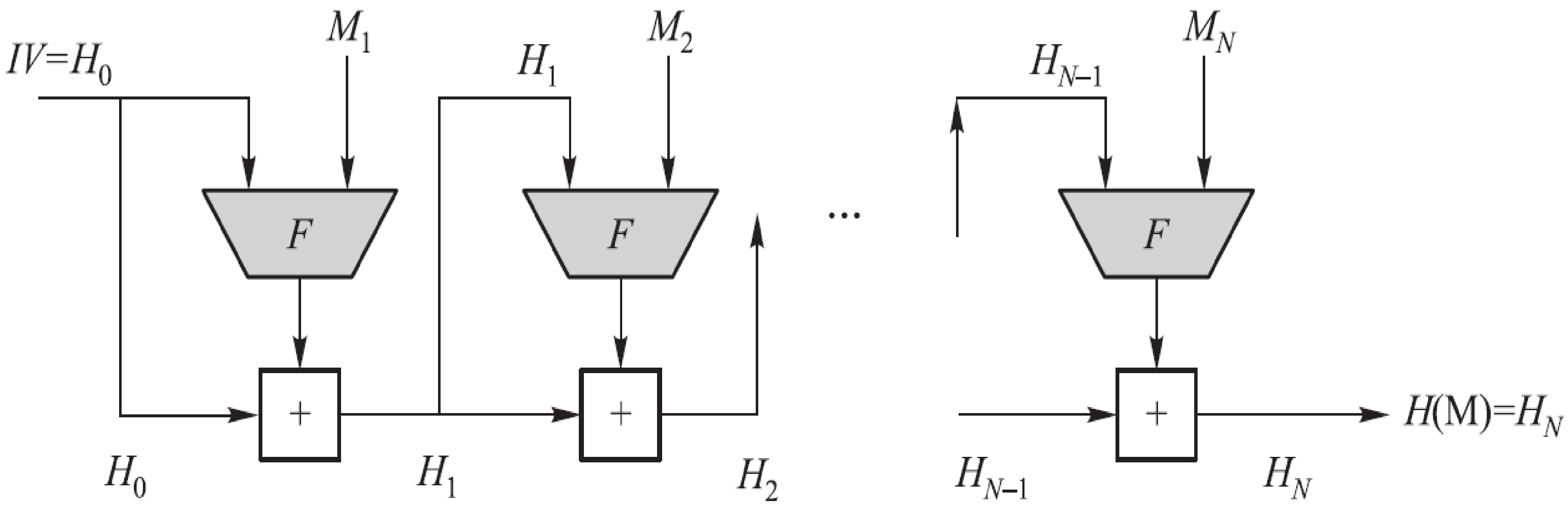
Merkle-Damgard Construction for Hash Functions

- SHA-1, SHA-2 (a series of hash functions), and WHIRLPOOL all have the same basic structure
- The heart of this basic structure is a *compression function F*
  - Different hash algorithms use different compression functions
  - Use a CBC mode of repeated applications of F without using secret keys



M is a plaintext block, IV is an initial vector, F is a compression function, and "+" is some form of modular addition operation

Merkle-Damgard Construction for Hash Functions



- The M's digital fingerprint is $H(M) = H_N$, where

$$\begin{aligned} H_0 &= \mathrm{IV}, \\ H_i &= H_{i-1} \oplus_{64} F(M_i, H_{i-1}), \\ i &= 1, 2, \ldots, N. \end{aligned}$$

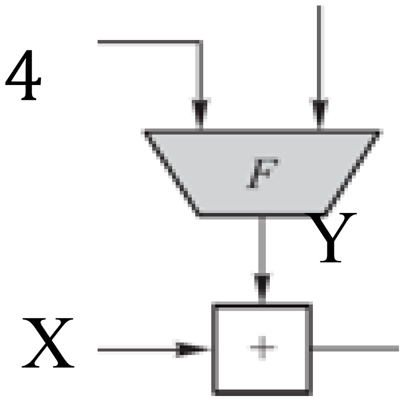
SHA-512

SHA-512 Algorithm

- Let $X = X_1X_2\ldots X_k$, $Y = Y_1Y_2\ldots Y_k$ be binary strings, where each X_i, Y_i is an l-bit binary string. Generalize the bitwise-XOR operation to an l-bitwise-XOR operation as follows:

$$X \oplus_l Y = [(X_1 + Y_1) \bmod 2^l][(X_2 + Y_2) \bmod 2^l]\cdots[(X_k + Y_k) \bmod 2^l]$$

