

Bahan Kuliah IF4020 Kriptografi

Beberapa Fungsi Hash

Oleh: Rinaldi M

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika(STEI)
ITB**



1. Secure Hash Algorithm (SHA)

Secure Hash Algorithm (SHA)

- *SHA* adalah fungsi *hash* satu-arah yang yang digunakan satu paket dengan *DSS* (*Digital Signature Standard*).
- NIST menetapkan *SHA* sebagai sebagai standard fungsi *hash* satu-arah.
- *SHA* dikembangkan dari *MD4* yang dibuat oleh Ronald L. Rivest dari *MIT*.
- *SHA* menerima masukan berukuran maksimum 2^{64} bit (2.147.483.648 *gigabyte*) dan menghasilkan *message digest* yang panjangnya 160 bit.
- *Message digest* dari *SHA* lebih panjang dari *message digest* yang dihasilkan oleh *MD5* (128 bit).

- Sebutan *SHA* mengacu pada keluarga fungsi *hash* satu-arah SHA.
- Enam varian *SHA*:
 - SHA-0
 - SHA-1
 - SHA-224
 - SHA-256
 - SHA-384
 - SHA-512
- SHA-0 sering diacu sebagai *SHA* saja
- Yang akan dibahas: SHA-1
- Detil algoritma SHA-1 dapat dibaca di dalam dokumen RFC 3174 (<http://www.faqs.org/rfcs/rfc3174.html>)

		Output size (bits)	Internal state size (bits)	Block size (bits)	Max message size (bits)	Word size (bits)	Rounds	Operations	<u>Collision</u> s found?
SHA-0									Yes
SHA-1		160	160	512	$2^{64} - 1$	32	80	+,and,or, xor,rot	Yes
SHA-2	<i>SHA-256/224</i>	256/224	256	512	$2^{64} - 1$	32	64	+,and,or, xor,shr,rot	No
	<i>SHA-512/384</i>	512/384	512	1024	$2^{128} - 1$	64	80		

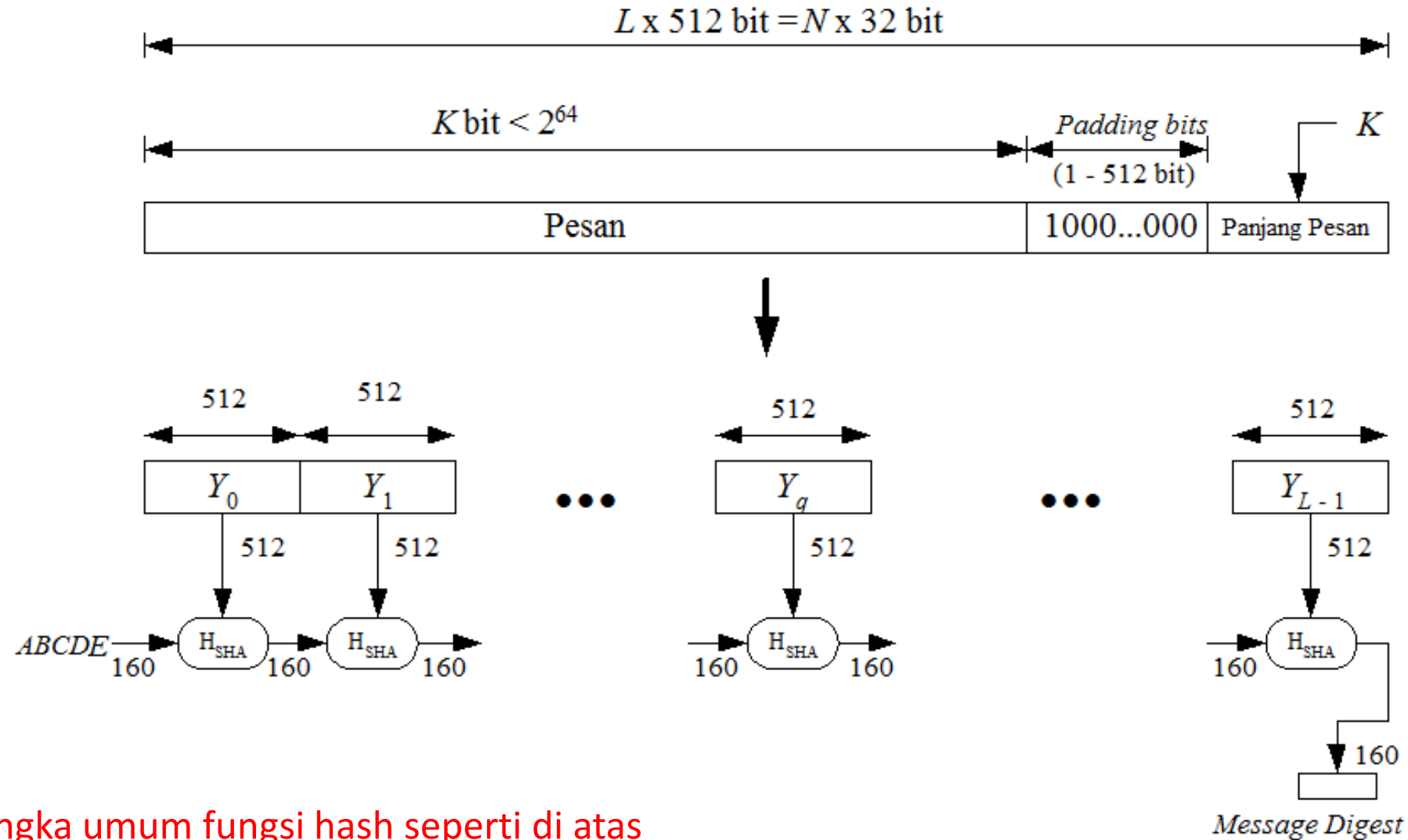
Table 11.3 Comparison of SHA Parameters

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

Note: All sizes are measured in bits.

https://www.brainkart.com/article/Secure-Hash-Algorithm-%28SHA%29_8450/

Gambaran Umum *SHA-1*

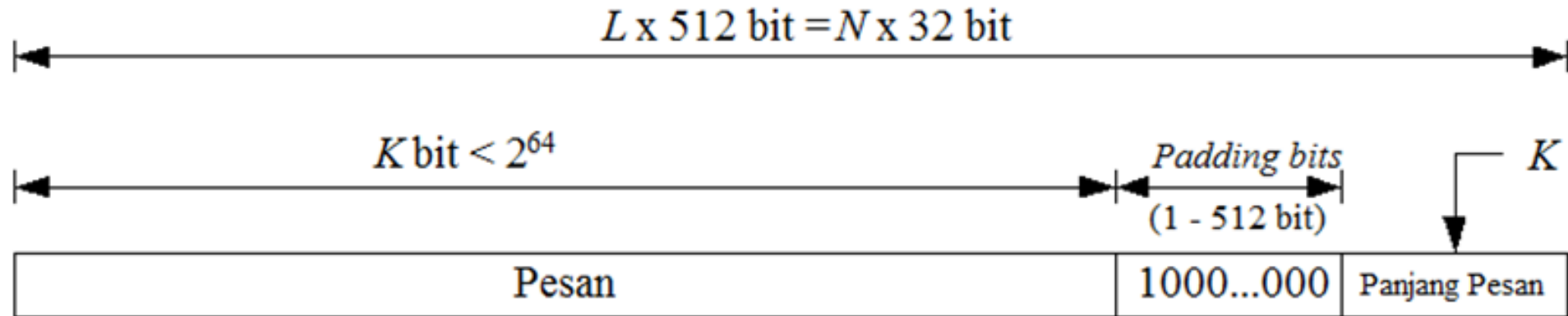


Kerangka umum fungsi hash seperti di atas dinamakan **Merkle–Damgård construction**

Langkah-langkah pembuatan *message digest* dengan SHA-1

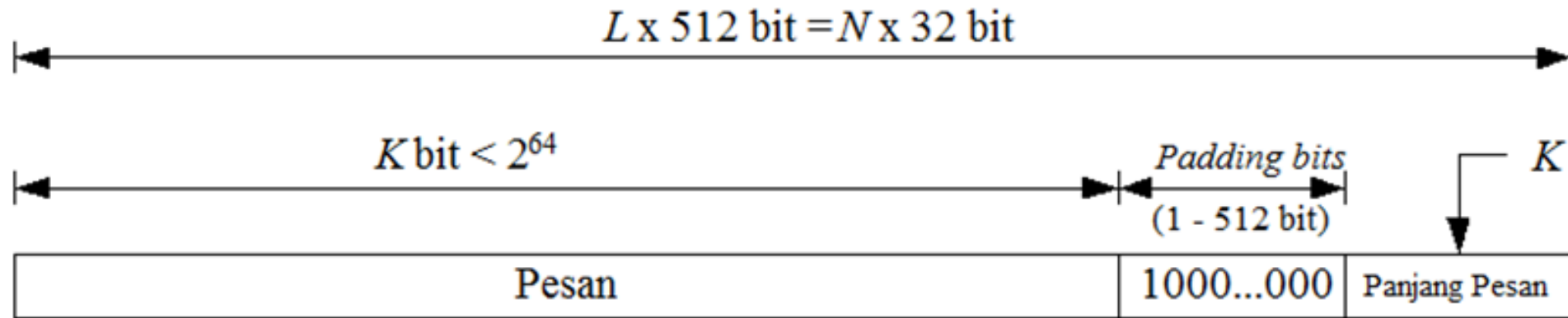
- Secara umum, arsitektur SHA-1 mirip dengan MD5, hanya berbeda dalam jumlah variabel *buffer* (5 buah) dan komputasi di dalam pemrosesan blok pesan 512-bit.
- Langkah-Langkah di dalam SHA-1 adalah sbb:
 1. Penambahan bit-bit pengganjal (*padding bits*).
 2. Penambahan ukuran panjang pesan semula.
 3. Inisialisasi variabel penyangga (*buffer*) MD.
 4. Pengolahan pesan dalam blok-blok berukuran 512 bit.

1. Penambahan Bit-bit Pengganjal



- Pes an ditambah dengan sejumlah bit pengganjal (*padding bits*) sedemikian sehingga panjang pes an (dalam satuan bit) kongruen dengan 448 (mod 512).
- Panjang bit-bit pengganjal adalah antara 1 sampai 512.
- Bit-bit pengganjal terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.
- Contoh: $K = 32000 \text{ bit} \rightarrow 32000 + 192 \text{ bit} = 32192 \rightarrow 32192 \bmod 512 = 448$

2. Penambahan Ukuran Panjang Pesan Semula



- Pesan yang telah diberi bit-bit pengganjal selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula.

Contoh: $K = 32000 = 1111101000000000_2 = 000...1111101000000000$

- Setelah ditambah dengan 64 bit, panjang pesan sekarang menjadi kelipatan 512 bit.

