Bahan Kuliah II4021 Kriptografi

Beberapa Fungsi Hash

Oleh: Dr. Rinaldi Munir

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika(STEI)

ITB

1. Fungsi Hash MD5

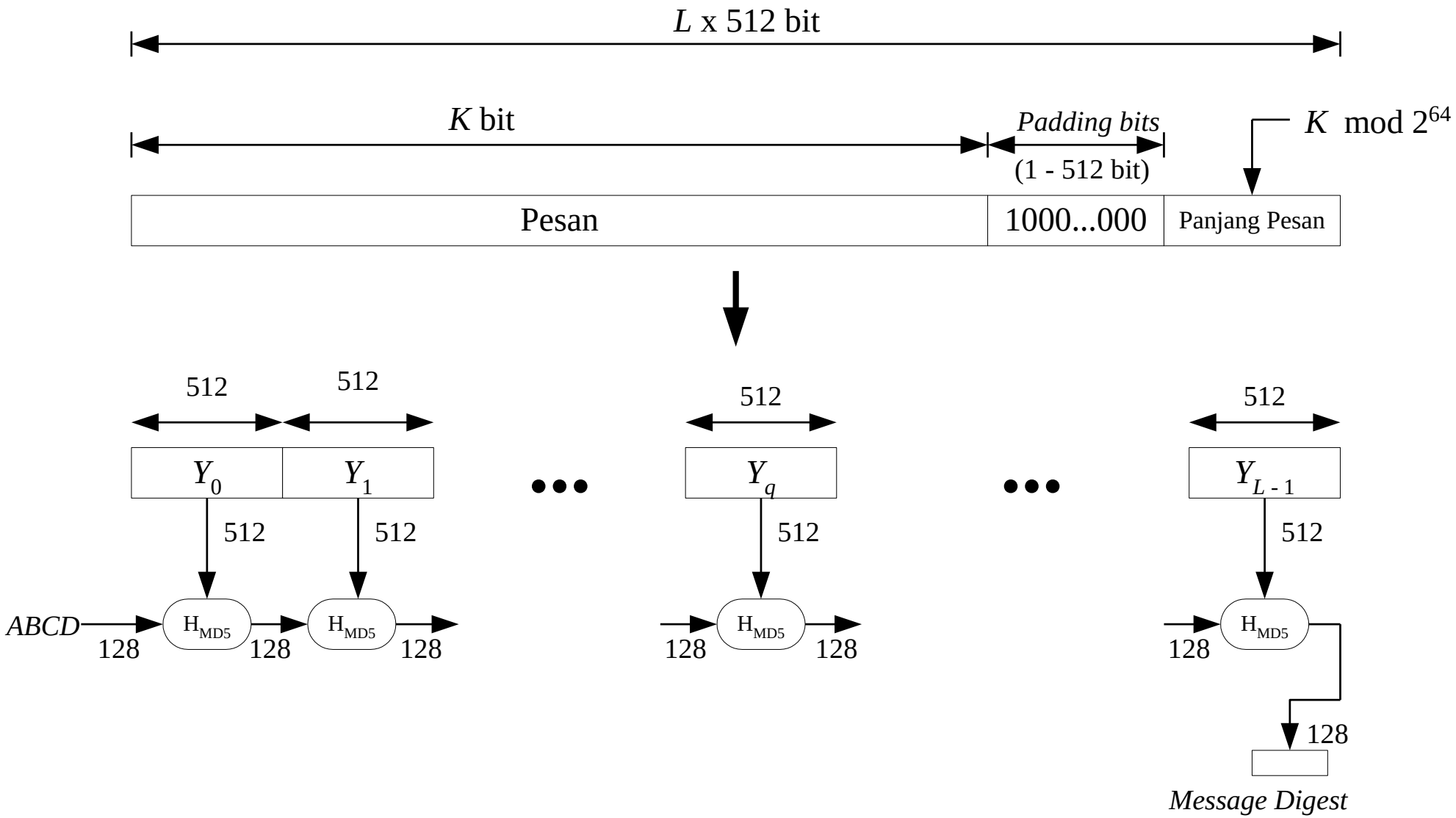
Pendahuluan

- *MD5* adalah fungsi *hash* satu-arah yang dibuat oleh Ron Rivest.

- *MD5* merupakan perbaikan dari *MD4* setelah *MD4* ditemukan kolisinya.

- Algoritma *MD5* menerima masukan berupa pesan dengan ukuran sembarang dan menghasilkan *message digest* yang panjangnya 128 bit.

Gambaran umum MD5



Langkah-langkah pembuatan *message digest* secara garis besar:

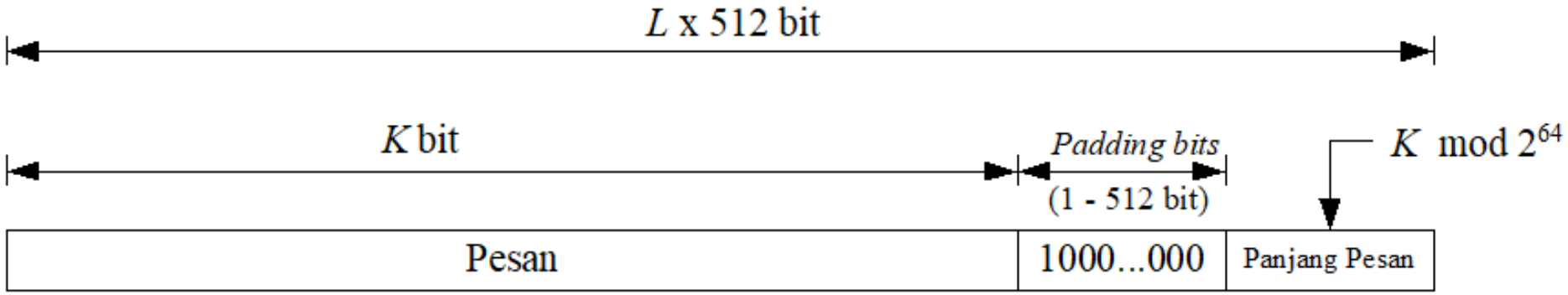
1. Penambahan bit-bit pengganjal (*padding bits*).

2. Penambahan nilai panjang pesan semula.

3. Inisialisasi penyangga (*buffer*) *MD*.

4. Pengolahan pesan dalam blok berukuran 512 bit.

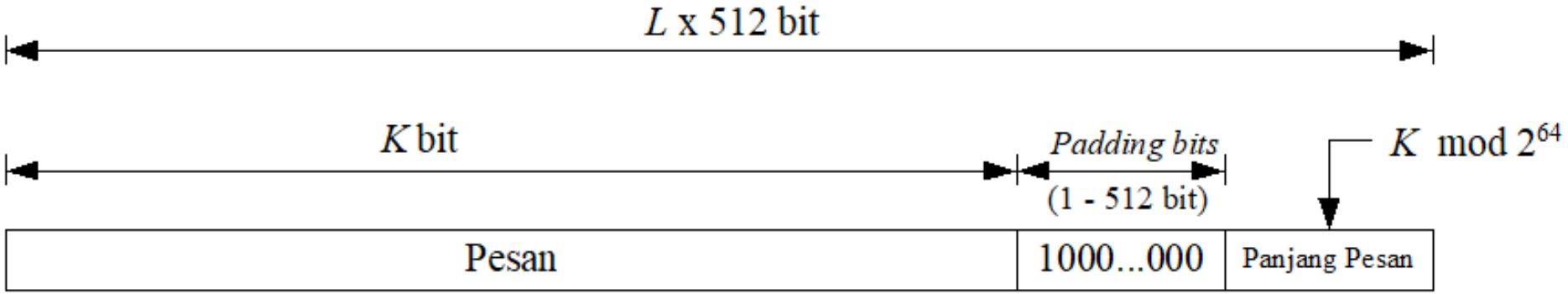
1. Penambahan Bit-bit Pengganjal



- Pesan ditambah dengan sejumlah bit pengganjal (*padding bits*) sedemikian sehingga panjang pesan (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 bit → 32000 + 192 bit = 32192 → 32192 mod 512 = 448

2. Penambahan Nilai Panjang Pesan



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

  Contoh: K = 32000 = 111110100000000_2 = 000...111110100000000

- Jika K > 2^{64} maka yang diambil adalah K mod 2^{64}.
- Setelah ditambah dengan 64 bit, panjang pesan sekarang menjadi kelipatan 512 bit.