For SHA-512: $l = 64$



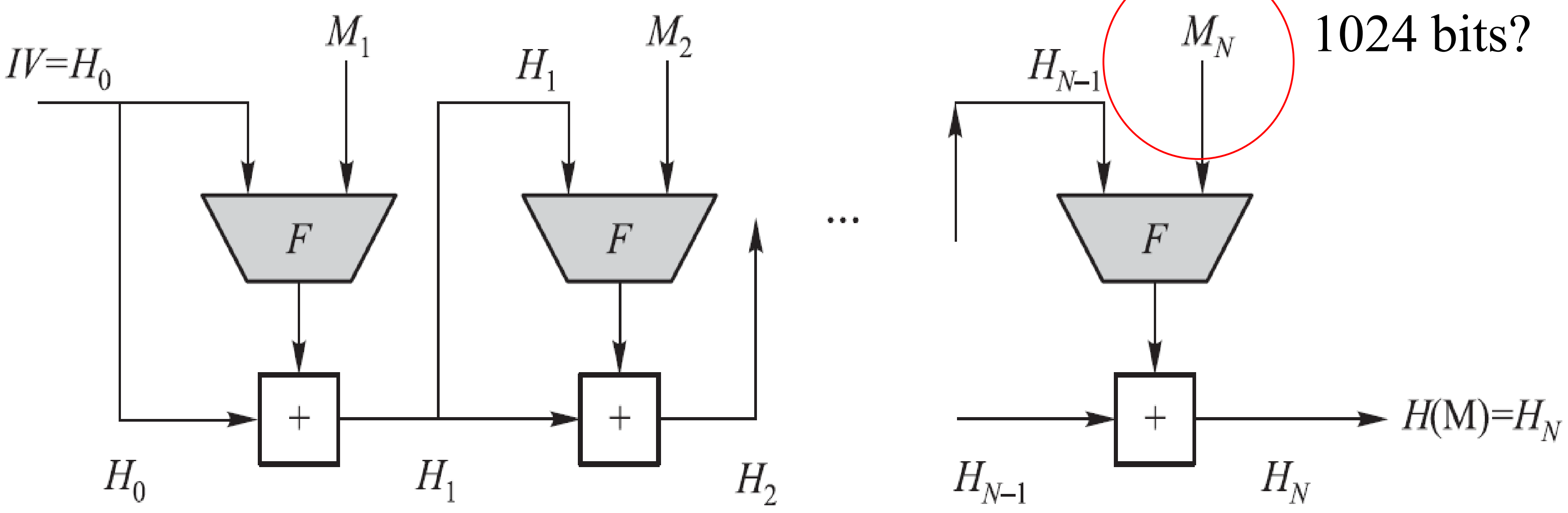
- Padding?
- Initial vector $IV = H_0$?
- Function F?

SHA-512 Initial Process (I)

Padding process

$\mathbf{M} \longrightarrow \mathbf{M'}= M_1M_2\ldots M_N$

M_i:1024-bit block



- **Length(M)=L**
- $\mathbf{M' = M \,||\, 1(0^{\ell}) \,||\, b_{128}(L)}$**, where** $\ell \geq 0$

SHA-512 Initial Process (I)

Padding process

Example: M=abc $\longrightarrow$ **M' (1024 bits)**

- **Length(M)=24**
- **M' = M || 1(0^{ℓ}) || b_{128}(L), where ℓ =1024 -24 -1 -128 = 871**

$$\underbrace{01100001}_{\text{“a”}}\quad \underbrace{01100010}_{\text{“b”}}\quad \underbrace{01100011}_{\text{“c”}}\quad 1\quad \overbrace{00\ldots00}^{871}\quad \overbrace{00\ldots0\underbrace{11000}_{L\,=\,24}}^{128}$$

SHA-512 Initial Process (II)

