Cryptographic Hash Functions: Design, Analysis & Applications

Dhananjoy Dey

Indian Institute of Information Technology, Lucknow
ddey@iiitl.ac.in

February 16, 2021

Disclaimers

1

All the pictures used in this presentation are taken from freely available websites.

2

If there is a reference on a slide all of the information on that slide is attributable to that source whether quotation marks are used or not.

3

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement nor does it imply that the products mentioned are necessarily the best available for the purpose.

Outline

Outline

Definition & Type

Definition & Type

- A function satisfies the following conditions:
  (i) 'easy' to compute (*efficient* & *deterministic* algorithm)
  (ii) taking an input of arbitrary length gives a fixed length of output

Definition & Type

- A function satisfies the following conditions:
  (i) 'easy' to compute (*efficient* & *deterministic* algorithm)
  (ii) taking an input of arbitrary length gives a fixed length of output

Definition

The hash function is a function $h : D \rightarrow R$ *where* $D = \{0,1\}^*$ *and* $R = \{0,1\}^n$ *for some* $n \geq 1$.

- Type of hash functions:
  (a) Perfect hash function
  (b) Minimal perfect hash function
  (c) Cryptographic hash function

Non-cryptographic Hash

Definition

Let $D = \{d_0, d_1, \ldots, d_{m-1}\}$ and $R = \{r_0, r_1, \ldots, r_{n-1}\}$ be sets with $m \leq n$.

*The hash function $h : D \rightarrow R$ is called a perfect hash function (**PHF**), if for all $x, y \in D$ and $x \neq y \Rightarrow h(x) \neq h(y)$.*

*In particular, if $m = n$, h is called a minimal perfect hash function (**MPHF**).*

Cryptographic Hash

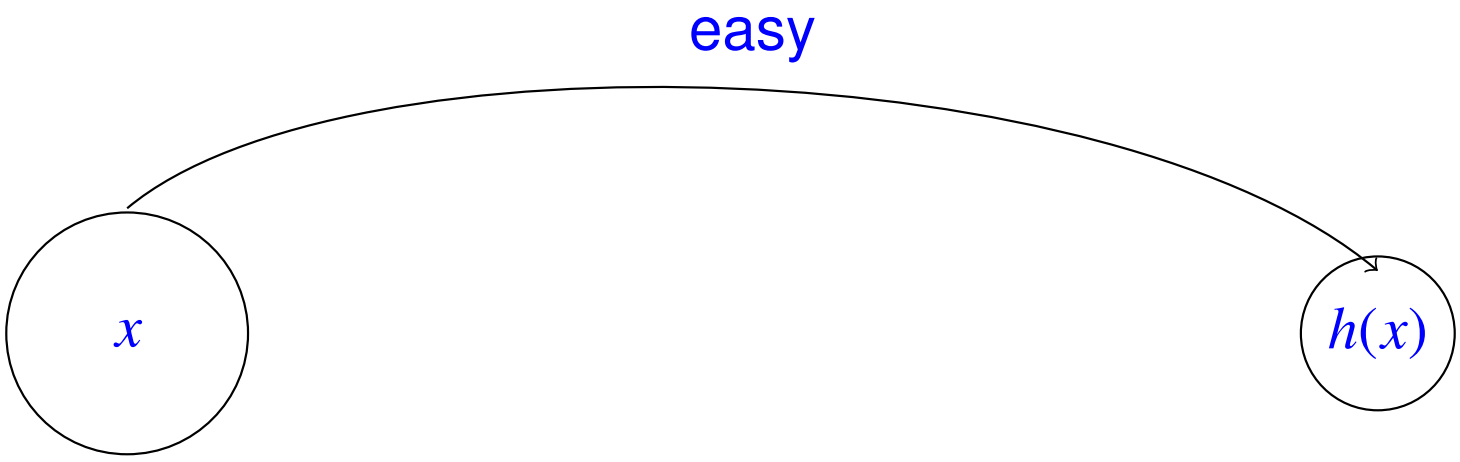
Definition

The (Cryptographic) hash function is a function $h : D \to R$ *where* $D = \{0,1\}^*$ *and* $R = \{0,1\}^n$ *for some* $n \geq 1$.

Cryptographic Hash

Definition

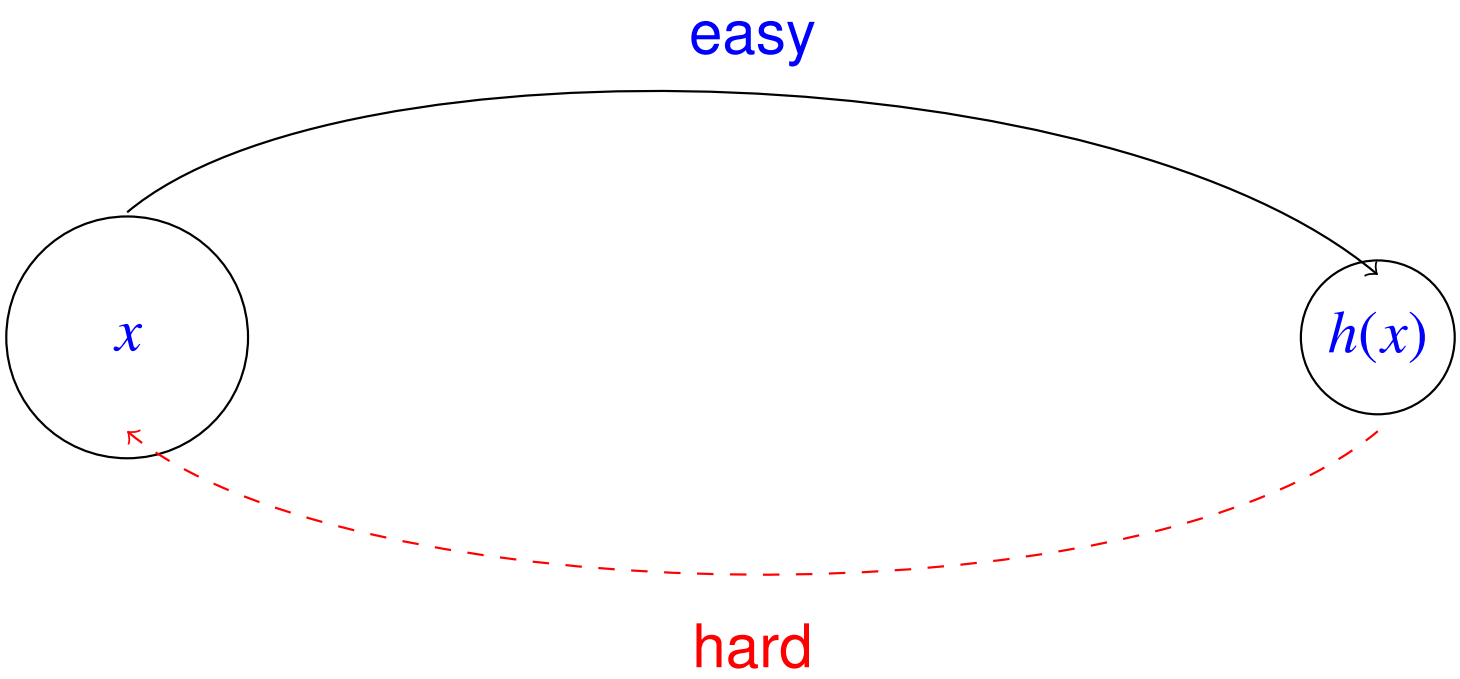
The (Cryptographic) hash function is a function $h : D \to R$ *where* $D = \{0,1\}^*$ *and* $R = \{0,1\}^n$ *for some* $n \geq 1$.



Cryptographic Hash

Definition

The (Cryptographic) hash function is a function $h : D \to R$ *where* $D = \{0,1\}^*$ *and* $R = \{0,1\}^n$ *for some* $n \geq 1$.



Ideal Cryptographic Hash

(i) **Ease of computation:** It is '*easy*' to compute the hash value for any given message.

(ii) **Compression:** It takes arbitrary length of input and gives a fixed length of output.

(iii) **Preimage resistance:** It is infeasible to find a message that has a given hash.

(iv) **Second preimage resistance:** It is infeasible to modify a message without changing its hash.

(v) **Collision resistance:** It is infeasible to find 2 different messages with the same hash.

Ideal Cryptographic Hash

(i) **Ease of computation:** It is '*easy*' to compute the hash value for any given message.

(ii) **Compression:** It takes arbitrary length of input and gives a fixed length of output.

(iii) **Preimage resistance:** It is infeasible to find a message that has a given hash.

(iv) **Second preimage resistance:** It is infeasible to modify a message without changing its hash.

(v) **Collision resistance:** It is infeasible to find 2 different messages with the same hash.

$$(i) - (iv) \Rightarrow OWHF, \quad (i) - (v) \Rightarrow CRHF$$

Ideal Cryptographic Hash

(vi) **Avalanche:** Flipping 1 bit in an input would change approximately 50% the output bits.

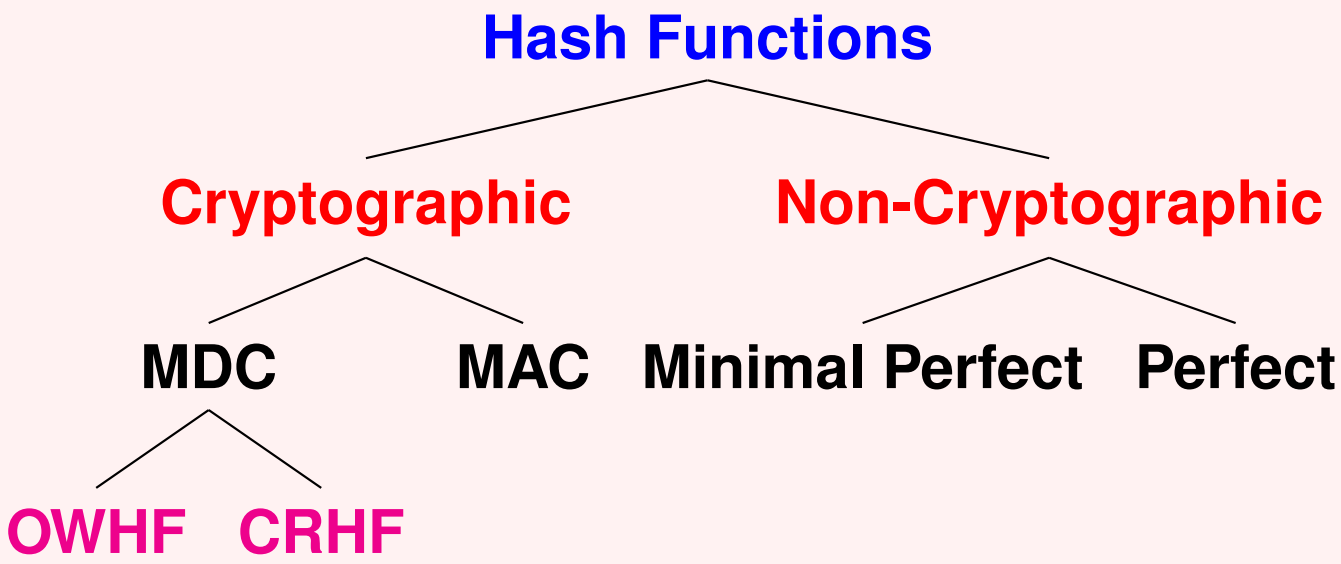
(vii) **Near-collision resistance:** It is computationally infeasible to find 2 input strings x and x' s/t $h(x)$ and $h(x')$ hardly differ.

(viii) **Partial-preimage resistance:** It is computationally infeasible to find any substring of input string x for any given output string s even for any given distinct substring of input string x.

(ix.) **Non-correlation:** Input string x and output string $h(x)$ are not correlated in any way.

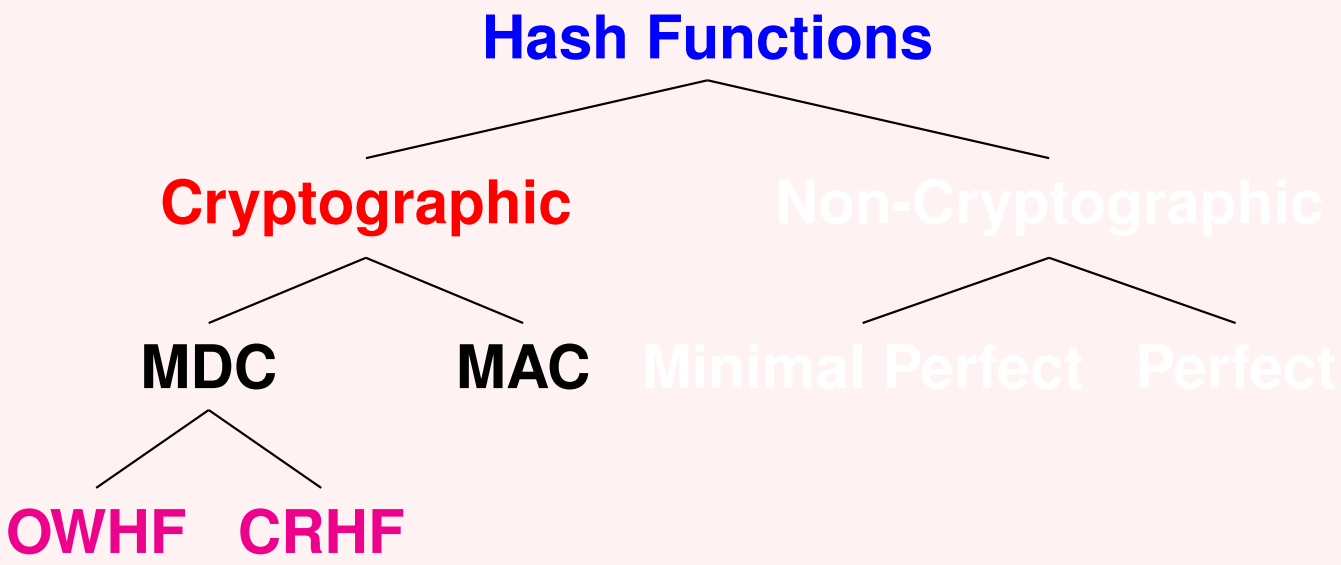
Types of Hash Functions

Hash Functions



Types of Hash Functions

Hash Functions



MAC

A MAC is a function h that satisfies the following:

(i) **Compress:** x can be of arbitrary length and $h(k, x)$ has a fixed length of n bits, where k is a fixed length of ℓ bits.

(ii) **Ease of computation:** Given h, k and an input x, the computation of $h(k, x)$ must be easy.

(iii) **'Preimage resistance':** Given a message x, it must be hard to determine $h(k, x)$, when k is not given; even when a large set of pairs $\{x_i, h(k, x_i)\}$ is known.

Requirements

- Knowing a message and MAC, is infeasible to find another message with same MAC.
- MACs should be uniformly distributed.
- MAC should depend equally on all bits of the message.

Requirements

- Knowing a message and MAC, is infeasible to find another message with same MAC.
- MACs should be uniformly distributed.
- MAC should depend equally on all bits of the message.

Definition

A MAC is a function $h : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{R}$, *s/t* $\mathcal{K} = \{0,1\}^{\ell}$ *is the key space,* $\mathcal{M} = \{0,1\}^{*}$ *is the message space and* $\mathcal{R} = \{0,1\}^{n}$ *is the range, for some* $\ell, n \geq 1$.

Required Output Length for a Hash Function

An n-bit hash function is said to have **ideal security** if the following conditions hold:

Required Output Length for a Hash Function

An n-bit hash function is said to have **ideal security** if the following conditions hold:

i. The expected workload of generating *a collision* $= 2^{n/2}$.

ii. Given a hash value, the expected workload of *finding a preimage* $= 2^n$.

iii. Given a message and its hash result, the expected workload of *finding a second preimage* $= 2^n$.

Generic Algorithm: Pre-Image

- Model H as a uniform random function, i.e., on distinct inputs, the outputs of H are independent and uniformly distributed over $\{0,1\}^n$.
- Finding pre-image: input y.

Generic Algorithm: Pre-Image

- Model H as a uniform random function, i.e., on distinct inputs, the outputs of H are independent and uniformly distributed over $\{0,1\}^n$.
- Finding pre-image: input y.
- Choose M; compute $H(M)$; if $H(M) = y$, return M.

Generic Algorithm: Pre-Image

- Model H as a uniform random function, i.e., on distinct inputs, the outputs of H are independent and uniformly distributed over $\{0,1\}^n$.
- Finding pre-image: input y.
- Choose M; compute $H(M)$; if $H(M) = y$, return M.
- Probability of success: $Pr[H(M) = y] = 1/2^n$.
- Expected number of trials: 2^n.
- Similarly, for finding 2^{nd} pre-image, the expected number of trials is also 2^n.

Generic Algorithm: Collision

Birthday Attack

Problem

Let there be $m+1$ people $\{P_1, P_2, \ldots, P_{m+1}\}$ in a room. What should be the value of m so that the probability that atleast one of the persons $\{P_2, P_3, \ldots, P_{m+1}\}$ shares birthday with P_1 is greater than $\frac{1}{2}$?

Generic Algorithm: Collision

Birthday Attack

Problem

Let there be $m+1$ people $\{P_1, P_2, \ldots, P_{m+1}\}$ in a room. What should be the value of m so that the probability that atleast one of the persons $\{P_2, P_3, \ldots, P_{m+1}\}$ shares birthday with P_1 is greater than $\frac{1}{2}$?

Problem

How many people must be there in a room, so that the probability of atleast 2 of them sharing the same birthday is greater than $\frac{1}{2}$?

Generic Algorithm: Collision

- Choose distinct $M_1, M_2, \cdots, M_q$;
- compute $y_1 = H(M_1), y_2 = H(M_2), \cdots, y_q = H(M_q)$;
- if $y_i = y_j$, return M_i, M_j.

Generic Algorithm: Collision

- Choose distinct $M_1, M_2, \cdots, M_q$;
- compute $y_1 = H(M_1), y_2 = H(M_2), \cdots, y_q = H(M_q)$;
- if $y_i = y_j$, return M_i, M_j.

$$Pr[Coll] = 1 - Pr[Distinct(y_1, \cdots, y_q)].$$

Generic Algorithm: Collision

- Choose distinct $M_1, M_2, \cdots, M_q$;
- compute $y_1 = H(M_1), y_2 = H(M_2), \cdots, y_q = H(M_q)$;
- if $y_i = y_j$, return M_i, M_j.

$$Pr[Coll] = 1 - Pr[Distinct(y_1, \cdots, y_q)].$$

$$Pr[Distinct(y_1, \cdots, y_q)] =$$

$$\left(1 - \frac{1}{2^n}\right) \times \cdots \times \left(1 - \frac{q-1}{2^n}\right)$$

- Using standard approximations and simplifications, for $q \approx 2^{n/2}$, a collision occurs with constant probability.

Relations Among Properties

- If one can find 2^{nd} pre-images, then one can find collisions.

Relations Among Properties

- If one can find 2^{nd} pre-images, then one can find collisions.
  - Suppose $\mathcal{A}$ is an algorithm to find 2^{nd} pre-images.
  - take an arbitrary x_1;
  - apply $\mathcal{A}$ on x_1 to find a 2^{nd} pre-image x_2;
  - return x_1 and x_2.

Relations Among Properties

- If one can find 2^{nd} pre-images, then one can find collisions.
  - Suppose $\mathcal{A}$ is an algorithm to find 2^{nd} pre-images.
  - take an arbitrary x_1;
  - apply $\mathcal{A}$ on x_1 to find a 2^{nd} pre-image x_2;
  - return x_1 and x_2.
- Collision resistance $\Rightarrow$ 2^{nd} pre-image resistance.
- Collision resistance $\not\Rightarrow$ pre-image resistance.