3. Inisialisasi Variabel Penyangga MD

- *SHA* menggunakan 5 buah variable penyangga (*buffer*) yang masing-masing panjangnya 32 bit. Kelima variable itu disebut variabel MD
- Total panjang penyangga adalah $5 \times 32 = 160$ bit.
- Kelima variabel *MD* ini diberi nama *A*, *B*, *C*, *D*, dan *E*. Setiap penyangga diinisialisasi dengan nilai-nilai (dalam notasi Hex) sebagai berikut:

A = 67452301

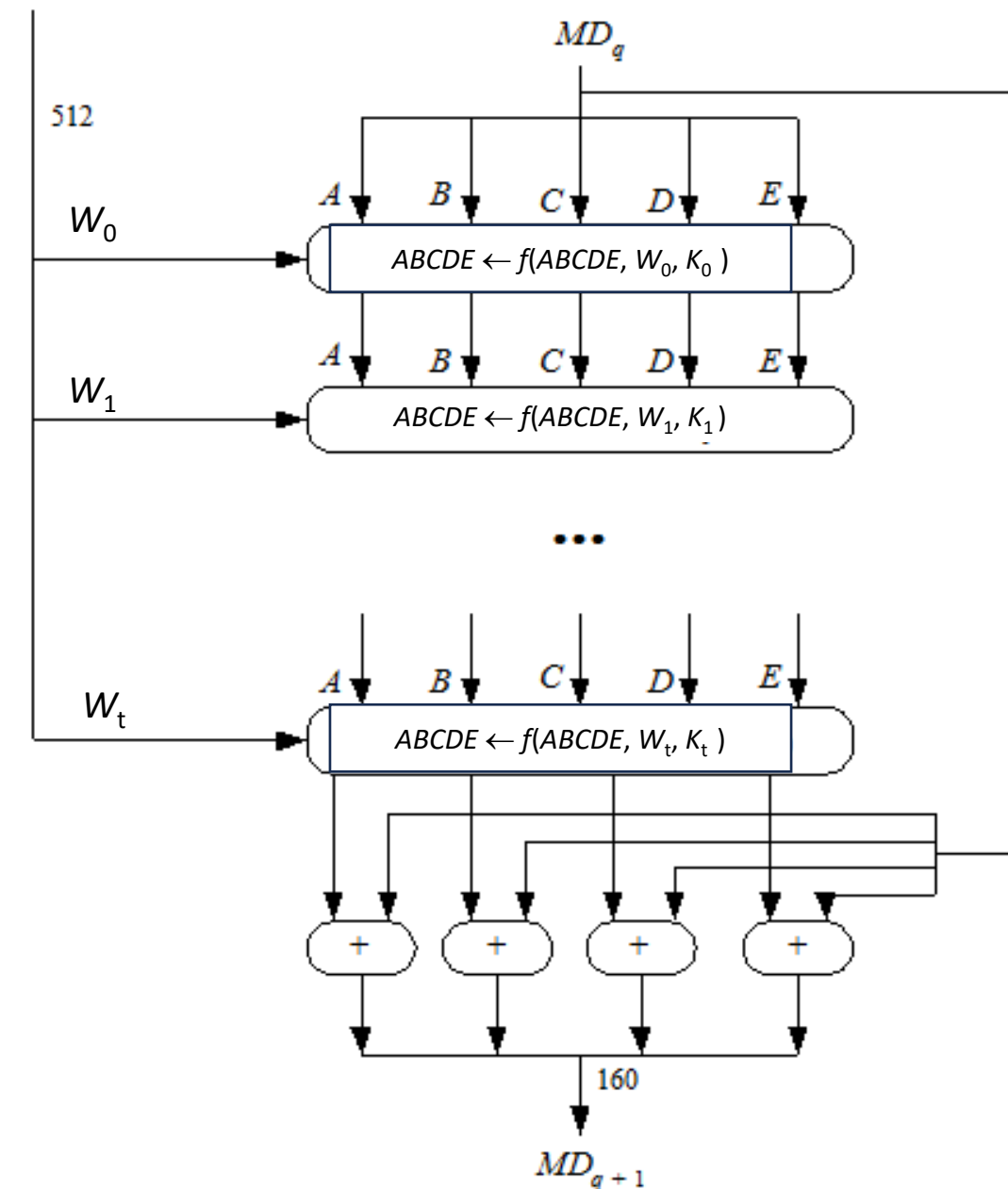
B = EFCDAB89

C = 98BADCFE

D = 10325476

E = C3D2E1F0

4. Pemrosesan Pesan dalam Blok-blok Berukuran 512 bit.



- Proses H_{SHA} terdiri dari 80 buah putaran
- Masing-masing putaran menggunakan bilangan penambah K_t , yaitu:

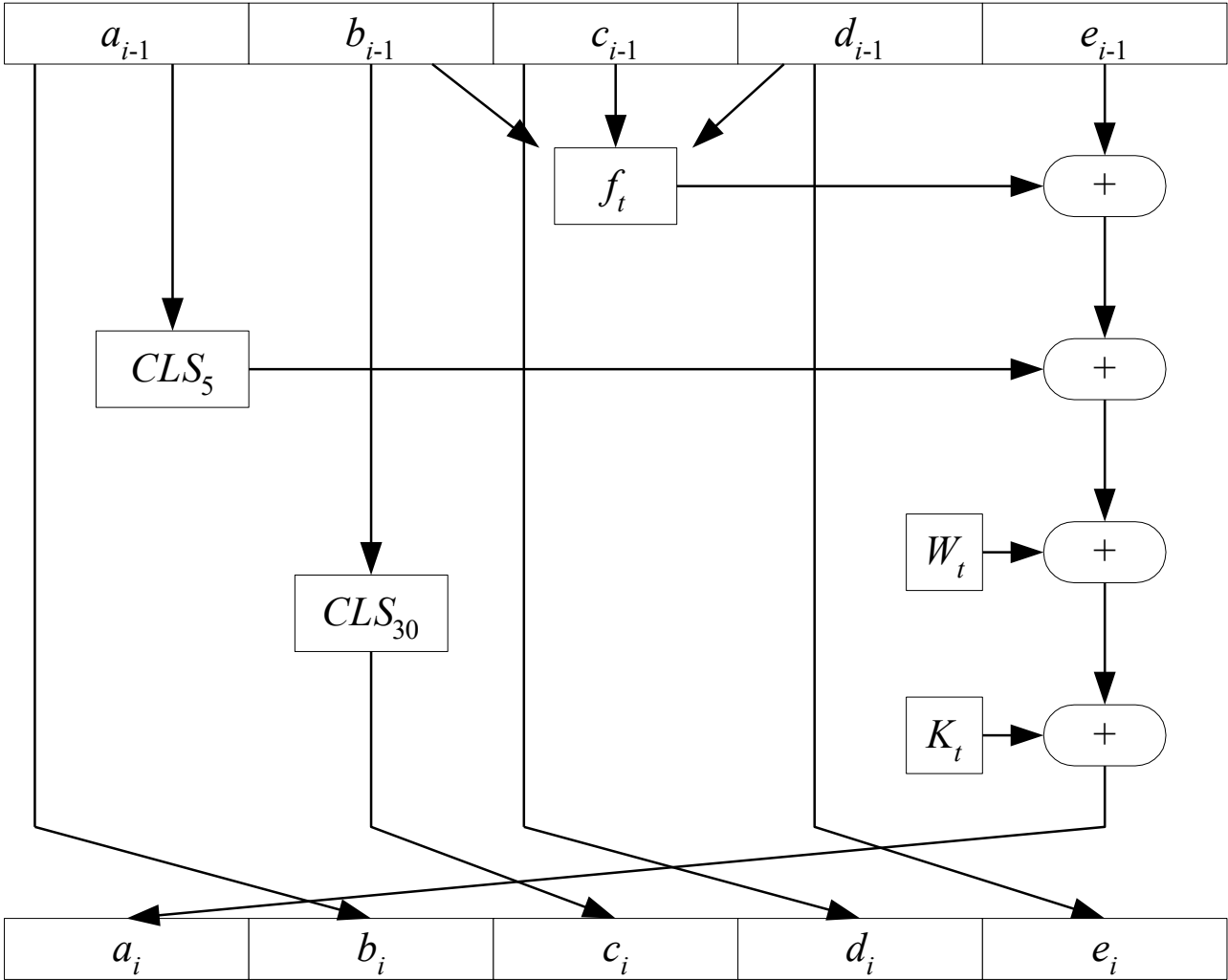
Putaran $0 \leq t \leq 19$ $K_t = 5A827999$

Putaran $20 \leq t \leq 39$ $K_t = 6ED9EBA1$

Putaran $40 \leq t \leq 59$ $K_t = 8F1BBCDC$

Putaran $60 \leq t \leq 79$ $K_t = CA62C1D6$

Operasi dasar pada setiap putaran:



Tabel 1. Fungsi logika f_i pada setiap putaran

Putaran	$f_i(b, c, d)$
0 .. 19	$(b \wedge c) \vee (\sim b \wedge d)$
20 .. 39	$b \oplus c \oplus d$
40 .. 59	$(b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
60 .. 79	$b \oplus c \oplus d$

$$\begin{aligned} e &\leftarrow d \\ d &\leftarrow c \\ c &\leftarrow CLS_{30}(b) \\ b &\leftarrow a \\ a &\leftarrow (CLS_5(a) + f_t(b, c, d) + e + W_t + K_t) \end{aligned}$$

Nilai W_1 sampai W_{16} berasal dari 16 word pada blok yang sedang diproses (1 word = 32 bit), sedangkan nilai W_t berikutnya didapatkan dari persamaan

$$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$$

Demo SHA1 online: <https://emn178.github.io/online-tools/sha1.html>

🔄

🔒 <https://emn178.github.io/online-tools/sha1.html>

☆

📧

⬇

Online Tools Hash Encoding Misc Contact

SHA1

This SHA1 online tool helps you calculate hash from string or binary. You can input UTF-8, UTF-16, Hex to SHA1. It also supports HMAC.

Input Type UTF-8 ▼

Libur lebaran sudah usai, yuk kuliah lagi gaesss...

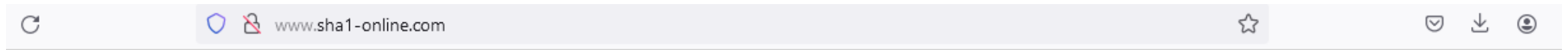
☐ Remember Input
☐ Enable HMAC

Hash ☒ Auto Update

f0ff7687efc93dc9e49ee36ccbec8e8b3e7893a3

SHA1
SHA1
SHA1 File

Demo SHA1 online lainnya: <http://www.sha1-online.com/>



[Home Page](#) | [SHA1 in JAVA](#) | [Secure password generator](#) | [Linux](#) | [Privacy Policy](#)

SHA1 and other hash functions online generator

Balik kampung hari raya ini kah? hash

gost ▼

Result for

gost: 9787c5e68b989d10a418e565192a4fcae7f4f1e5e7aa73126a8e4987aa2e09c4

[SHA-1](#) [MD5](#) on Wikipedia

[We love SPAIN](#) and [oldpics.org](#)

Contoh *message digest* beberapa pesan yang dihasilkan oleh SHA-1:

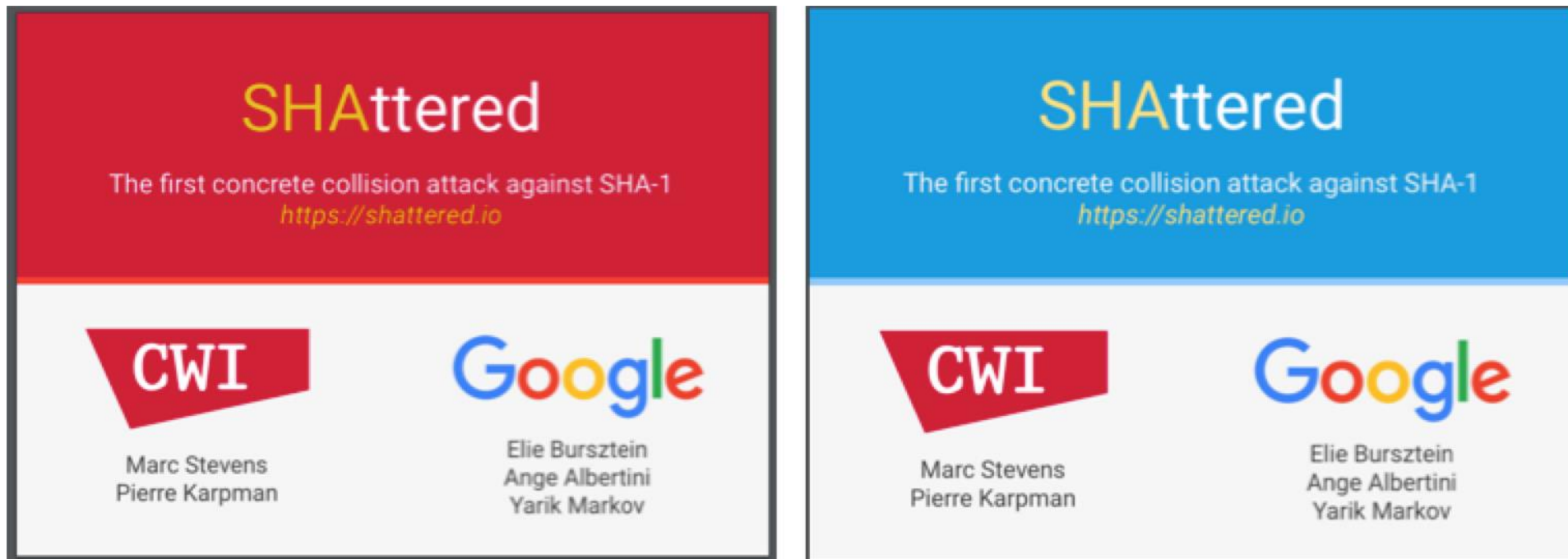
sha1("halo")= 33b1eac210971fb02a3b90afce9dbff758be794d

sha1("halo, apa kabar teman? ") =
03d17abd2962dbed1037d1230f012037cd25fe91

sha1("The quick brown fox jumps over the lazy dog") =
2fd4e1c67a2d28fced849ee1bb76e7391b93eb12

Kriptanalisis *SHA-1*

- Pada tanggal 23 February 2017, kolisi pertama SHA1 diumumkan.
- Pranala: <https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html>
<https://lockandcode.com/miscellaneous/sha1-collision-df-implications>



shattered-1.pdf dan shattered-2.pdf memiliki nilai hash SHA-1 yang sama

SHA1 File Checksum

SHA1 online hash file checksum function

shattered-1.pdf

Hash ☒ Auto Update

38762cf7f55934b34d179ae6a4c80cadccbb7f0a

SHA1 File Checksum

SHA1 online hash file checksum function

shattered-2.pdf

Hash ☒ Auto Update

38762cf7f55934b34d179ae6a4c80cadccbb7f0a|

- Rizki Wicaksono, seorang *hacker* alumni Informatika ITB Angkatan 1999 mendemonstrasikan cara membentuk dua file berbeda yang memiliki nilai *hash* SHA-1 sama (silakan baca tulisannya pada pranala:

<http://www.ilmuhacking.com/cryptography/membuat-sha1-collision-file/>



Rizki Wicaksono

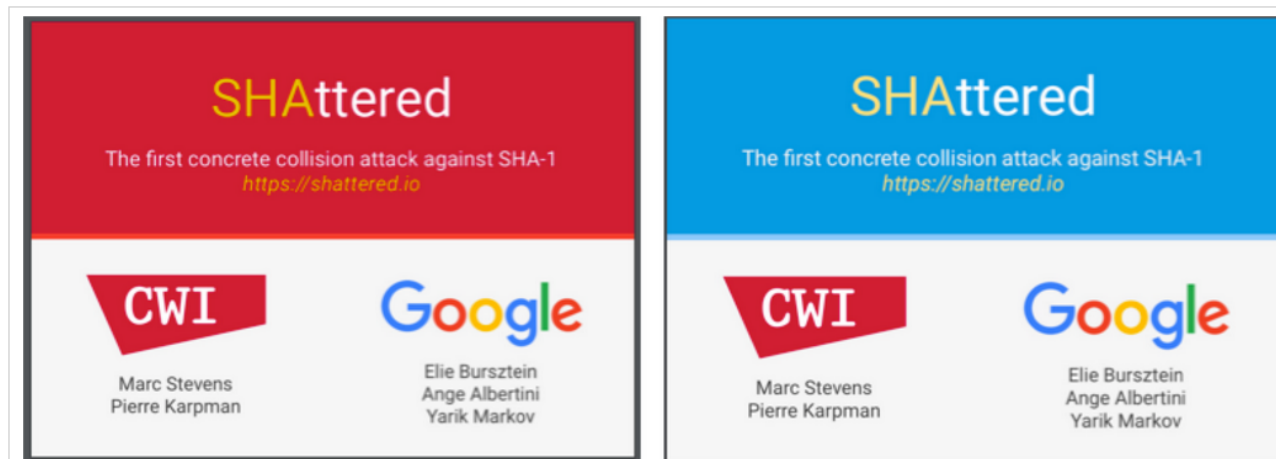
Published

March 21, 2017

in Cryptography

Membuat file dengan SHA1 collision

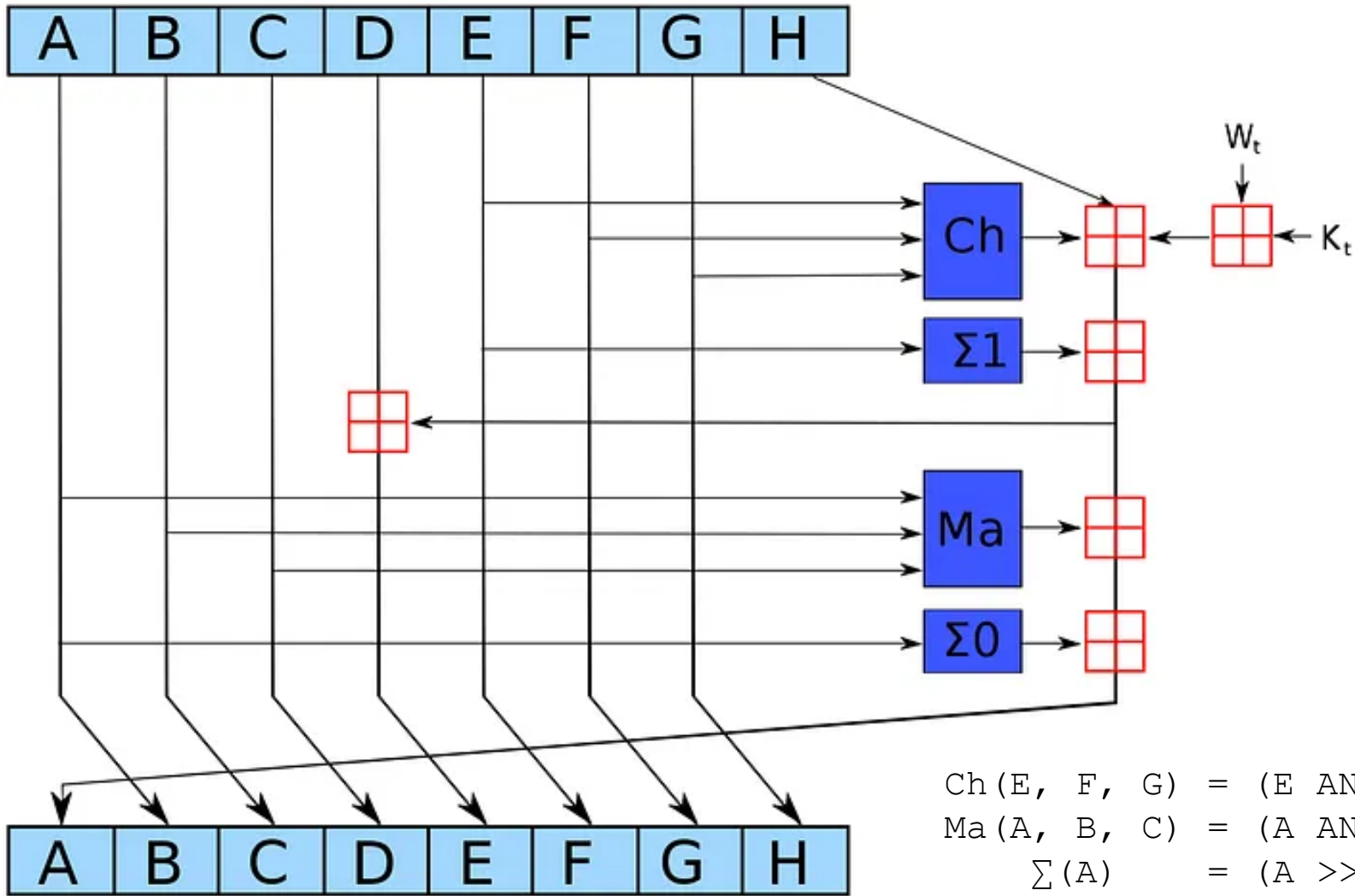
Sejumlah ilmuwan dengan didukung kekuatan komputasi Google berhasil melakukan serangan **SHA1 collision terhadap dokumen PDF**. Mereka menciptakan dua file PDF dengan nilai SHA1 hash yang identik walaupun isi filenya berbeda.



shattered-1.pdf dan shattered-2.pdf memiliki SHA1 yang sama

SHA-256

Aspek	SHA-1	SHA-256
Tipe struktur	Merkle–Damgård construction	Merkle–Damgård construction
Ukuran blok pesan (block size)	512 bit	512 bit
Ukuran word	32 bit	32 bit
Jumlah langkah (rounds)	80 round	64 round
Jumlah register internal (working variables)	5 (A, B, C, D, E)	8 (A, B, C, D, E, F, G, H)
Fungsi boolean yang digunakan	4 fungsi berbeda tergantung range round (AND, OR, XOR, majority)	Fungsi logika tetap: Ch, Maj, Σ_0 , Σ_1
Konstanta round (K)	4 nilai berbeda (tergantung range round)	64 konstanta berbeda (dari akar kubik bilangan prima pertama)
Operasi utama	Rotasi dan penambahan modular 2^{32}	Rotasi, shift, dan penambahan modular 2^{32}



$$\begin{aligned}
 \text{Ch}(E, F, G) &= (E \text{ AND } F) \text{ XOR } ((\text{NOT } E) \text{ AND } G) \\
 \text{Ma}(A, B, C) &= (A \text{ AND } B) \text{ XOR } (A \text{ AND } C) \text{ XOR } (B \text{ AND } C) \\
 \Sigma(A) &= (A \ggg 2) \text{ XOR } (A \ggg 13) \text{ XOR } (A \ggg 22) \\
 \Sigma(E) &= (E \ggg 6) \text{ XOR } (E \ggg 11) \text{ XOR } (E \ggg 25) \\
 &+ \text{addition modulo } 2^{32}
 \end{aligned}$$

- Inisialisasi buffer

```
a = 0x6a09e667
b = 0xbb67ae85
c = 0x3c6ef372
d = 0xa54ff53a
e = 0x510e527f
f = 0x9b05688c
g = 0x1f83d9ab
h = 0x5be0cd19
```

- Nilai K[0..63}

```
k[0..63] :=
  0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
  0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
  0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
  0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
  0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
  0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
  0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
  0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befeffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
```