- Set $\Gamma = 2^{128} - 1$ and $\gamma = 512$
- M is a binary with $|M| = L \leq \Gamma$
- Represent L as a 128-bit binary string, denoted by $b_{128}(L)$
- Pad M to produce a new binary string M' as follows:

$$\mathbf{M' = M \,||\, 1(0^{\ell}) \,||\, b_{128}(L), \text{ where } \ell \geq 0}$$

  such that $|M'|$ (denoted by L') is divisible by 1024. We have

$$L' = L + (1 + \ell) + 128 = L + \ell + 129 = L + (1024 - 895) + \ell$$

- L can be represented as

$$L = 1024 \cdot \left\lfloor \frac{L}{1024} \right\rfloor + \left(L \bmod 1024\right)$$

- Hence, ℓ can be determined as follows:

$$\ell = \begin{cases} 895 - L \bmod 1024, & \text{if } 895 \geq L \bmod 1024, \\ 895 + (1024 - L \bmod 1024), & \text{if } 895 < L \bmod 1024. \end{cases}$$

- Thus, L' is divisible by 1024. Let $L' = 1024N$ and write as a sequence of 1024-bit blocks: $M' = M_1 M_2 \ldots M_N$

Padding process

$$M \longrightarrow M'$$

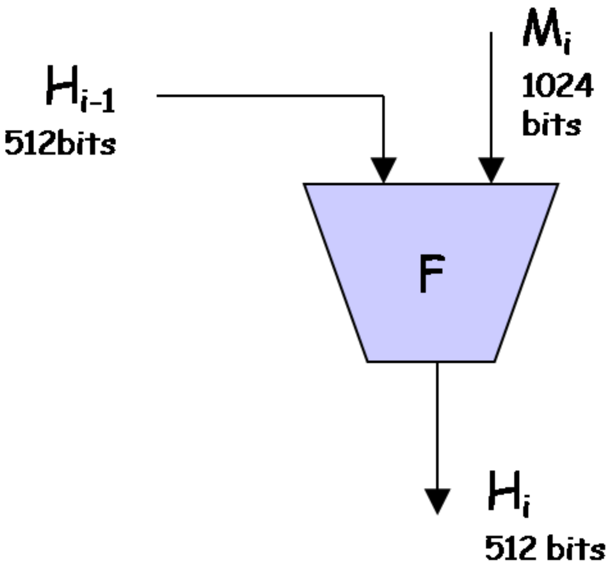
$$\mathbf{M' = M \,||\, 1(0^{\ell}) \,||\, b_{128}(L),}$$

where $\ell \geq 0$

$$M' = M_1 M_2 \ldots M_N$$

$$len(M_i) = 1024$$

SHA-512 Initial Process (II)



- SHA-512 uses a 512-bit IV
- Let r_1, r_2, r_3, r_4, r_5, r_6, r_7, and r_8 be eight 64-bit registers
  - Initially they are set to, respectively, the 64-bit binary string in the prefix of the fractional component of the square root of the first 8 prime numbers:

$$\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}, \sqrt{11}, \sqrt{13}, \sqrt{17}, \sqrt{19}$$

$$H^{(0)} = \text{IV}$$

$H_0^{(0)}$ = 6a09e667f3bcc908

$H_1^{(0)}$ = bb67ae8584caa73b

$H_2^{(0)}$ = 3c6ef372fe94f82b

$H_3^{(0)}$ = a54ff53a5f1d36f1

$H_4^{(0)}$ = 510e527fade682d1

$H_5^{(0)}$ = 9b05688c2b3e6c1f

$H_6^{(0)}$ = 1f83d9abfb41bd6b

$H_7^{(0)}$ = 5be0cd19137e2179

SHA-512 Compression Function (III)

$\wedge$	Bitwise AND operation.
$\vee$	Bitwise OR ("inclusive-OR") operation.
$\oplus$	Bitwise XOR ("exclusive-OR") operation.
$\neg$	Bitwise complement operation.
$+$	Addition modulo 2^w.
$<<$	Left-shift operation, where $x << n$ is obtained by discarding the left-most n bits of the word x and then padding the result with n zeroes on the right.
$>>$	Right-shift operation, where $x >> n$ is obtained by discarding the right-most n bits of the word x and then padding the result with n zeroes on the left.

