Week 10: Hash Function and Message Authentication Codes (P2)

PhD. Ngoc-Tu Nguyen

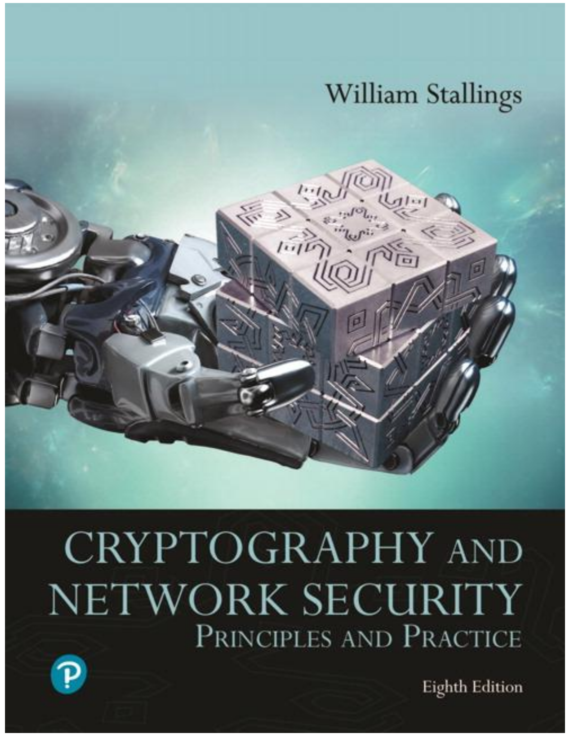
tunn@uit.edu.vn

Outline

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

Textbooks and References

- Text books



[1] Chapter 11,12

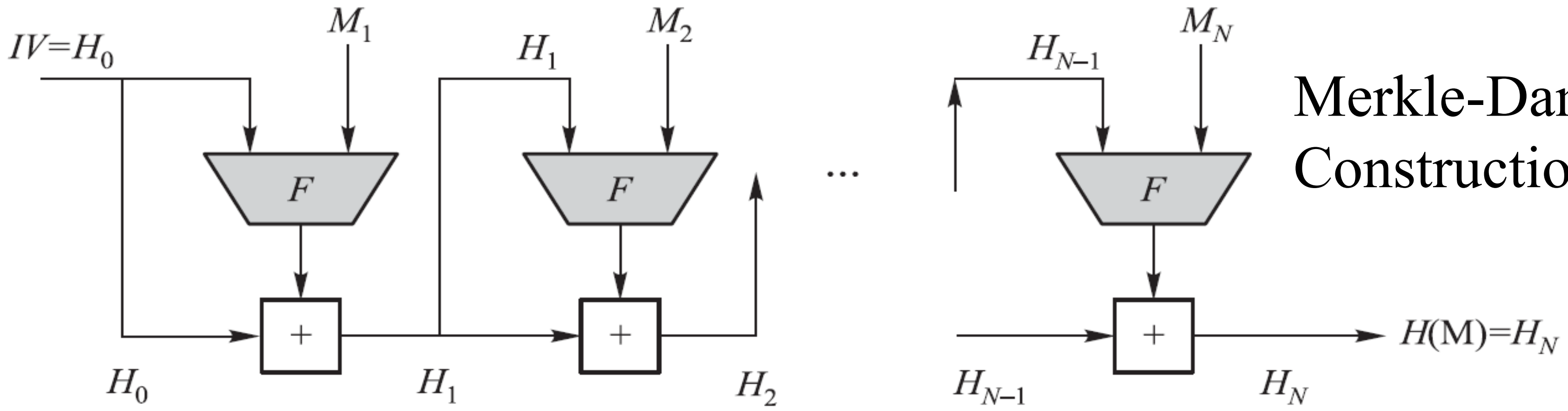
SHA-1, SHA-2, SHA-3

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_Algorithm

SHA-1, SHA-2

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
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

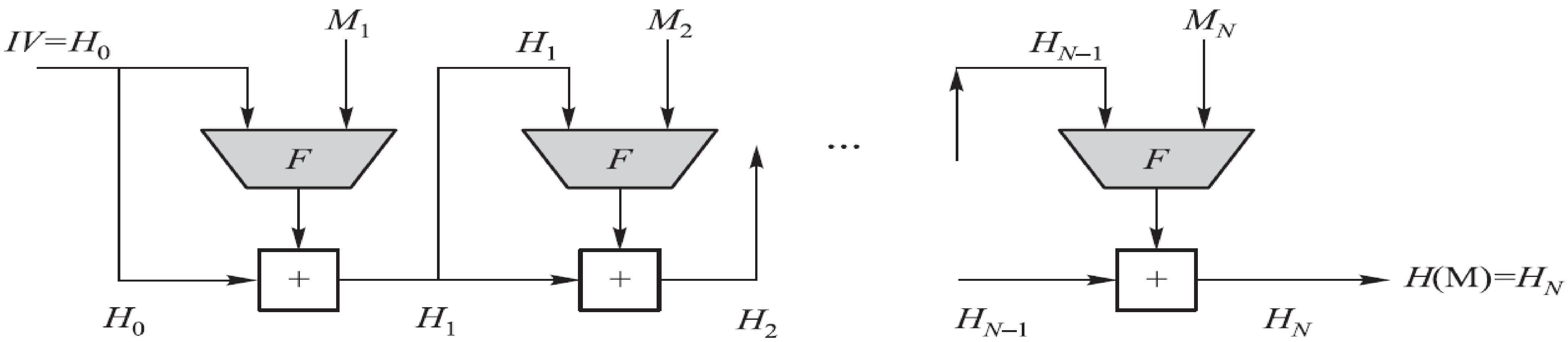


Merkle-Damgard Construction

https://csrc.nist.gov/publications/fips#fips180-4

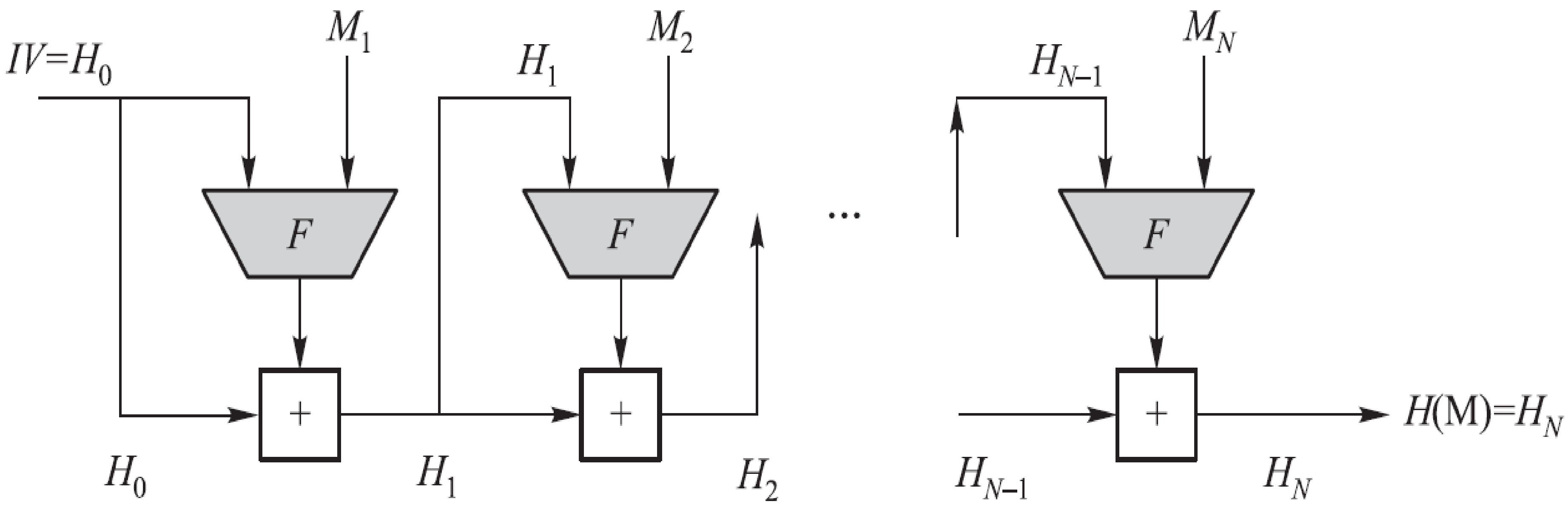
Merkle-Damgard Construction for Hash Functions

- 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

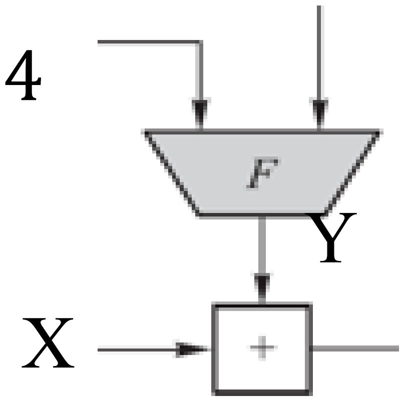
SHA-512 Algorithm

- 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 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.
$\ll$	Left-shift operation, where $x \ll n$ is obtained by discarding the left-most n bits of the word x and then padding the result with n zeroes on the right.
$\gg$	Right-shift operation, where $x \gg n$ 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 : $\overline{X} = \overline{x}_1\overline{x}_2\cdots\overline{x}_l$

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 $1 \le i \le N$

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

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

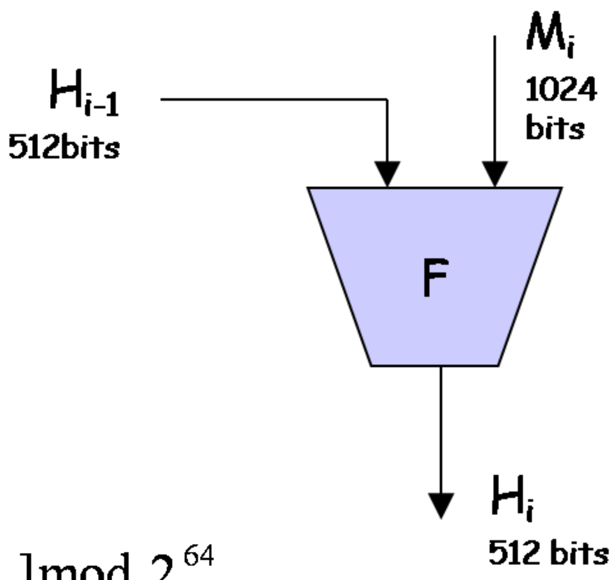
$$\text{generate } W_{16} \cdot W_{17} \cdots W_{79} \text{ as follows}$$