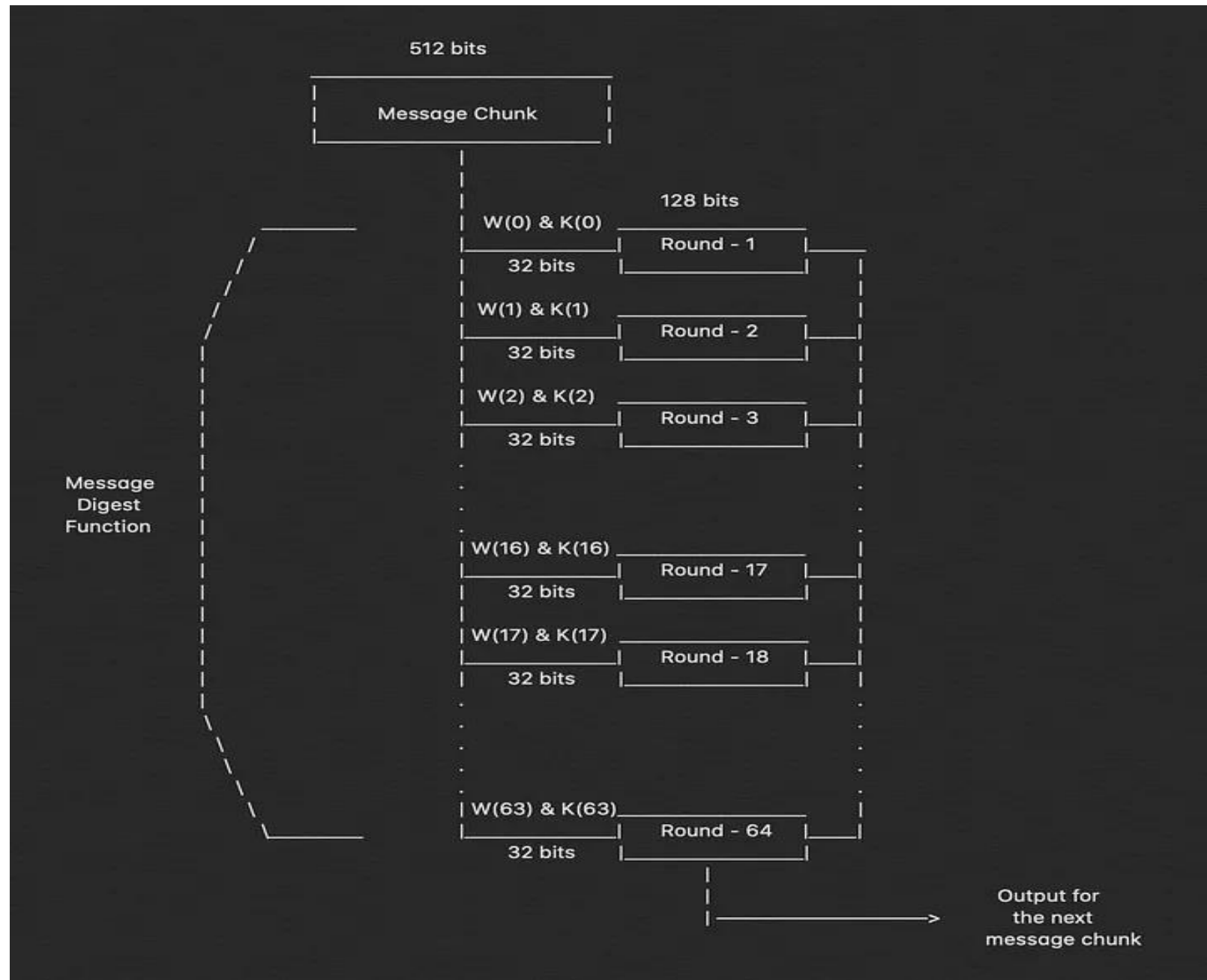
W_j ($16 \leq j \leq 63$). $\sigma_0(x)$ and $\sigma_1(x)$ are computed as eq. (2) and eq. (3). Note that $S^n(x)$ and $R^n(x)$ denote the right rotation and the right shift of data x by n bits, respectively.

$$W_j = \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16} \quad (1)$$

$$\sigma_0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x) \quad (2)$$

$$\sigma_1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x) \quad (3)$$

- 64 putaran



Aspek	SHA-1	SHA-256
Collision resistance	Sudah ditembus (collision ditemukan oleh Google & CWI tahun 2017, "SHAttered")	Belum ada collision yang diketahui
Preimage resistance	Melemah, tidak lagi direkomendasikan	Masih sangat kuat
Status keamanan	Tidak aman, tidak direkomendasikan untuk kriptografi modern	Aman, direkomendasikan oleh NIST
Keluarga algoritma	SHA-1 (keluarga SHA-1)	SHA-2 (keluarga SHA-2, termasuk SHA-224/256/384/512)

SHA-512

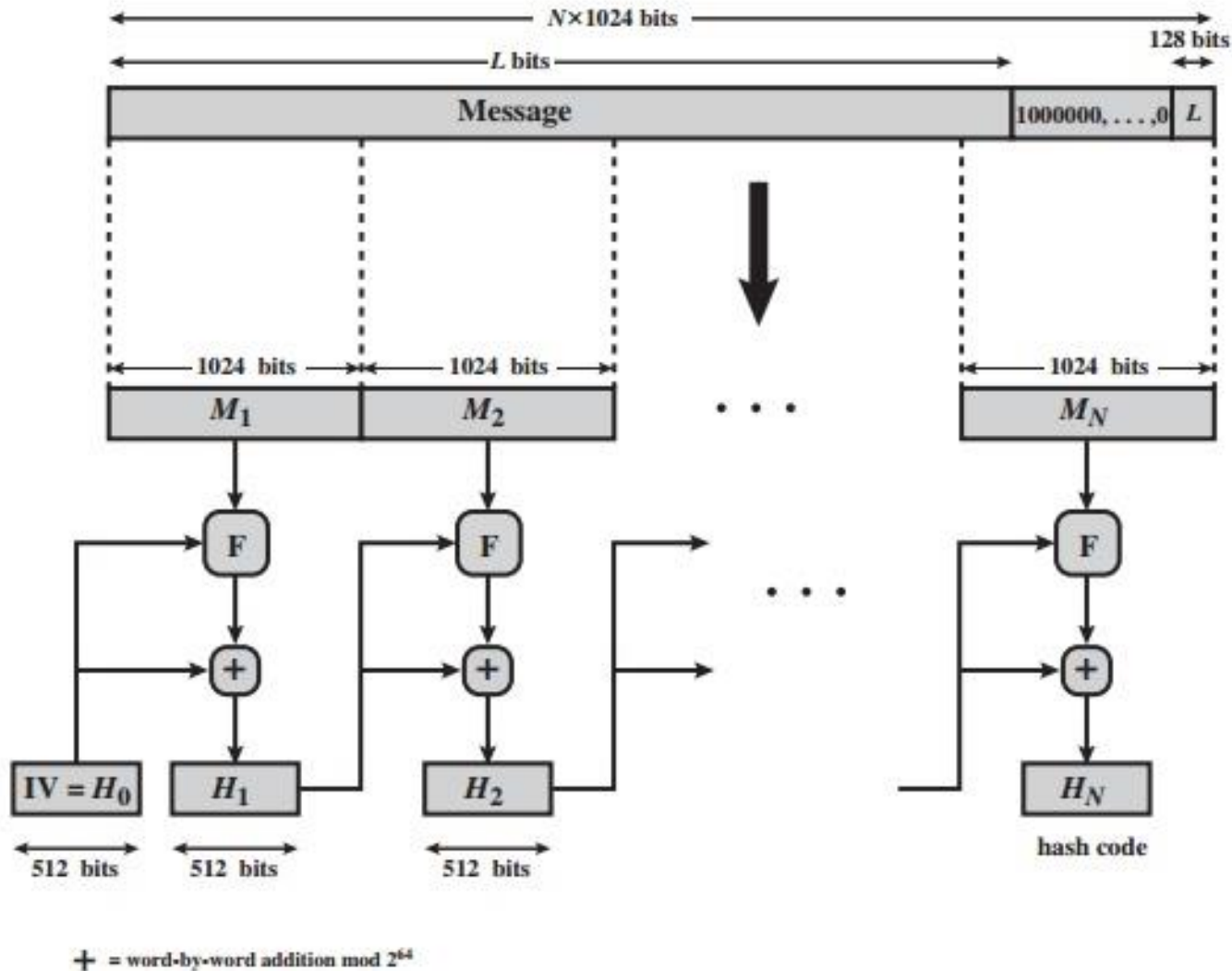


Figure 11.8 Message Digest Generation Using SHA-512

Inisialisasi variable buffer a, b, c, d, e, f, g, h:

H₀

a	6a09e667f3bcc908	f6afceb8bcfcddf5	1320f8c9fb872cc0
b	bb67ae8584caa73b	6a09e667f3bcc908	f6afceb8bcfcddf5
c	3c6ef372fe94f82b	bb67ae8584caa73b	6a09e667f3bcc908
d	a54ff53a5f1d36f1	3c6ef372fe94f82b	bb67ae8584caa73b
e	510e527fade682d1	58cb02347ab51f91	c3d4ebfd48650ffa
f	9b05688c2b3e6c1f	510e527fade682d1	58cb02347ab51f91
g	1f83d9abfb41bd6b	9b05688c2b3e6c1f	510e527fade682d1
h	5be0cd19137e2179	1f83d9abfb41bd6b	9b05688c2b3e6c1f

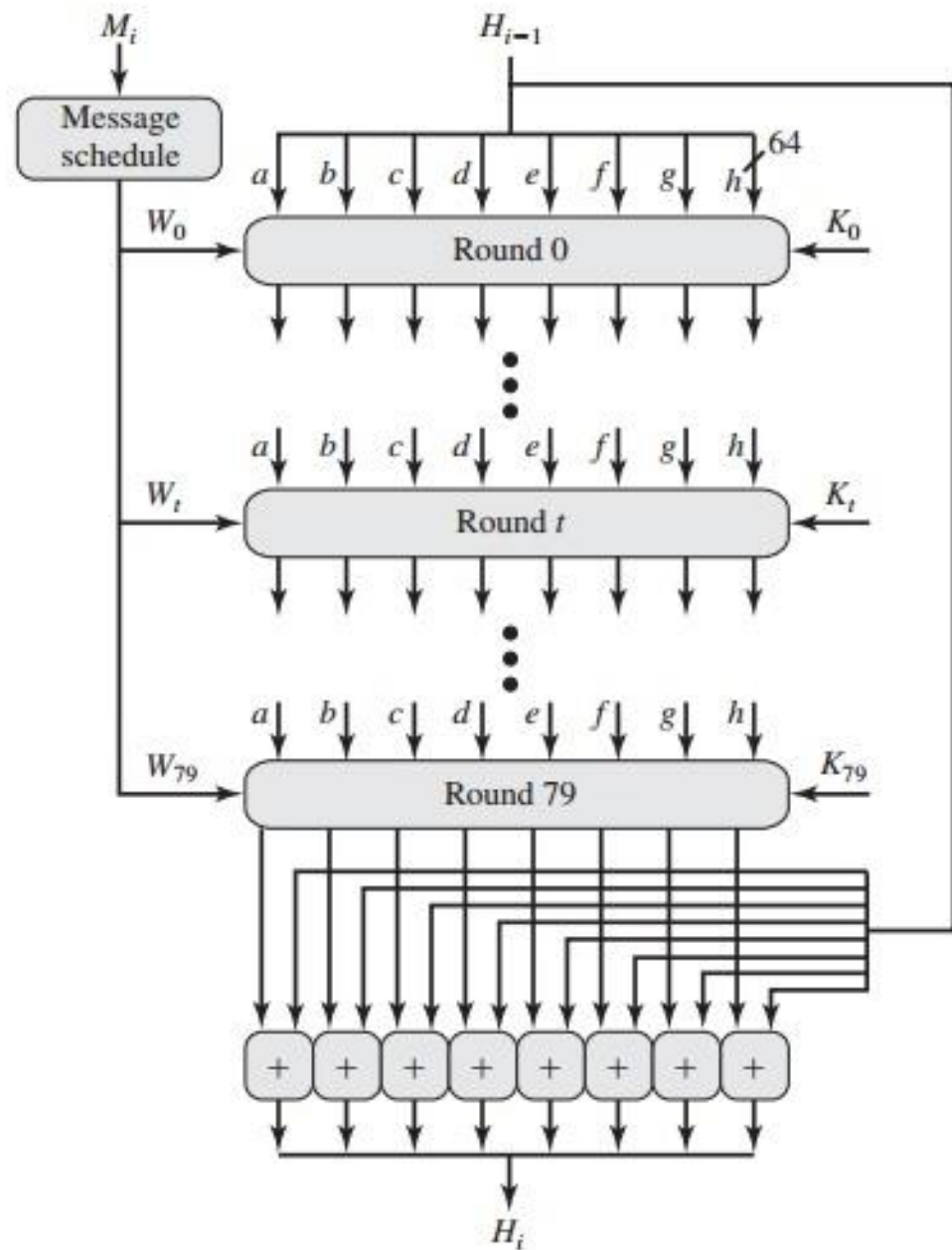


Figure 11.9 SHA-512 Processing of a Single 1024-Bit Block

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

where

$$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$$

$$\sigma_1^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$$

$\text{ROTR}^n(x)$ = circular right shift (rotation) of the 64-bit argument x by n bits