Relations Among Properties

- No clear deterministic relation between finding pre-images and finding collisions.
- There is, however, a probabilistic relation.

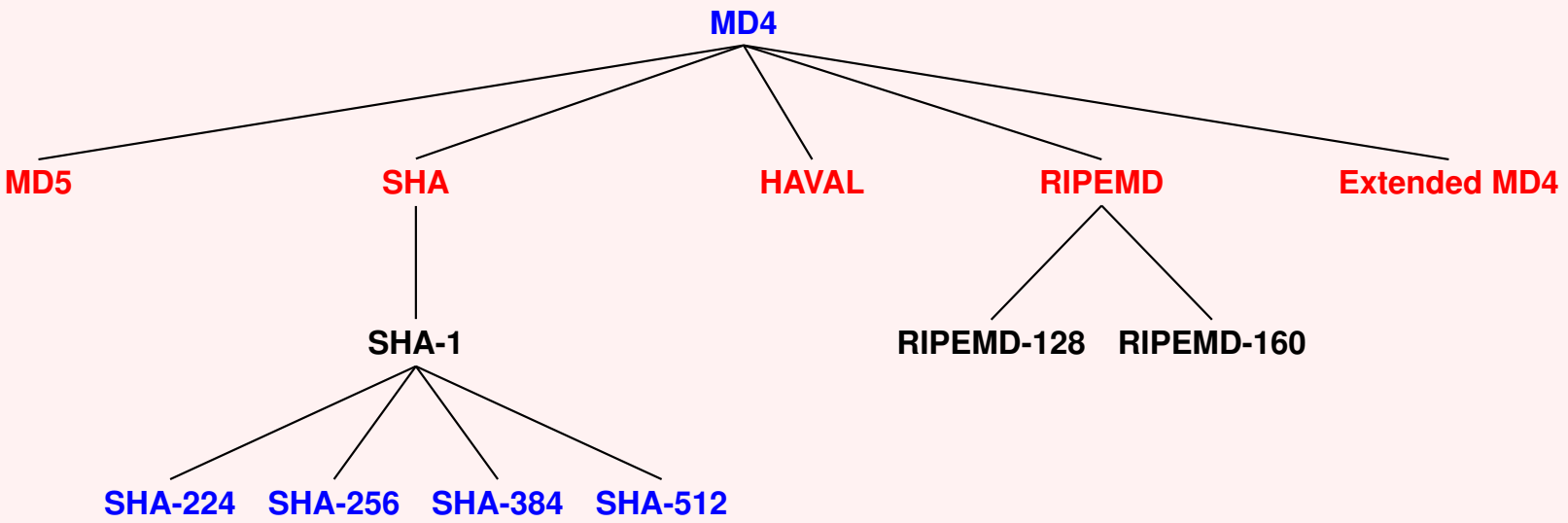
Relations Among Properties

- No clear deterministic relation between finding pre-images and finding collisions.
- There is, however, a probabilistic relation.
  - Suppose $\mathcal{B}$ is an algorithm to find pre-images.
  - take an arbitrary x_1;
  - compute $y = H(x_1)$;
  - apply $\mathcal{B}$ on y to find a pre-image x_2;
  - return x_1 and x_2.
- Under some assumptions, x_2 is different from x_1 with significant probability.

Outline

MD4 Family

MD4 Family



MD4 Family

- **MD4**
  - -> 3 rounds of 16 steps, output bit-length is 128.
- **MD5**
  - -> 4 rounds of 16 steps, output bit-length is 128.

Designed by Ron Rivest in 1991 & 1992 rsp

MD4 Family

- **MD4**
  - -> 3 rounds of 16 steps, output bit-length is 128.
- **MD5**
  - -> 4 rounds of 16 steps, output bit-length is 128.

Designed by Ron Rivest in 1991 & 1992 rsp

- **SHA-1**
  - -> 4 rounds of 20 steps, output bit-length is 160.

Designed by NIST in 1995 (FIPS-180-1)

MD4 Family

- **MD4**
  - -> 3 rounds of 16 steps, output bit-length is 128.
- **MD5**
  - -> 4 rounds of 16 steps, output bit-length is 128.

Designed by Ron Rivest in 1991 & 1992 rsp

- **SHA-1**
  - -> 4 rounds of 20 steps, output bit-length is 160.

Designed by NIST in 1995 (FIPS-180-1)

- **RIPEMD-160**
  - -> 5 rounds of 16 steps, output bit-length is 160.

Designed by Dobbertin, Bosselaers & Preneel in 1995 (RIPE-RACE 1040)

MD4 Family

- **MD4**
  - -> 3 rounds of 16 steps, output bit-length is 128.
- **MD5**
  - -> 4 rounds of 16 steps, output bit-length is 128.

Designed by Ron Rivest in 1991 & 1992 rsp

- **SHA-1**
  - -> 4 rounds of 20 steps, output bit-length is 160.

Designed by NIST in 1995 (FIPS-180-1)

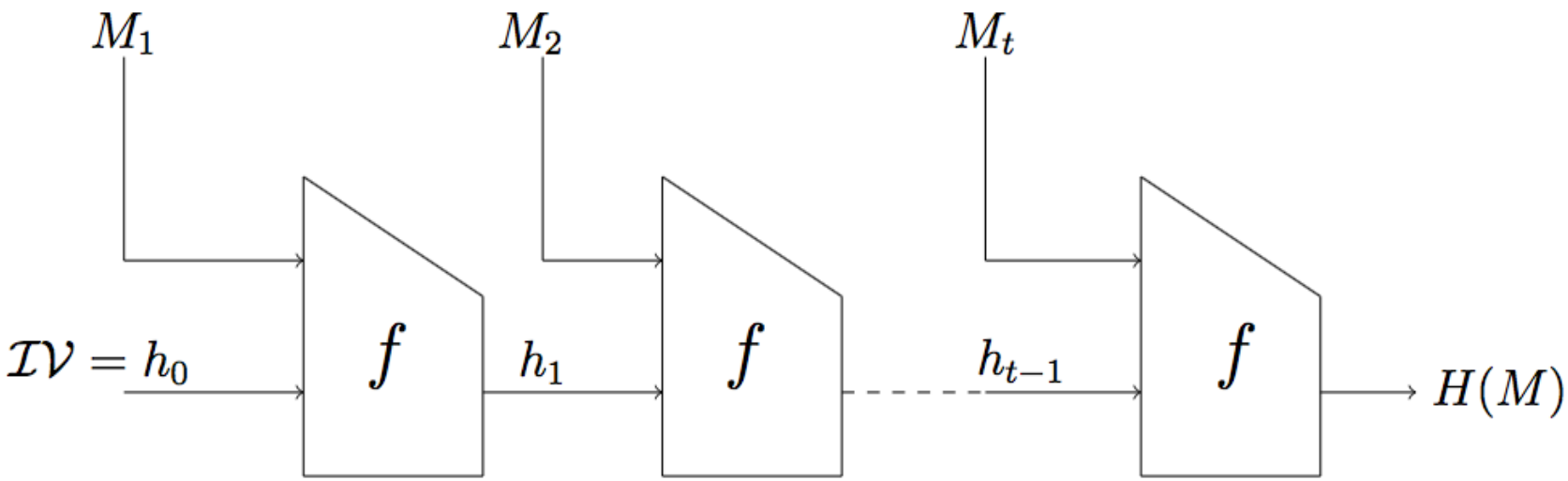
- **RIPEMD-160**
  - -> 5 rounds of 16 steps, output bit-length is 160.

Designed by Dobbertin, Bosselaers & Preneel in 1995 (RIPE-RACE 1040)

- **SHA-2**
  - -> Produces various output bit-lengths: 224, 256, 384 and 512.

Designed by NIST in 2002 (FIPS-180-2)

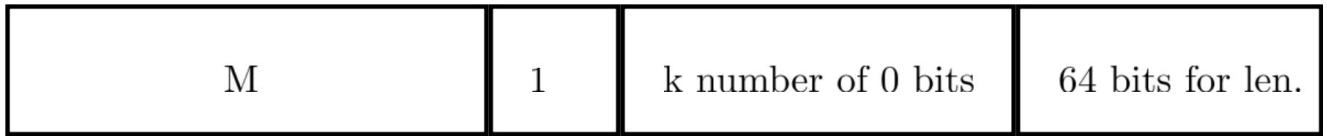
Merkle-Damgård



$$M||Pad(M) = M_1||M_2||\cdots||M_t$$

MD5 Hash

Padding



Word Permutation

$$
\begin{aligned}
p[16 \cdots 31] &= [1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12], \\
p[32 \cdots 47] &= [5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2], \\
p[48 \cdots 63] &= [0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9].
\end{aligned}
$$

MD5 Hash

Algorithm

$$b \leftarrow b + rotl_{r_t}(a + f_t(b,c,d) + K_t + W_{p(t)})$$

$$a \leftarrow d$$

$$d \leftarrow c$$

$$c \leftarrow b$$

$$h_0^{(i)} = a + h_0^{(i-1)},\ h_1^{(i)} = b + h_1^{(i-1)},\ h_2^{(i)} = c + h_2^{(i-1)},\ h_3^{(i)} = d + h_3^{(i-1)}.$$

MD5 Hash

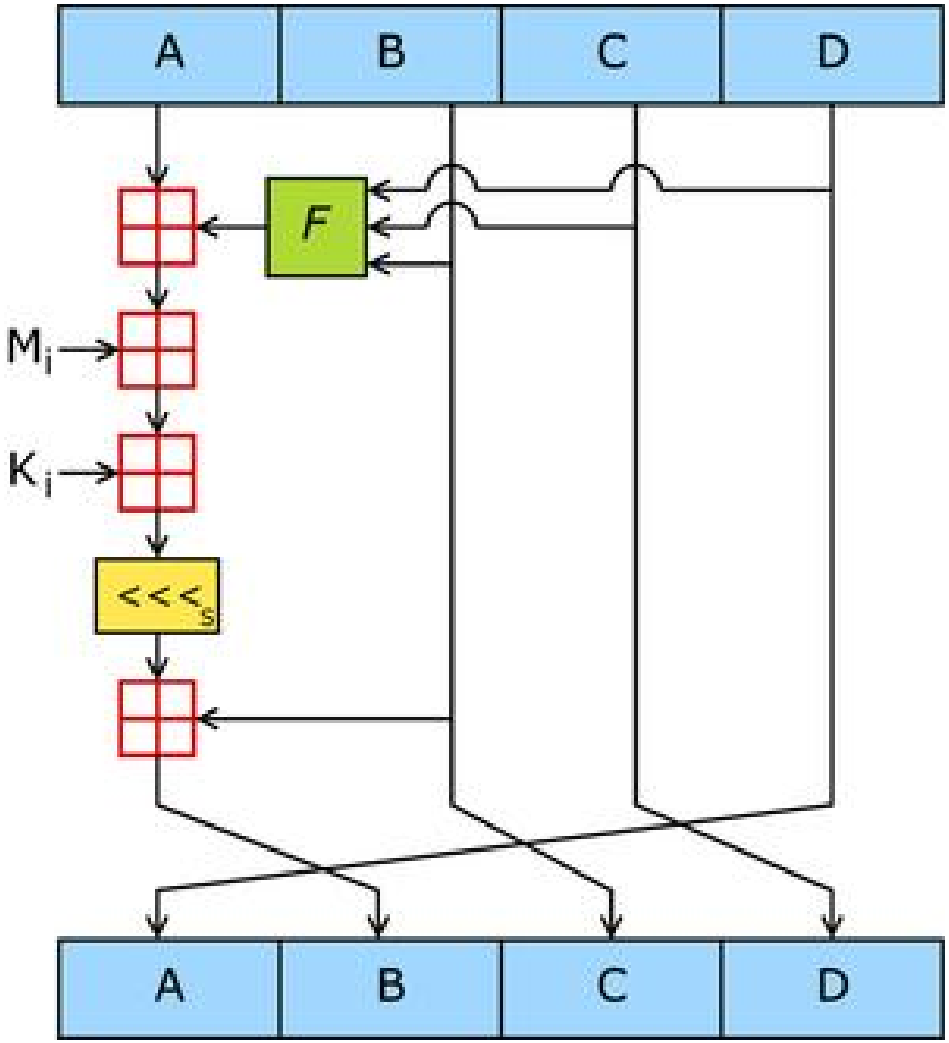
Round Functions

$$
\begin{aligned}
f_t(x,y,z) &= (x \wedge y) \vee (\neg x \wedge z) && 0 \leq t \leq 15 \\
f_t(x,y,z) &= (x \wedge z) \vee (y \wedge \neg z) && 16 \leq t \leq 31 \\
f_t(x,y,z) &= x \oplus y \oplus z && 32 \leq t \leq 47 \\
f_t(x,y,z) &= y \oplus (x \vee \neg z) && 48 \leq t \leq 63
\end{aligned}
$$

Round Constants

$$K_t = first\ 32\ bits\ of\ binary\ value\ of\ |sin(t+1)|, \quad 0 \leq t \leq 63$$

Step Transformation of MD5



Description of SHA-1

Padding

M	1	k number of 0 bits	64 bits for len.

Message Expansion

$$W_t = M_t^{(i)} \qquad 0 \leq t \leq 15$$

$$W_t = rotl^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \qquad 16 \leq t \leq 79$$

Description of SHA-1

Round Operation of Compression Function

$$
\begin{aligned}
T &\leftarrow rotl^5(a) + f_t(b,c,d) + e + K_t + W_t \\
e &\leftarrow d \\
d &\leftarrow c \\
c &\leftarrow rotl^{30}(b) \\
b &\leftarrow a \\
a &\leftarrow T
\end{aligned}
$$

$$
h_0^{(i)} = a + h_0^{(i-1)},\ h_1^{(i)} = b + h_1^{(i-1)},\ h_2^{(i)} = c + h_2^{(i-1)},\ h_3^{(i)} = d + h_3^{(i-1)},
$$
$$
h_4^{(i)} = e + h_4^{(i-1)}.
$$

Description of SHA-1

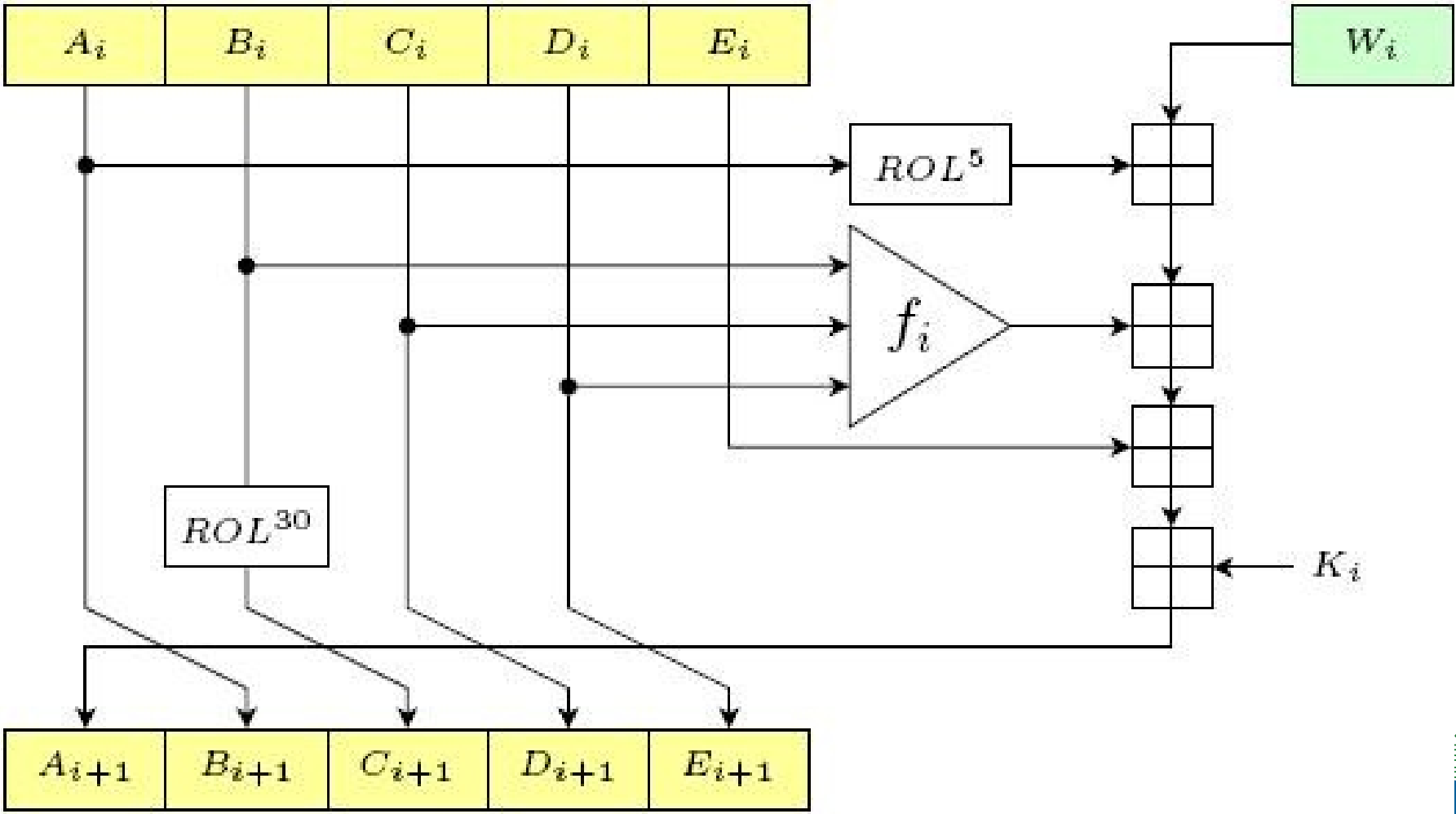
Additive Constants

$$
\begin{aligned}
K_t &= \text{0x5a827999}, && 0 \le t \le 19 \\
K_t &= \text{0x6ed9eba1}, && 20 \le t \le 39 \\
K_t &= \text{0x8f1bbcdc}, && 40 \le t \le 59 \\
K_t &= \text{0xca62c1d6}, && 60 \le t \le 79
\end{aligned}
$$

Round Functions

$$
\begin{aligned}
f_t(x,y,z) &= (x \wedge y) \vee (\neg x \wedge z) && 0 \le t \le 19 \\
f_t(x,y,z) &= (x \oplus y \oplus z) && 20 \le t \le 39 \\
f_t(x,y,z) &= (x \wedge y) \vee (y \wedge z) \vee (z \wedge x) && 40 \le t \le 59 \\
f_t(x,y,z) &= (x \oplus y \oplus z) && 60 \le t \le 79
\end{aligned}
$$

Step Transformation of SHA-1



Description of SHA-256

Padding

M	1	k number of 0 bits	64 bits for len.