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

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

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

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



$W >>> n$: **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)

$$H_{i-1} = r_1r_2r_3r_4r_5r_6r_7r_8, W_0, W_1, \dots, W_{79}, K_0, K_1, \dots, K_{79}$$

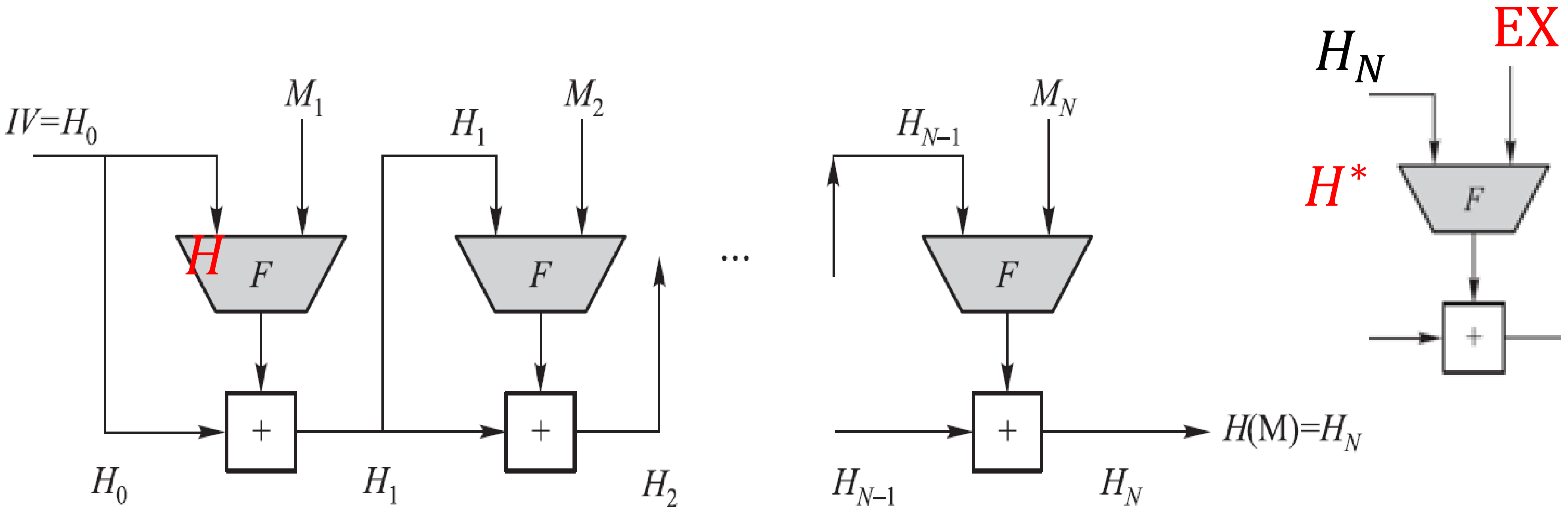
For each i is executed 80 rounds: t=0,1,2,..79

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

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

$$F(Mi, H_{i-1}) = r_1r_2r_3r_4r_5r_6r_7r_8$$

Length extension attack on SHA2



$$H(K||M) = H(K||M||padded)$$

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

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

$$H(M||K) \longrightarrow OK$$

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

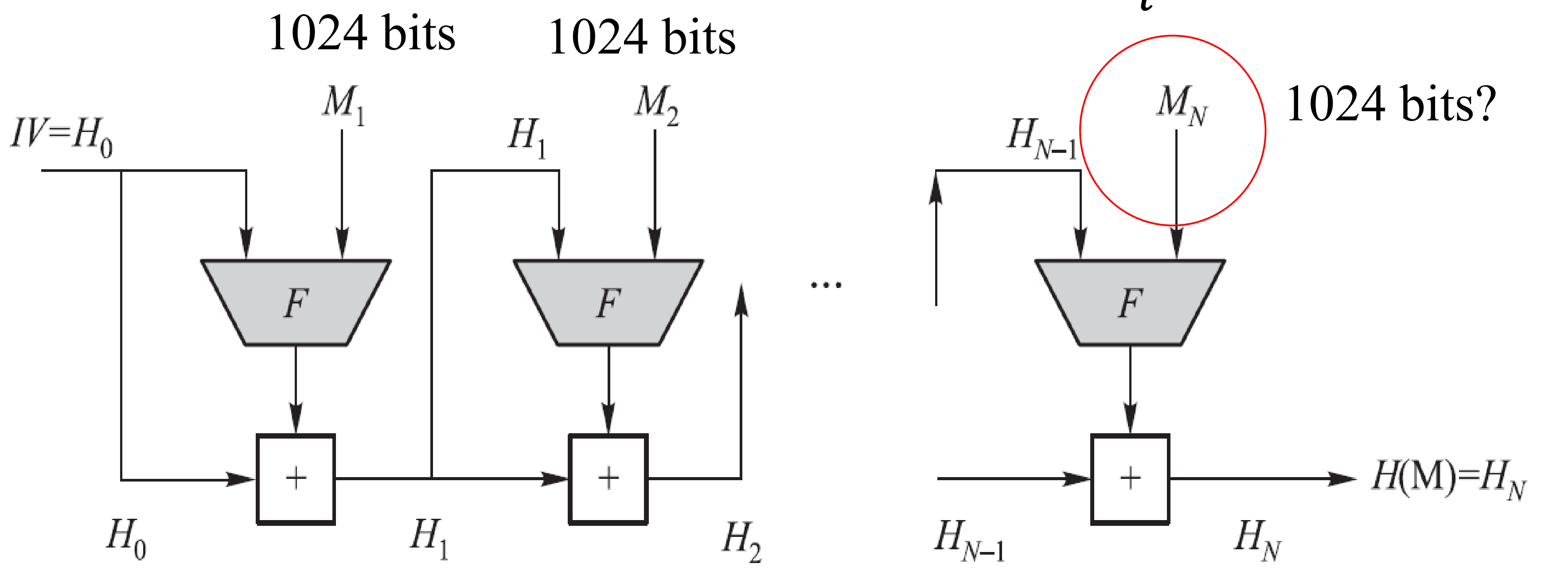
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

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** $\boldsymbol{\ell \geq 0}$

SHA-512 Initial Process (I)

Padding process

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

- **Length(M)=24**
- **$M' = M \,||\, 1(0^{\ell}) \,||\, b_{128}(L)$, 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

$$\mathbf{M \longrightarrow M'}$$

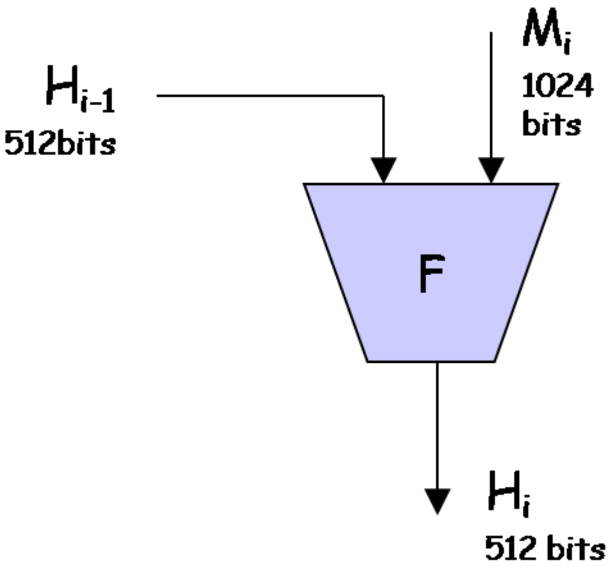
$$\mathbf{M' = M \,||\, 1(0^{\ell}) \,||\, b_{128}(L),}$$

where $\boldsymbol{\ell \geq 0}$

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

$$len(M_i) = 1024$$

SHA-512 Initial Process (II)



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

- 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:

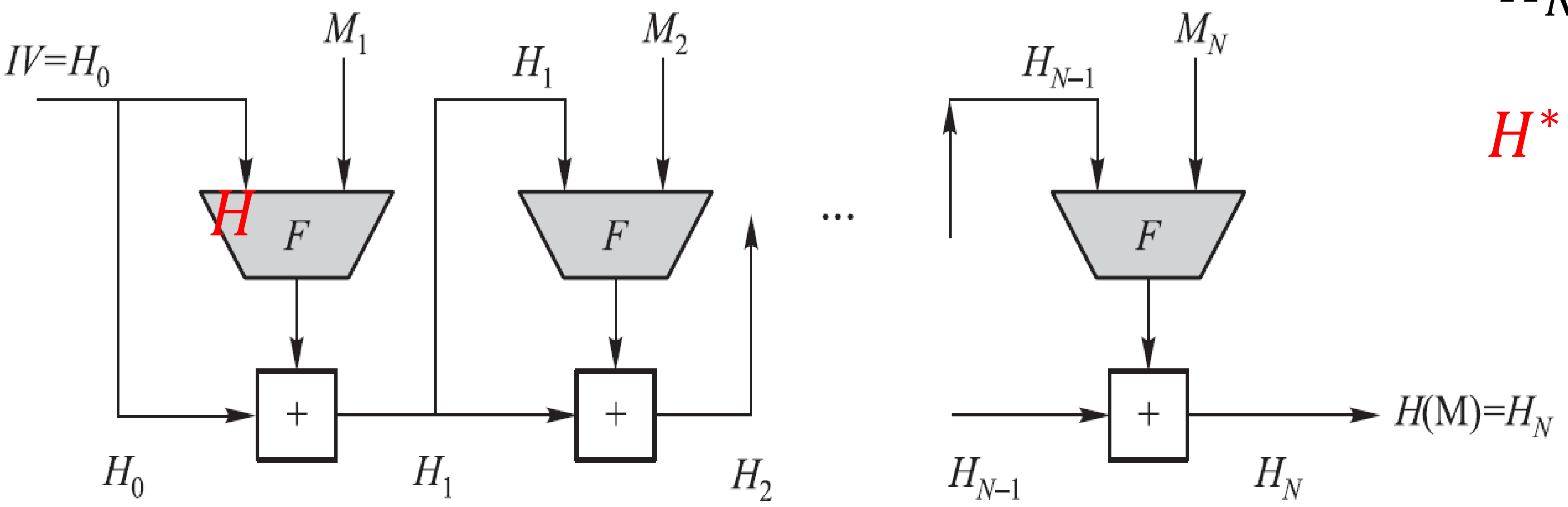
$$\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}, \sqrt{11}, \sqrt{13}, \sqrt{17}, \sqrt{19}$$