3. Inisialisasi Penyangga MD

- *MD5* membutuhkan 4 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit. Total panjang penyangga adalah $4 \times 32 = 128$ bit. Keempat penyangga ini menampung hasil antara dan hasil akhir.

- Keempat penyangga ini diberi nama *A*, *B*, *C*, dan D. Setiap penyangga diinisialisasi dengan nilai-nilai (dalam notasi HEX) sebagai berikut:

  A = 01234567

  B = 89ABCDEF

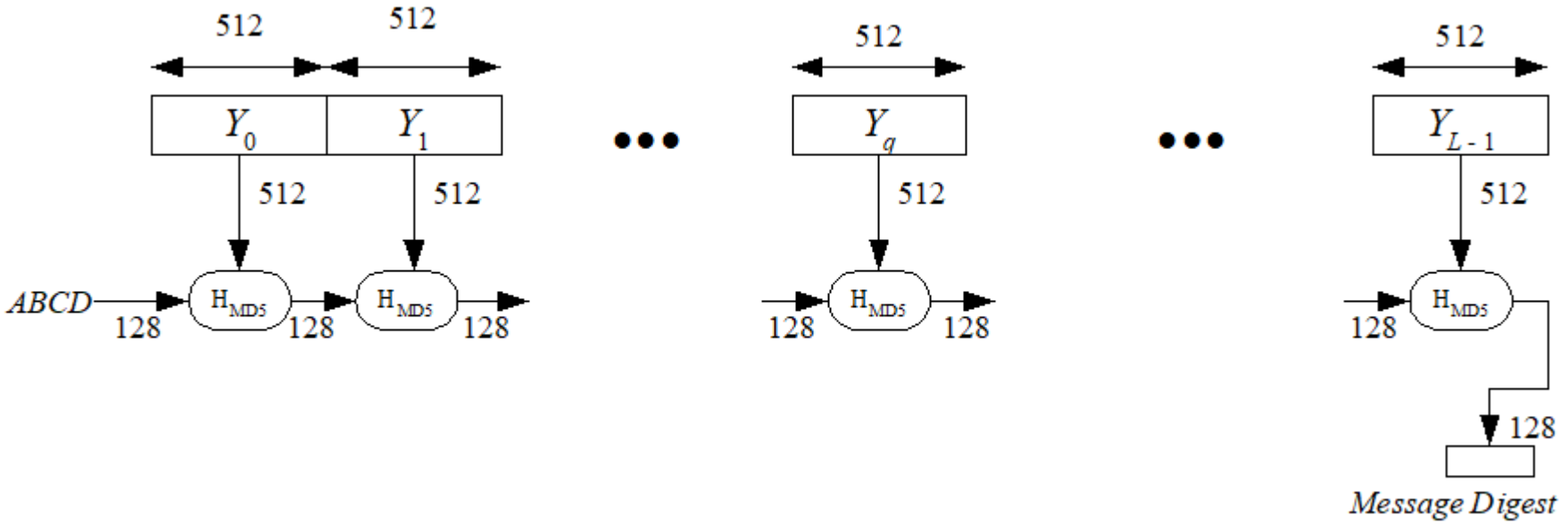
  C = FEDCBA98

  D = 76543210

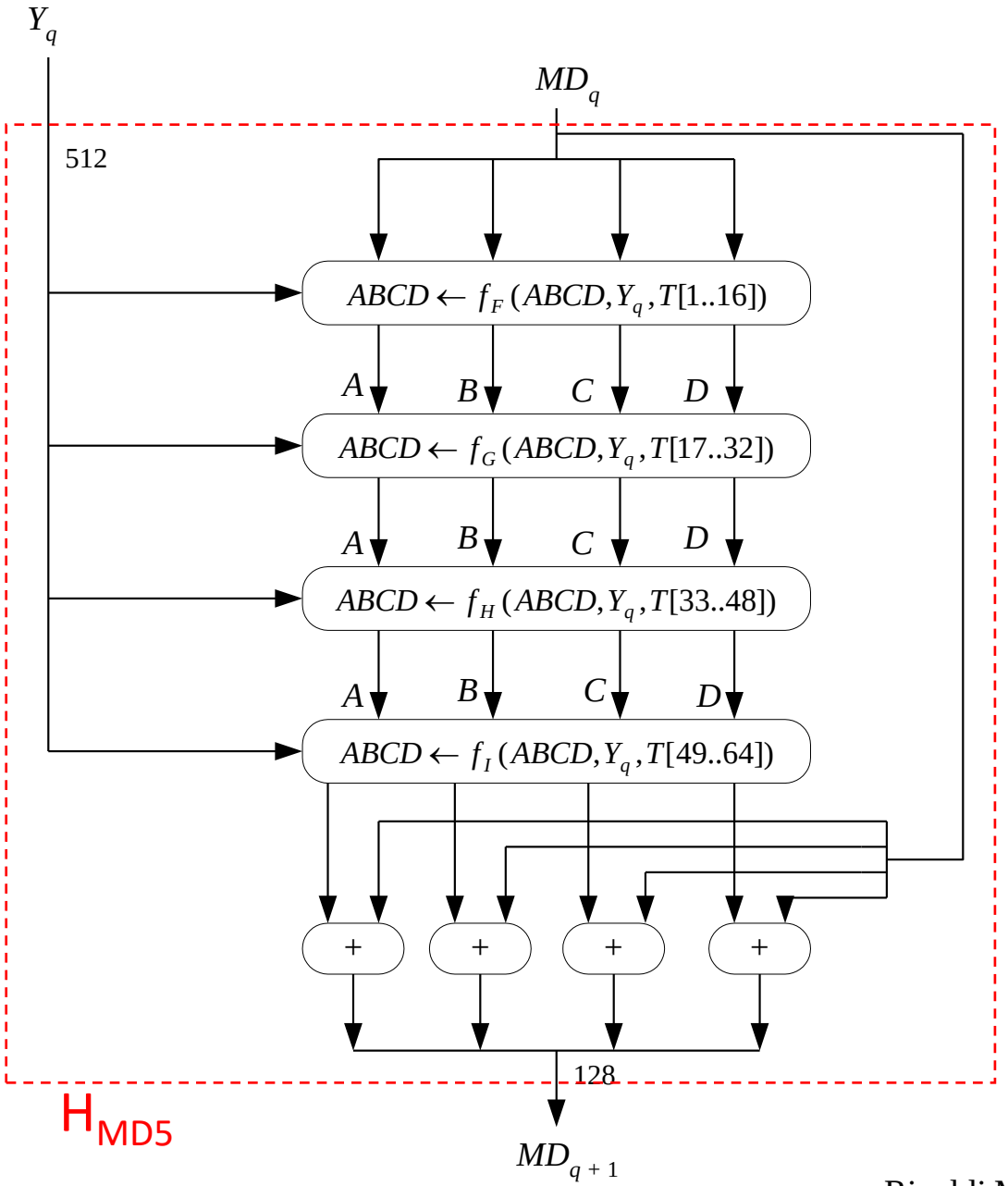
- Untuk mempersingkat penyebutan, keempat penyangga itu disebut MD saja

4. Pengolahan Pesan dalam Blok Berukuran 512 bit.

- Pesan dibagi menjadi *L* buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai Y_{L-1}).