SHA-512 Compression Function (III)

Bitwise operations

Define :

logical conjunction : $X \wedge Y = (x_1 \wedge y_1)(x_2 \wedge y_2)\cdots(x_l \wedge y_l)$

logical disjunction : $X \vee Y = (x_1 \vee y_1)(x_2 \vee y_2)\cdots(x_l \vee y_l)$

logical negation : $\overline{X} = \bar{x}_1\bar{x}_2\cdots\bar{x}_l$

conditional predicate : $ch(X,Y,Z) = (X \wedge Y) \vee (\overline{X} \wedge Z)$

majority predicate : $maj(X,Y,Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z)$

$$\Delta_0(r) = (r >>> 28) \oplus (r >>> 34) \oplus (r >>> 39)$$

$$\Delta_1(r) = (r >>> 14) \oplus (r >>> 18) \oplus (r >>> 41)$$

SHA-512 Compression Function (III)

- **Two inputs**:
  - a 1024-bit plaintext block M_i
  - a 512-bit string $H_{i\text{-}1}$, where $1 \leq i \leq N$

$$H_{i-1} = r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8$$

$$M_i = W_0 \cdot W_1 \cdots W_{15}, |W_i| = 64$$

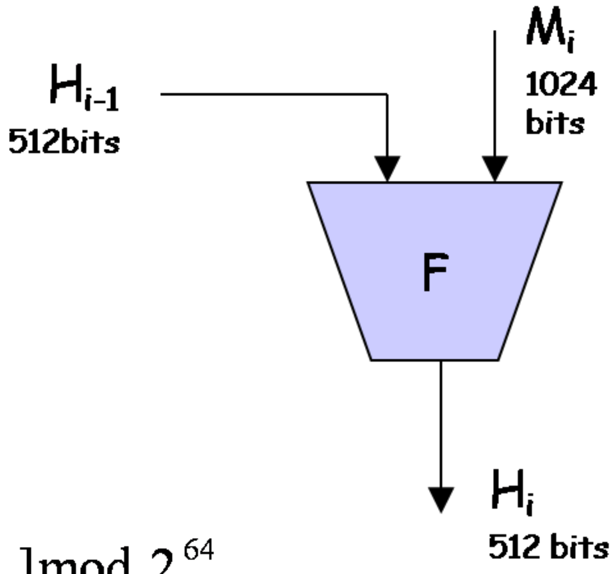
$$\text{generate } W_{16} \cdot W_{17} \cdots W_{79} \text{ as follows}$$

$$W_t = [\sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}] \bmod 2^{64}$$

$$t = 16, \cdots, 79,$$

$$\sigma_0(W) = (W >>> 1) \oplus (W >>> 8) \oplus (W << 7)$$

$$\sigma_1(W) = (W >>> 19) \oplus (W >>> 61) \oplus (W << 6)$$



$W>>>n$: **circularly right shift** W for n times

$W<<n$: **linearly left shift** W for n times (with the n-bit suffix of filled with 0's)

SHA-512 Compression Function (III)

$K_0^{\{512\}}, K_1^{\{512\}}, \ldots, K_{79}^{\{512\}}$: first sixty-four bits of the fractional parts of the cube roots of the first eighty prime numbers

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcbd41fbd4	76f988da831153b5
983e5152ee66dfab	a831c66d2db43210	b00327c898fb213f	bf597fc7beef0ee4
c6e00bf33da88fc2	d5a79147930aa725	06ca6351e003826f	142929670a0e6e70
27b70a8546d22ffc	2e1b21385c26c926	4d2c6dfc5ac42aed	53380d139d95b3df
650a73548baf63de	766a0abb3c77b2a8	81c2c92e47edaee6	92722c851482353b
a2bfe8a14cf10364	a81a664bbc423001	c24b8b70d0f89791	c76c51a30654be30
d192e819d6ef5218	d69906245565a910	f40e35855771202a	106aa07032bbd1b8
19a4c116b8d2d0c8	1e376c085141ab53	2748774cdf8eeb99	34b0bcb5e19b48a8
391c0cb3c5c95a63	4ed8aa4ae3418acb	5b9cca4f7763e373	682e6ff3d6b2b8a3
748f82ee5defb2fc	78a5636f43172f60	84c87814a1f0ab72	8cc702081a6439ec
90befffa23631e28	a4506cebde82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273eceea26619c	d186b8c721c0c207	eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a	5fcb6fab3ad6faec	6c44198c4a475817