$H_0^{(0)}$ = 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

Length extension attack on SHA2

H_N EX

H^*

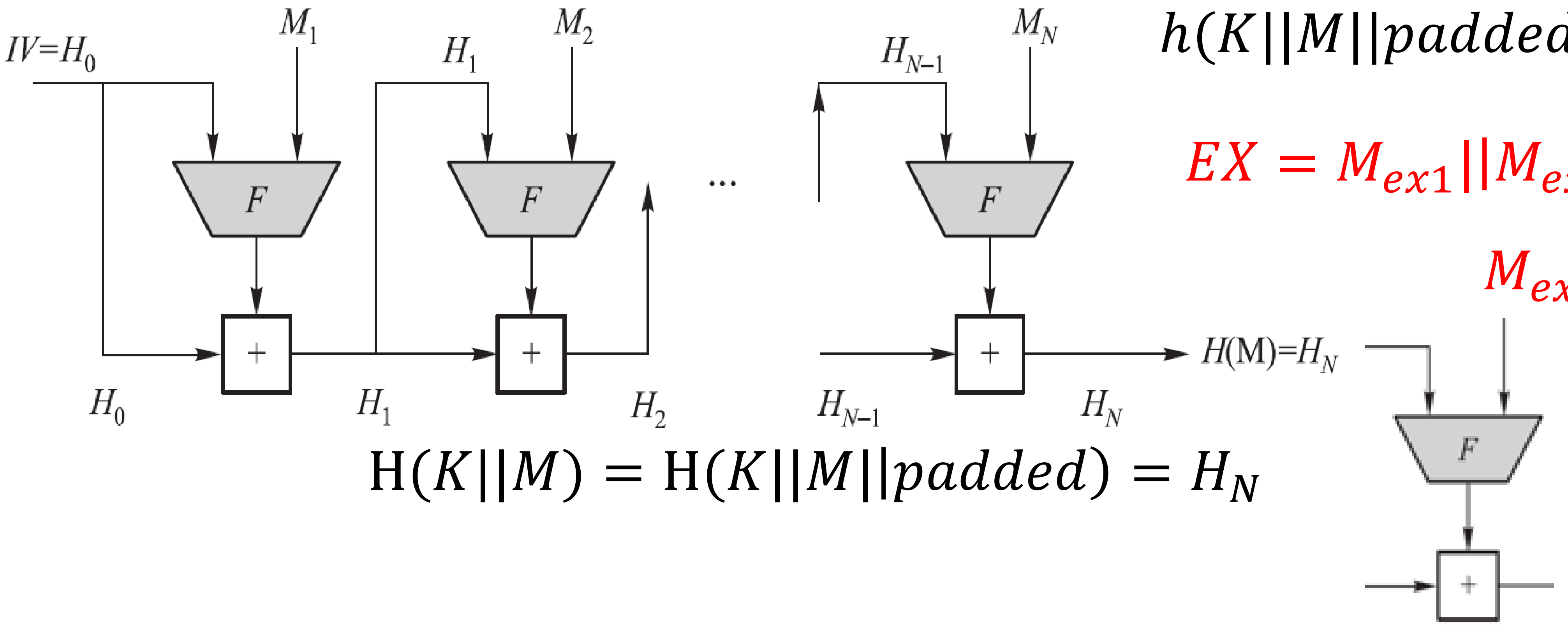


$$(\mathrm{M}, \text{tag}{=}\mathrm{H}(K||M) = \mathrm{H}(K||M||padded))$$

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

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

Length extension attack on SHA2



$h(K||M||padded||EX)$?

$EX = M_{ex1}||M_{ex2}|| \dots$

M_{ex1}

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

- **Compute** $\mathrm{H}(K||M||padded||\mathrm{EX})$ **without knowing the input** $\boldsymbol{K, M}$

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

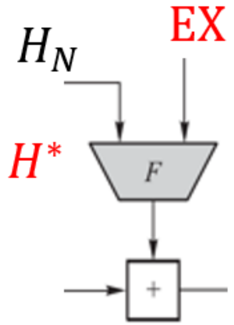
$$\text{where IV of } H^* = H_N$$

Length extension attack on SHA2

K

$$M, \text{tag} = h(K||M) = H_N$$

K



$$M \rightarrow M' = M||padding||EX, \text{tag}' = h(K||M') = H^*(EX)$$

K

$$M, \text{tag} = h(K||M) = H_N$$

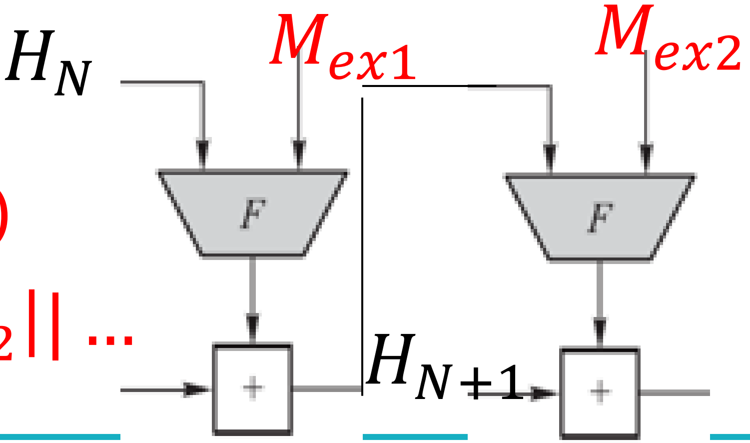
$$M', \text{tag}' = H^*(EX)$$

K

M', tag'

$$h(K||M') = h(K||M||padding||\text{EX}) = h^*(EX)$$

$$EX = M_{ex1}||M_{ex2}||\ldots$$



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

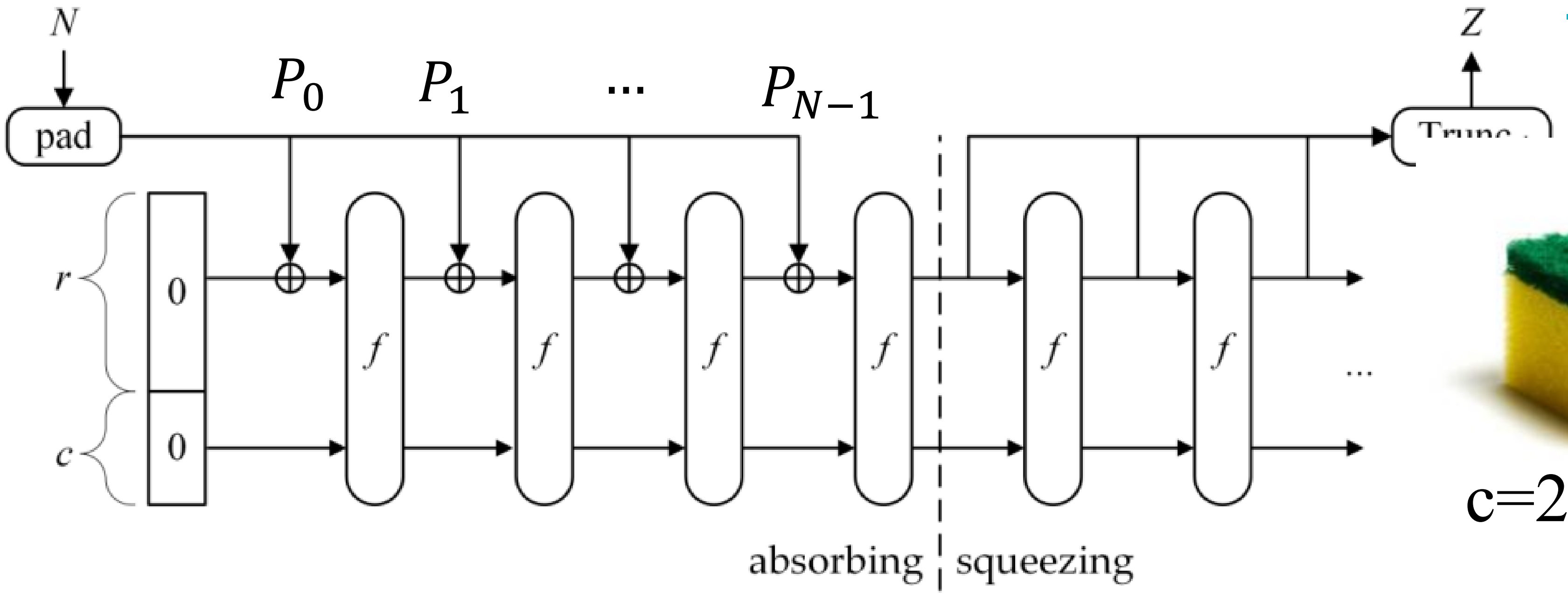
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

NIST SHA-3

- 2007: Request for submissions of new hash functions
- 2008: Submissions deadline. Received 64 entries. Announced first-round selections of 51 candidates.
- 2009: After First SHA-3 candidate conference in Feb, announced 14 Second Round Candidates in July.
- 2010: After one year public review of the algorithms, hold second SHA-3 candidate conference in Aug. Announced 5 Third-round candidates in Dec.
- 2011: Public comment for final round
- 2012: October 2, NIST selected SHA3
  - Keccak (pronounced “catch-ack”) created by Guido Bertoni, Joan Daemen and Gilles Van Assche, Michaël Peeters

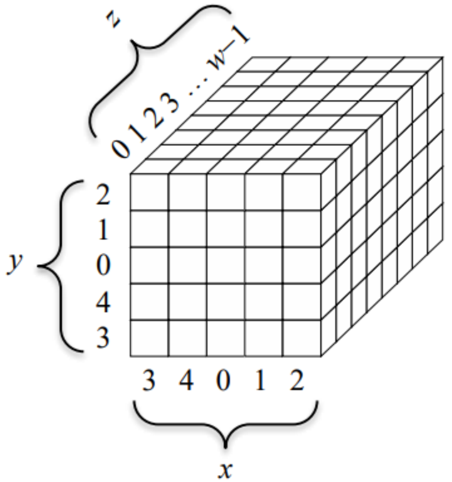
SHA3 Standard

Sponge construction (Keccak)



r (block size)

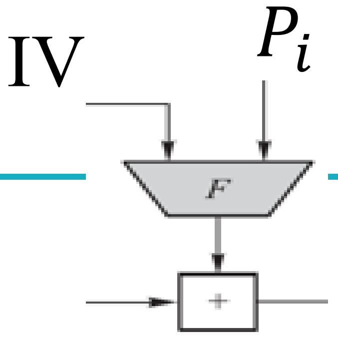
c=2d = 2.(output length)



- Let M be the input string; d = the hash length.
- b = r + c, where c = 2d
  - ✓ r is called **rate** and c **capacity**
- Where $b = 25 \times 2^{l}$ with $0 \le l \le 6$
  $b \in \{25, 50, 100, 200, 400, 800, 1600\}$

Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., & Van Keer, R. (2012). Keccak implementation overview. *URL: http://keccak. neokeon. org/Keccak-implementation-3.2. pdf*.