$\text{SHR}^n(x)$ = left shift of the 64-bit argument x by n bits with padding by zeros on the right

$+$ = addition modulo 2^{64}

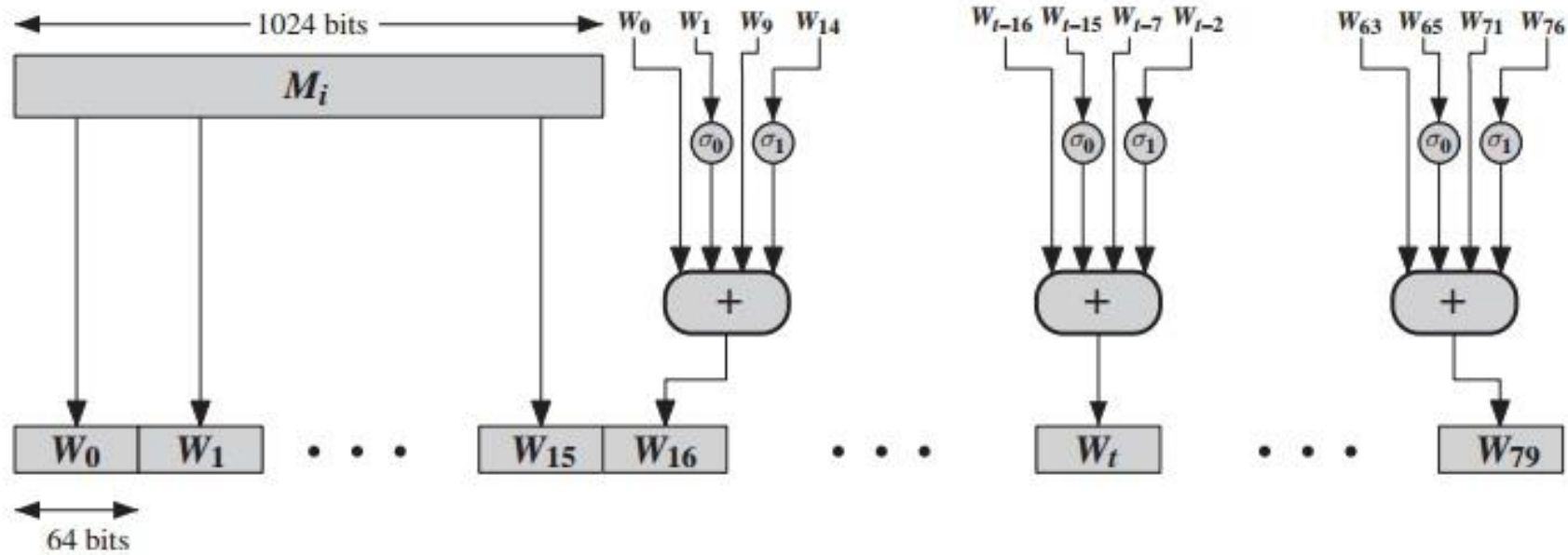


Figure 11.11 Creation of 80-word Input Sequence for SHA-512 Processing of Single Block

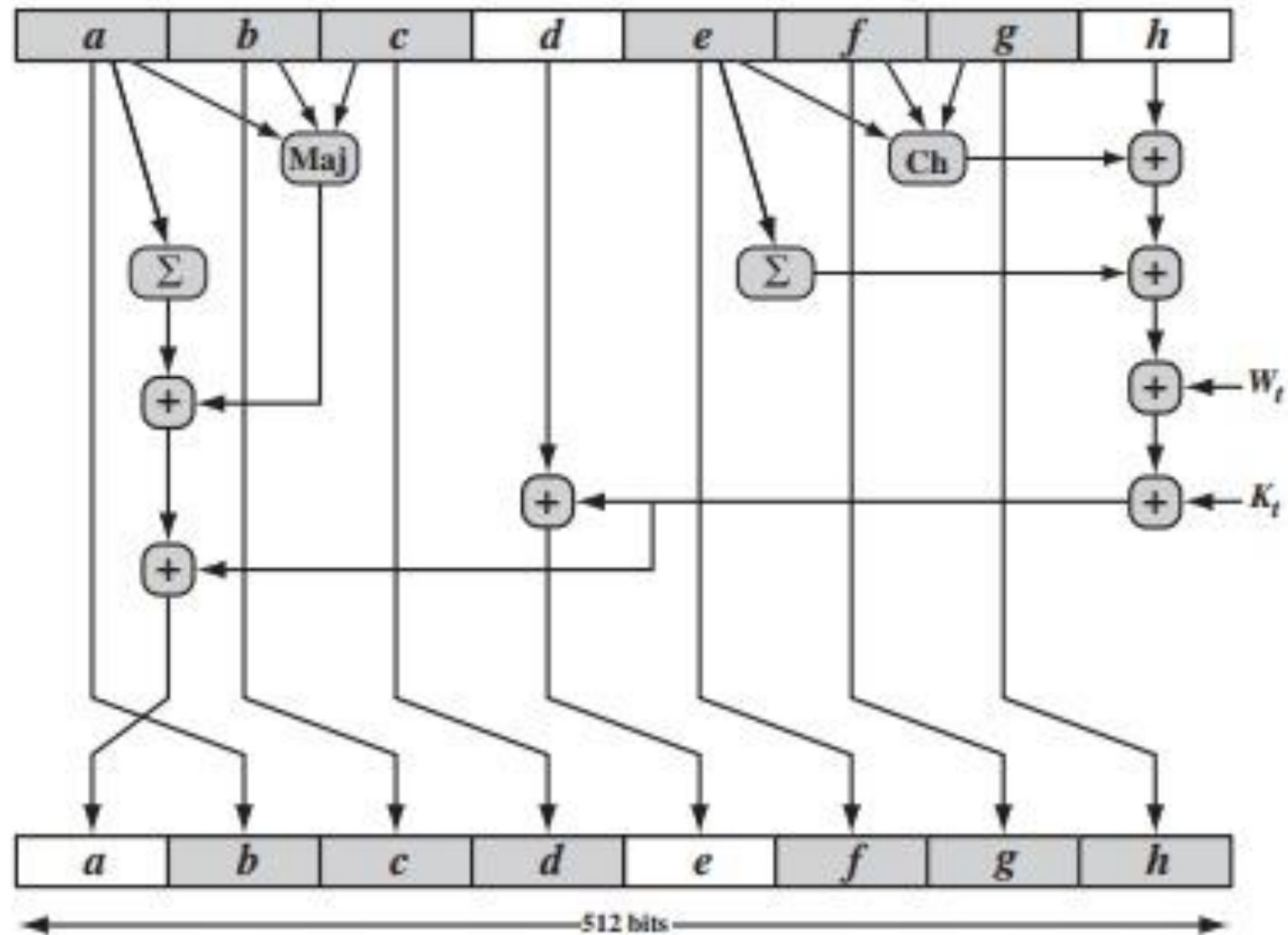


Figure 11.10 Elementary SHA-512 Operation (single round)

$$T_1 = h + \text{Ch}(e, f, g) + \left(\sum_1^{512} e \right) + W_t + K_t$$

$$T_2 = \left(\sum_0^{512} a \right) + \text{Maj}(a, b, c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = T_1 + T_2$$

where

t = step number; $0 \leq t \leq 79$

$\text{Ch}(e, f, g) = (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$

the conditional function: If e then f else g

Table 11.4 SHA-512 Constants

428a2f98d728ae22	7137449123ef65cd	b5c0fbcfec4d3b2f	e9b5dba58189dbbc
3956c25bf348b538	59f111f1b605d019	923f82a4af194f9b	ab1c5ed5da6d8118
d807aa98a3030242	12835b0145706fbe	243185be4ee4b28c	550c7dc3d5ffb4e2
72be5d74f27b896f	80deb1fe3b1696b1	9bdc06a725c71235	c19bf174cf692694
e49b69c19ef14ad2	efbe4786384f25e3	0fc19dc68b8cd5b5	240ca1cc77ac9c65
2de92c6f592b0275	4a7484aa6ea6e483	5cb0a9dcbbd41fbd4	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
90befe5fa23631e28	a4506cebbe82bde9	bef9a3f7b2c67915	c67178f2e372532b
ca273ecee26619c	d186b8c721c0c207	eada7dd6cde0eb1e	f57d4f7fee6ed178
06f067aa72176fba	0a637dc5a2c898a6	113f9804bef90dae	1b710b35131c471b
28db77f523047d84	32caab7b40c72493	3c9ebe0a15c9bebc	431d67c49c100d4c
4cc5d4becb3e42b6	597f299cfc657e2a	5fcb6fab3ad6faec	6c44198c4a475817

The padded message consists blocks $M_1, M_2, \dots, M_N$. Each message block M_i consists of 16 64-bit words $M_{i,0}, M_{i,1}, \dots, M_{i,15}$. All addition is performed modulo 2^{64} .

$$H_{0,0} = 6A09E667F3BCC908$$

$$H_{0,4} = 510E527FADE682D1$$

$$H_{0,1} = BB67AE8584CAA73B$$

$$H_{0,5} = 9B05688C2B3E6C1F$$

$$H_{0,2} = 3C6EF372FE94F82B$$

$$H_{0,6} = 1F83D9ABFB41BD6B$$

$$H_{0,3} = A54FF53A5F1D36F1$$

$$H_{0,7} = 5BE0CDI9137E2179$$

for $i = 1$ **to** N

1. Prepare the message schedule W

for $t = 0$ **to** 15

$$W_t = M_{i,t}$$

for $t = 16$ **to** 79

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

2. Initialize the working variables

$$a = H_{i-1,0} \quad e = H_{i-1,4}$$

$$b = H_{i-1,1} \quad f = H_{i-1,5}$$

$$c = H_{i-1,2} \quad g = H_{i-1,6}$$

$$d = H_{i-1,3} \quad h = H_{i-1,7}$$

3. Perform the main hash computation

for $t = 0$ **to** 79

$$T_1 = h + \text{Ch}(e, f, g) + \left(\sum_1^{512} e \right) + W_t + K_t$$

$$T_2 = \left(\sum_0^{512} a \right) + \text{Maj}(a, b, c)$$

Kode Program Hash dengan Python (1)

Input teks dari dalam kode program:

```
import hashlib

hashMD5 = hashlib.md5(b'Yogyakarta kota kenangan').hexdigest() #Karakter b mengubah string menjadi bytes
hashSHA1 = hashlib.sha1(b'Yogyakarta kota kenangan').hexdigest()
hashSHA256 = hashlib.sha256(b'Yogyakarta kota kenangan').hexdigest()
hashSHA512 = hashlib.sha512(b'Yogyakarta kota kenangan').hexdigest()

print("MD5 :", hashMD5)
print("SHA1 :", hashSHA1)
print("SHA256 :", hashSHA256)
print("SHA512 :", hashSHA512)
```