SHA-512 Compression Function (III)

$$H_{i-1} = r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8, W_0, W_1, \dots, W_{79}, K_0, K_1, \dots, K_{79}$$

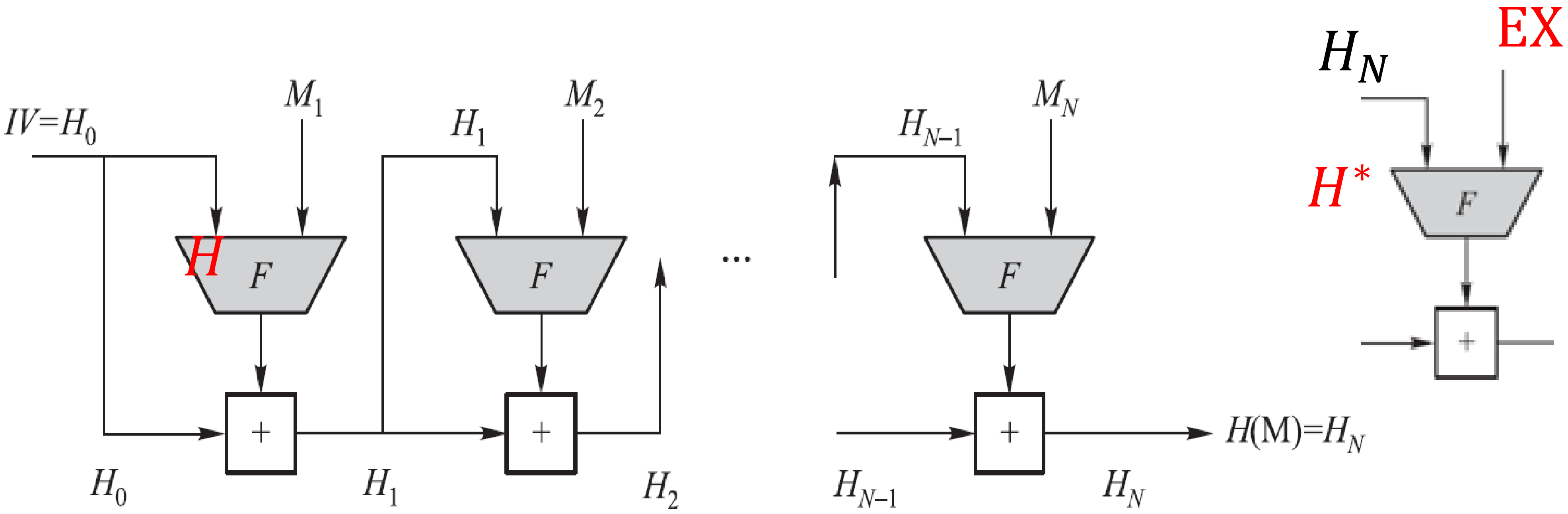
For each i is executed 80 rounds: t=0,1,2,..79

$$T_1 \leftarrow [r_8 + ch(r_5, r_6, r_7) + \Delta_1(r_5) + W_t + K_t] \bmod 2^{64},$$
$$T_2 \leftarrow [\Delta_0(r_1) + maj(r_1, r_2, r_3)] \bmod 2^{64},$$
$$r_8 \leftarrow r_7,$$
$$r_7 \leftarrow r_6,$$
$$r_6 \leftarrow r_5,$$
$$r_5 \leftarrow (r_4 + T_1) \bmod 2^{64},$$
$$r_4 \leftarrow r_3,$$
$$r_3 \leftarrow r_2,$$
$$r_2 \leftarrow r_1,$$
$$r_1 \leftarrow (T_1 + T_2) \bmod 2^{64}.$$