Example



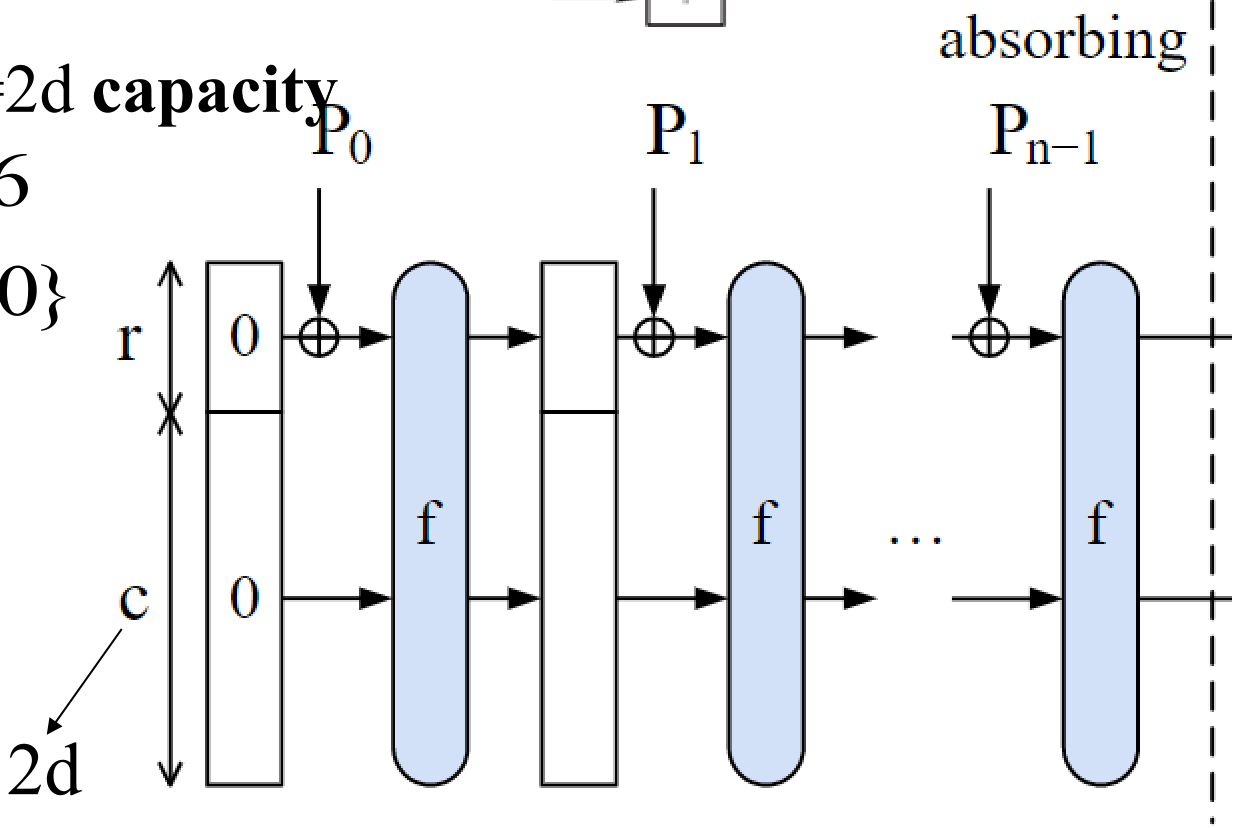
- Let M be the input string; d = the hash length.
- b = r + c, where c = 2d
  - ✓ r (**block size**) is called **rate,** and c=2d **capacity**
- Where $b = 25 \times 2^l$ with $0 \le l \le 6$

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

Ex. $d = 512$, then c = 1024.

Choose b = 1600, then r = 576.

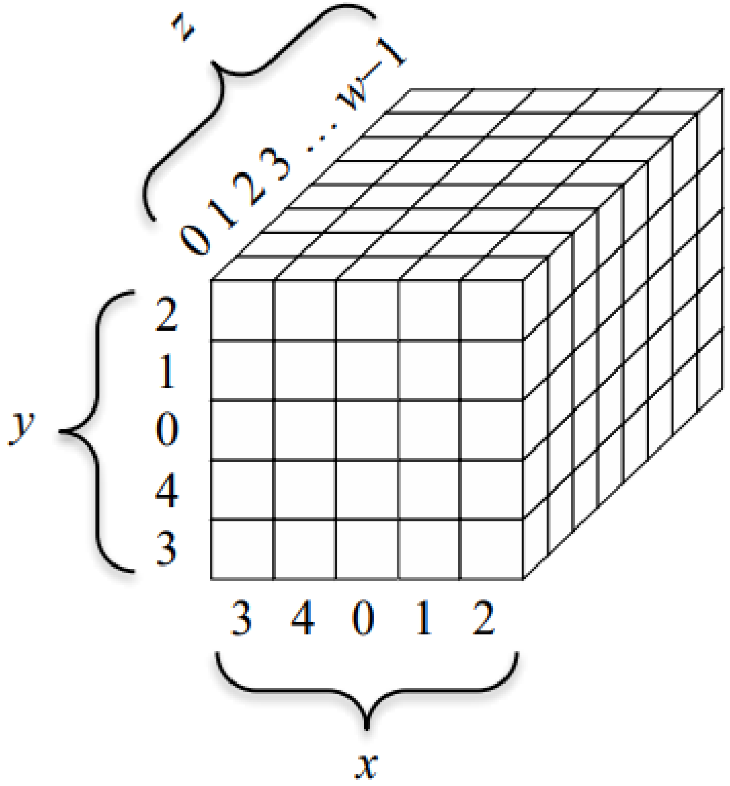
Instance	Output size d	Rate r = block size
SHA3-256(M)	256	1088
SHA3-384(M)	384	832
SHA3-512(M)	512	576



$$\| P_0 \| = \| P_1 \| = \ldots \| P_{N-1} \| = r$$

Setup

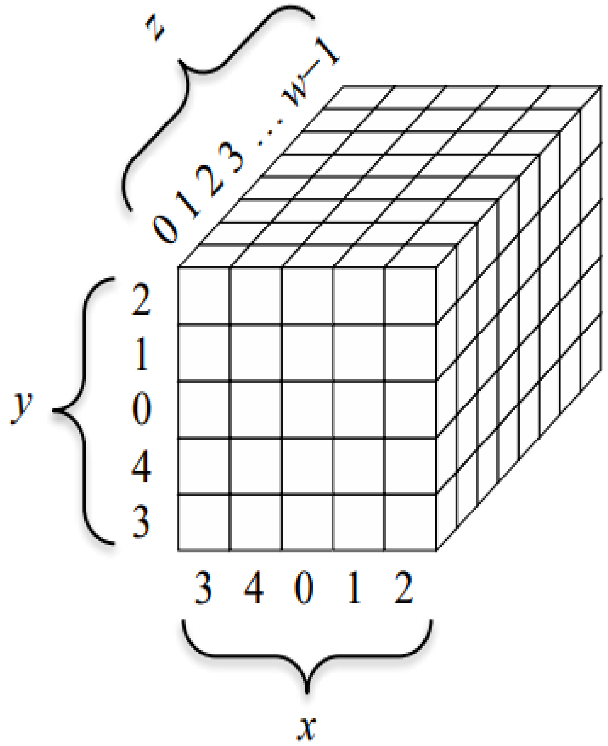
- Pad M by appending 1{0}*1 to produce M' such that | divisible by r.
- Divide M' into N = |M'|/r blocks: $M_1, \ldots, M_N$
- Let A be a b-bit string and denote A as a 5X5 matrix
- Let $a_{i,j,k}$ denote the k^{th} bit in $a_{i,j}$
- Let f_b be a fixed-length permutation on b-bit inputs
- Let $p_r = pfx_r$, $s_c = sfx_c$



$$A = (a_{i,j,k})_{5 \times 5 \times w}$$

Absorb and Squeeze

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.



$$\theta \to \rho \to \pi \to \chi \to \iota$$

Repeat Round times

$i_r = 0$

$i_r += 1$

https://chemejon.io/sha-3-explained/

$$Rounds = 12 + 2\ell$$

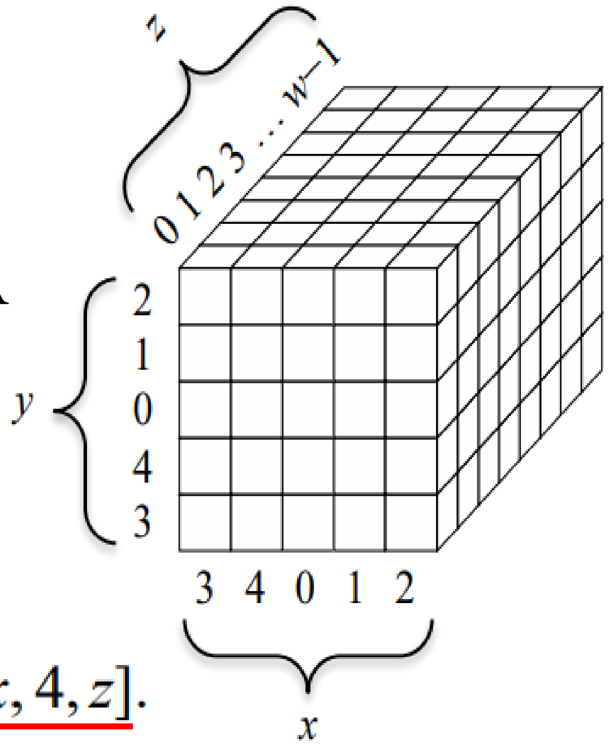
Absorb and Squeeze

$$\theta \to \rho \to \pi \to \chi \to \iota$$

Repeat Round times
$i_r = 0$
$i_r \mathrel{+}= 1$

$$Rounds = 12 + 2\ell$$

A



Algorithm 1: θ(**A**)