- Setiap blok 512-bit diproses bersama dengan penyangga *MD* menjadi keluaran 128-bit, dan ini disebut proses H_{MD5}.

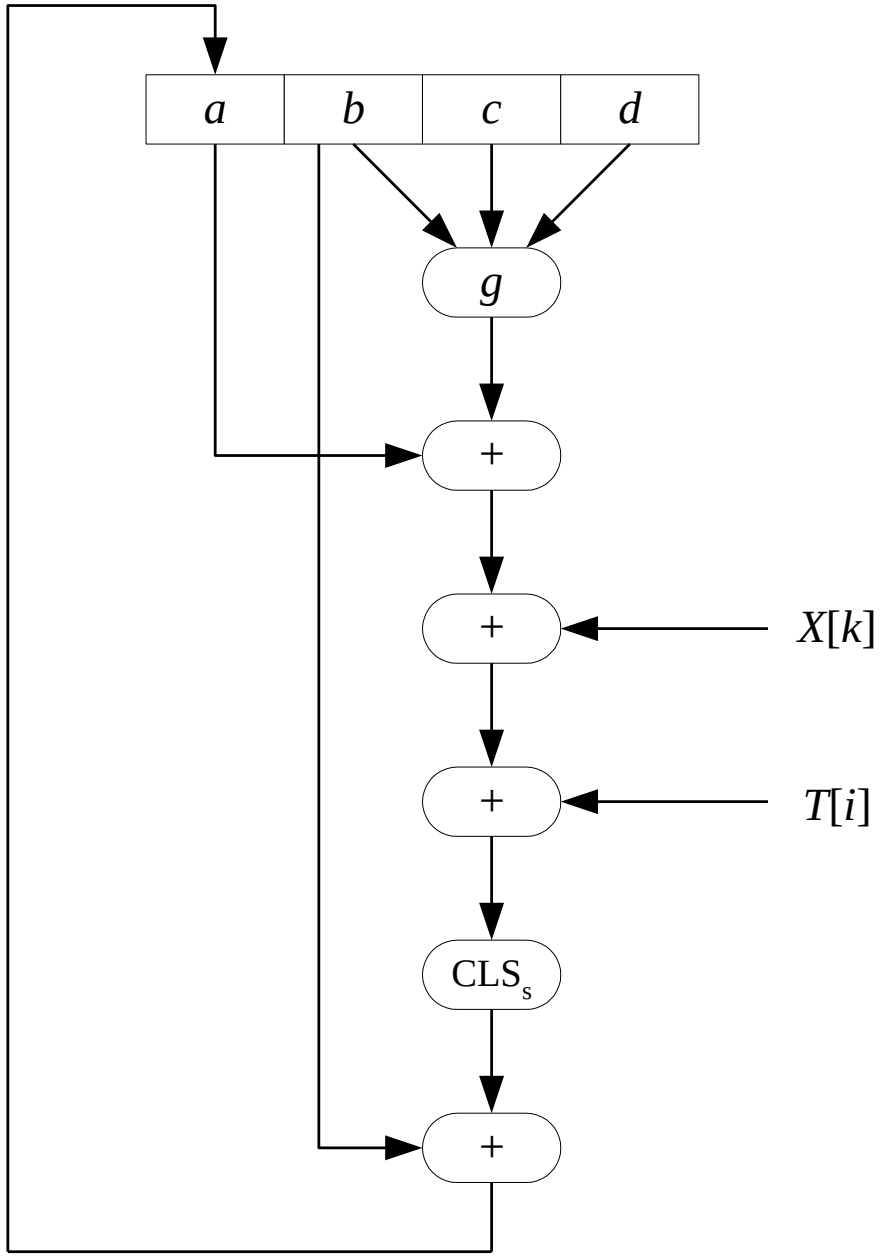


Pada Gambar tersebut, Y_q menyatakan blok 512-bit ke-q

MD_q adalah nilai *message digest* 128-bit dari proses H_{MD5} ke-q. Pada awal proses, MD_q berisi nilai inisialisasi penyangga *MD*.

Proses H_{MD5} terdiri dari 4 buah putaran:

- Setiap putaran melakukan operasi dasar *MD5* sebanyak 16 kali
- Operasi dasar melibatkan fungsi f_F, f_G, f_H, dan f_I
- Setiap operasi dasar memakai sebuah elemen *T*
- Jadi setiap putaran memakai 16 elemen Tabel *T*.



Operasi dasar *MD5* pada gambar di samping dapat ditulis dengan sebuah persamaan:

$$a \leftarrow b + \text{CLS}_s(a + g(b, c, d) + X[k] + T[i])$$

a, b, c, d = empat buah peubah penyangga 32-bit A, B, C, D

g = salah satu fungsi F, G, H, I

CLS_s = *circular left shift* sebanyak s bit

$X[k]$ = kelompok 32-bit ke-k dari blok 512 bit *message* ke-q. Nilai k = 0 sampai 15.

$T[i]$ = elemen Tabel T ke-i (32 bit)