After 80 rounds of executions, the output is 512-bit string

$$F(Mi, H_{i-1}) = r_1 r_2 r_3 r_4 r_5 r_6 r_7 r_8$$

Length extension attack on SHA2



$$\mathrm{H}(K||M) = \mathrm{H}(K||M||padded)$$

$$\Rightarrow \mathrm{H}(K||M||padded||\mathrm{EX}) = H^*(EX)$$

can compute $\mathrm{H}(M||padded||\mathrm{EX})$ **without knowing the input** $\boldsymbol{M}$

$$\mathrm{H}(M||K) \dashrightarrow OK$$

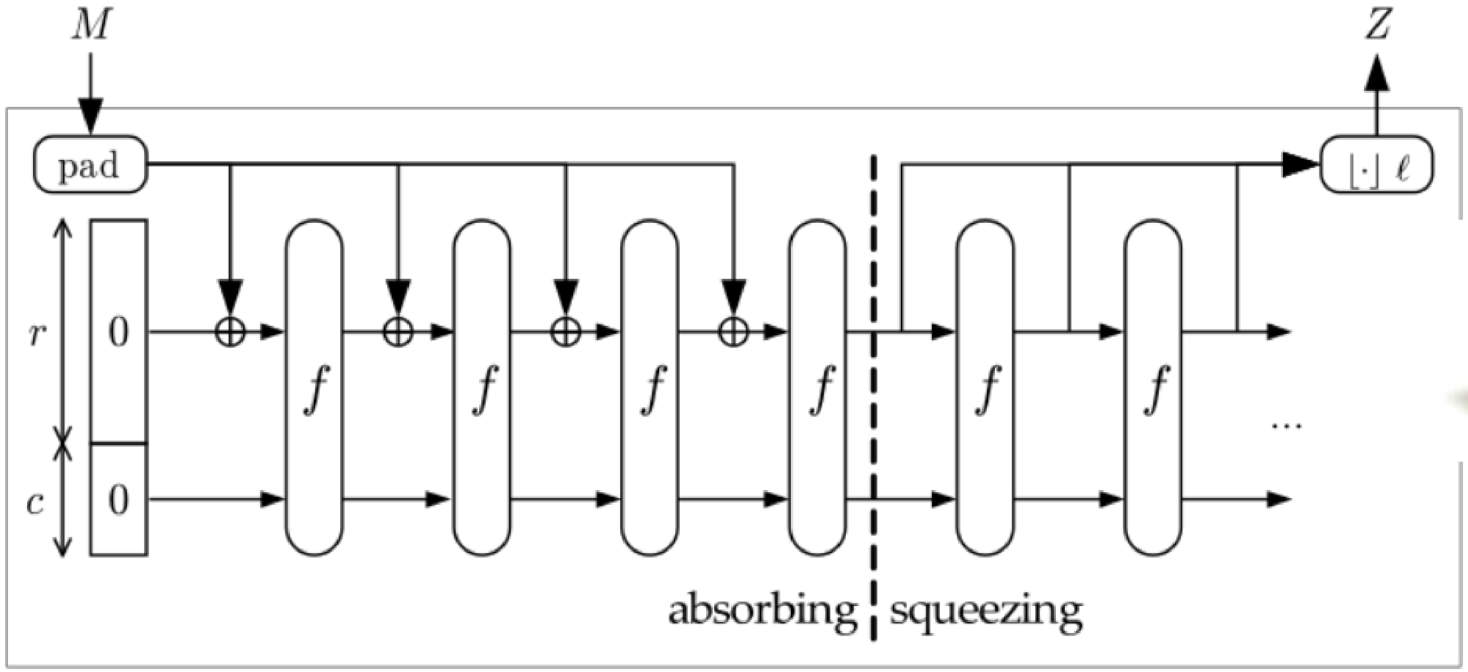
SHA3 Standard

- SHA-3 provides an alternative to SHA-2, and is drop-in compatible with any system using SHA-2

- SHA-3 uses a **sponge construction**, instead of the CBC mode of repeated compressions used by SHA-1, SHA-2, and Whirlpool

https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

SHA3 Standard



Sponge construction

- Let M be the input string; γ = the hash length.
- b = r + c, where c = 2γ
  - ✓ r is called **rate** and c **capacity**
- Where $b = 25 \times 2^{l}$ with $0 \le l \le 6$

  $b \in \{25, 50, 100, 200, 400, 800, 1600\}$