Output:

```
MD5 : df6ec566eb228237a59e012e343ecb13
SHA1 : aa8adef498e0ffd2222f49a0021d645b9f8cfa4d
SHA256 : c2494851d6345fac381077cc6ae6a283ad8c02ac7a4bbd8a9f57d5fb2145707f
SHA512 :
e3edf0e24a4dc15291073b8b187ae41e21d30e5ec5db5a057a62950630a602c33c027eceb69d6695ccb041e0713c0f2dd27a2f65
3a6bf0b0da55313d3def17f9
```

Kode Program Hash dengan Python (2)

Input teks dari keyboard:

```
import hashlib

message = input('Ketikkan teks: ')

# Konversi string message menjadi representasi byte menggunakan encoding UTF-8.
# fungsi encode('utf-8') mengubah string menjadi bytes, karena algoritma hash
# seperti MD5 dan SHA bekerja dengan data biner, bukan string.
encoded_message = message.encode('utf-8')

hashMD5 = hashlib.md5(encoded_message).hexdigest()
hashSHA1 = hashlib.sha1(encoded_message).hexdigest()
hashSHA256 = hashlib.sha256(encoded_message).hexdigest()
hashSHA512 = hashlib.sha512(encoded_message).hexdigest()

print("MD5 :", hashMD5)
print("SHA1 :", hashSHA1)
print("SHA256 :", hashSHA256)
print("SHA512 :", hashSHA512)
```

Output:

Ketikkan teks: Mereka pergi ke Gunung Tangkubanperahu pukul 9.00

MD5 : 9f672f83fd302dcc7c5fe8cde648df7a

SHA1 : 1f73011345a09b10f8bd5fc0863679f3d0afb0ac

SHA256 : 389384279fda7612c44045fb433ccb8069c642f59049395c3b5a5490fd4ccb4c

SHA512 :

cf2b7075c1c82eb83c5c1cf88d4818c02f6a9e7799bd111960f62fa247a28aa48933203645cda413eac702b8
da64e415acbc11d2e303069d17a2db81e763d1ce

Kode Program Hash dengan Python (3)

Input dari file:

```
def hash_file(filepath):  
    # Menghitung nilai hash pesan berupa file, misalkan dengan SHA1  
    # buat objek sha1  
    hash = hashlib.sha256()  
  
    # buka file dan baca dalam mode biner  
    with open(filepath, 'rb') as file:  
        # loop sampai akhir file  
        chunk = 0  
        while chunk != b'':  
            # baca 1024 byte setiap kali  
            chunk = file.read(1024)  
            hash.update(chunk)  
  
    # return nilai hash dalam kode hexadecimal  
    return hash.hexdigest()  
  
# Program utama  
namafile = input("Nama file beserta path-nya: ")  
digest = hash_file(namafile)  
print("Nilai hash SHA1 : ", digest)
```

Output:

Nama file beserta path-nya: E:/STIN/Kriptografi 2025/07-AES-(2025).pptx

Nilai hash SHA1 : 386e7a95c3b3e73d3fcd4e05e7e05efb7cd926844741e5cf434e66c16efe1fe7

Kode Program Hash dengan Python (4)

Memeriksa integritas (keaslian) pesan, missal menggunakan SHA-256:

```
import hashlib

hashSHA256_1 = hashlib.sha256(b'Yogyakarta kota kenangan').hexdigest()
print("SHA256 (1) :", hashSHA256_1)

#Ubah pesan "kota" jadi "koti"
hashSHA256_2 = hashlib.sha256(b'Yogyakarta koti kenangan').hexdigest()
print("SHA256 (2) :", hashSHA256_2)

if hashSHA256_1 == hashSHA256_2:
    print("Pesan masih utuh, belum dimanipulasi")
else:
    print("Pesan sudah diubah")
```

Output:

```
SHA256 (1) : c2494851d6345fac381077cc6ae6a283ad8c02ac7a4bbd8a9f57d5fb2145707f
SHA256 (2) : f1212f92c60b4950fa78d7783653e487cefaa5c8d7eb8f7faedce647d76420df
Pesan sudah diubah
```


Latar Belakang

- Seperti sejarah AES, *National Institute of Standards and Technology* (NIST) menyelenggarakan kompetisi terbuka untuk mengembangkan fungsi *hash* yang baru, bernama SHA-3
- SHA-3 menjadi komplementer SHA-1 dan SHA-2
- Kompetisi diumumkan pada tahun 2007 dan berakhir pada Oktober 2012 dengan memilih pemenang.

Proses Pemilihan

- Proses pemilihan terdiri dari 2 putaran dan final
- Jumlah *submission* adalah 64 rancangan fungsi *hash*.
- Putaran pertama (penyisihan): dipilih 51 *submission*
- Putaran kedua (semi final): dipilih 14 *submission*
- Babak final: 5 finalis

Finalis

1. BLAKE
2. Grøstl
3. JH
4. Keccak
5. Skein

BLAKE

- Designers: Jean-Philippe Aumasson, Luca Henzen, Willi Meier, Raphael C.-W. Phan
- Detail: Digest sizes 224, 256, 384 or 512 bits
- Rounds 14 or 16

Grøstl

- Designers: Praveen Gauravaram, Lars Knudsen, [Krystian Matusiewicz](#), [Florian Mendel](#), [Christian Rechberger](#), [Martin Schläffer](#), and Søren S. Thomsen
- Detail: Digest sizes 256 and 512

JH

- Designers: [Hongjun Wu](#)
- Detail: Digest sizes 224, 256, 384, 512

Keccak

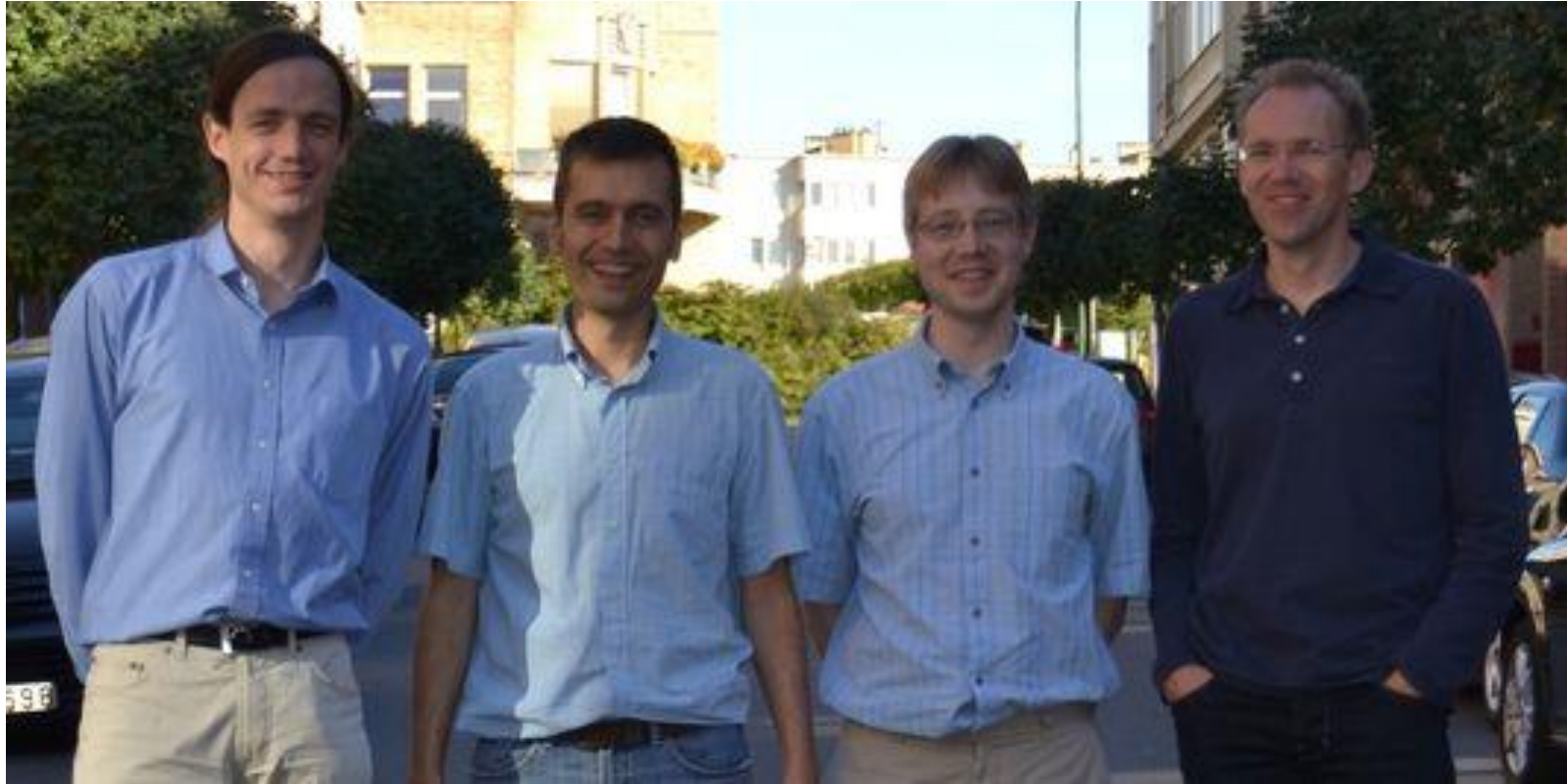
- Designers: [Guido Breton](#), Joan Daemen, [Michaël Peeters](#) and [Gilles Van Assche](#).
- Detail: [Digest sizes](#) arbitrary

SKEIN

- Designers: [Bruce Schneier](#), [Stefan Lucks](#), [Niels Ferguson](#), [Doug Whiting](#), [Mihir Bellare](#), [Tadayoshi Kohno](#), [Jon Callas](#) and [Jesse Walker](#).
- Detail: [Digest sizes](#) arbitrary
- Rounds: 72 (256 & 512 block size), 80 (1024 block size)

dan pemenangnya adalah...

Keccak



[Guido Breton](#), Joan Daemen, [Michaël Peeters](#) and [Gilles Van Assche](#).