$+$ = operasi penjumlahan dalam modulo 2^{32}

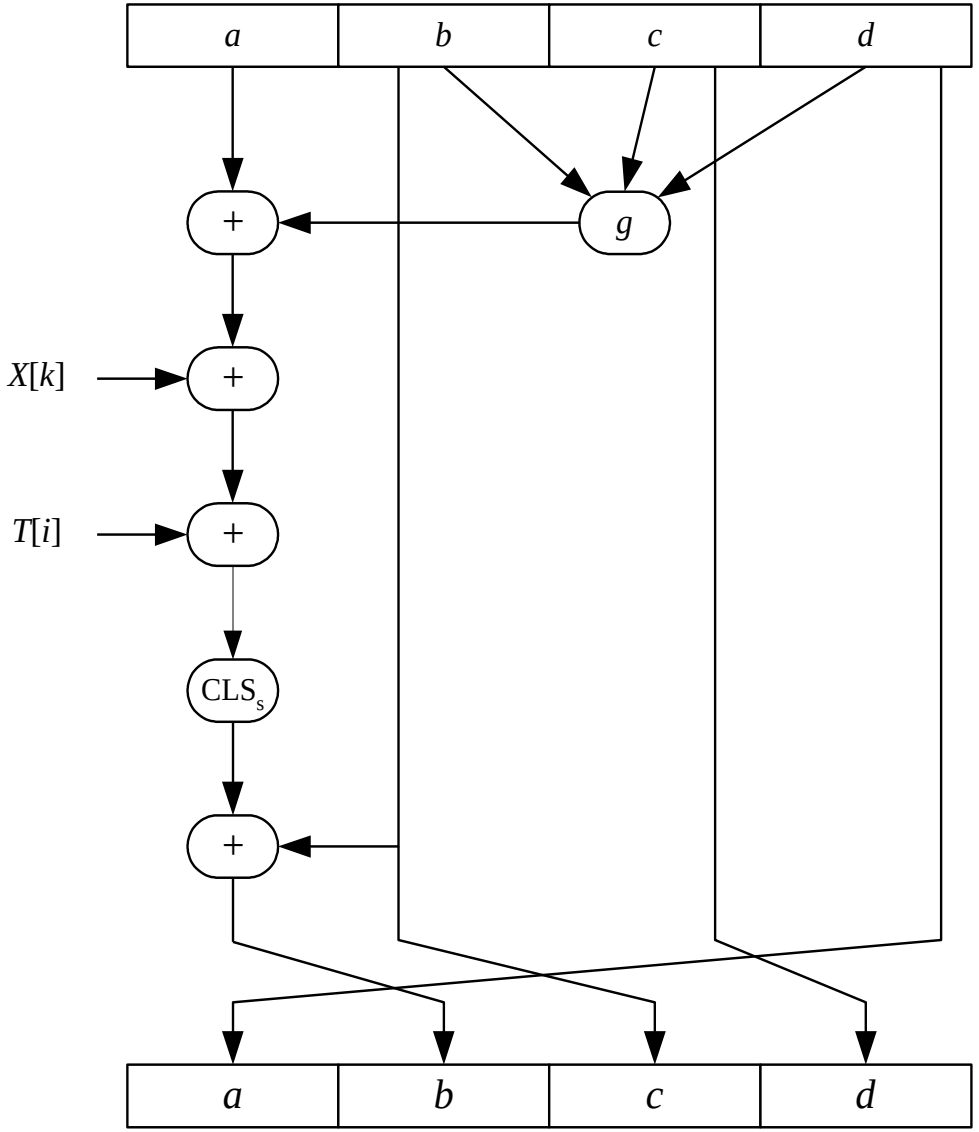
Tabel 1. Fungsi-fungsi dasar MD5

Nama	**Notasi**	***g*(*b*, *c*, *d*)**
f_F	$F(b, c, d)$	$(b \wedge c) \vee (\sim b \wedge d)$
f_G	$G(b, c, d)$	$(b \wedge d) \vee (c \wedge \sim d)$
f_H	$H(b, c, d)$	$b \oplus c \oplus d$
f_I	$I(b, c, d)$	$c \oplus (b \wedge \sim d)$

Catatan: operator logika AND, OR, NOT, XOR masing-masing dilambangkan dengan $\wedge$, $\vee$, $\sim$, $\oplus$

- Karena ada 16 kali operasi dasar, maka setiap kali selesai satu operasi dasar, penyangga-penyangga itu digeser ke kanan secara sirkuler dengan cara pertukaran sebagai berikut:

```
temp ← d
d ← c
c ← b
b ← a
a ← temp
```



Tabel 2. Nilai $T[i]$

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E5A51	T[35] = 69D96122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57C0FAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBCB	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

- Misalkan notasi

$$[abcd \quad k \quad s \quad i]$$

menyatakan operasi

$$a \leftarrow b + \text{CLS}_s(a + g(b, c, d) + X[k] + T[i])$$

maka operasi dasar pada masing-masing putaran dapat ditabulasikan sebagai berikut:

Putaran 1: 16 kali operasi dasar dengan $g(b, c, d) = F(b, c, d)$ dapat dilihat pada Tabel 3.

Tabel 3. Rincian operasi pada fungsi $F(b, c, d)$

No.	[*abcd*	*k*	*s*	*i*]
1	[ABCD	0	7	1]
2	[DABC	1	12	2]
3	[CDAB	2	17	3]
4	[BCDA	3	22	4]
5	[ABCD	4	7	5]
6	[DABC	5	12	6]
7	[CDAB	6	17	7]
8	[BCDA	7	22	8]
9	[ABCD	8	7	9]
10	[DABC	9	12	10]
11	[CDAB	10	17	11]
12	[BCDA	11	22	12]
13	[ABCD	12	7	13]
14	[DABC	13	12	14]
15	[CDAB	14	17	15]
16	[BCDA	15	22	16]

$$a \leftarrow b + \mathrm{CLS}_s(a + g(b, c, d) + X[k] + T[i])$$

Putaran 2: 16 kali operasi dasar dengan $g(b, c, d) = G(b, c, d)$ dapat dilihat pada Tabel 4.

Tabel 4. Rincian operasi pada fungsi $G(b, c, d)$

No.	[*abcd*	*k*	*s*	*i*]
1	[ABCD	1	5	17]
2	[DABC	6	9	18]
3	[CDAB	11	14	19]
4	[BCDA	0	20	20]
5	[ABCD	5	5	21]
6	[DABC	10	9	22]
7	[CDAB	15	14	23]
8	[BCDA	4	20	24]
9	[ABCD	9	5	25]
10	[DABC	14	9	26]
11	[CDAB	3	14	27]
12	[BCDA	8	20	28]
13	[ABCD	13	5	29]
14	[DABC	2	9	30]
15	[CDAB	7	14	31]
16	[BCDA	12	20	32]

$$a \leftarrow b + \mathrm{CLS}_s(a + g(b, c, d) + X[k] + T[i])$$

Putaran 3: 16 kali operasi dasar dengan $g(b, c, d) = H(b, c, d)$ dapat dilihat pada Tabel 5.

Tabel 5. Rincian operasi pada fungsi $H(b, c, d)$

No.	[*abcd* *k* *s* *i*]
1	[ABCD 5 4 33]
2	[DABC 8 11 34]
3	[CDAB 11 16 35]
4	[BCDA 14 23 36]
5	[ABCD 1 4 37]
6	[DABC 4 11 38]
7	[CDAB 7 16 39]
8	[BCDA 10 23 40]
9	[ABCD 13 4 41]
10	[DABC 0 11 42]
11	[CDAB 3 16 43]
12	[BCDA 6 23 44]
13	[ABCD 9 4 45]
14	[DABC 12 11 46]
15	[CDAB 15 16 47]
16	[BCDA 2 23 48]

$$a \leftarrow b + \mathrm{CLS}_s(a + g(b, c, d) + X[k] + T[i])$$

Putaran 4: 16 kali operasi dasar dengan $g(b, c, d) = I(b, c, d)$ dapat dilihat pada Tabel 6.