Message Expansion

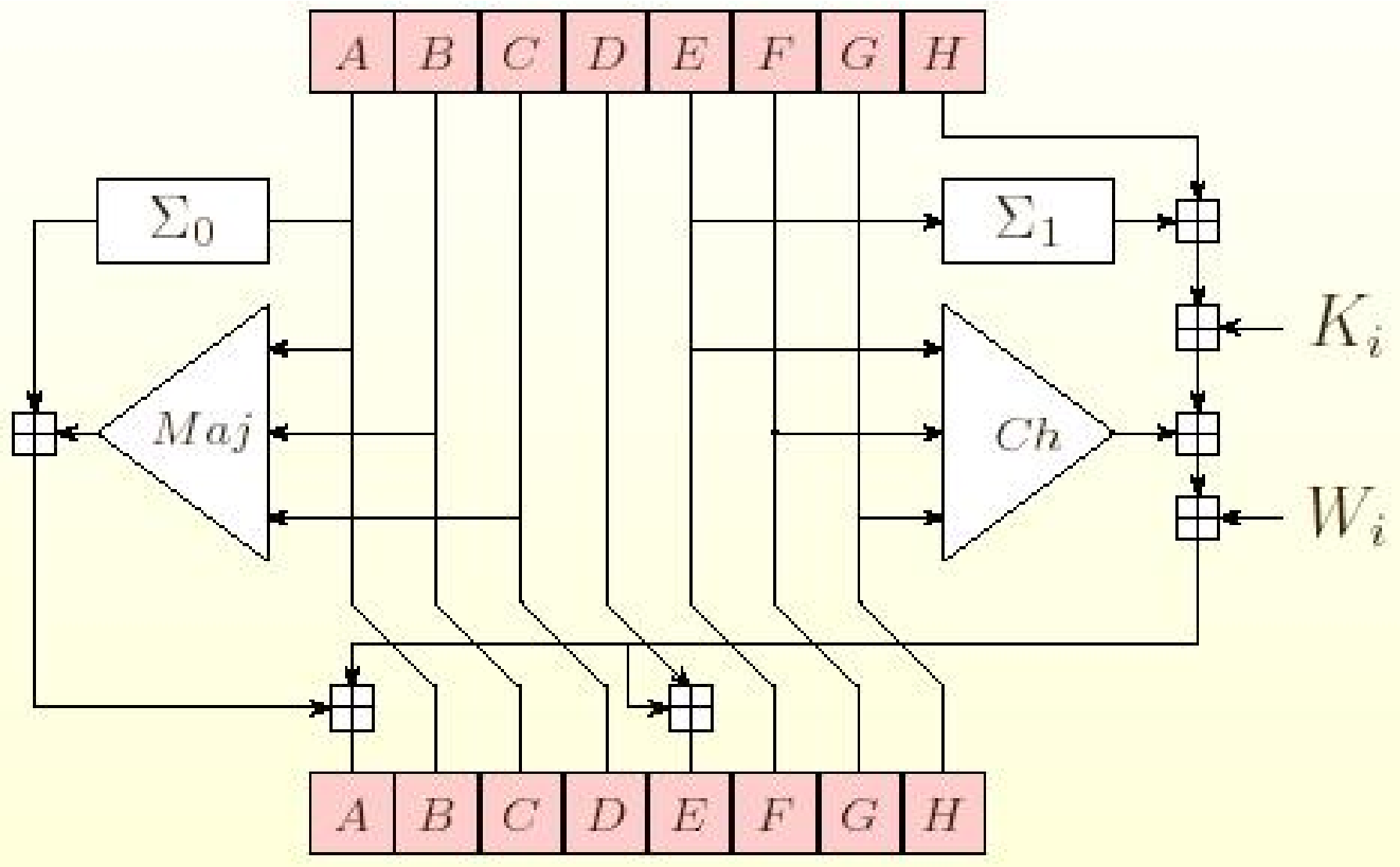
$$W_t = M_t^{(i)} \qquad 0 \le t \le 15$$

$$W_t = \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} \qquad 16 \le t \le 63$$

$$\sigma_0(x) = Rotr_7(x) \oplus Rotr_{18}(x) \oplus Shr_3(x)$$

$$\sigma_1(x) = Rotr_{17}(x) \oplus Rotr_{19}(x) \oplus Shr_{10}(x)$$

Step Transformation of SHA-256



Round Operation of Compression Function of SHA-256

$$
\begin{aligned}
T_1 &\leftarrow H + \Sigma_1(E) + Ch(E,F,G) + K_t + W_t \\
T_2 &\leftarrow \Sigma_0(A) + Maj(A,B,C) \\
H &\leftarrow G \\
G &\leftarrow F \\
F &\leftarrow E \\
E &\leftarrow D + T_1 \\
D &\leftarrow C \\
C &\leftarrow B \\
B &\leftarrow A \\
A &\leftarrow T_1 + T_2
\end{aligned}
$$

Round Operation of Compression Function of SHA-256

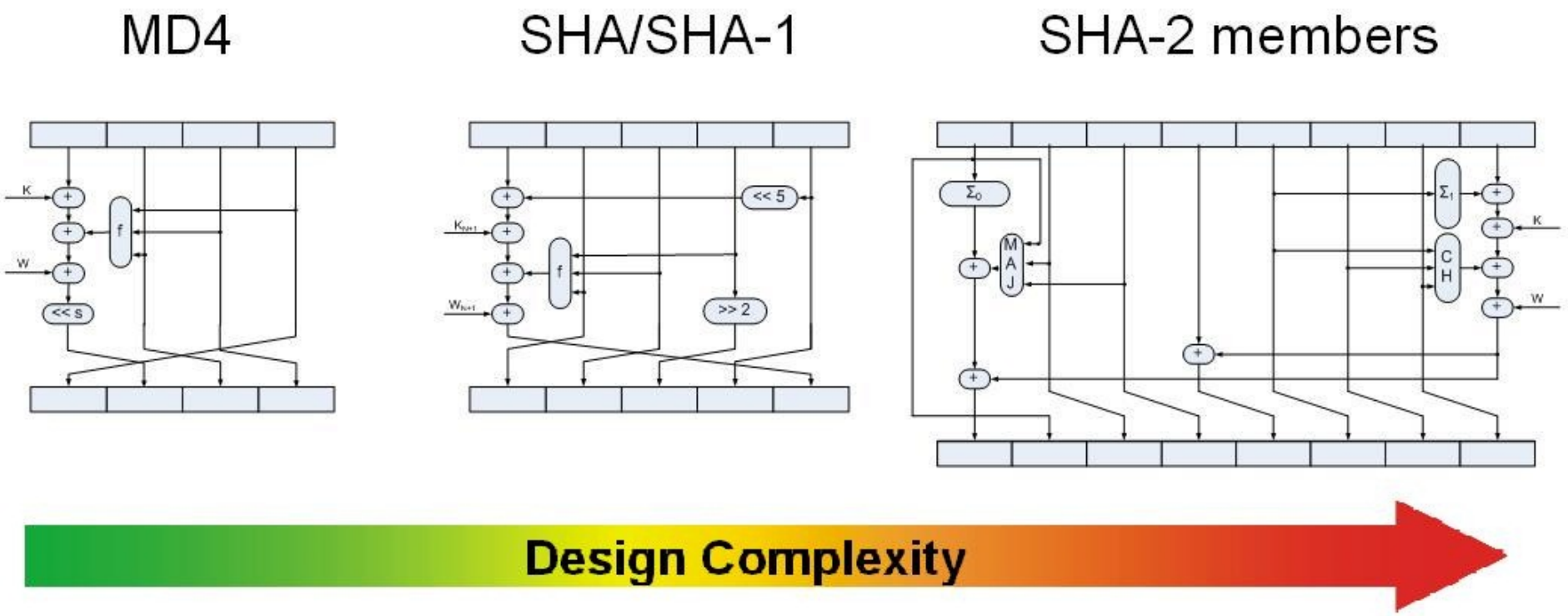
$$\Sigma_0(x) = Rotr_2(x) \oplus Rotr_{13}(x) \oplus Rotr_{22}(x)$$

$$\Sigma_1(x) = Rotr_6(x) \oplus Rotr_{11}(x) \oplus Rotr_{25}(x)$$

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

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

Evolution of MD4



Standard Hash Functions at a Glance

Name	Block Size (bits)	Word Size (bits)	Output Size (bits)	Rounds	Year of the Standard
MD5	512	32	128	64	1992
RIPEMD	512	32	128	48	1992
SHA-0	512	32	160	80	1993
SHA-1	512	32	160	80	1995
RIPEMD-128	512	32	128	64	1995
RIPEMD-160	512	32	160	80	1997
SHA-256	512	32	256	64	2002
SHA-384	1024	64	384	80	2002
SHA-512	1024	64	512	80	2002
SHA-224	512	32	224	64	2004
SHA-3	1600	64	224, 256, 384, 512	24	2015

SHA Family

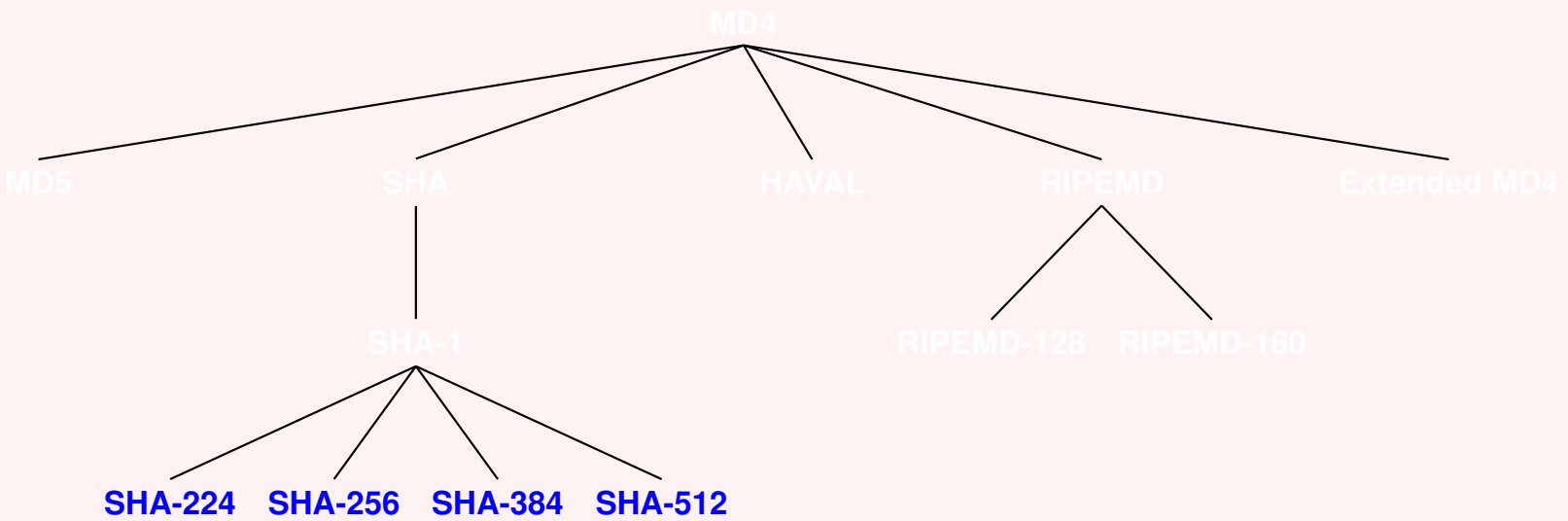
Secure Hash Standard

- SHA-1 (32-bit)
- SHA-224 & SHA-256 Functions (32-bit)
- SHA-384, SHA-512, SHA-512/224 & SHA-512/256 Functions (64-bit)

NIST,
Secure Hash Standard (SHS), FIPS PUB 180-4, 2012.

MD4 Family

MD4 Family



Hash Stew

Pour the initial value in a big cauldron and place it over a nice fire. Now slowly add salt if desired and stir well. Marinade your input bit string by appending some strengthened padding. Now chop the resulting bit string into nice small pieces (512-bit) of the same size and stretch each piece to at least 4 times its original length. Slowly add each single piece while continually stirring at the speed given by rotation constants and spicing it up with some addition constants. When the ***hash stew*** *is ready, extract a nice portion of at least 224 bits* [1] *and present this hash value on warm with some garnish.*

[1] Earlier it was 160 bits

Hash Stew

Pour the initial value in a big cauldron and place it over a nice fire. Now slowly add salt if desired and stir well. Marinade your input bit string by appending some strengthened padding. Now chop the resulting bit string into nice small pieces (512-bit) of the same size and stretch each piece to at least 4 times its original length. Slowly add each single piece while continually stirring at the speed given by rotation constants and spicing it up with some addition constants. When the ***hash stew*** *is ready, extract a nice portion of at least 224 bits* [1] *and present this hash value on warm with some garnish.*

· · · **Marc Stevens**

[1] Earlier it was 160 bits

Hash Stew

Pour the initial value in a big cauldron and place it over a nice fire. Now slowly add salt if desired and stir well. Marinade your input bit string by appending some strengthened padding. Now chop the resulting bit string into nice small pieces (512-bit) of the same size and stretch each piece to at least 4 times its original length. Slowly add each single piece while continually stirring at the speed given by rotation constants and spicing it up with some addition constants. When the ***hash stew*** *is ready, extract a nice portion of at least 224 bits* [1] *and present this hash value on warm with some garnish.*

· · · **Marc Stevens**

Shattered: The first collision for full SHA-1

[1] Earlier it was 160 bits

Recommended Hash Functions

Primitive	Output Length	Recommendation	
		Legacy	Future
SHA-2	256, 384, 512	✓	✓
SHA3	256, 384, 512	✓	✓
Whirlpool	512	✓	✓
SHA3	224	✓	×
SHA-2	224	✓	×
RIPEMD-160	160	✓	×
SHA-1	160	×	×
MD-5	128	×	×
RIPEMD-128	128	×	×

Algorithms, key size and parameters report – 2014

www.enisa.europa.eu

Recommended Hash Functions

Legacy × — Attack exists or security considered not sufficient.
Mechanism should be replaced in Fielded products as a matter of urgency.

Legacy ✓ — No known weaknesses at present.
Better alternatives exist.
Lack of security proof or limited key size.

Future ✓ — Mechanism is well studied (often with security proof).
Expected to remain secure in 10-50 year lifetime.

Outdated MD5

A quarter of major CMSs use outdated MD5 as the default password hashing scheme

Offenders include WordPress, osCommerce, SuiteCRM, miniBB, SugarCRM, and others.

By Catalin Cimpanu for Zero Day | June 17, 2019 -- 17:48 GMT (23:18 IST) | Topic: Security

https://www.zdnet.com/article/a-quarter-of-major-cmss-use-outdated-md5-as-the-default-password-hashing-scheme/

Outline

How to Build a Hash Function

How to Build a Hash Function

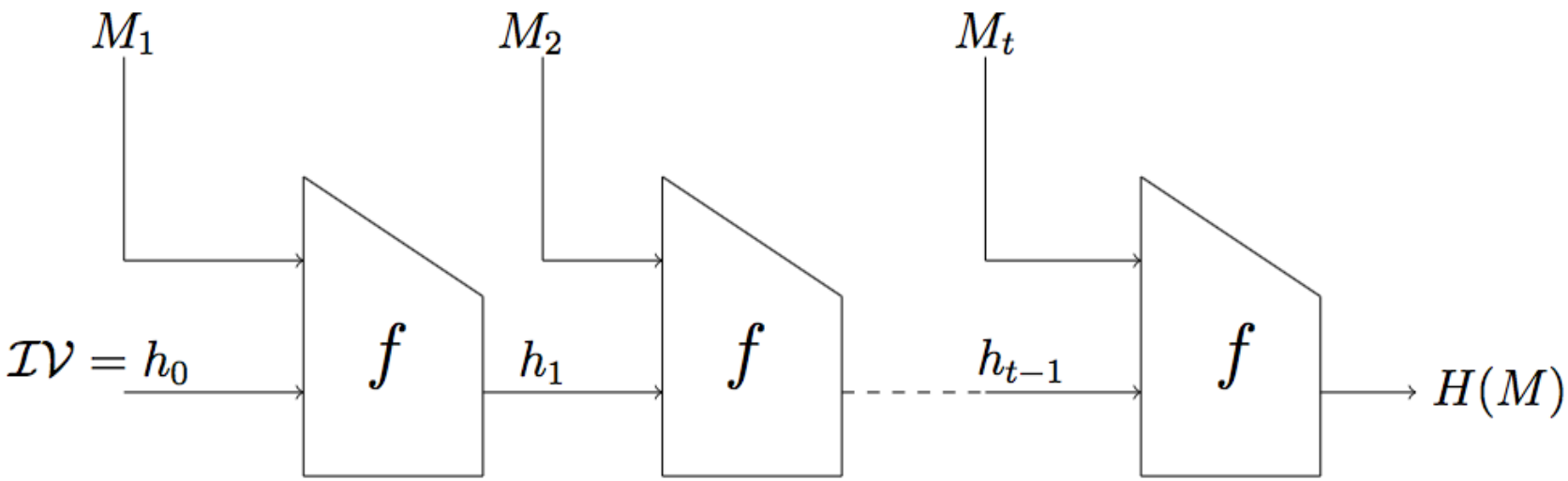
- Design a compression function (a black box that accepts $n + b$-bit & produces n-bit).
- Find a good mode of iteration (a way to handle messages of length longer or shorter than $n + b$-bit).
- Combine the two.

How to Build a Hash Function

- Design a compression function (a black box that accepts $n + b$-bit & produces n-bit).
- Find a good mode of iteration (a way to handle messages of length longer or shorter than $n + b$-bit).
- Combine the two.

Merkle-Damgård Construction

Merkle-Damgård Construction



$$M||Pad(M) = M_1||M_2||\cdots||M_t$$

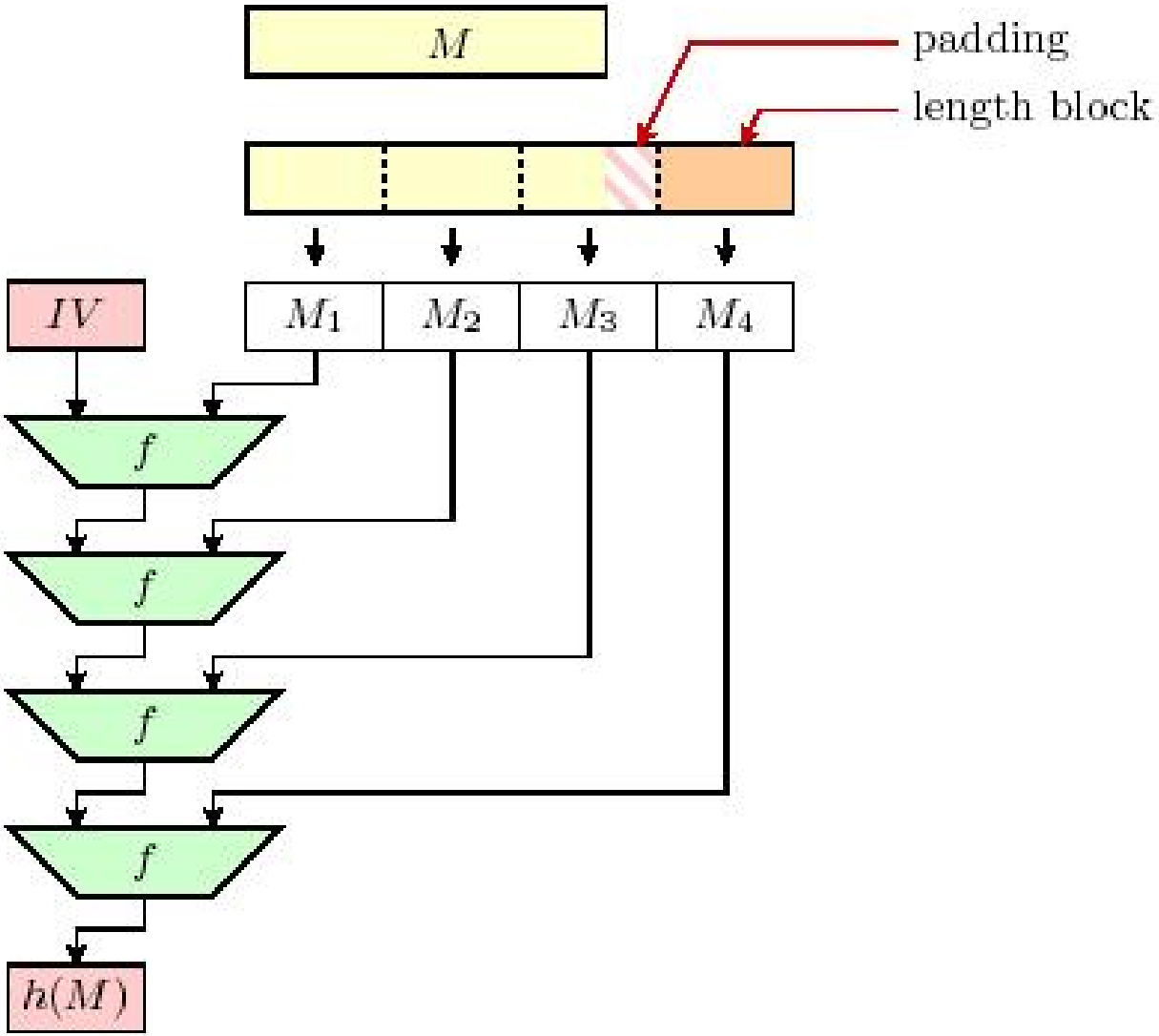
Iterative hash function

- *Compression function* is a function $f : \mathcal{D} \rightarrow \mathcal{R}$, where $\mathcal{D} = \{0,1\}^a \times \{0,1\}^b$ & $\mathcal{R} = \{0,1\}^c$ for some $a, b, c \geq 1$ with $(a + b) \geq c$.
- *Output transformation* is a function $g : \mathcal{D} \rightarrow \mathcal{R}$, where $\mathcal{D} = \{0,1\}^a$ & $\mathcal{R} = \{0,1\}^n$ for some $a, n \geq 1$ with $a \geq n$.