1. For all pairs (x, z) such that $0 \le x < 5$ and $0 \le z < w$, let
$$C[x,z] = \mathbf{A}[x,0,z] \oplus \mathbf{A}[x,1,z] \oplus \mathbf{A}[x,2,z] \oplus \mathbf{A}[x,3,z] \oplus \mathbf{A}[x,4,z].$$
2. For all pairs (x, z) such that $0 \le x < 5$ and $0 \le z < w$ let
$$D[x,z] = C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z-1) \bmod w].$$
3. For all triples (x, y, z) such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let
$$\mathbf{A'}[x,y,z] = \mathbf{A}[x,y,z] \oplus D[x,z].$$

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

Absorb and Squeeze

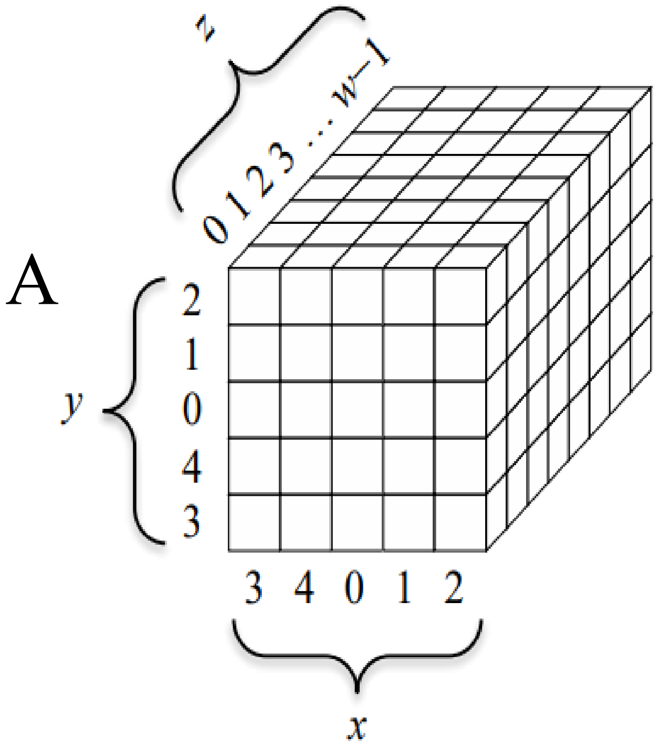
$$\theta \to \rho \to \pi \to \chi \to \iota$$

Repeat Round times

$i_r = 0$

$i_r \mathrel{+}= 1$

$$Rounds = 12 + 2\ell$$



<u>Algorithm 2: ρ(**A**)</u>

1. For all z such that $0 \le z < w$, let $\mathbf{A}'[0, 0, z] = \mathbf{A}[0, 0, z]$.
2. Let $(x, y) = (1, 0)$.
3. For t from 0 to 23:
   a. for all z such that $0 \le z < w$, let $\mathbf{A}'[x, y, z] = \mathbf{A}[x, y, (z-(t+1)(t+2)/2) \bmod w]$;
   b. let $(x, y) = (y, (2x+3y) \bmod 5)$.
4. Return $\mathbf{A}'$.

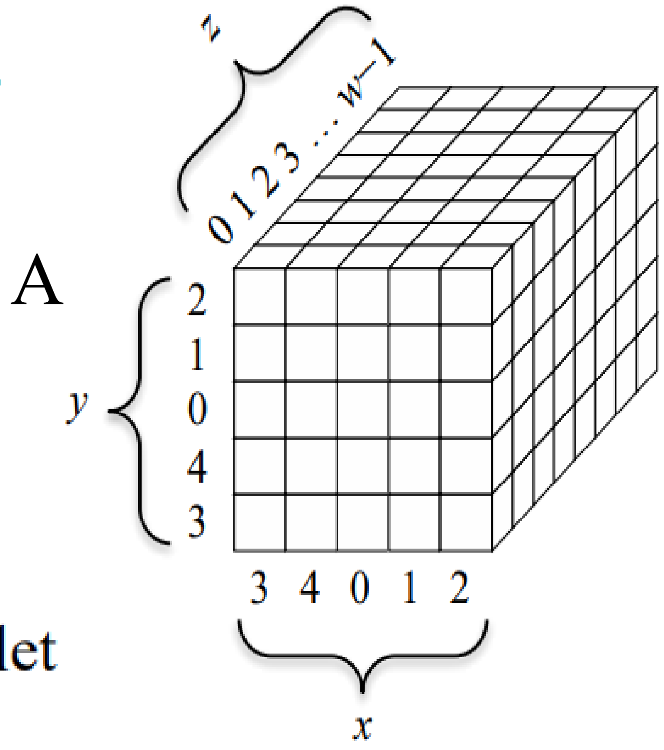
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

Absorb and Squeeze

$$\theta \rightarrow \rho \rightarrow \pi \rightarrow \chi \rightarrow \iota$$

Repeat Round times
$i_r = 0$
$i_r \mathrel{+}= 1$

$$Rounds = 12 + 2\ell$$



<u>Algorithm 3: π(**A**)</u>

1. For all triples (x, y, z) such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let
   $\mathbf{A}'[x, y, z] = \mathbf{A}[(x + 3y) \bmod 5, x, z]$.
2. Return $\mathbf{A}'$.

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

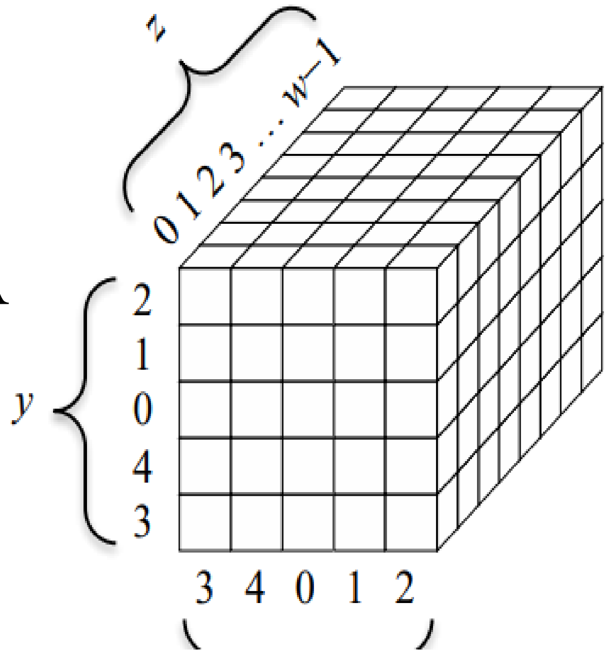
Absorb and Squeeze

$$\theta \rightarrow \rho \rightarrow \pi \rightarrow \chi \rightarrow \iota$$

Repeat Round times
$i_r = 0$
$i_r \mathrel{+}= 1$

$$Rounds = 12 + 2\ell$$

A



__Algorithm 4: χ(A)__

1. For all triples (x, y, z) such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let
$$\mathbf{A}'[x,y,z] = \mathbf{A}[x,y,z] \oplus ((\mathbf{A}[(x+1) \bmod 5, y, z] \oplus 1) \cdot \mathbf{A}[(x+2) \bmod 5, y, z]).$$
2. Return $\mathbf{A}'$.

“.” “ integer multiplication

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

Absorb and Squeeze

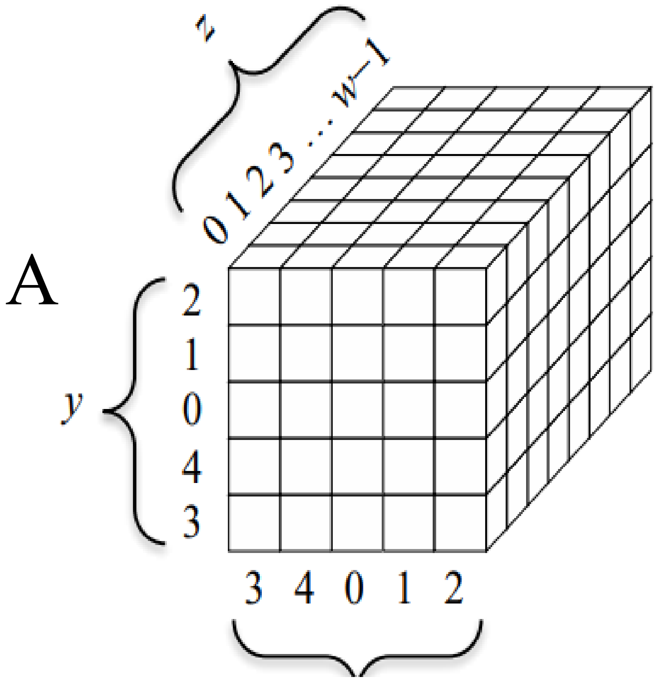
$$\theta \rightarrow \rho \rightarrow \pi \rightarrow \chi \rightarrow \iota$$

Repeat Round times

$i_r = 0$

$i_r += 1$

$$Rounds = 12 + 2\ell$$



<u>Algorithm 6: ι(**A**, i_r)</u>

1. For all triples (x, y, z) such that $0 \le x < 5$, $0 \le y < 5$, and $0 \le z < w$, let $\mathbf{A'}[x, y, z] = \mathbf{A}[x, y, z]$.
2. Let $RC = 0^w$.
3. For j from 0 to ℓ, let $RC[2^j - 1] = rc(j + 7i_r)$.
4. For all z such that $0 \le z < w$, let $\mathbf{A'}[0, 0, z] = \mathbf{A'}[0, 0, z] \oplus RC[z]$.
5. Return $\mathbf{A'}$.