Tabel 6. Rincian operasi pada fungsi $I(b, c, d)$

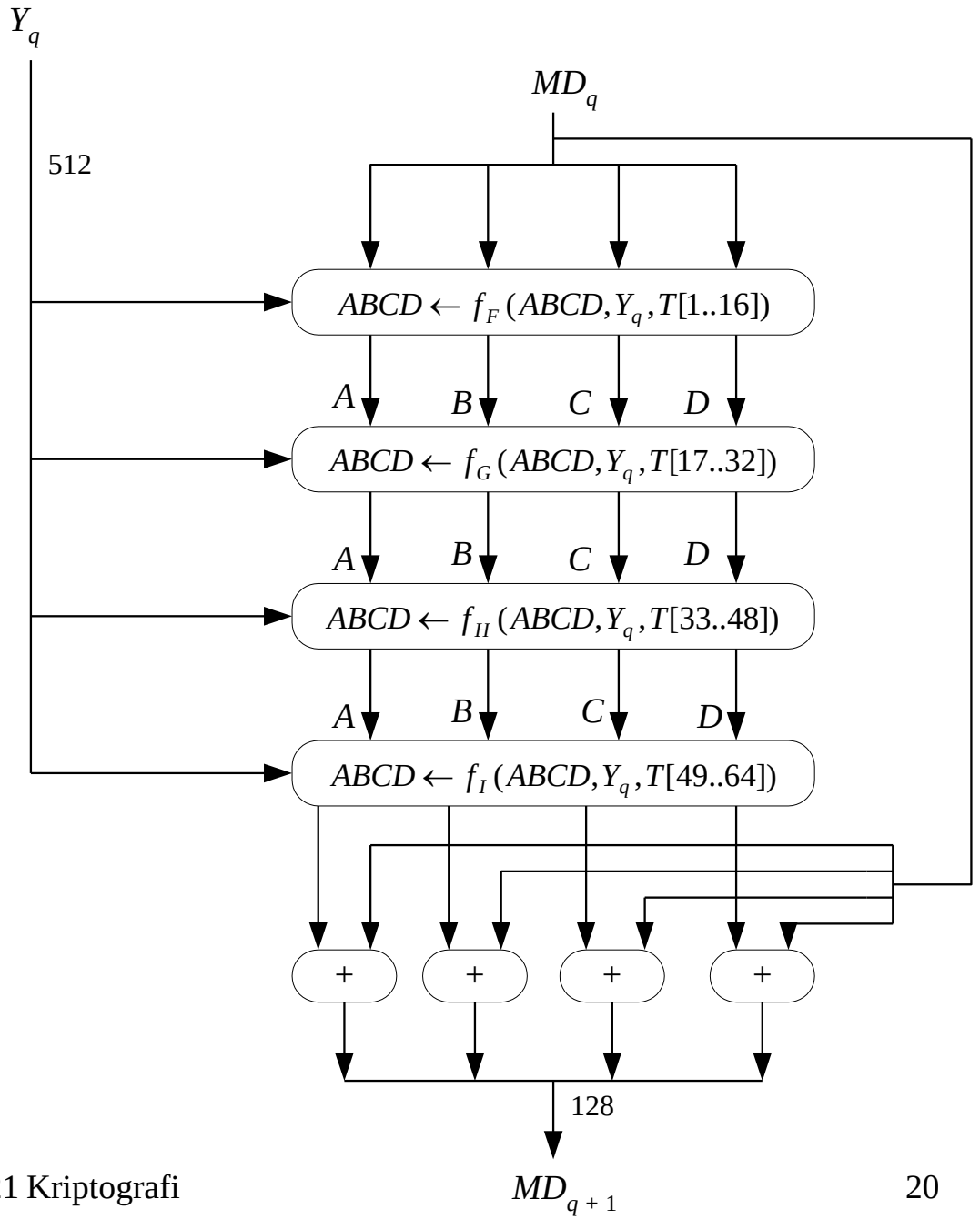
No.	[*abcd* *k* *s* *i*]
1	[ABCD 0 6 49]
2	[DABC 7 10 50]
3	[CDAB 14 15 51]
4	[BCDA 5 21 52]
5	[ABCD 12 6 53]
6	[DABC 3 10 54]
7	[CDAB 10 15 55]
8	[BCDA 1 21 56]
9	[ABCD 8 6 57]
10	[DABC 15 10 58]
11	[CDAB 6 15 59]
12	[BCDA 13 21 60]
13	[ABCD 4 6 61]
14	[DABC 11 10 62]
15	[CDAB 2 15 63]
16	[BCDA 9 21 64]

$$a \leftarrow b + \mathrm{CLS}_s(a + g(b, c, d) + X[k] + T[i])$$

- Setelah putaran ke-4, *a*, *b*, *c*, dan *d* ditambahkan ke *A*, *B*, *C*, dan *D.*

- Selanjutnya algoritma memproses untuk blok data berikutnya (Y_{q+1}).

- Luaran akhir dari algoritma *MD5* adalah hasil penyambungan bit-bit di *A*, *B*, *C*, dan *D*.



- Contoh *message digest* beberapa pesan yang dihasilkan oleh MD5:

md5(“halo”) = 57f84228617109485 5e51fc3a541c1e2

md5(“halo, apa kabar teman?”) =
d29029a1dcf256466290d773f5dbfc0f

md5(“The quick brown fox jumps over the lazy dog”) =
9e107d9d372bb6826bd81d3542a419d6

Contoh. Contoh message digest dari sebuah file, `bandung.txt`, sebagai berikut:

```
Pada bulan Oktober 2004 ini, suhu udara kota Bandung terasa lebih panas dari
hari-hari biasanya. Menurut laporan Dinas Meteorologi Kota Bandung, suhu
tertinggi kota Bandung adalah 33 derajat Celcius pada Hari Rabu, 17 Oktober
yang lalu. Suhu terseut sudah menyamai suhu kota Jakarta pada hari-hari biasa.
Menurut Kepala Dinas Meteorologi, peningkatan suhu tersebut terjadi karena
posisi bumi sekarang ini lebih dekat ke matahari daripada hari-hari biasa.

Sebutan Bandung sebagai kota sejuk dan dingin mungkin tidak lama lagi akan
tinggal kenangan. Disamping karena faktor alam, jumlah penduduk yang padat,
polusi dari pabrik di sekita Bandung, asap knalpot kendaraan, ikut menambah
kenaikan suhu udara kota.
```

Message digest dari arsip `bandung.txt` yang dihasilkan oleh algoritma *MD5* adalah 128-bit:

```
0010 1111 1000 0010 1100 0000 1100 1000 1000 0100 0101 0001 0010 0001
1011 0001 1011 1001 0101 0011 1101 0101 0111 1101 0100 1100 0101 1001
0001 1110 0110 0011
```

atau, dalam notasi HEX adalah:

```
2F82D0C845121B953D57E4C3C5E91E63
```

Kriptanalisis MD5

Review kembali sifat-sifat fungsi *hash H*:

a) collision resistance : sangat sukar menemukan dua input *a* dan *b* sedemikian sehingga $H(a) = H(b)$

b) preimage resistance: untuk sembarang output *y*, sukar menemukan input *a* sedemikian sehingga $H(a) = y$

c) second preimage resistance – untuk input *a* dan output $y = H(a)$, sukar menemukan input kedua *b* sedemikian sehingga $H(b) = y$

Kriptanalisis MD5

- Secara teori, menemukan kolisi pada fungsi *hash* sangatlah sukar dilakukan.

- Pada awalnya penemu algoritma *MD5* menganggap usaha tersebut hampir tidak mungkin dilakukan karena membutuhkan waktu yang sangat lama.

- Tetapi, pada tahun 1996, Dobbertin melaporkan penemuan kolisi pada algoritma *MD5* meskipun kecacatan ini bukan kelemahan yang fatal.

- Pada tanggal 1 Maret 2005, Arjen Lenstra, Xiaoyun Wang, dan Benne de Weger mendemostraskan pembentukan dua buah pesan berbeda (berupa sertifikat X.509, yang akan dijelaskan di dalam bab *Infrastruktur Kunci Publik)* tetapi mempunyai nilai *hash* yang sama (lihat publikasinya di dalam hhtp://eprint.iacr.org/2005/067).