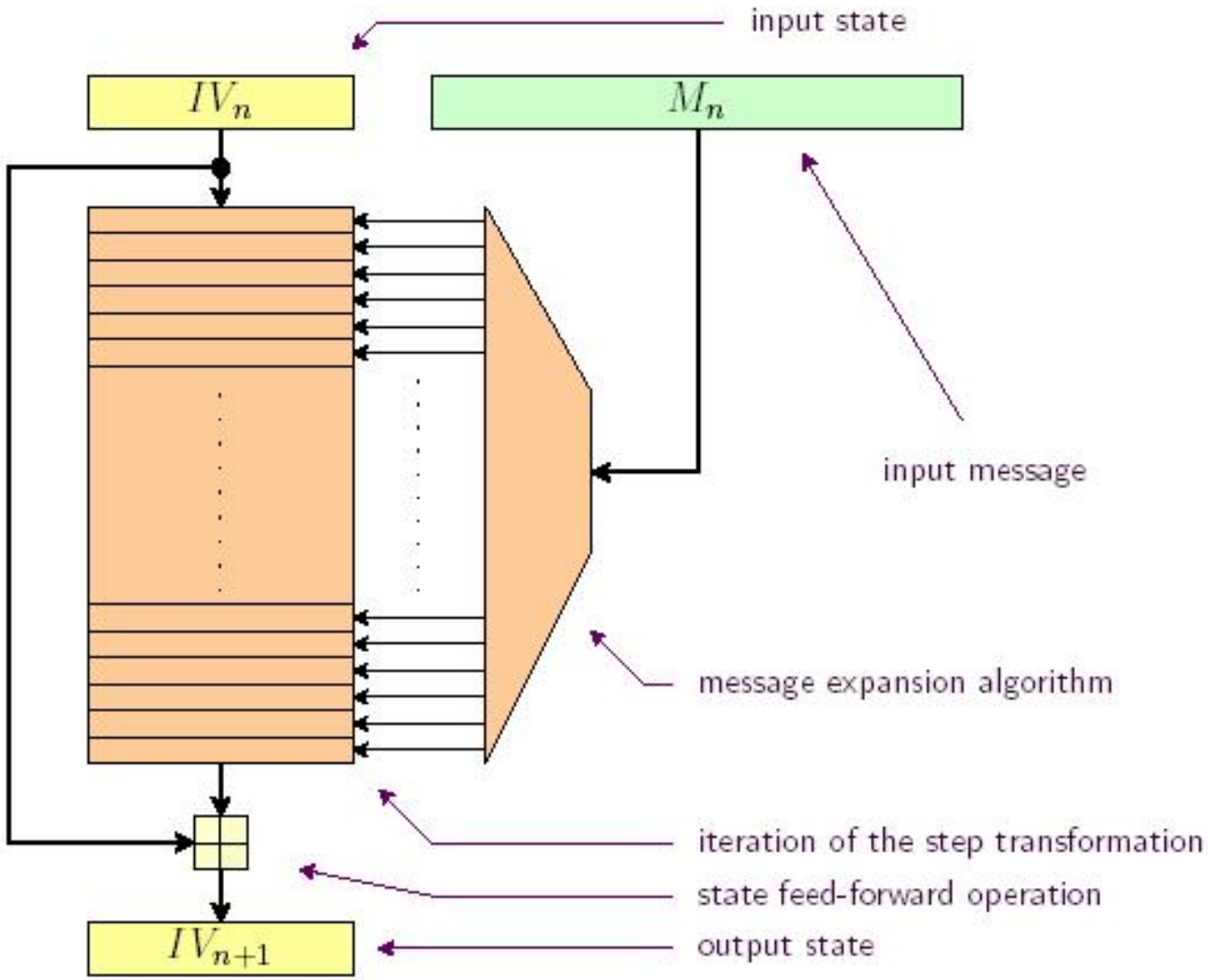
Iterative hash function

- *Compression function* is a function $f : \mathcal{D} \rightarrow \mathcal{R}$, where $\mathcal{D} = \{0,1\}^a \times \{0,1\}^b$ & $\mathcal{R} = \{0,1\}^c$ for some $a, b, c \geq 1$ with $(a + b) \geq c$.
- *Output transformation* is a function $g : \mathcal{D} \rightarrow \mathcal{R}$, where $\mathcal{D} = \{0,1\}^a$ & $\mathcal{R} = \{0,1\}^n$ for some $a, n \geq 1$ with $a \geq n$.
- *Iterative hash function* $h : (\{0,1\}^b)^* \rightarrow \{0,1\}^n$ defined by $h(X_0 \ldots X_{t-1}) = g(H_t)$, where $H_{i+1} = f(H_i, X_i)$ for $0 \leq i \leq t-1$ and the chaining value $H_0 = \mathcal{IV} \in \{0,1\}^c$.

Iterative hash function

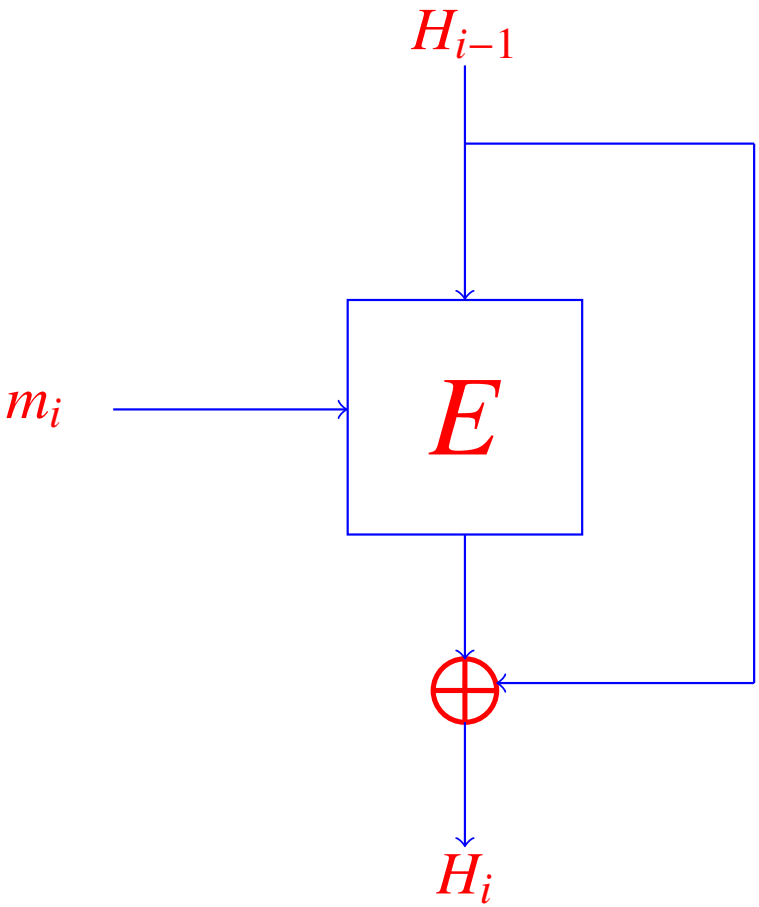


MD & SHA


input state
IV_n
M_n
input message
message expansion algorithm
iteration of the step transformation
state feed-forward operation
output state
IV_{n+1}

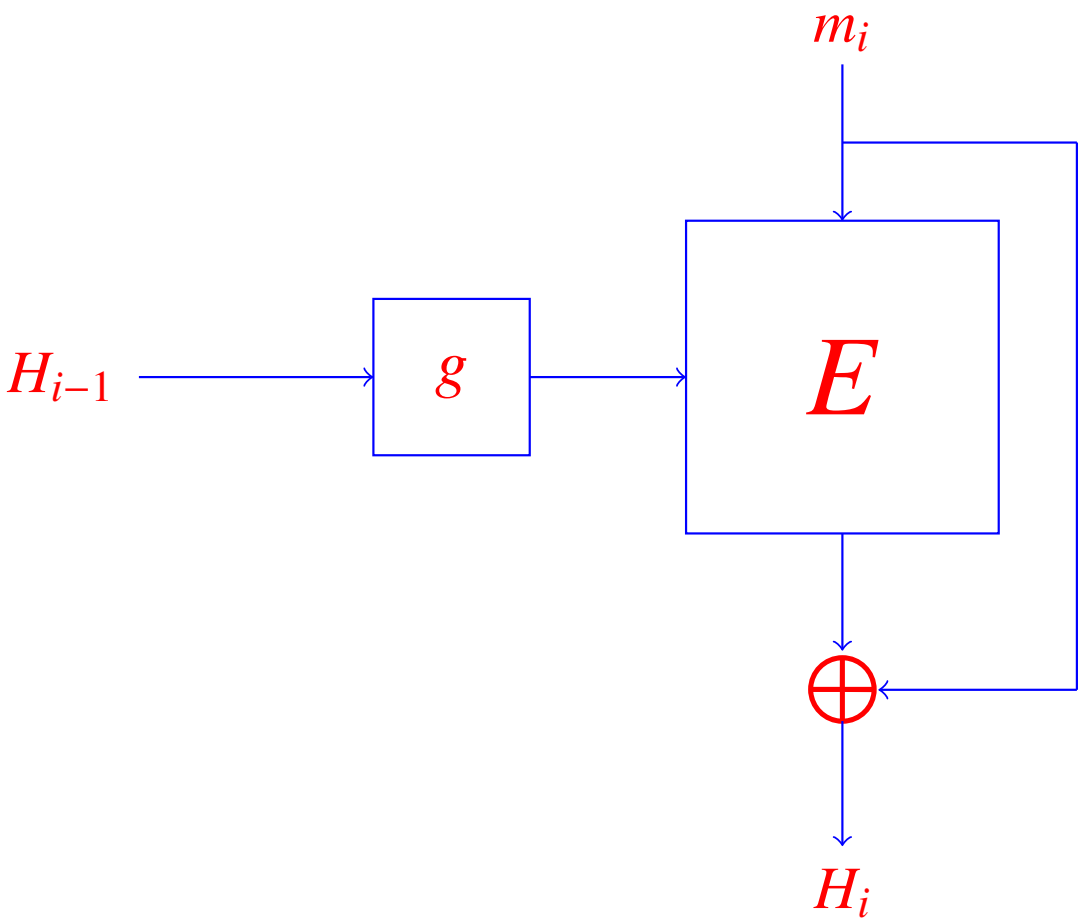
Compression Function Mode

Davis-Meyer Construction



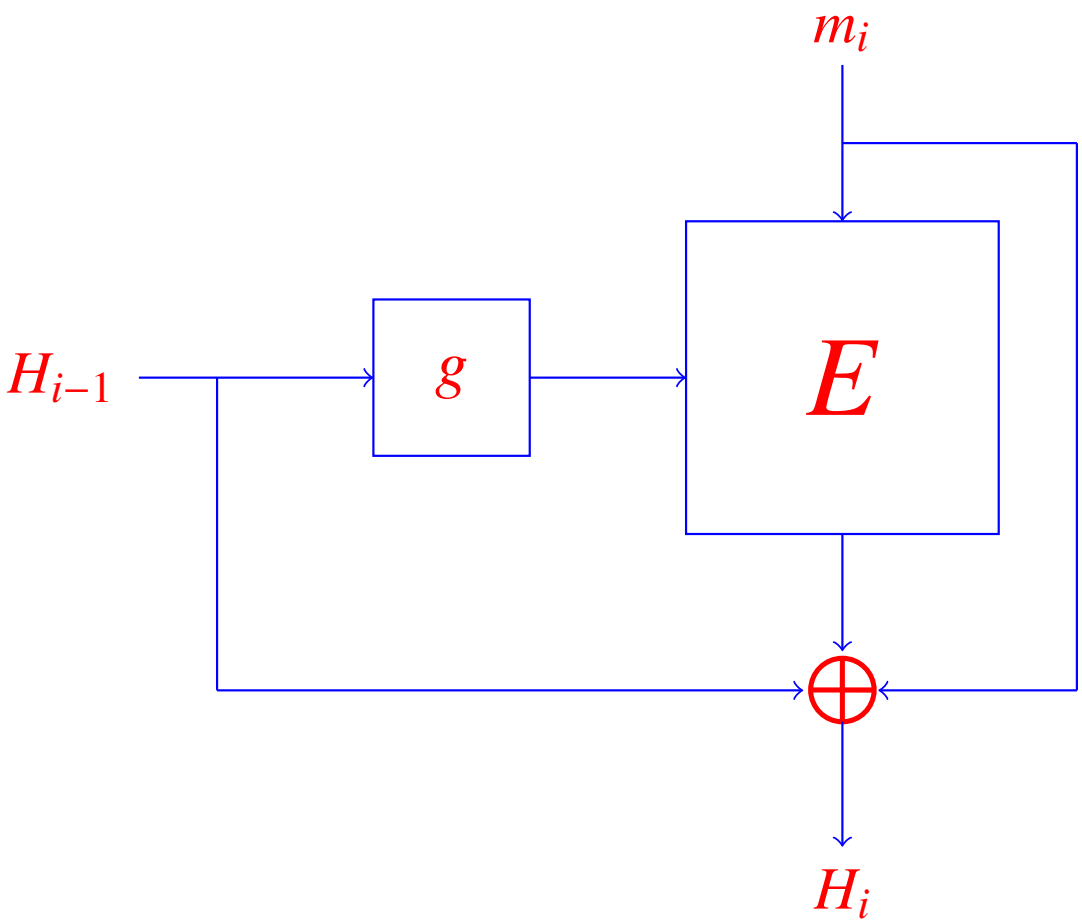
Compression Function Mode

Matyas-Meyer-Oseas



Compression Function Mode

Miyaguchi-Preneel



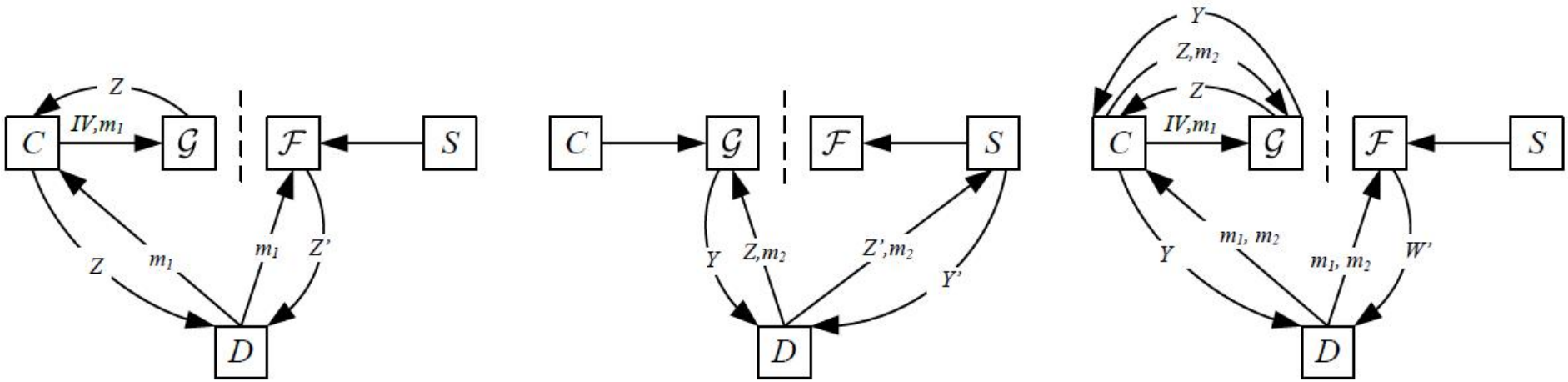
Security of Iterative Hash Function

i. The choice of initial value i.e. $\mathcal{IV}$
   - If $\mathcal{IV}$ is not fixed, collision can be found.

ii. The choice of padding rule
   - If padding procedure does not include length of the input, fixed point attack is possible.

Weaknesses in MD Construction

Indifferentiability Attack



Weaknesses in MD Construction

Length Extension Attack

- Given $h(m)$ and length of the message m.
- m is not known.
- One can compute $h(m||m')$.

Weaknesses in MD Construction

Length Extension Attack

- Given $h(m)$ and length of the message m.
- m is not known.
- One can compute $h(m||m')$.

The HMAC construction works around these problems.

$$HMAC_k(X) = h((k \oplus opad)||h((k \oplus ipad)||X))$$

Weaknesses in MD Construction

One collision $\Longrightarrow$ Infinitely many collisions.