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

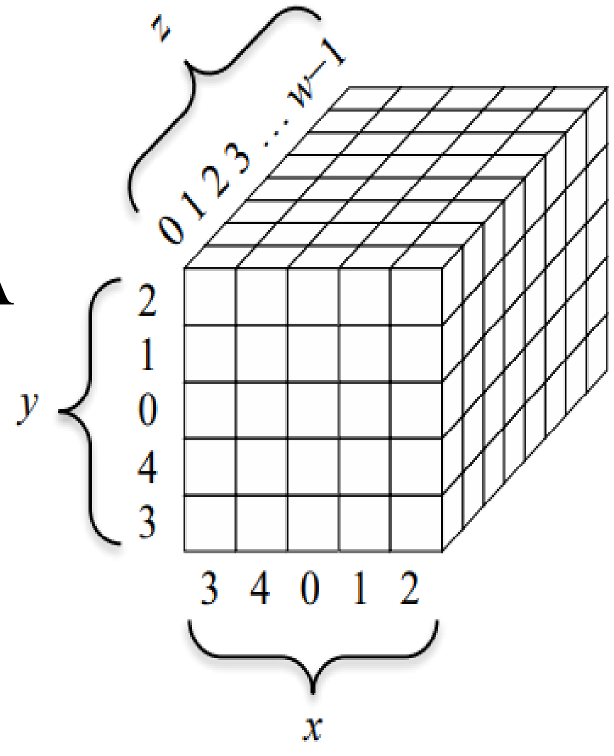
Absorb and Squeeze

Algorithm 5: $rc(t)$

round constant

1. If t mod 255 = 0, return 1.
2. Let R = 10000000.
3. For i from 1 to t mod 255, let:
   a. $R = 0 \parallel R$;
   b. $R[0] = R[0] \oplus R[8]$;
   c. $R[4] = R[4] \oplus R[8]$;
   d. $R[5] = R[5] \oplus R[8]$;
   e. $R[6] = R[6] \oplus R[8]$;
   f. $R = \mathrm{Trunc}_8[R]$.
4. Return $R[0]$.

A

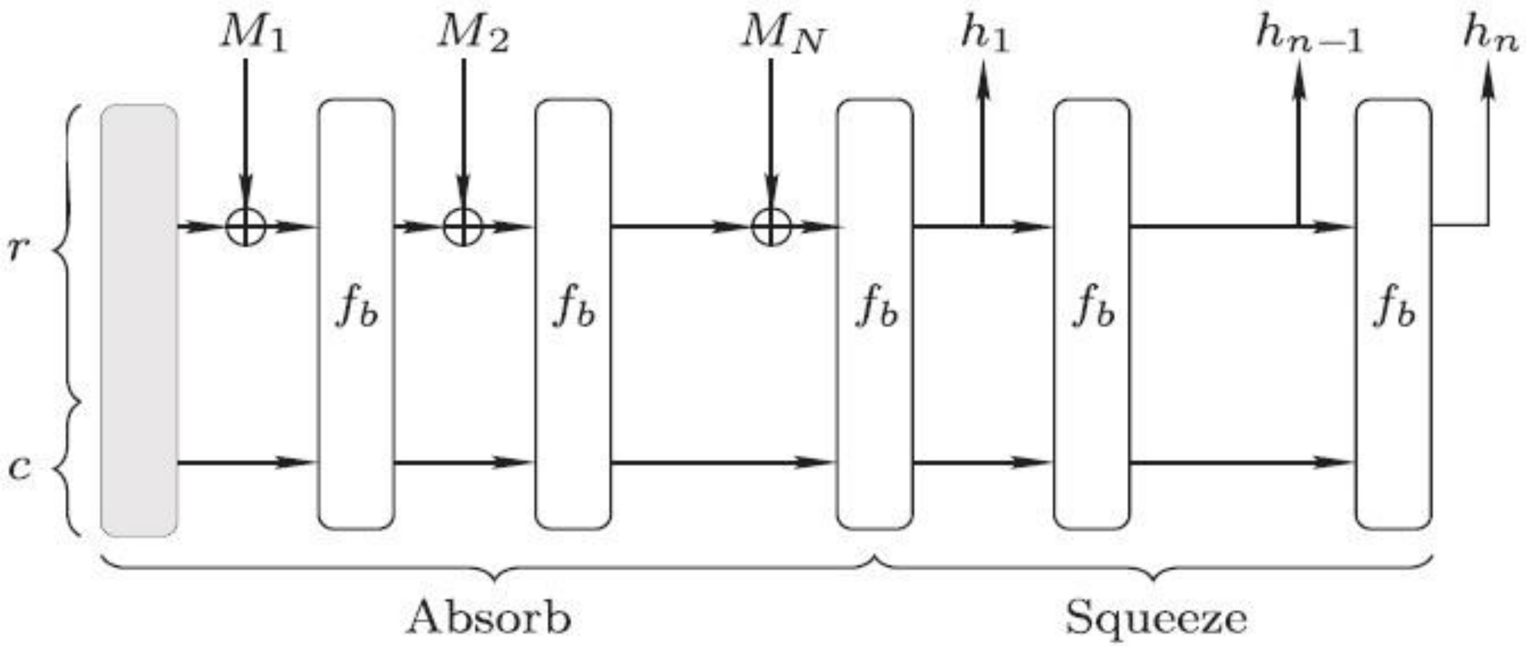


$\mathrm{Trunc}_s(X)$ — For a positive integer s and a string X, $\mathrm{Trunc}_s(X)$ is the string comprised of bits $X[0]$ to $X[s-1]$. For example, $\mathrm{Trunc}_2(10100)$=10.

Absorb and Squeeze

Absorb: $A_i = f_b((\mathsf{p}_r(M_i \oplus A_{i-1}) \parallel \mathsf{s}_c(A_{i-1})),$
$i = 1, \cdots, N.$

Squeeze: $A_{N+i} = f_b(A_{N+i-1}),$ $i = 1, \cdots, n.$ $h_i = \mathsf{p}_r(A_{N+i-1}),$ $i = 1, \cdots, n.$



SHA-3 Hash

Permitation $f_b(b)$

The KECCAK Family of Permutations

We now describe the KECCAK family of permutations f_b, where $b = 25 \times 2^\ell$ with $0 \le \ell \le 6$, and $b = r + c$.

The permutation f_b takes a 5×5 state matrix $\boldsymbol{A}$ as input and carries out the following five operations, where indices are computed modulo 4 and 2^ℓ where appropriate.

1. *Diffusion*: For all $0 \le i, j \le 4$ and $0 \le k \le 2^\ell - 1$, compute

$$a_{i,j,k} = a_{i,j,k} \oplus \bigoplus_{y=0}^{4} a_{i-1,y,k} \oplus \bigoplus_{y=0}^{4} a_{i+1,y,k-1}.$$

SHA-3 Hash

2. *Dispersion*(of bits in words): This operation is the following sequence of 24 steps:
   a) Set $i = 1$ and $j = 0$.
   b) For $t = 0$ to 23 do
      i. $a_{i,j} = a_{i,j} \oplus ((t+1)(t+2)/2)$.
      ii. Set $i = j$ and $j = (2i + 3j) \bmod 5$.
3. *Dispersion*(of words): For all $0 \le i, j \le 4$, compute

$$a_{i,(2i+3j) \bmod 5} = a_{i,j}.$$

4. *Nonlinear Map*: For each $0 \le i, j \le 4$, compute

$$a_{i,j} = a_{i,j} \oplus \left(\overline{a_{(i+1) \bmod 5, j}} \wedge a_{(i+2) \bmod 5, j}\right).$$

   This map provides resistance to linear cryptanalysis.
5. *Symmetry Disruption*: During the l-th round, compute

$$a_{0,0} = a_{0,0} \oplus \mathrm{RC}_l,$$

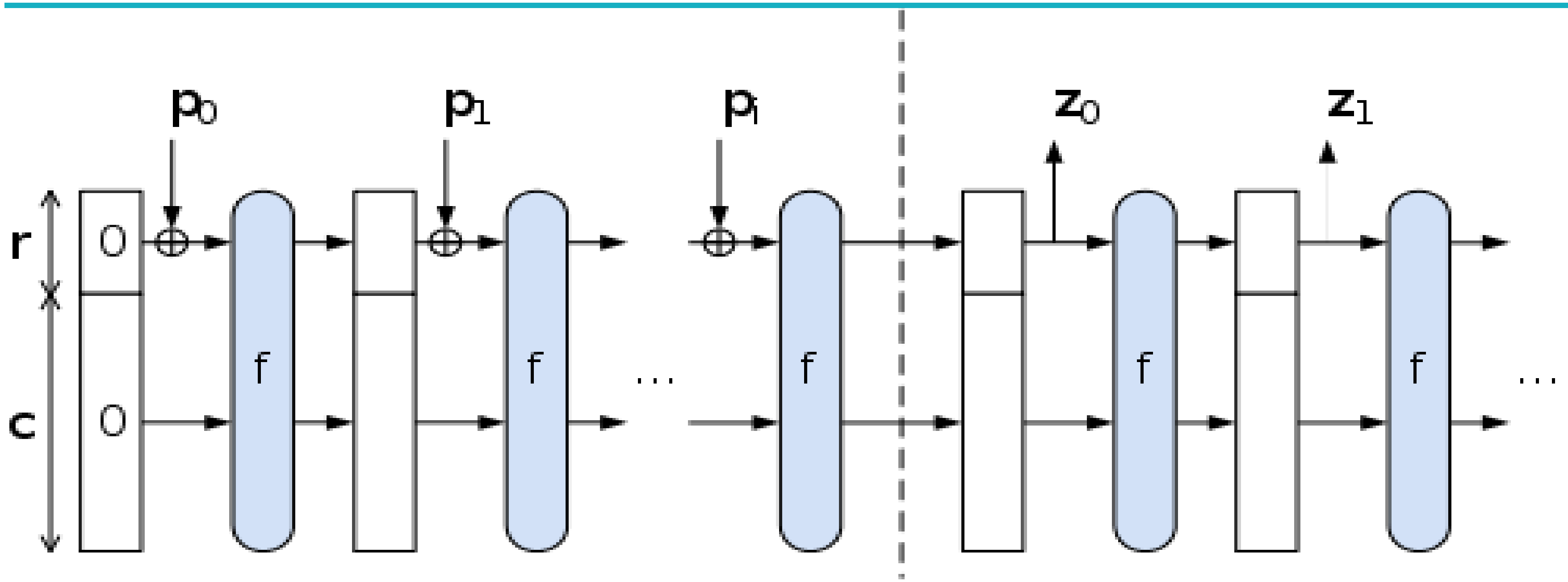
   where RC_l is the round constant for the l-th round.
   Table 4.2 lists the round constants for $\ell = 6$, where the length of each word is 64-bit long.