- Beberapa hari kemudian, Vlastimil Klima (http://eprint.iacr.org/2005/075) memperbaiki algoritma Lenstra dkk yang dapat menghasilkan kolisi *MD5* hanya dalam waktu beberapa jam dengan menggunakan komputer *PC.*

(Sumber: Wikipedia)

Contoh dua pesan yang hanya berbeda 6 bit tetapi memiliki nilai *hash* sama (Sumber: *Eric Rescorla (2004-08-17). "A real MD5 collision"*):

M1 =

d131dd02c5e6eec4693d9a0698aff95c2fcab5**8**712467eab400458
3eb8fb7f8955ad340609f4b30283e4888325**7**1415a085125e8f7cd
c99fd91dbd**f**280373c5bd8823e3156348f5bae6dacd436c919c6dd
53e2**b**487da03fd02396306d248cda0e99f33420f577ee8ce54b670
80**a**80d1ec69821bcb6a8839396f965**2**b6ff72a70

M2 =

d131dd02c5e6eec4693d9a0698aff95c2fcab5**0**712467eab400458
3eb8fb7f8955ad340609f4b30283e4888325**f**1415a085125e8f7cd
c99fd91dbd**7**280373c5bd8823e3156348f5bae6dacd436c919c6dd
53e2**3**487da03fd02396306d248cda0e99f33420f577ee8ce54b670
80**2**80d1ec69821bcb6a8839396f965**a**b6ff72a70

Kedua pesan di atas memiliki nilai *hash* **79054025255fb1a26e4bc422aef54eb4.**

Dua buah gambar biner yang memiliki nilai hash MD5 yang sama

Sumber:
https://natmchugh.blogspot.com/2015/05/how-to-make-two-binaries-with-same-md5.html

2. Secure Hash Algorithm (SHA)

Secure Hash Algorithm (*SHA*)

- *SHA* adalah fungsi *hash* satu-arah yang dibuat oleh *NIST* dan digunakan bersama *DSS* (*Digital Signature Standard*).

- Oleh *NSA*, *SHA* dinyatakan sebagai standard fungsi *hash* satu-arah.

- *SHA* didasarkan pada *MD4* yang dibuat oleh Ronald L. Rivest dari *MIT*.

- Algoritma *SHA* menerima masukan berukuran maksimum 2^{64} bit (2.147.483.648 *gigabyte*) dan menghasilkan *message digest* yang panjangnya 160 bit.

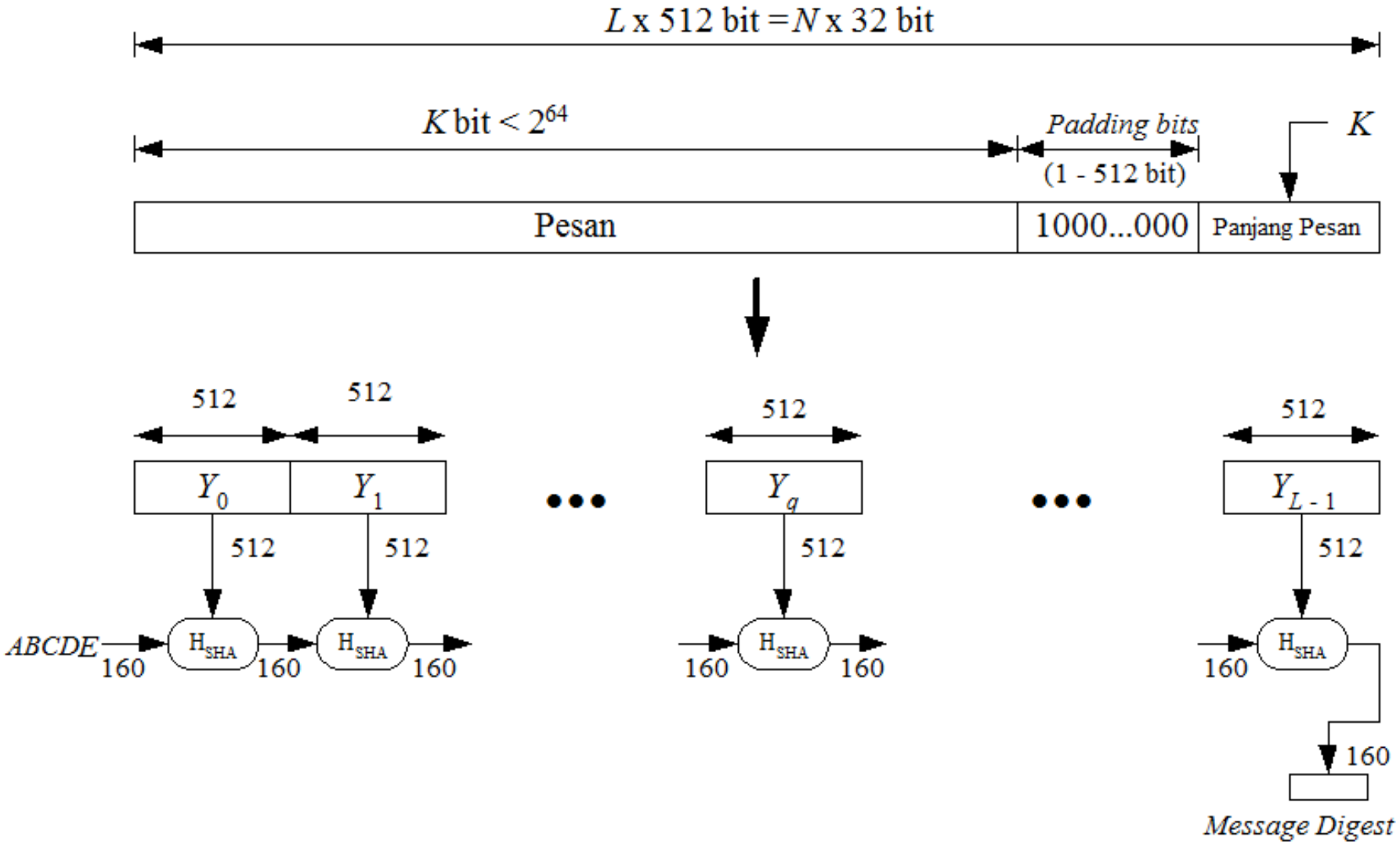
- *Message digest* dari SHA lebih panjang dari *message digest* yang dihasilkan oleh *MD5* (128 bit).

- 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	Collisions found?
SHA-0									Yes
SHA-1		160	160	512	$2^{64} - 1$	32	80	+,and,or, xor,rot	Yes
SHA-2	*SHA-256/224*	256/224	256	512	$2^{64} - 1$	32	64	+,and,or, xor,shr,rot	No
	SHA-512/384	512/384	512	1024	$2^{128} - 1$	64	80		

Gambaran Umum *SHA-1*



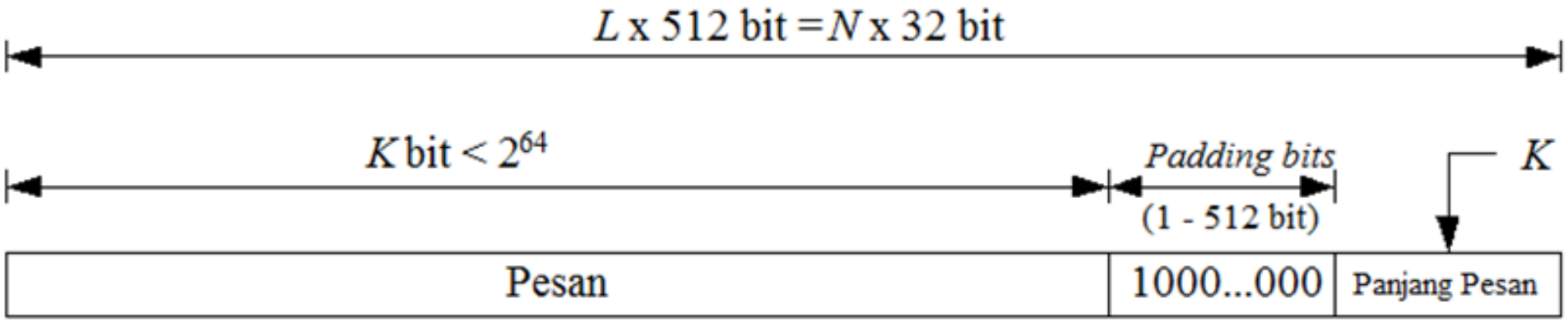
Langkah-langkah pembuatan *message digest* dengan SHA-1

- Secara umum, SHA-1 mirip dengan MD5, hanya berbeda dalam jumlah variabel *buffer* (5 buah) dan komputasi di dalam pengolahan blok pesan 512-bit.