Weaknesses in MD Construction

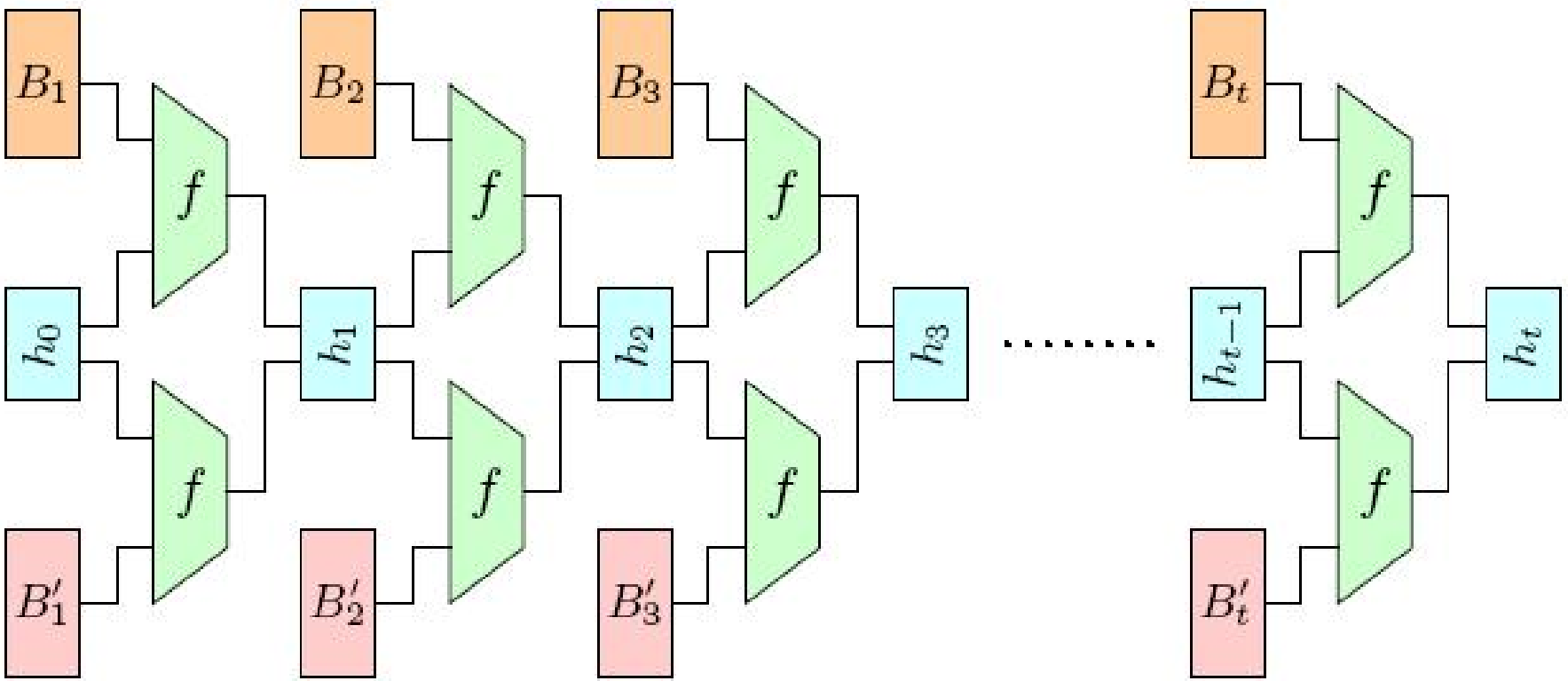
One collision $\Longrightarrow$ Infinitely many collisions.

$$Suppose \quad h(m) = h(m'), \quad where \quad m \neq m' \ \& \ |m| = |m'|$$

$$\Longrightarrow h(m||x) = h(m'||x), \quad \forall \ x.$$

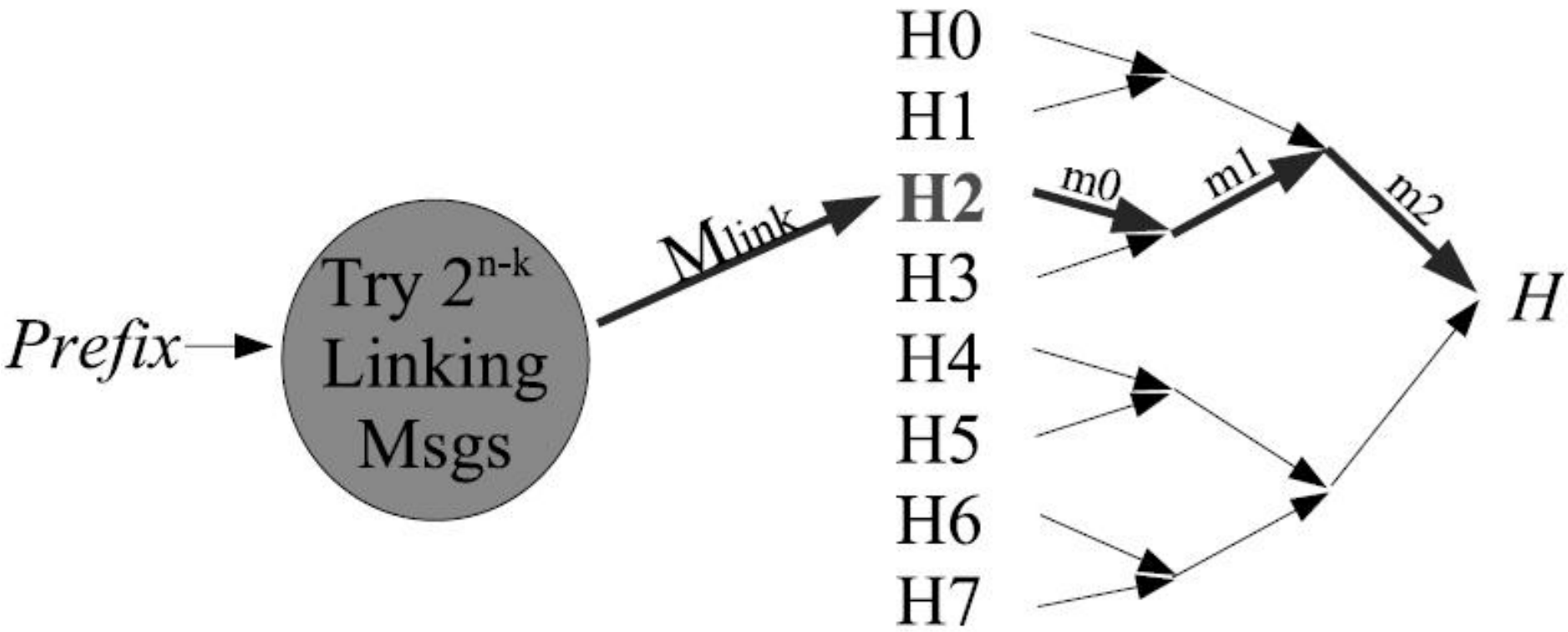
Weaknesses in MD Construction

$$t\ compression\ function\ collisions \Longrightarrow 2^t - multicollision$$



Weaknesses in MD Construction

Herding Attack



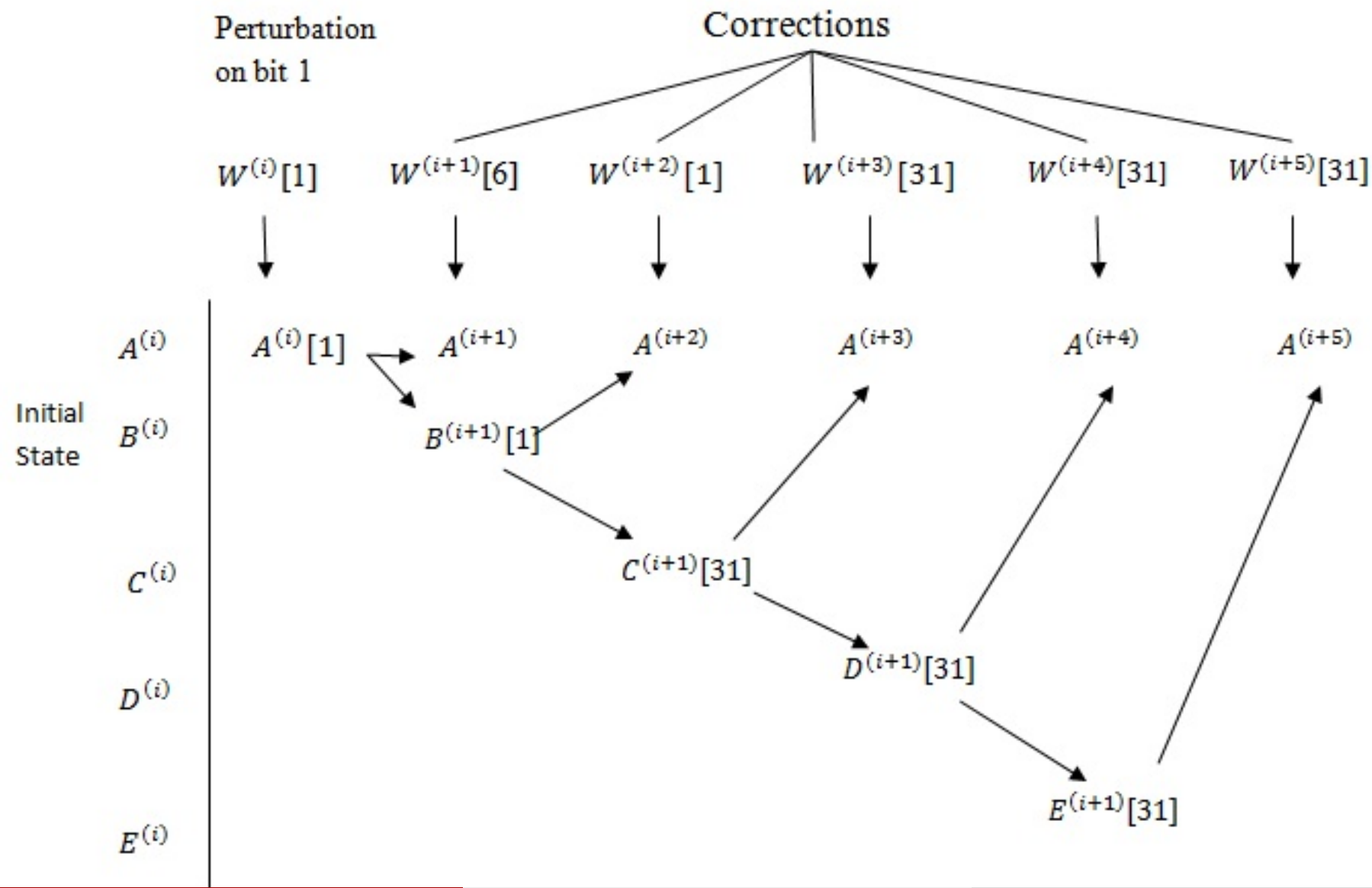
Weaknesses in MD Construction

Herding Attack

Hash Function	output size	diamond width(k)	suffix length (blocks)	work
MD5	128	41	48	2^{87}
SHA-1	160	52	59	2^{108}
SHA-256	256	84	92	2^{172}

J. Kelsey & T. Kohno,
Herding Hash Functions and the Nostradamus Attack,
EUROCRYPT'06, LNCS 4004

Differential Attack of Chabaud & Joux



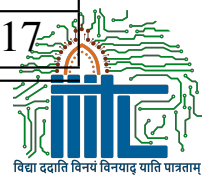
Attacking Step Reduced SHA-2 Family

Cross Dependence Equation

$$E_i = A_i + A_{i-4} - \sum_0(A_{i-1}) - Maj(A_{i-1}, A_{i-2}, A_{i-3}).$$

Attacks on Standard Hash Functions

Hash	Attack			
	Author	Type	Complexity	Year
MD4	Dobbertin	collision	2^{22}	1996
	Wang et. al.	collision	2^{8}	2005
MD5	dan Boer & Bosselaers	pseudo-collision	2^{16}	1993
	Dobbertin	free-start	2^{34}	1996
	Wang et. al.	collision	2^{39}	2005
SHA-0	Chabaud & Joux	collision	2^{61} (theory)	1998
	Biham & Chen	near-collision	2^{40}	2004
	Biham et. al.	collision	2^{51}	2005
	Wang et. al.	collision	2^{39}	2005
SHA-1	Biham et. al.	collision (40 rounds)	very low	2005
	Biham et. al.	collision (58 rounds)	2^{75} (theory)	2005
	Wang et. al.	collision (58 rounds)	2^{33}	2005
	Wang et. al.	collision	2^{63} (theory)	2005
	Stevens et. al.	collision	$< 2^{63.1}$ (practical)	2017

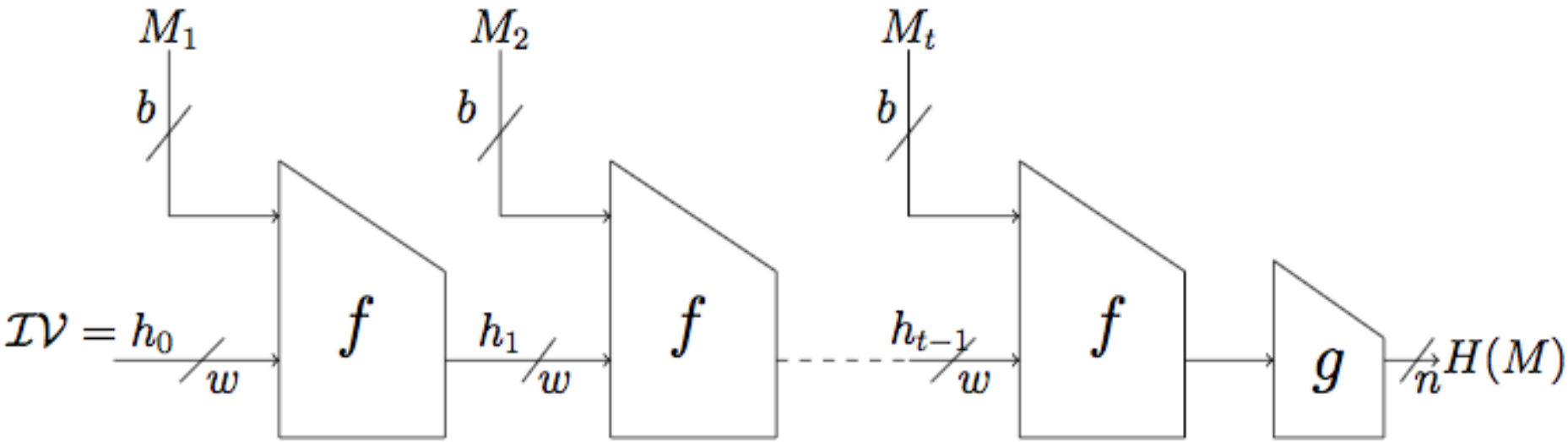


Attacks on Standard Hash Functions

Hash	Attack			
	Author	Type	Complexity	Year
SHA-256	Sarkar et. al.	collision(24 rounds)	$2^{15.5}$	2008
	Sasaki et. al.	preimage(41-step)	$2^{253.5}$	2009
SHA-512	Sarkar et. al.	collision(24 rounds)	$2^{22.5}$	2008
	Sasaki et. al.	preimage(46-step)	$2^{511.5}$	2009

Widepipe/ChopMD

- S. Lucks proposed this design in 2005.
- Designed the hash functions using two compression functions
  i. $f : \{0,1\}^{w+b} \rightarrow \{0,1\}^{w}$
  ii. $g : \{0,1\}^{w} \rightarrow \{0,1\}^{n}$, where $w > n$.



$$M||Pad(M) = M_1||M_2||\cdots||M_t$$

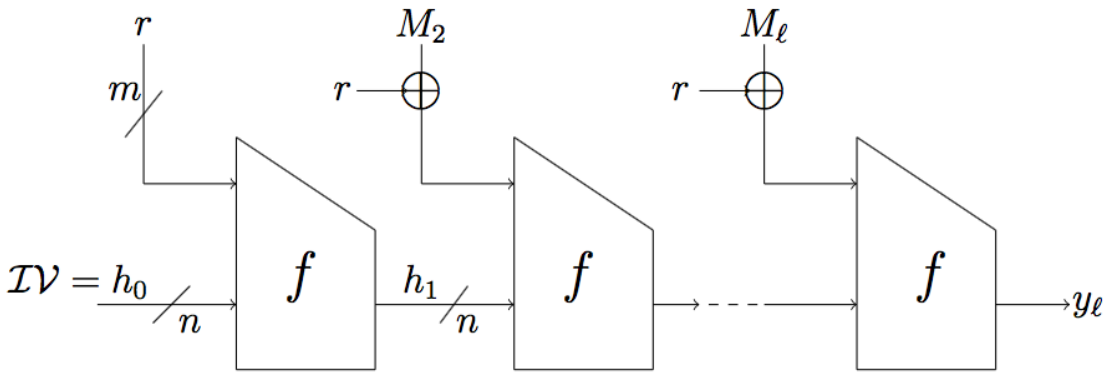
Randomised Hashing

- This was proposed by Halevi and Krawczyk in 2006.
- Designed to strengthen the MD construction.
- Introduced two ways to design this

i. Each message block M_i is XORed with a random block r

$$h_{i+1} := f(h_i, M_i \oplus r).$$

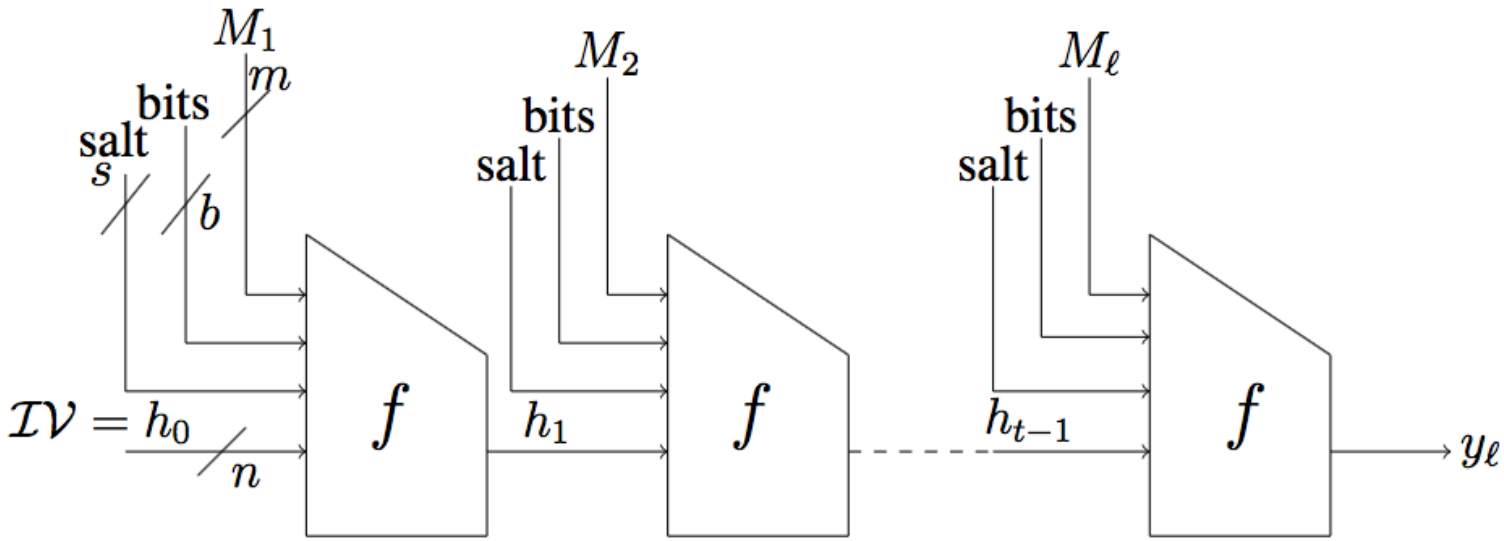
ii. Used a random block r as prefix of the message while still performing XOR with r for all message blocks.