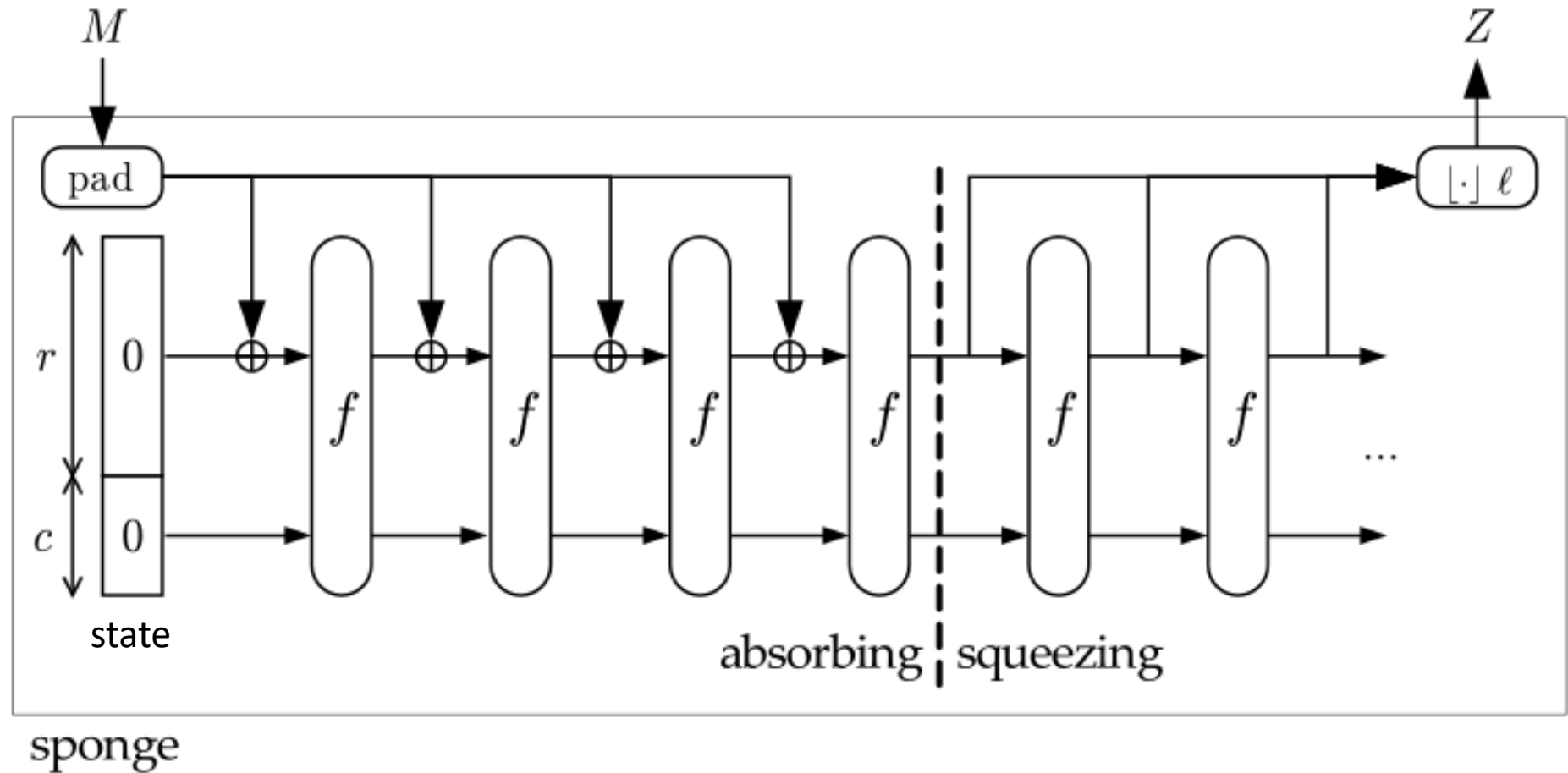
Keccak terpilih sebagai SHA-3

Sekilas Keccak

- Nama 'Keccak' berasal dari kata 'Kecak', sebuah tari dari Bali.
- Keccak berbeda dari finalis SHA3 lainnya dalam hal menggunakan konstruksi 'spons' (*sponge construction*). Jika desain lainnya bergantung pada 'fungsi kompresi, Keccak menggunakan fungsi non-kompresi untuk menyerap dan kemudian 'memeras' *digest*.
- Desain Keccak berbeda dari pendekatan yang ada. NIST menganggap bahwa dalam kasus ini, yang berbeda adalah lebih baik.



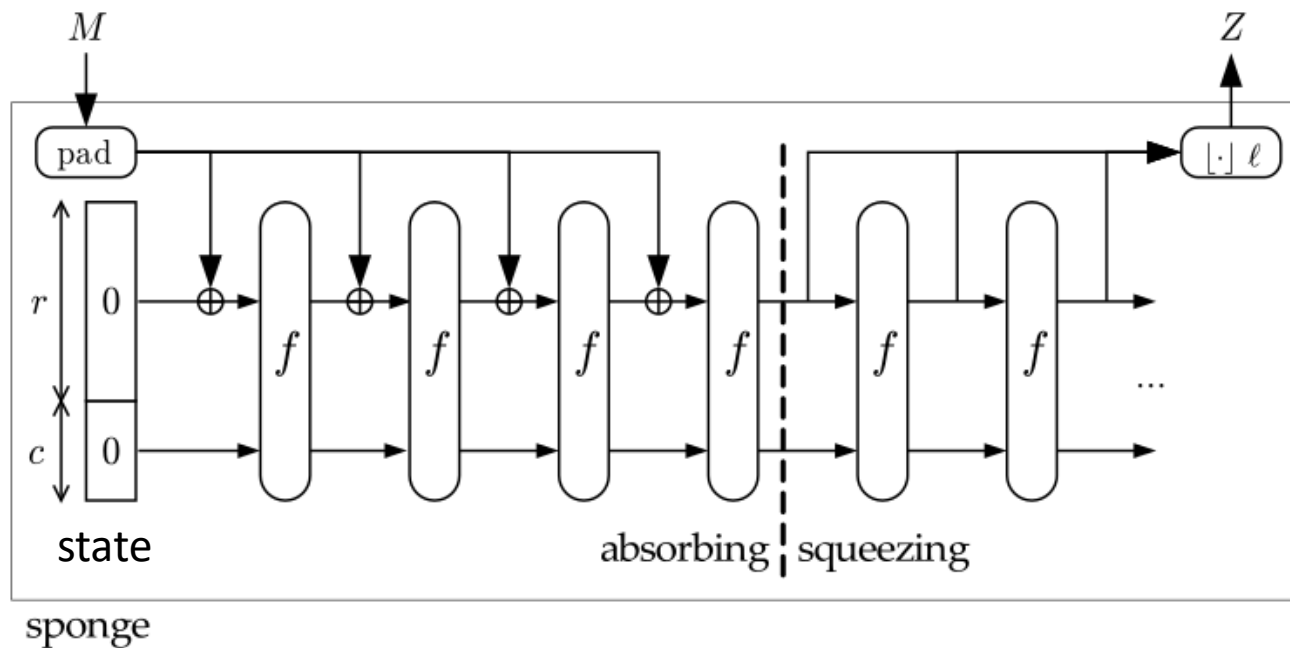
Konstruksi spons



Praproses:

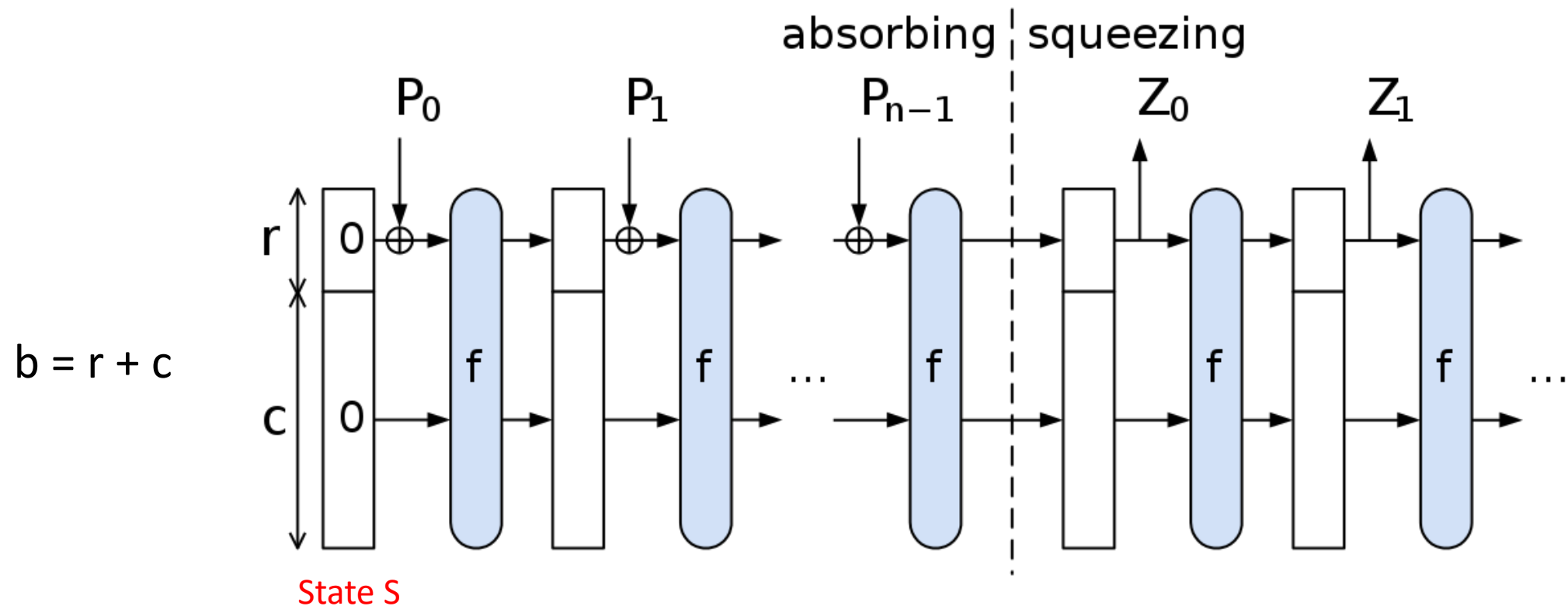
- Misalkan panjang *digest* yang diinginkan adalah d bit, panjang setiap blok r bit.
- Pertama-tama, pesan M ditambah dengan bit-bit pengganjal (*padding*) menjadi string P sehingga panjangnya habis dibagi dengan r , yaitu $n = \text{length}(P) \mathbf{div} r$

- r (*bitrate*): adalah ukuran blok dalam bit yang digunakan untuk memasukkan data (*absorb*) atau mengambil keluaran (*squeeze*) pada setiap iterasi.
- c (*capacity*) adalah bagian *state* yang tidak digunakan secara langsung untuk *absorb* atau *squeeze* dan berperan dalam keamanan algoritma.
- $r + c$ adalah ukuran total *state* di dalam Keccak.



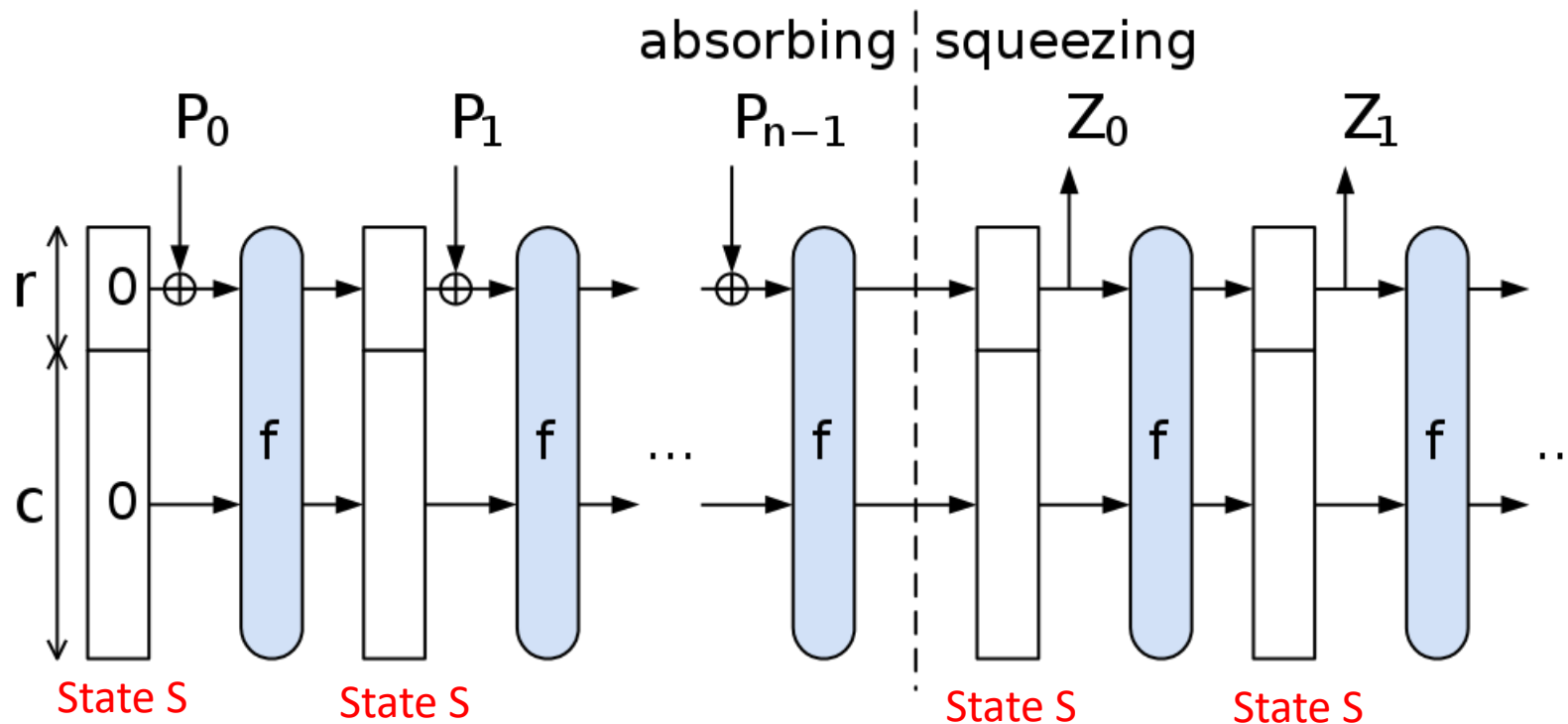
- Pada SHA3-256, parameter yang digunakan adalah:
 $r=1088$ bit = 136 byte
 $c=512$ bit = 64 byte
 sehingga $1088+512=1600$ bit
- Semakin besar nilai c , semakin tinggi tingkat keamanan yang dapat dicapai terhadap serangan

- Setelah dilakukan padding, selanjutnya, P dipotong menjadi blok-blok P_i berukuran r -bit.
- Kemudian, b -bit dari peubah status (*state*) S diinisialisasi menjadi nol dan konstruksi spons berlangsung dalam dua fase: *absorbing* dan *squeezing*.