SHA-3 Hash

Table 4.2: Round constants for the symmetry disruption phase of the f_b for $\ell = 6$. For $\ell < 6$, use the prefix of these round constants to obtain the round constants of appropriate length

l	**Value**	l	**Value**
0	0x0000000000000001	12	0x000000008000808B
1	0x0000000000008082	13	0x800000000000008B
2	0x800000000000808A	14	0x8000000000008089
3	0x8000000080008000	15	0x8000000000008003
4	0x000000000000808B	16	0x8000000000008002
5	0x0000000080000001	17	0x8000000000000080
6	0x8000000080008081	18	0x000000000000800A
7	0x8000000000008009	19	0x800000008000000A
8	0x000000000000008A	20	0x8000000080008081
9	0x0000000000000088	21	0x8000000000008080
10	0x0000000080008009	22	0x0000000080000001
11	0x000000008000000A	23	0x8000000080008008

The Sponge Construction: Used by SHA-3

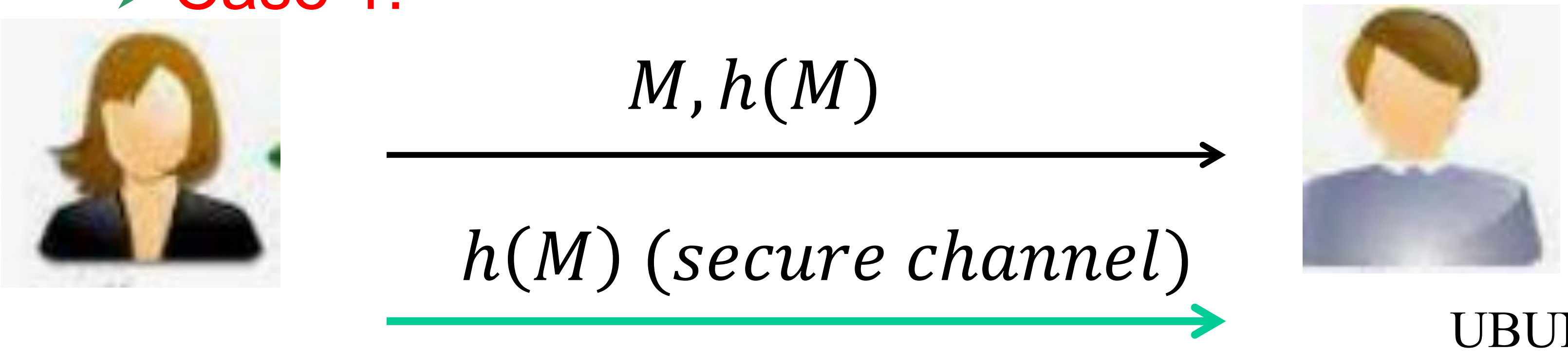


- Each round, the next r bits of message is XOR'ed into the first r bits of the state, and a function f is applied to the state.
- After message is consumed, output r bits of each round as the hash output; continue applying f to get new states
- SHA-3 uses 1600 bits for state size

Choosing the length of Hash outputs

- The Weakest Link Principle:
  - A system is only as secure as its weakest link.
- Hence all links in a system should have similar levels of security.
- Because of the birthday attack, the length of hash outputs in general should double the key length of block ciphers
  - SHA-224 matches the 112-bit strength of triple-DES (encryption 3 times using DES)
  - SHA-256, SHA-384, SHA-512 match the new key lengths (128,192,256) in AES
  - SHAKE: ouput length d

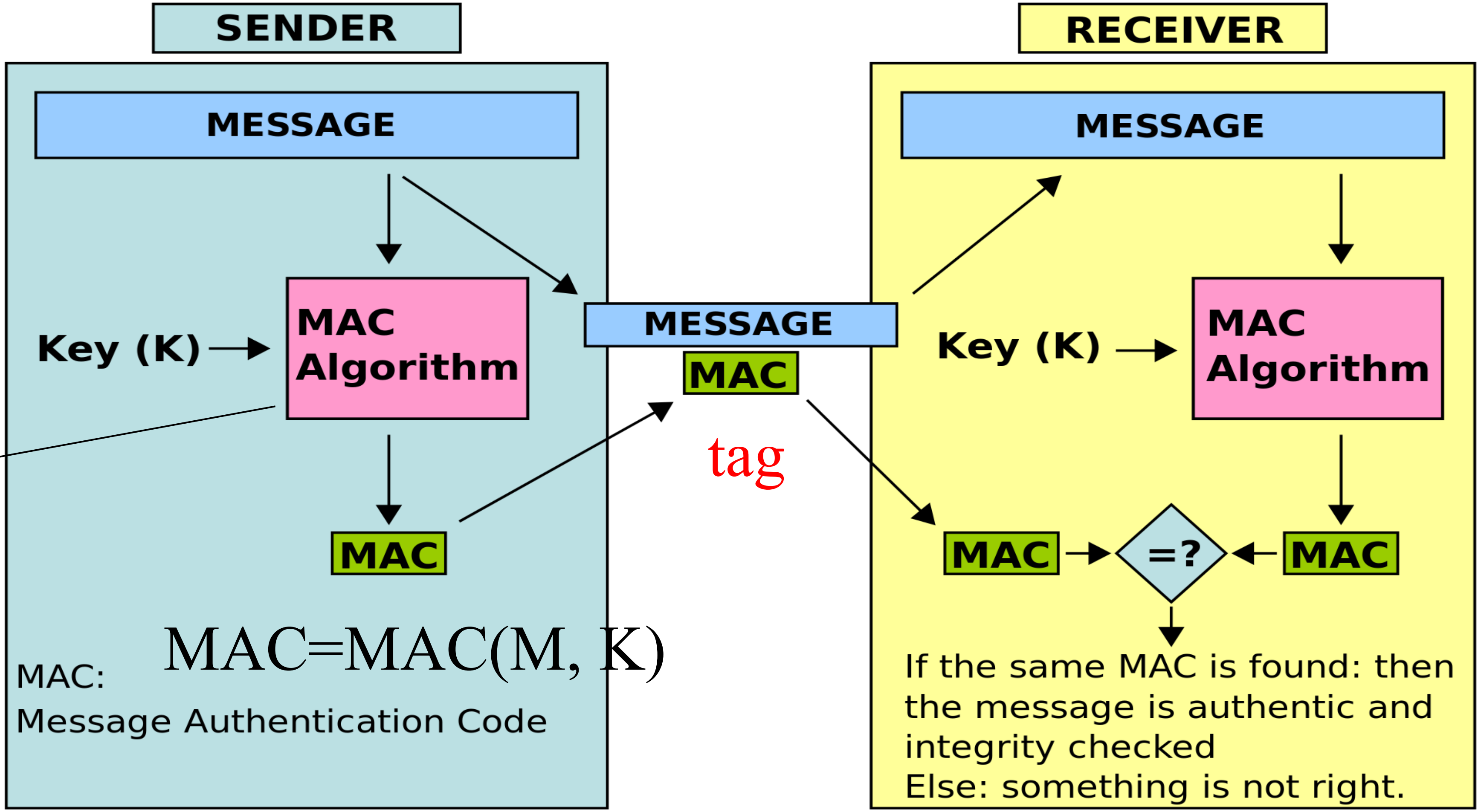
- Is this secure scheme (M cannot be modified)?
  - Case 1:



UBUNTO 20 ISO

- Require an authentic channel to transmit the hash of a message
  - Without such a channel, it is insecure, because anyone can compute the hash value of any message, as the hash function is public
  - Such a channel may not always exist
- How to address this?
  - use more than one hash functions
  - use a key to select which one to use

Message Authentication Code



Collision resistance

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

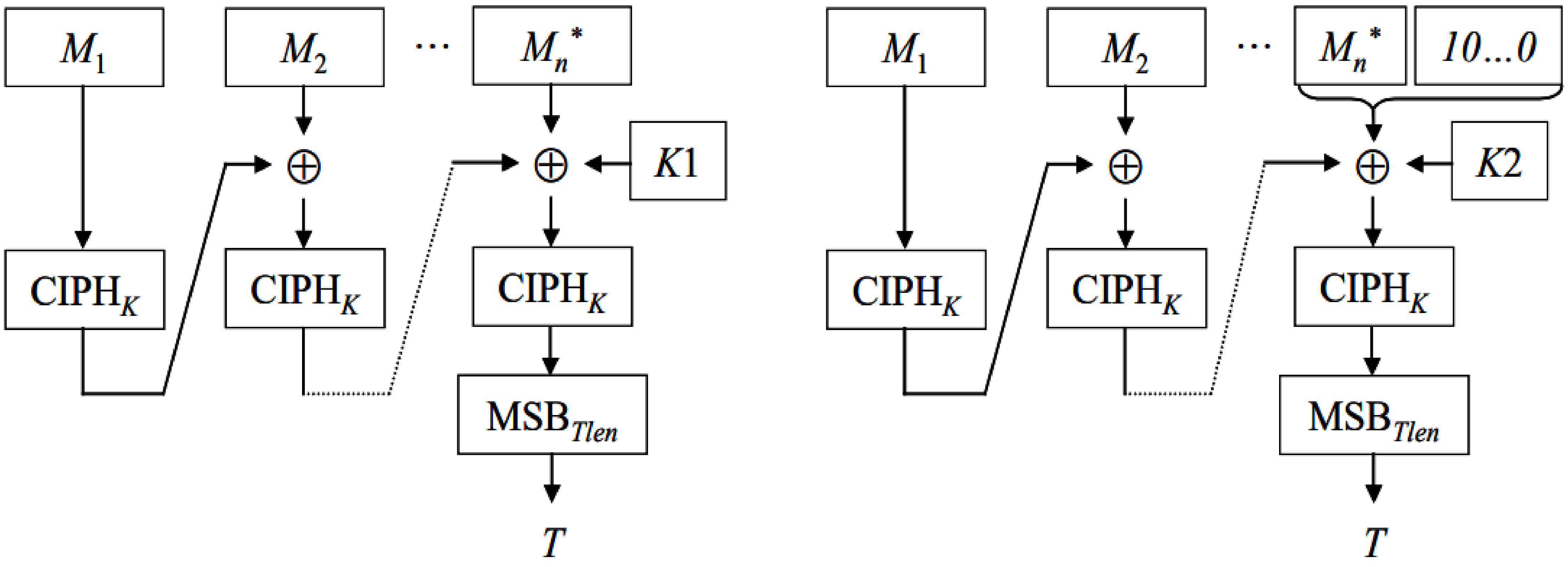
Message Authentication Code

other block ciphers	SEED, RC5, SHACAL-2, SIMECK, SIMON (64/128), Skipjack, SPECK (64/128), Simeck, SM4,Threefish (256/512/1024), Triple-DES (DES-EDE2 and DES-EDE3), TEA, XTEA
block cipher modes of operation	ECB, CBC, CBC ciphertext stealing (CTS), CFB, OFB, counter mode (CTR), XTS
message authentication codes	BLAKE2b, BLAKE2s, CMAC, CBC-MAC, DMAC, GMAC (GCM), HMAC, Poly1305, SipHash, Two-Track-MAC, VMAC
hash functions	BLAKE2b, BLAKE2s, Keccack (F1600), SHA-1, SHA-2, SHA-3, SHAKE (128/256), SipHash, LSH (128/256), Tiger, RIPEMD (128/160/256/320), SM3, WHIRLPOOL
public-key cryptography	RSA, DSA, Determinsitic DSA (RFC 6979), ElGamal, Nyberg-Rueppel (NR), Rabin-Williams (RW), EC-based German Digital Signature (ECGDSA), LUC, LUCELG, DLIES (variants of DHAES), ESIGN