HAIFA (HAsh Iterative FrAmework)

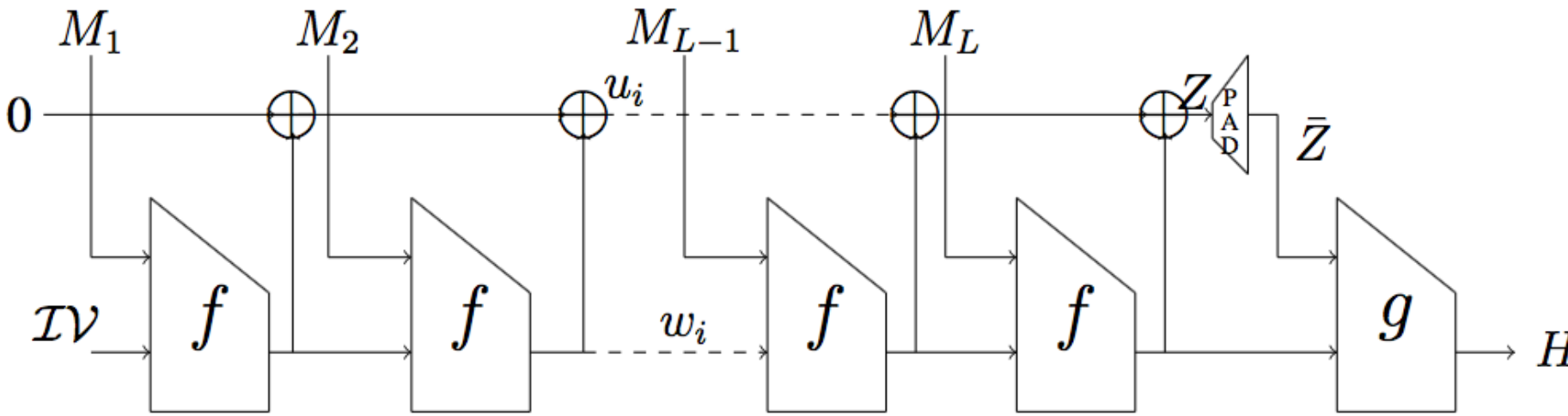
1. It was proposed by Biham and Dunkelman in 2006.
2. Compression function $f : \{0,1\}^{n+m+b+s} \rightarrow \{0,1\}^n$

$$h_{i+1} := f(h_i \parallel M_i \parallel \#bits \parallel salt)$$

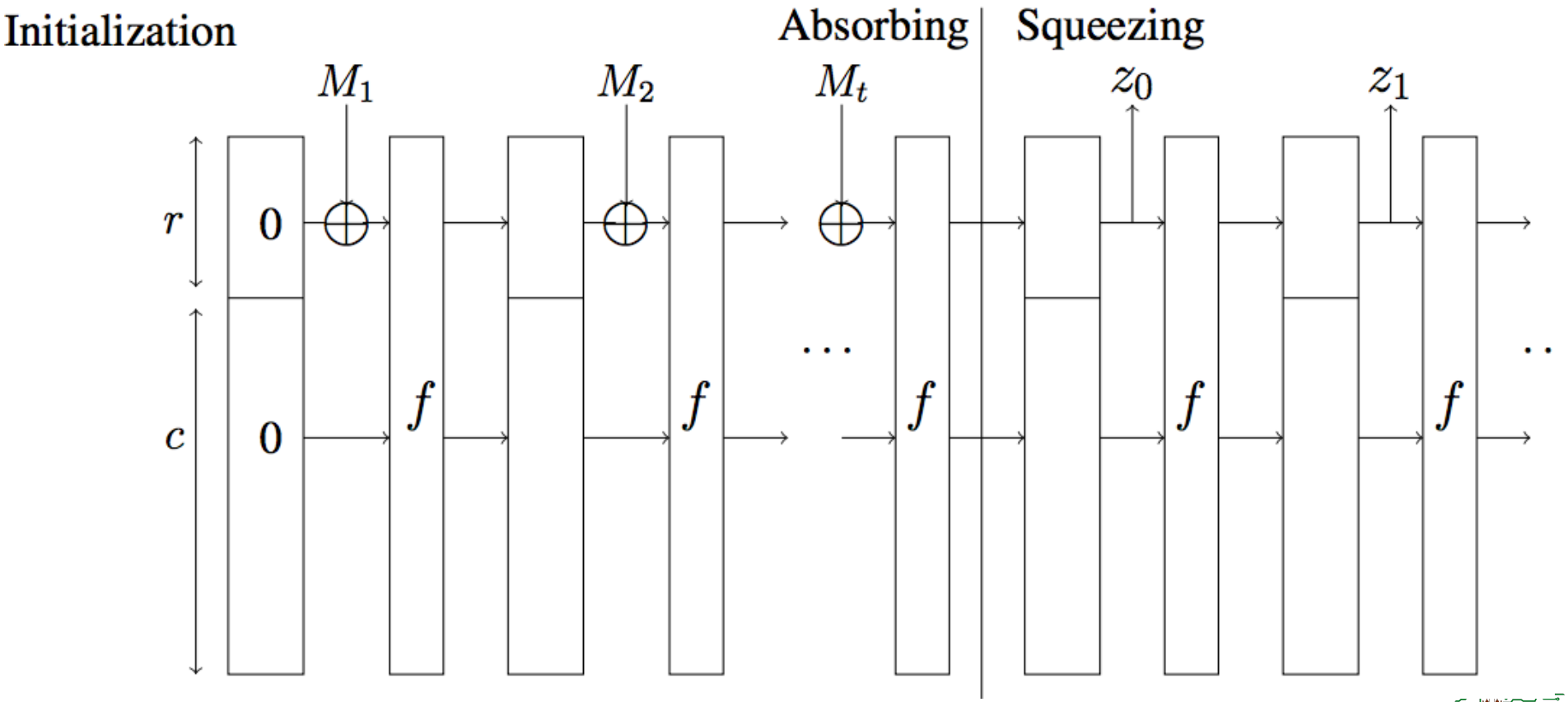


3C Constructions

- Gauravaram proposed this designs in 2006.
- Aimed at strengthening the Merkle-Damgård construction against multi-block collision attacks.



Sponge Construction



Outline

Requirements for SHA-3

- Plug-compatible with SHA-2 in current applications
- Support digests of 224, 256, 384, and 512 bits,
- Support messages of at least 2^{64} bits
- Support digital signatures, hash-based MACs, PRFs, RNGs, KDFs, etc.
- Required security properties

Requirements for SHA-3

- Plug-compatible with SHA-2 in current applications
- Support digests of 224, 256, 384, and 512 bits,
- Support messages of at least 2^{64} bits
- Support digital signatures, hash-based MACs, PRFs, RNGs, KDFs, etc.
- Required security properties
  - Collision resistance of approximately $n/2$ bits,
  - Preimage resistance of approximately n bits,
  - $2nd$-preimage resistance of approximately $n - k$ bits for any message shorter than 2^k bits,
  - Resistance to length-extension attacks.

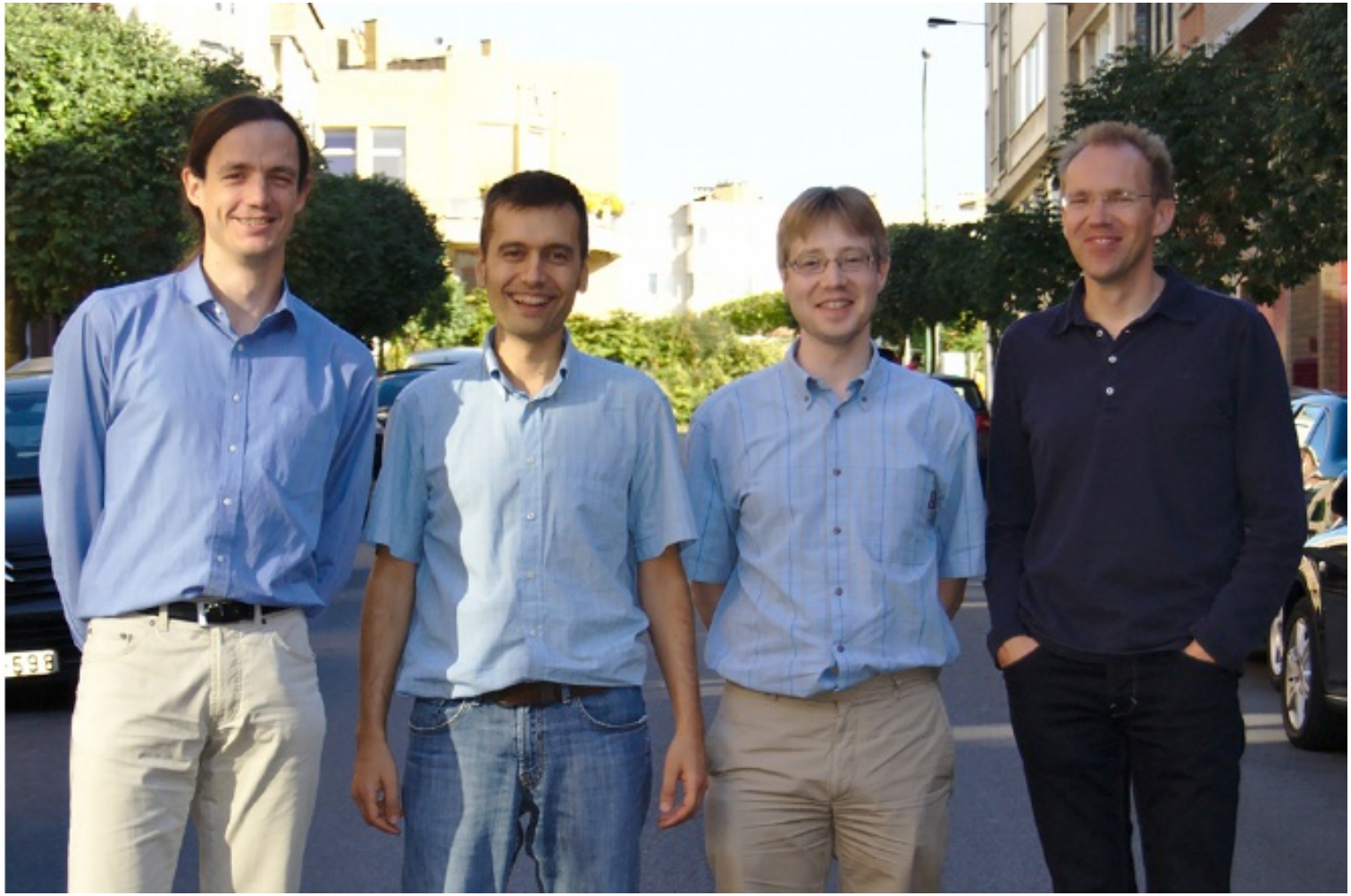
Time Line of Major Events

31 Oct 08	:	SHA-3 Submission Deadline.
09 Dec 08	:	Announced 51 First round candidates
24 Jul 09	:	Announced 14 Second round candidates
09 Dec 10	:	Announced 5 Third round candidates
02 Oct 12	:	Announced the winner - Keccak
31 May 2014	:	Published draft of FIPS 202
5 Aug 2015	:	SHA-3 Standardised, FIPS-202: Permutation based hash and Extendable-output functions (XOFs). SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 and SHAKE256.

Final Round of SHA-3

Algorithm Name	Principal Submitter
BLAKE	Jean-Philippe Aumasson
Grøstl	Lars Ramkilde Knudsen
JH	Hongjun Wu
Keccak	Joan Daemen
Skein	Bruce Schneier

Keccak Team



(L to R) Michaël Peeters, Guido Bertoni, Gilles Van Assche and Joan Daemen

SHA-3 Hash: Keccak

- NIST chose Keccak over the 4 other excellent finalists for its

SHA-3 Hash: Keccak

- NIST chose Keccak over the 4 other excellent finalists for its
  - elegant design,
  - large security margin,
  - good general performance,
  - excellent efficiency in hardware implementations and for its flexibility.

SHA-3 Hash: Keccak

- NIST chose Keccak over the 4 other excellent finalists for its
  - elegant design,
  - large security margin,
  - good general performance,
  - excellent efficiency in hardware implementations and for its flexibility.
- Keccak uses a new “sponge construction" chaining mode, based on a fixed permutation, that can readily be adjusted to trade generic security strength for throughput, and can generate larger or smaller hash outputs as required.

SHA-3 Hash: Keccak

- NIST chose Keccak over the 4 other excellent finalists for its
  - elegant design,
  - large security margin,
  - good general performance,
  - excellent efficiency in hardware implementations and for its flexibility.
- Keccak uses a new “sponge construction" chaining mode, based on a fixed permutation, that can readily be adjusted to trade generic security strength for throughput, and can generate larger or smaller hash outputs as required.
- The Keccak designers have also defined a modified chaining mode for Keccak that provides authenticated encryption.

SHA-3 Hash: Keccak

- Keccak family of hash functions are based on the sponge construction.
- They use as a building block a permutation from a set of 7 permutations $\{viz.,\ 25, 50, 100, 200, 400, 800, 1600\}$.

Algorithm	Rate (r)	Capacity (c)	Depth (d)
Keccak-224	1152	448	28
Keccak-256	1088	512	32
Keccak-384	832	768	48
Keccak-512	576	1024	64

XOFs: Extendable-Output Functions

- In Fips-202, SHA-3 family consists of six functions.
- Four cryptographic hash functions called SHA3-224, SHA3-256, SHA3-384 and SHA3-512 with two extendable-output functions called **SHAKE128** and **SHAKE256** which are

XOFs: Extendable-Output Functions

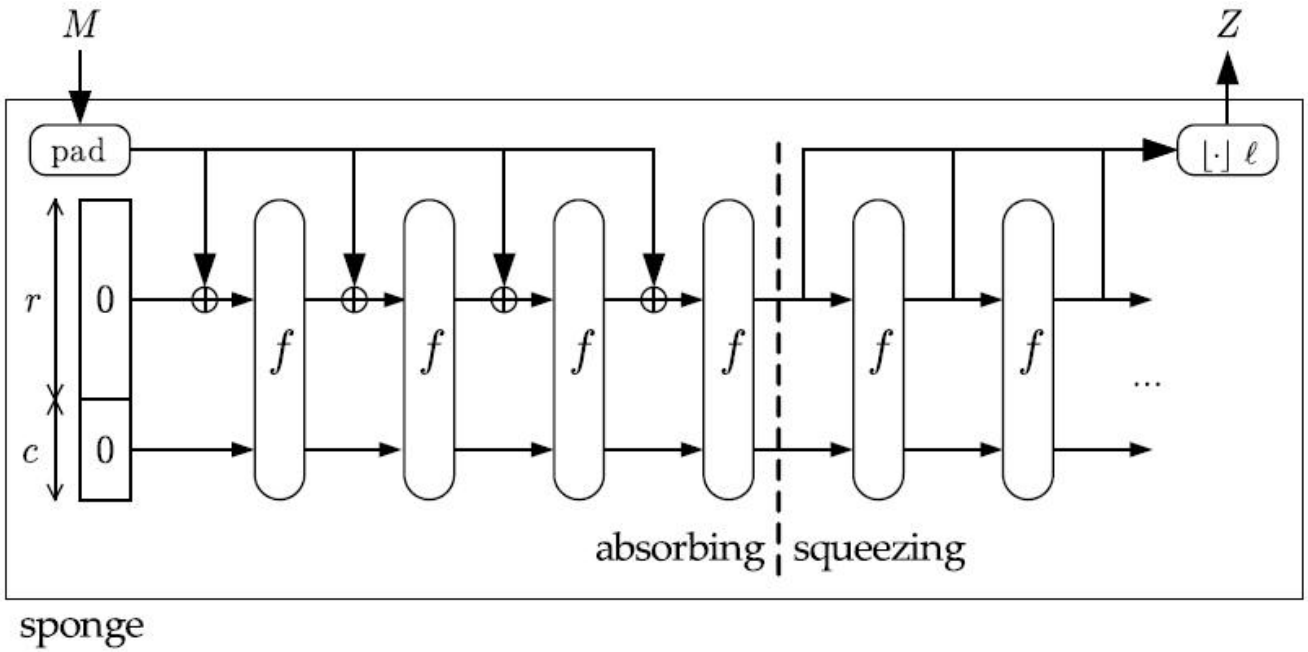
- In Fips-202, SHA-3 family consists of six functions.
- Four cryptographic hash functions called SHA3-224, SHA3-256, SHA3-384 and SHA3-512 with two extendable-output functions called **SHAKE128** and **SHAKE256** which are
  - the first XOFs that NIST have standardised
  - specialized to hash functions in which the output can be extended to any desired length
  - “128" and “256" indicate the security strength in SHAKE128 and SHAKE256

XOFs: Extendable-Output Functions

- In Fips-202, SHA-3 family consists of six functions.
- Four cryptographic hash functions called SHA3-224, SHA3-256, SHA3-384 and SHA3-512 with two extendable-output functions called **SHAKE128** and **SHAKE256** which are
  - the first XOFs that NIST have standardised
  - specialized to hash functions in which the output can be extended to any desired length
  - “128" and “256" indicate the security strength in SHAKE128 and SHAKE256

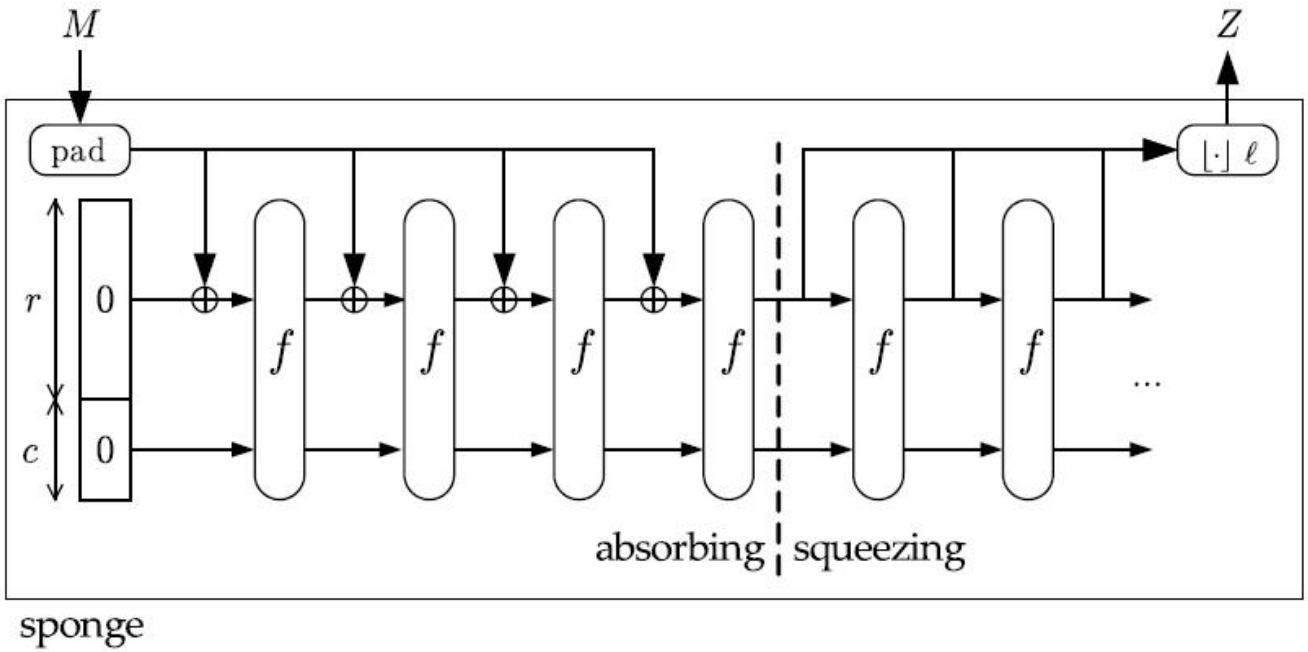
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

The sponge construction



- More general than a hash function:

The sponge construction

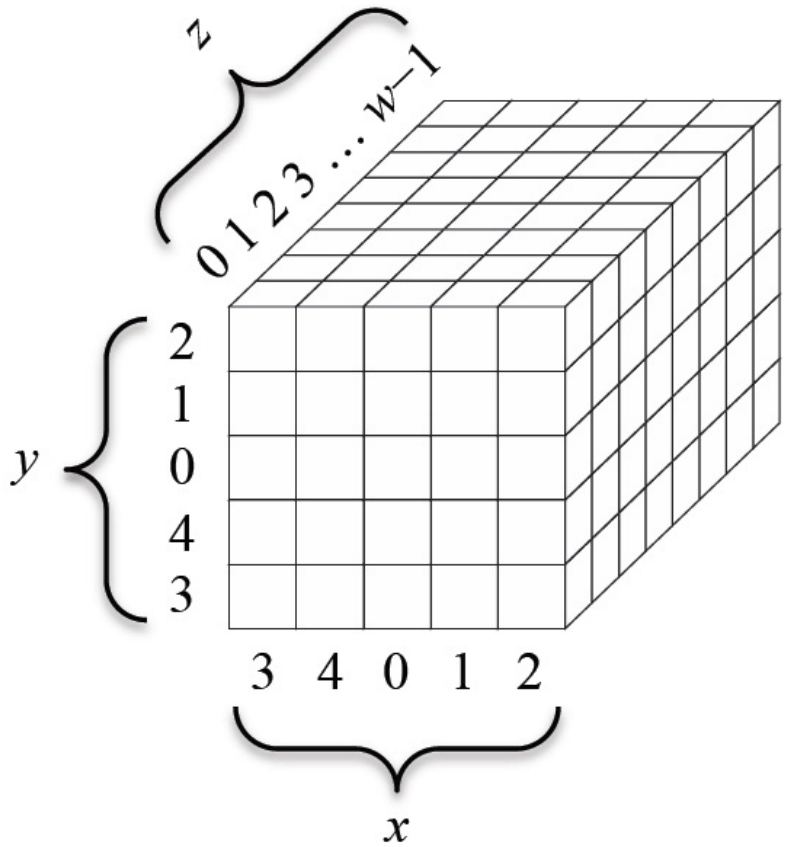


- More general than a hash function: arbitrary-length output
- Calls a b-bit permutation f, with $b = r + c$
  - r bits of rate
  - c bits of capacity (security parameter)