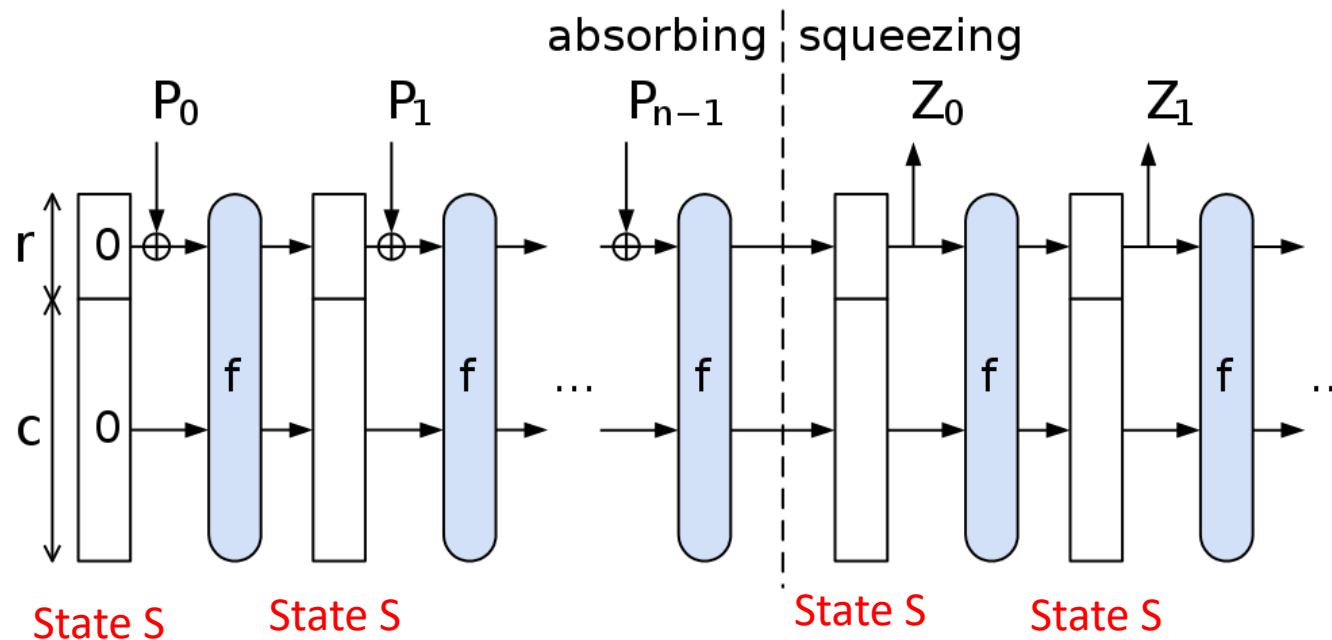
Fase penyerapan (*absorbing*)

- Untuk setiap blok masukan P_i berukuran r -bit, XOR-kan dengan r -bit pertama dari *state* S , lalu hasilnya, bersama-sama dengan c bit dari S dimasukkan ke dalam fungsi permutasi f untuk menghasilkan *state* baru S .
- Bila semua blok masukan selesai diproses, konstruksi spons beralih ke fase pemerasan (*squeezing*).

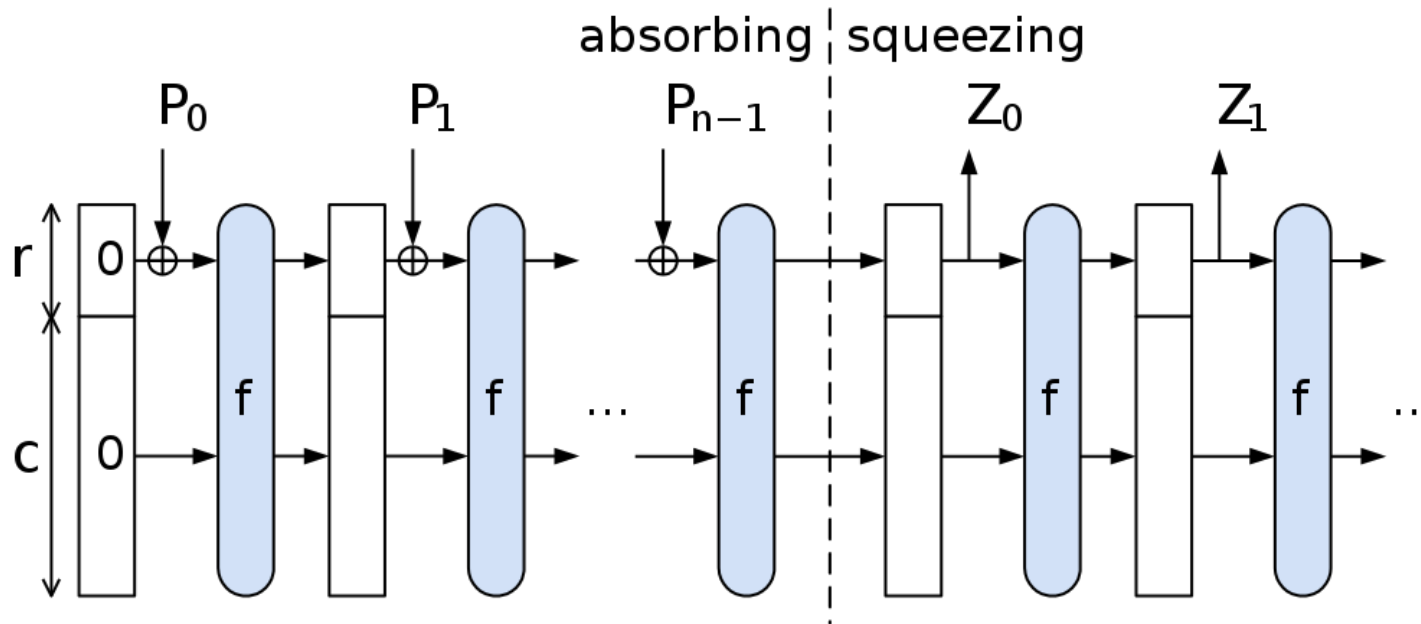


Fase pemerasan (*squeezing*)

- *Message digest* akan disimpan di dalam variabel Z .
- Inisialisasi Z dengan string kosong (*null string*).
- Selagi panjang Z belum sama dengan d , r -bit pertama dari *state* S disambungkan (*append*) ke Z .
- Jika panjang Z masih belum sama dengan d , masukkan S ke dalam fungsi permutasi f menghasilkan *state* baru S .



- Perhatikan bahwa c bit terakhir dari *state* tidak pernah terpengaruh secara langsung oleh blok masukan dan tidak pernah mengeluarkan luaran selama fase pemerasan. Hal ini untuk mencegah terjadinya kolisi.



- Spesifikasi Keccak (termasuk *source code*) dapat dilihat di:
http://keccak.noekeon.org/specs_summary.html

```
Keccak[r,c](Mbytes || Mbits) {
```

```
  # Padding
```

```
  d = 2^|Mbits| + sum for i=0..|Mbits|-1 of 2^i*Mbits[i]
```

```
  P = Mbytes || d || 0x00 || ... || 0x00
```

```
  P = P xor (0x00 || ... || 0x00 || 0x80)
```

```
  # Initialization
```

```
  S[x,y] = 0,                      for (x,y) in (0...4,0...4)
```

```
  # Absorbing phase
```

```
  for each block Pi in P
```

```
    S[x,y] = S[x,y] xor Pi[x+5*y],    for (x,y) such that x+5*y < r/w
```

```
    S = Keccak-f[r+c](S)
```

```
  # Squeezing phase
```

```
  Z = empty string
```

```
  while output is requested
```

```
    Z = Z || S[x,y],                for (x,y) such that x+5*y < r/w
```

```
    S = Keccak-f[r+c](S)
```

```
  return Z
```

```
}
```

https://keccak.team/keccak_specs_summary.html

```

Keccak-f[b](A) {
  for i in 0...n-1
    A = Round[b](A, RC[i])
  return A
}

```

```

Round[b](A,RC) {
  #  $\theta$  step
  C[x] = A[x,0] xor A[x,1] xor A[x,2] xor A[x,3] xor A[x,4], for x in 0...4
  D[x] = C[x-1] xor rot(C[x+1],1), for x in 0...4
  A[x,y] = A[x,y] xor D[x], for (x,y) in (0...4,0...4)

  #  $\rho$  and  $\pi$  steps
  B[y,2*x+3*y] = rot(A[x,y], r[x,y]), for (x,y) in (0...4,0...4)

  #  $\chi$  step
  A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y]), for (x,y) in (0...4,0...4)

  #  $\iota$  step
  A[0,0] = A[0,0] xor RC

  return A
}

```

https://keccak.team/keccak_specs_summary.html

Keccak-256

Keccak-256 online hash function

halo

Input type Text

Hash

☒ Auto Update

51e5ddea5dcdaa85bc8623b8ac163bed5c3b00e9e2ceaeca78f7f3cd8016d836

Hash	File Hash
CRC-16	CRC-16
CRC-32	CRC-32
MD2	MD2
MD4	MD4
MD5	MD5
SHA1	SHA1
SHA224	SHA224
SHA256	SHA256
SHA384	SHA384
SHA512	SHA512
SHA512/224	SHA512/224
SHA512/256	SHA512/256
SHA3-224	SHA3-224
SHA3-256	SHA3-256
SHA3-384	SHA3-384
SHA3-512	SHA3-512
Keccak-224	Keccak-224

Contoh *message digest* dengan Keccak-256

keccak256("halo")=
51e5ddea5dcdaa85bc8623b8ac163bed5c3b00e9e2ceaeca78f7f3cd8016d836

keccak256("halu")=
beccbf92062e8427947bfed81a546d4ccebba76d3f002bc254e19a6d3359d144

keccak256("halo, apa kabar teman? ") =
0055d097fad321072610be3f16147533355cd1d370577a21ffae735f9da47c48

keccak("The quick brown fox jumps over the lazy dog") =
4d741b6f1eb29cb2a9b9911c82f56fa8d73b04959d3d9d222895df6c0b28aa15

SELAMAT BELAJAR