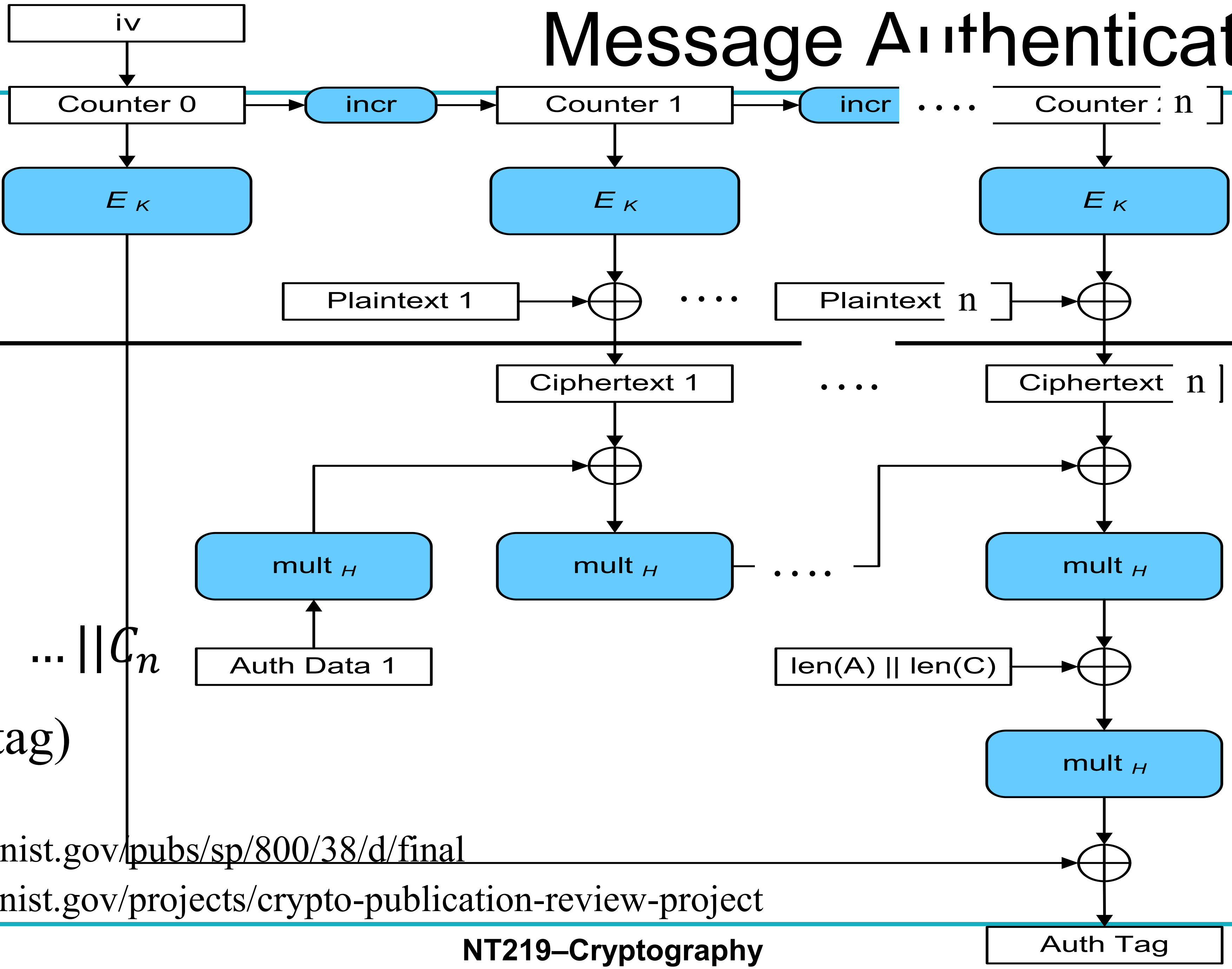
CMAC: https://csrc.nist.gov/pubs/sp/800/38/b/upd1/final

Message Authentication Code

CMAC



Message Authentication Code
GMAC



GCM

$C = C_1 \, || \, \ldots || C_n$

(C, auth tag)

Page 19 for $Mult_H$

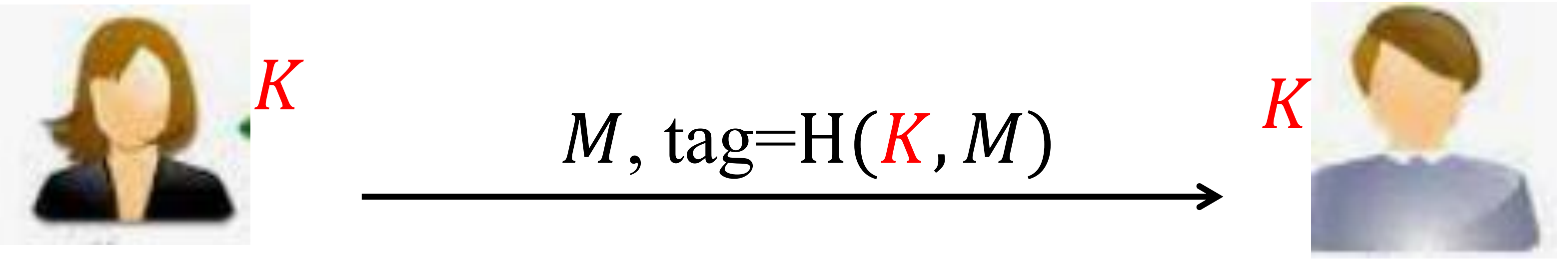
https://csrc.nist.gov/pubs/sp/800/38/d/final

https://csrc.nist.gov/projects/crypto-publication-review-project

Keyed-Hash Message Authentication Code

HMAC

- A hash family (H) use for message authentication
- $MAC(K,M) = H_K(M)$
- The sender and the receiver share secret K
- The sender sends $(M, H_k(M))$
- The receiver receives (X,Y) and verifies that $H_K(X)=Y$, if so, then accepts the message as from the sender
- To be secure, an adversary shouldn't be able to come up with (X',Y') such that $H_K(X')=Y'$.



$$M', tag' = H(K, M')?\!=\text{tag}$$

Security Requirements for MAC

- Resist the Existential Forgery under Chosen Plaintext Attack
  - Challenger chooses a random key K
  - Adversary chooses a number of messages M_1, M_2, .., M_n, and obtains t_j=MAC(K,M_j) for $1 \leq j \leq n$
  - Adversary outputs M' and t'
  - Adversary wins if $\forall j$ M'$\neq M_j$, and t'=MAC(K,M')

- Basically, adversary cannot create the MAC for a message for which it hasn't seen an MAC

Constructing MAC from Hash Functions

- Let h be a one-way hash function (SHA2)
- MAC(K,M) = h(K || M), where || denote concatenation
  - Insecure as MAC
  - Because of the Merkle-Damgard construction for hash functions, given M and t=h(K || M), adversary can compute M’=M||Pad(M)||X and t’, such that h(K||M’) = t’

Constructing MAC from Cryptographic Hash Functions

HMAC

$\text{HMAC}_K[M] = \text{Hash}[(K^+ \oplus \text{opad}) \,||\, \text{Hash}[(K^+ \oplus \text{ipad})||M)]]$

- $K^+ = K||\{0\}^*$ (to B bytes, the input block size of the hash function)
- ipad = 0x36 0x36 … 0x36 (repeated B times)
- opad = 0x5C 0x5C …. 0x5C (repeated B times).

At high level, $\text{HMAC}_K[M] = H(K \,||\, H(K \,||\, M))$

Authentication and Integrity checking

K 

$M, \text{tag} = h(K, M)$ $\longrightarrow$

 K

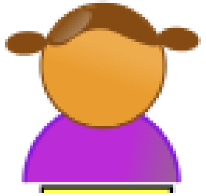
h(K, M) = h[($K^+ \oplus$ opad) || Hash[($K^+ \oplus$ ipad)||M)]]

$K^+ = K||\{0\}^*$ (to B bytes, the input block size of the hash function)
ipad = 0x36 0x36 … 0x36 (repeated B times)
opad = 0x5C 0x5C …. 0x5C (repeated B times).

Pros and Cons?

Authenticate user/end-devices case

(1) Password-based Authentication



$store: name,\ h(pw)$

$name, pw$

- **Login**

	UserName	HashedSaltedPwd	UserID
1	A-JayBibbins637	0x600FAD66D16A56AF3459D9C996DFF98B	10000
2	A-JayTorain976	0x4E30184141CB6CFF4120D6AC443CCE54	10001
3	AadamDobbin507	0xED2A51BD92E1EE8333314407817CD6F9	10002

- Provide $name, pw'$ to server?
  - Locate the row using name
  - Check $h(pw')? = h(pw)$

→ authenticate the user

HMAC Security

- If used with a secure hash functions (e.g., SHA3-256) and according to the specification (key size, and use correct output), no known practical attacks against HMAC

Dictionary attacks on hash function

h(password)=
b1b6a3de29ab9071536146 83d357b2db943a317d036ff25f7022d4707109005a

password=?

Guest password	sha256	sha384	sha512	…
abc123	…	…	…..	…
abcd1234	…	….	…	…
….				

Rainbow table

https://hashkiller.io/