Keccak

- Instantiation of a sponge function
- the permutation Keccak-f
  - 7 permutations: $b \in \{25, 50, 100, 200, 400, 800, 1600\}$
- Security-speed trade-offs using the same permutation, e.g.,
  - SHA-3 instance: $r = 1088$ and $c = 512$
    - permutation width: 1600
    - security strength 256: post-quantum sufficient
  - Lightweight instance: $r = 40$ and $c = 160$
    - permutation width: 200
    - security strength 80: same as SHA-1

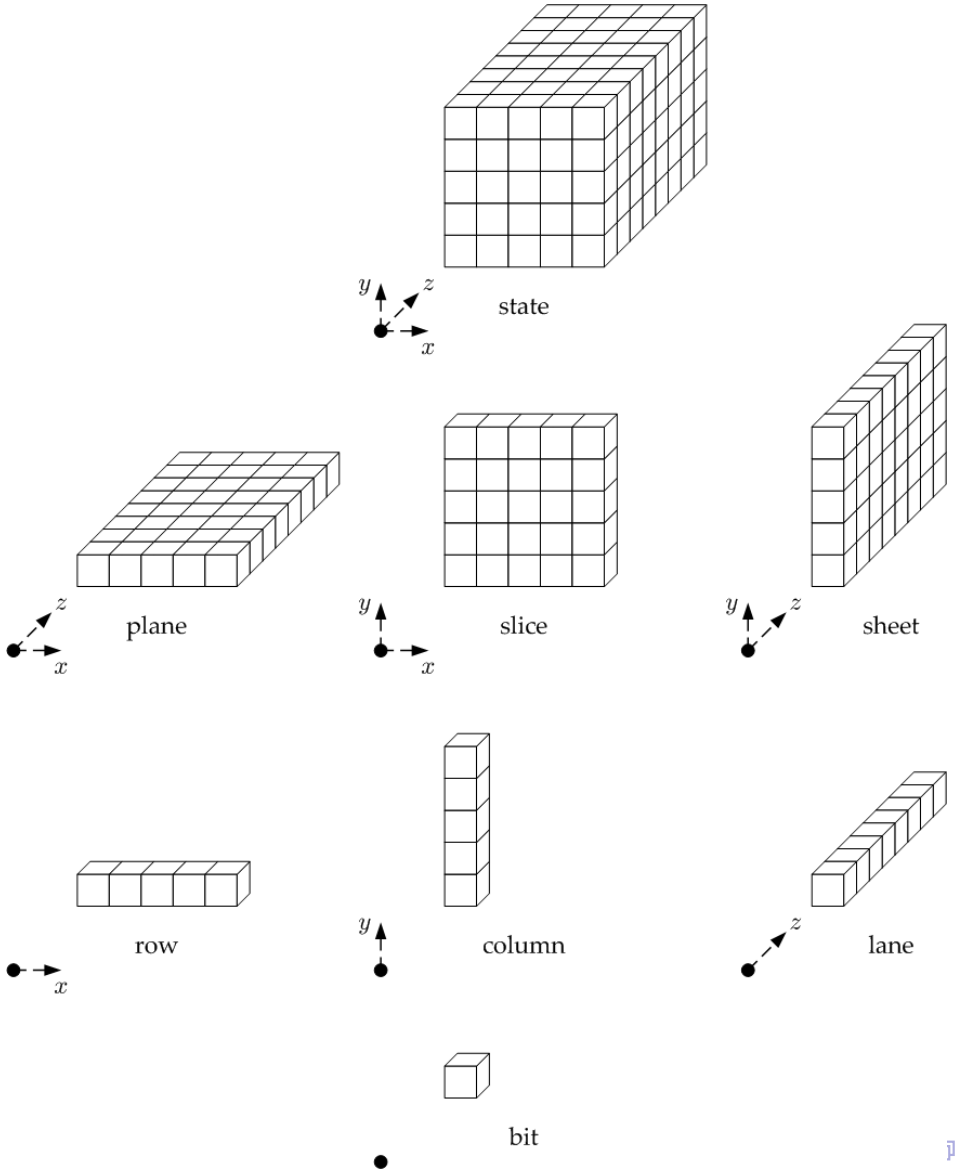
The state: an array of $5 \times 5 \times 2^{\ell}$ bits



- 5×5 lanes, each containing 2^{ℓ} bits (1, 2, 4, 8, 16, 32 or 64)
- (5×5)-bit slices, 2^{ℓ} of them

https://summerschool-croatia.cs.ru.nl/2015/SHA3.pdf

Pieces of State in Keccak



Keccak-f summary

- **Round function:**

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

- **Number of rounds:** $12 + 2\ell$
  - Keccak-f[25] has

Keccak-f summary

- **Round function:**

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

- **Number of rounds:** $12 + 2\ell$
  - Keccak-f[25] has 12 rounds
  - Keccak-f[1600] has

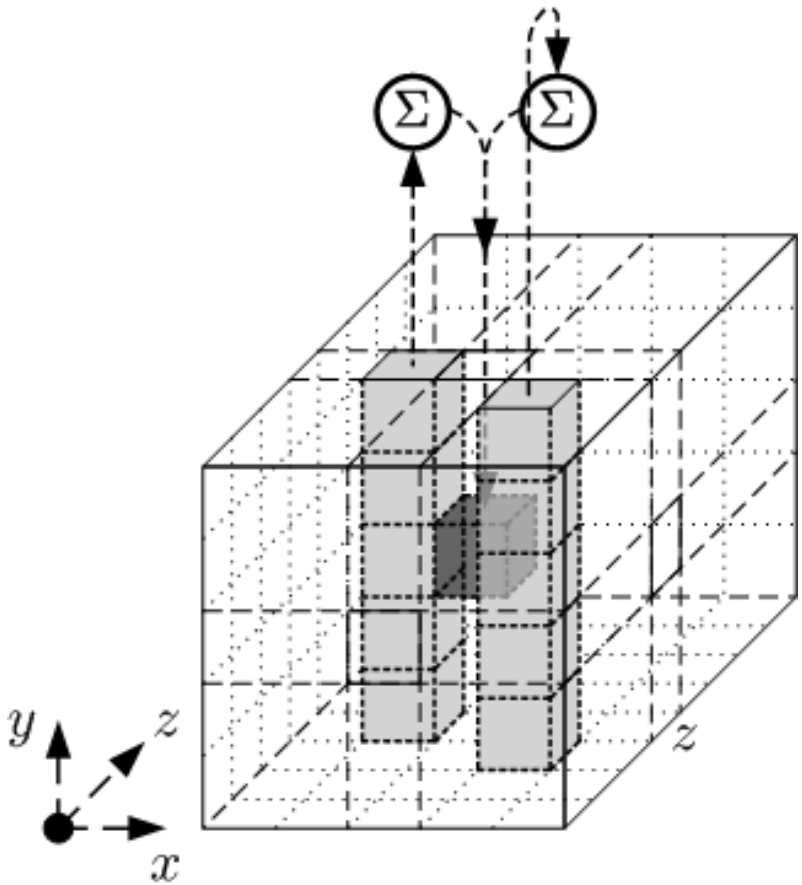
Keccak-f summary

- **Round function:**

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

- **Number of rounds:** $12 + 2\ell$
  - Keccak-f[25] has 12 rounds
  - Keccak-f[1600] has 24 rounds

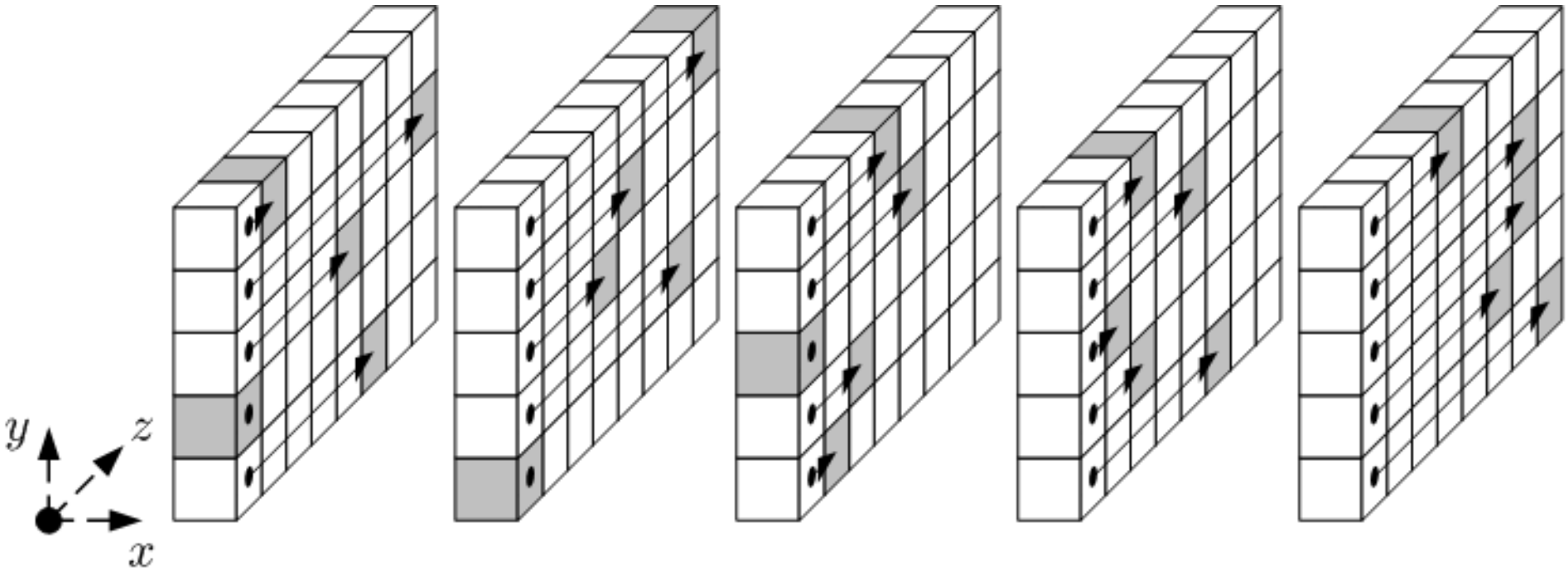
Diffusion of θ



The effect of θ is to XOR each bit in the state with the parities of two columns in the array

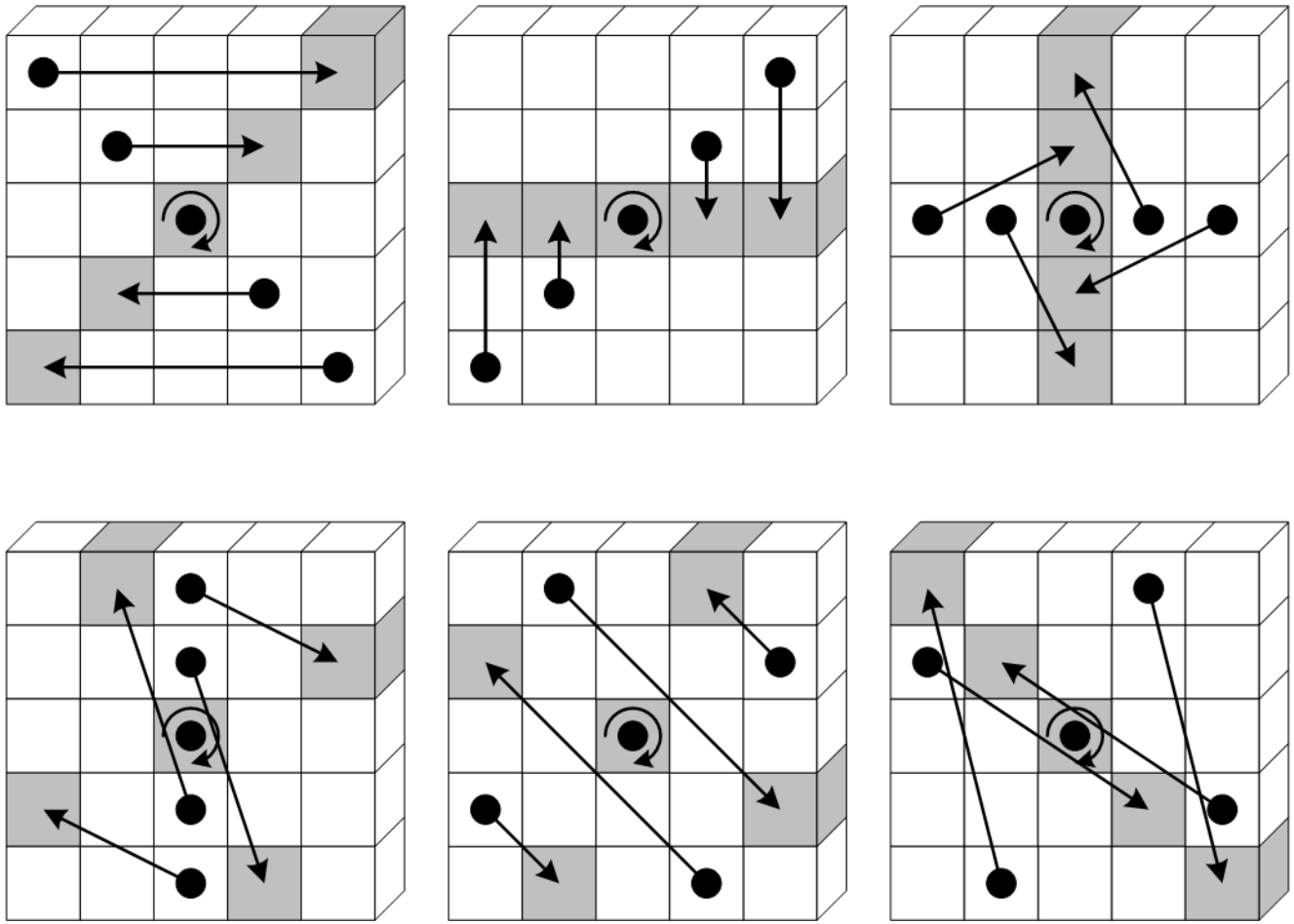
https://keccak.team/figures.html

ρ for inter-slice dispersion



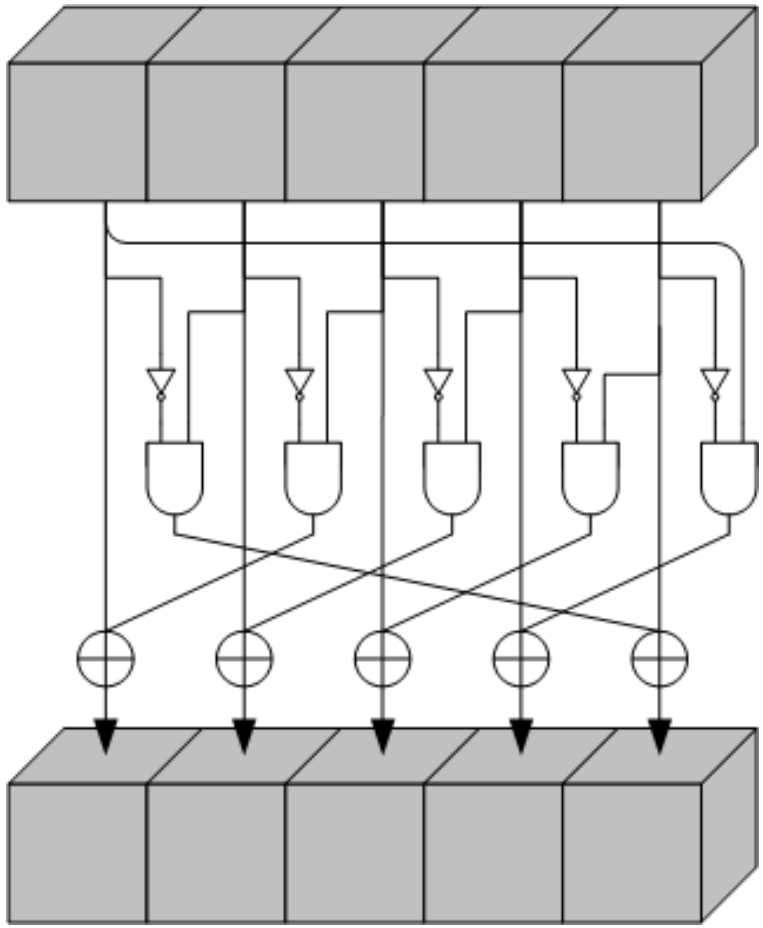
The effect of ρ is to rotate the bits of each lane by a length

π for disturbing horizontal/vertical alignment



The effect of π is to rearrange the positions of the lanes

χ – the nonlinear mapping in Keccak-f



The effect of χ is to XOR each bit with a non-linear function of two other bits in its row

ι to break symmetry

- XOR of round-dependent constant to lane in origin
- Without ι, the round mapping would be symmetric
- Without ι, all rounds would be the same
- Without ι, we get simple fixed points
- The effect of ι is to modify some of the bits of $Lane(0,0)$ in a manner that depends on the round index. The other 24 lanes are not affected by ι.

Outline

Applications of Hash Functions

Applications of Hash Functions

- Truncated Message Digest
- Digital Signatures
- Message Authentication Codes (MAC)

Applications of Hash Functions

- Truncated Message Digest
- Digital Signatures
- Message Authentication Codes (MAC)
- Key Derivation Functions (KDF)
- Pseudo-Random Bit Generation (PRBG)

Applications of Hash Functions

- Truncated Message Digest
- Digital Signatures
- Message Authentication Codes (MAC)
- Key Derivation Functions (KDF)
- Pseudo-Random Bit Generation (PRBG)


Quynh Dang,
Recommendation for Applications Using Approved Hash Algorithms, NIST SP 800-107, 2012.

SHA-3 Derived Functions

NIST recommended four types of SHA-3 derived functions which are mentioned as follows:

- **cSHAKE:** customizable variant of SHAKE function
- **KMAC:** Keccak Message Authentication Code
- **TupleHash:** a variable-length hash function designed to hash tuples of input strings without trivial collisions
- **ParallelHash:** a variable-length hash function that can hash very long messages in parallel

SHA-3 Derived Functions

NIST recommended four types of SHA-3 derived functions which are mentioned as follows:

- **cSHAKE:** customizable variant of SHAKE function
- **KMAC:** Keccak Message Authentication Code
- **TupleHash:** a variable-length hash function designed to hash tuples of input strings without trivial collisions
- **ParallelHash:** a variable-length hash function that can hash very long messages in parallel

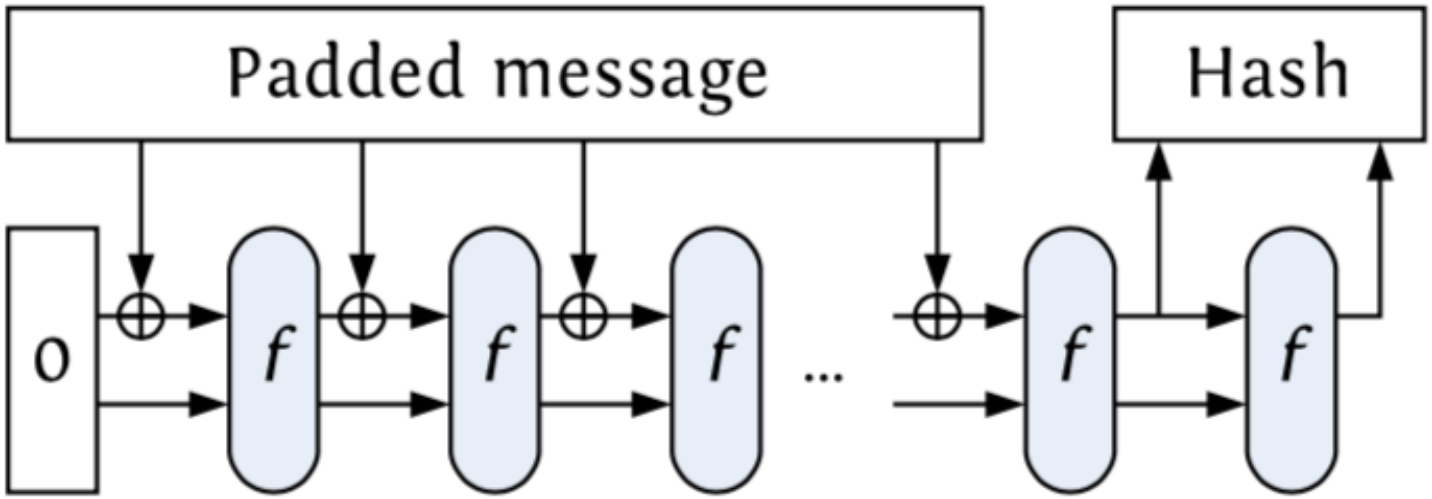
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf

Applications of Sponge Function

- Regular hashing
- Salted hashing
- Mask generation function
- Message authentication codes
- Stream cipher
- Single pass authenticated encryption

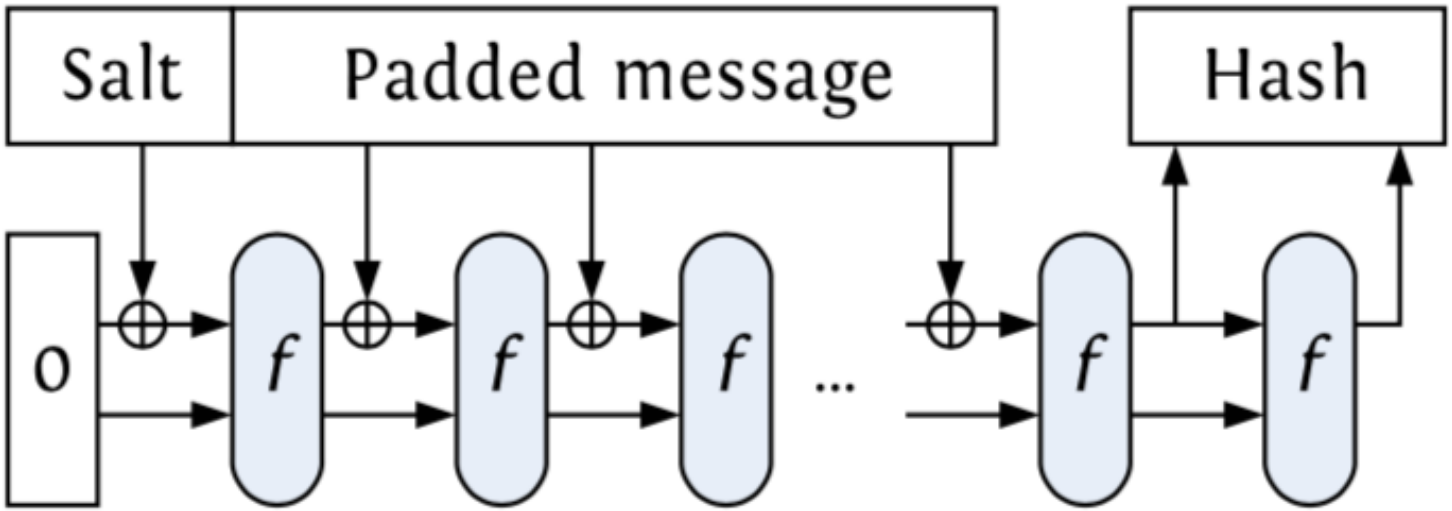
Applications of Sponge Function

Regular hashing



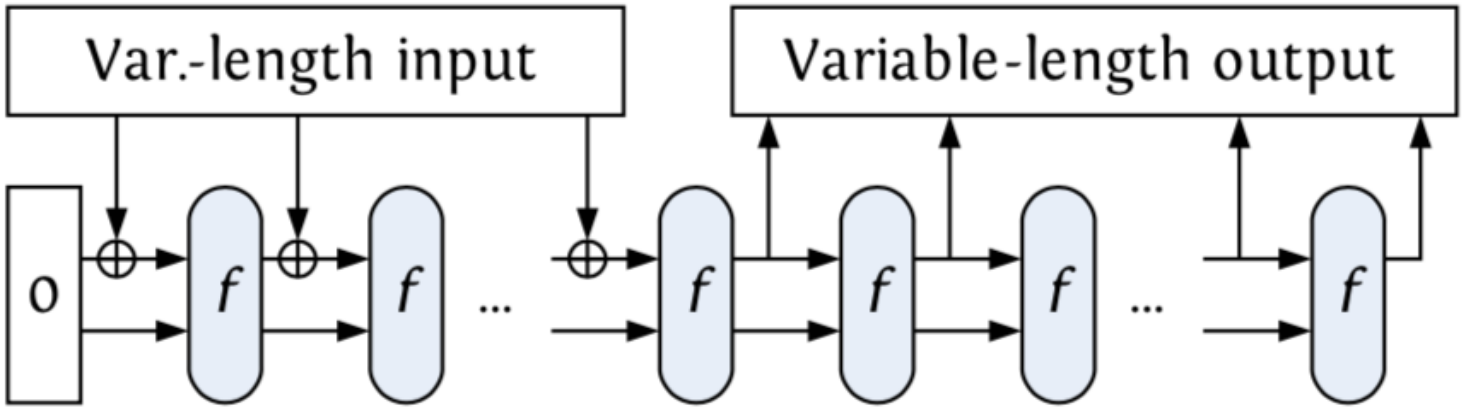
Applications of Sponge Function

Salted hashing



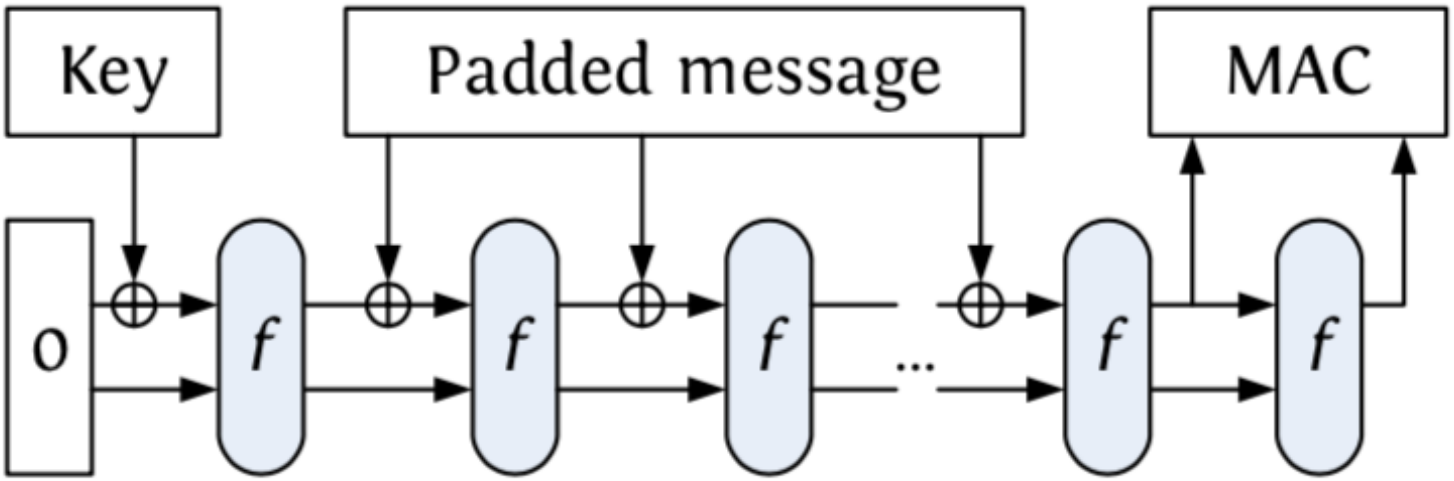
Applications of Sponge Function

Mask generation function



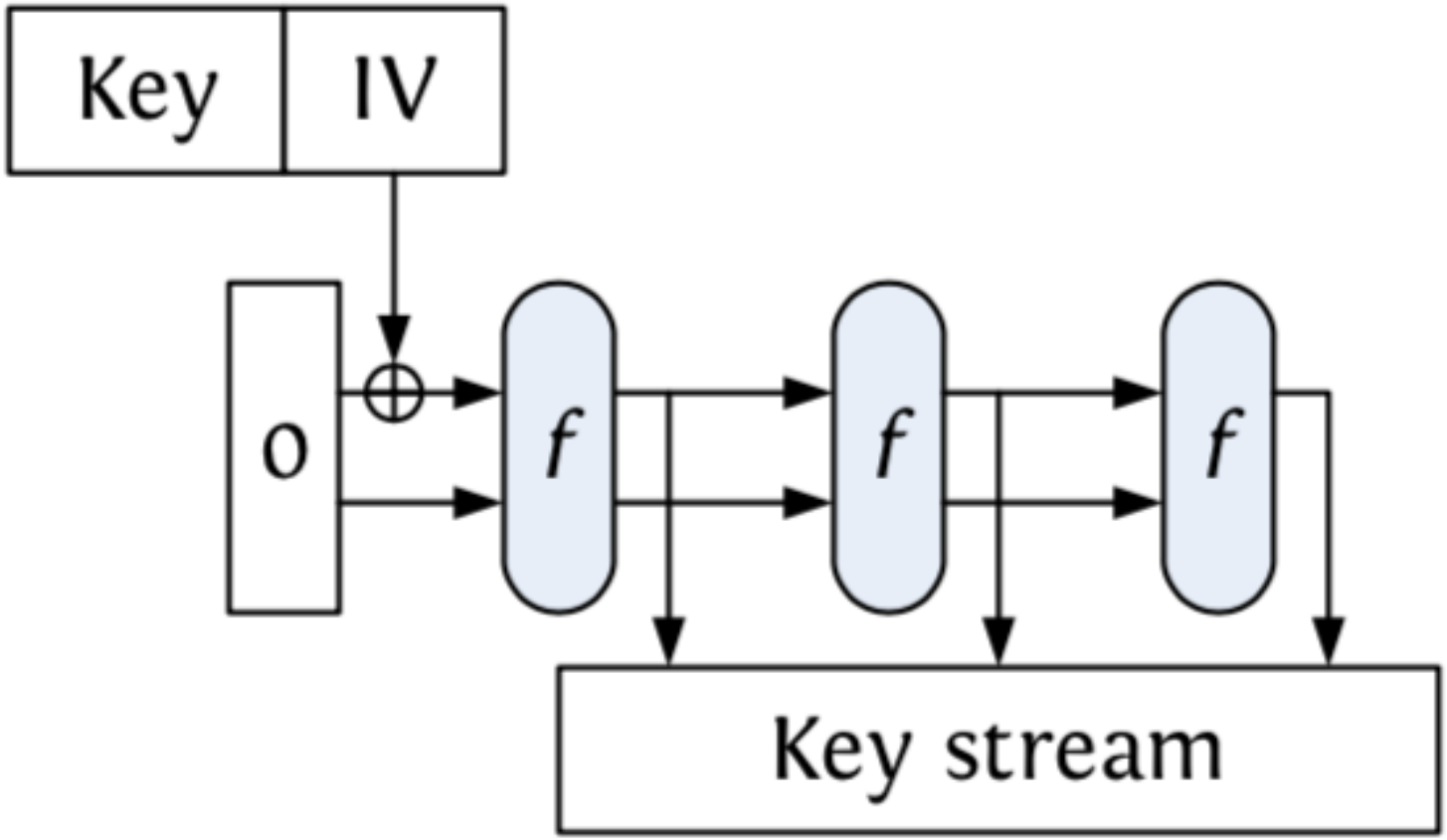
Applications of Sponge Function

MAC



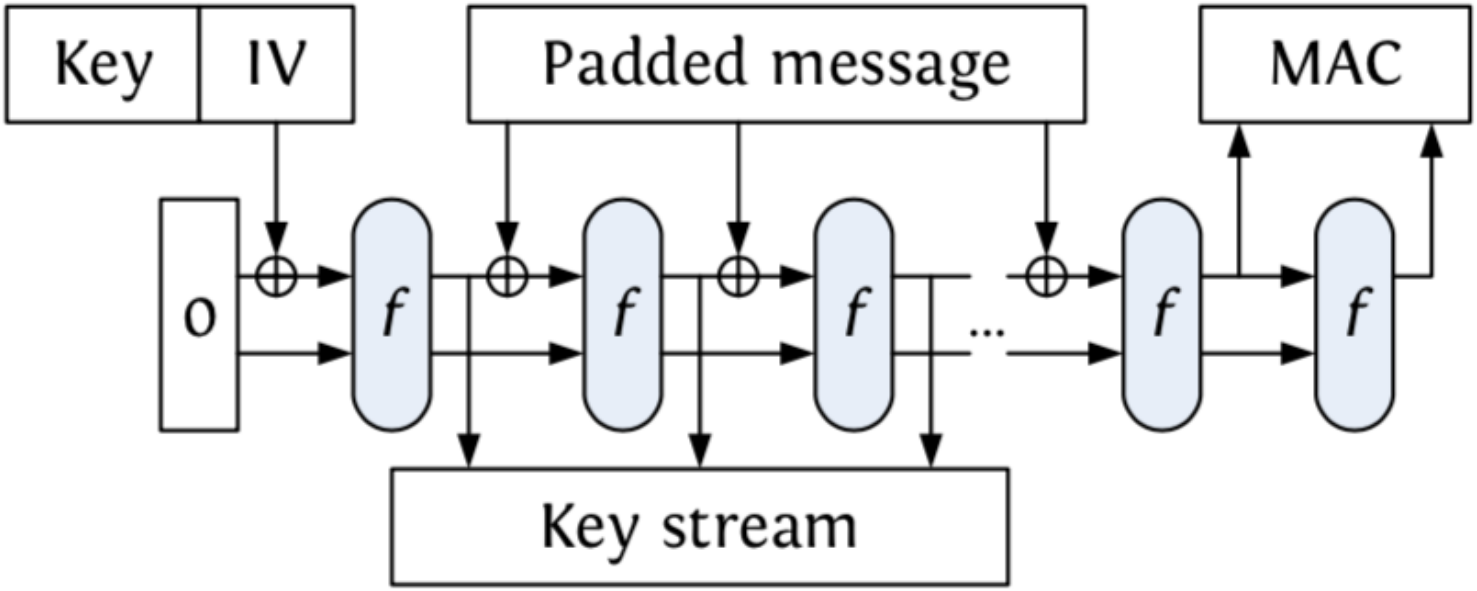
Applications of Sponge Function

Stream cipher



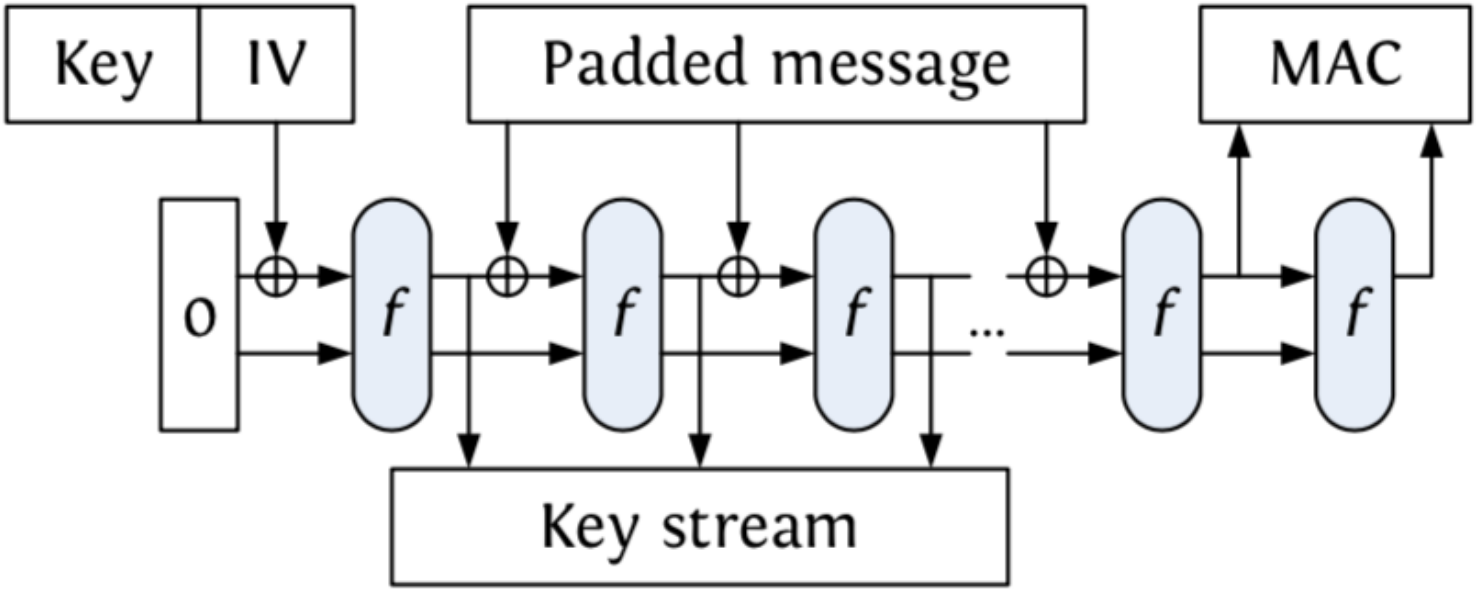
Applications of Sponge Function

Single pass authenticated encryption



Applications of Sponge Function

Single pass authenticated encryption



All the pictures related to Applications are taken from the presentation slide of Keccak Team

References


E. Fleischmann, C. Forler & M. Gorski,
Classification of the SHA-3 Candidates. Available online at
http://eprint.iacr.org/2008/511


A. Joux,
Algorithmic Cryptanalysis, CRC Press, 2009.


K. Matusiewicz,
Analysis of Modern Dedicated Cryptographic Hash Functions, Ph. D. Thesis, 2007.

References


M. Nandi et. al.,
Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition, NISTIR 7620, NIST Report, 2009.


B. Preneel,
Analysis and Design of Cryptographic Hash Functions, PhD thesis, 1993.


B. Rompay,
Analysis and Design of Cryptographic Hash Functions, MAC Algorithms and Block Ciphers, PhD Thesis, 2004.


D R Stinson & M B Paterson,
Cryptography – Theory and Practice, Fourth Edition, CRC Press, 2019.

The End

Thanks a lot for your attention

and

QUESTIONS Please!