- Langkah-Langkah di dalam SHA-1 adalah sbb:

  1. Penambahan bit-bit pengganjal (*padding bits*).
  2. Penambahan nilai panjang pesan semula.
  3. Inisialisasi penyangga (*buffer*) *MD*.
  4. Pengolahan pesan dalam blok berukuran 512 bit.

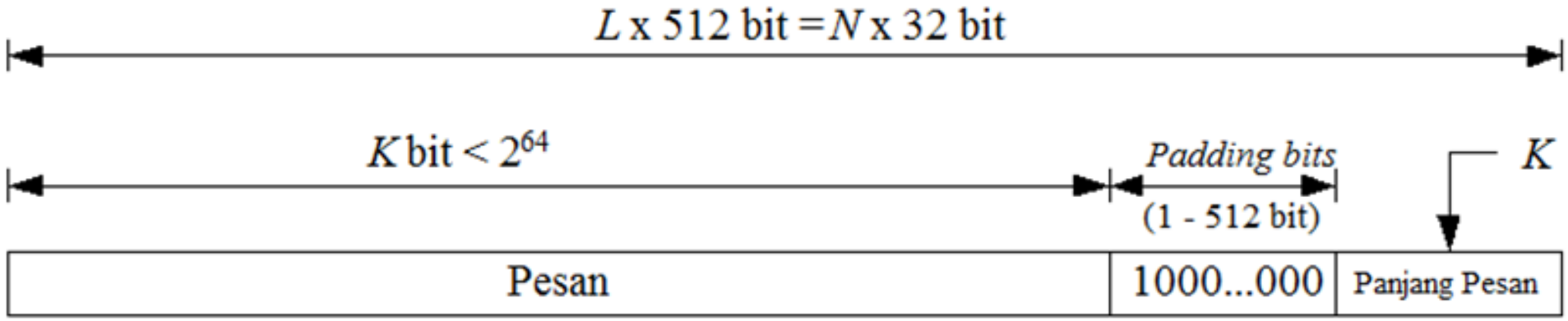
1. Penambahan Bit-bit Pengganjal



- Pesan ditambah dengan sejumlah bit pengganjal (*padding bits*) sedemikian sehingga panjang pesan (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 bit → 32000 + 192 bit = 32192 → 32192 mod 512 = 448

2. Penambahan Nilai Panjang Pesan



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

  Contoh: K = 32000 = 111110100000000_2 = 000…111110100000000

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

3. Inisialisasi Penyangga MD

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

  A = 67452301

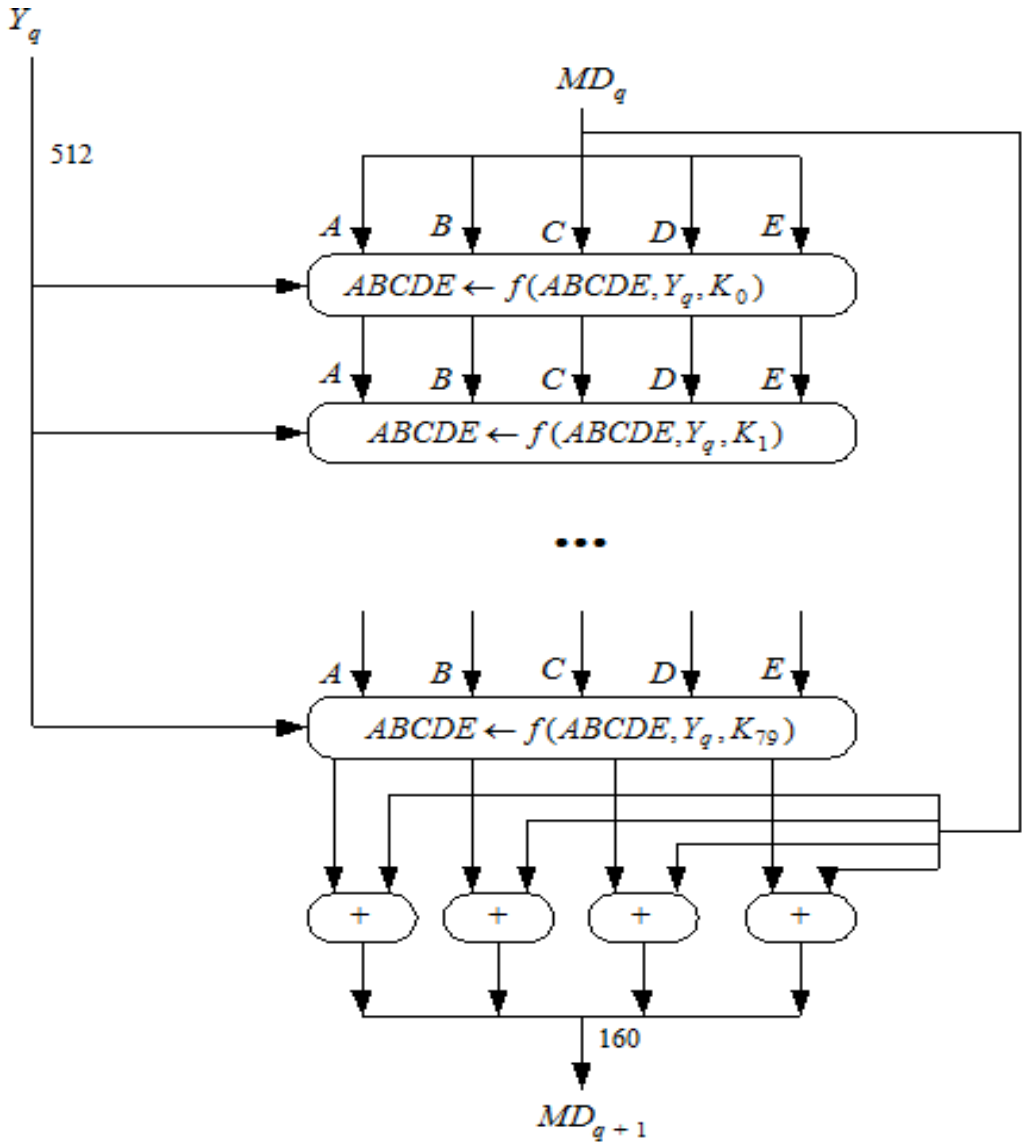
  B = EFCDAB89

  C = 98BADCFE

  D = 10325476

  E = C3D2E1F0

4. Pengolahan Pesan dalam Blok Berukuran 512 bit.



- Proses H_{SHA} terdiri dari 80 buah putaran (*MD5* hanya 4 putaran)

- Masing-masing putaran menggunakan bilangan penambah K_t, yaitu:

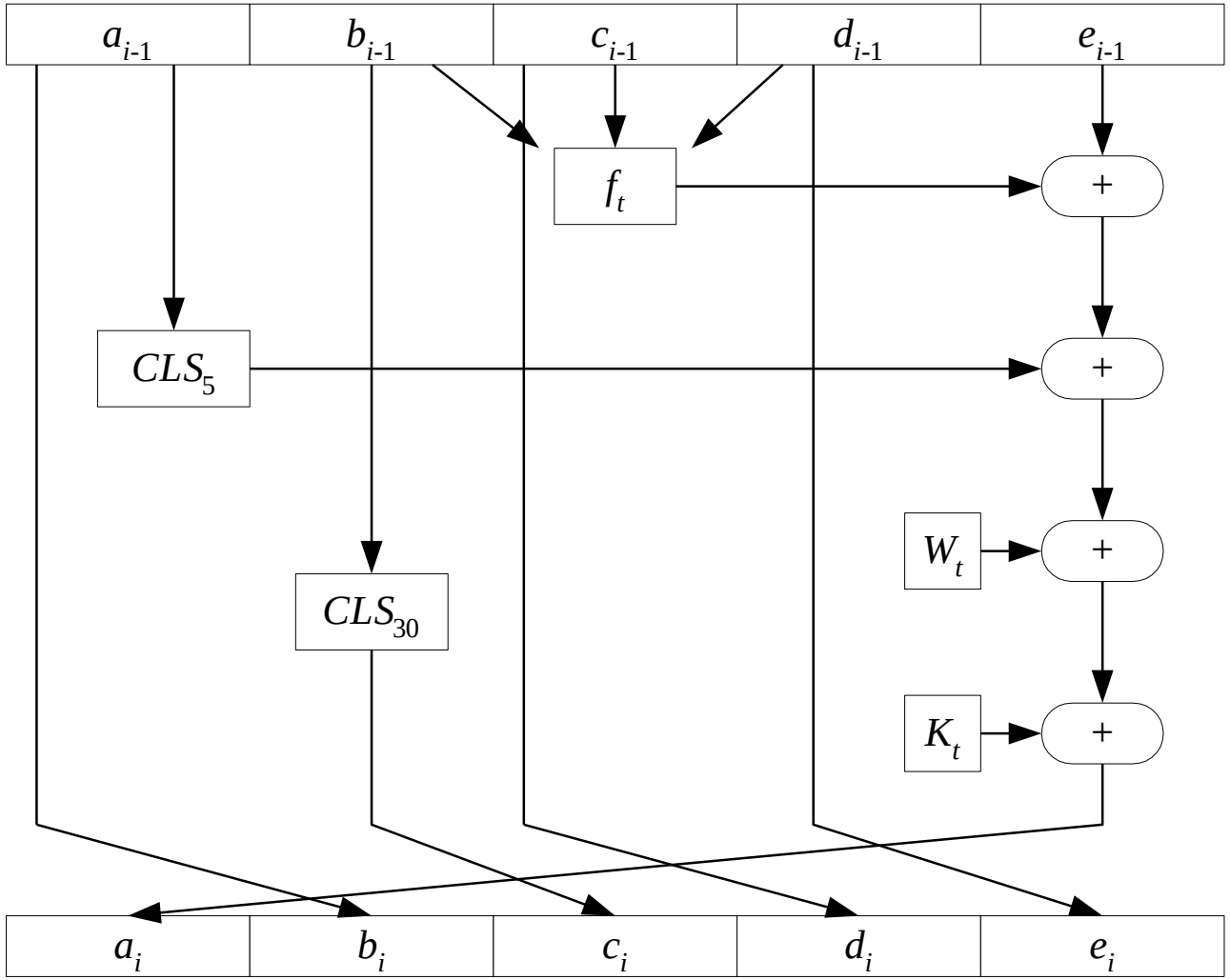
Putaran $0 \leq t \leq 19$ K_t = 5A827999

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_t pada setiap putaran

Putaran	$f_t(b, c, d)$
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	$b \oplus c \oplus d$

$e \leftarrow d$

$d \leftarrow c$

$c \leftarrow \mathrm{CLS}_{30}(b)$

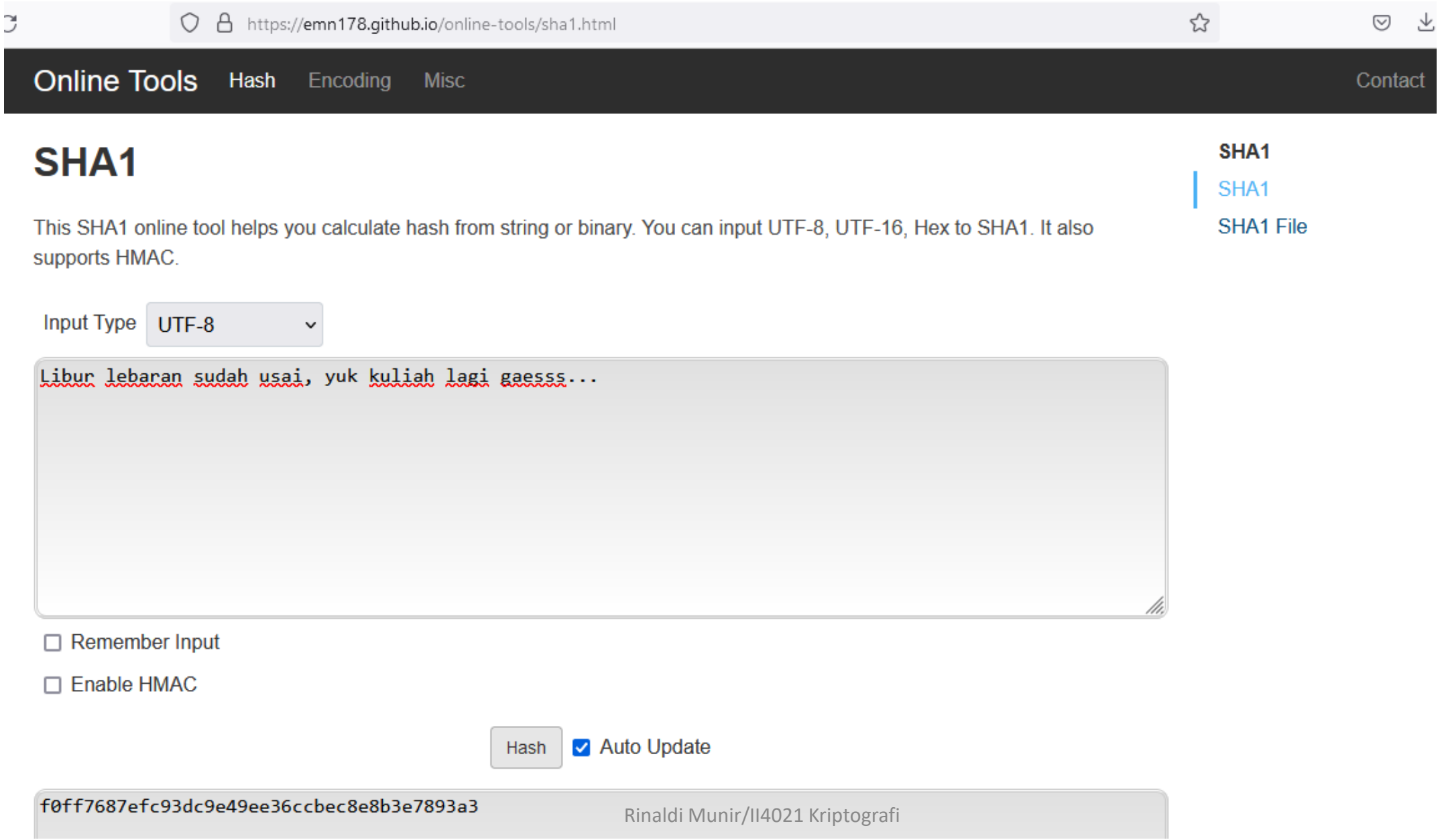
$b \leftarrow a$

$a \leftarrow (\mathrm{CLS}_5(a) + f_t(b, c, d) + e + W_t + K_t)$

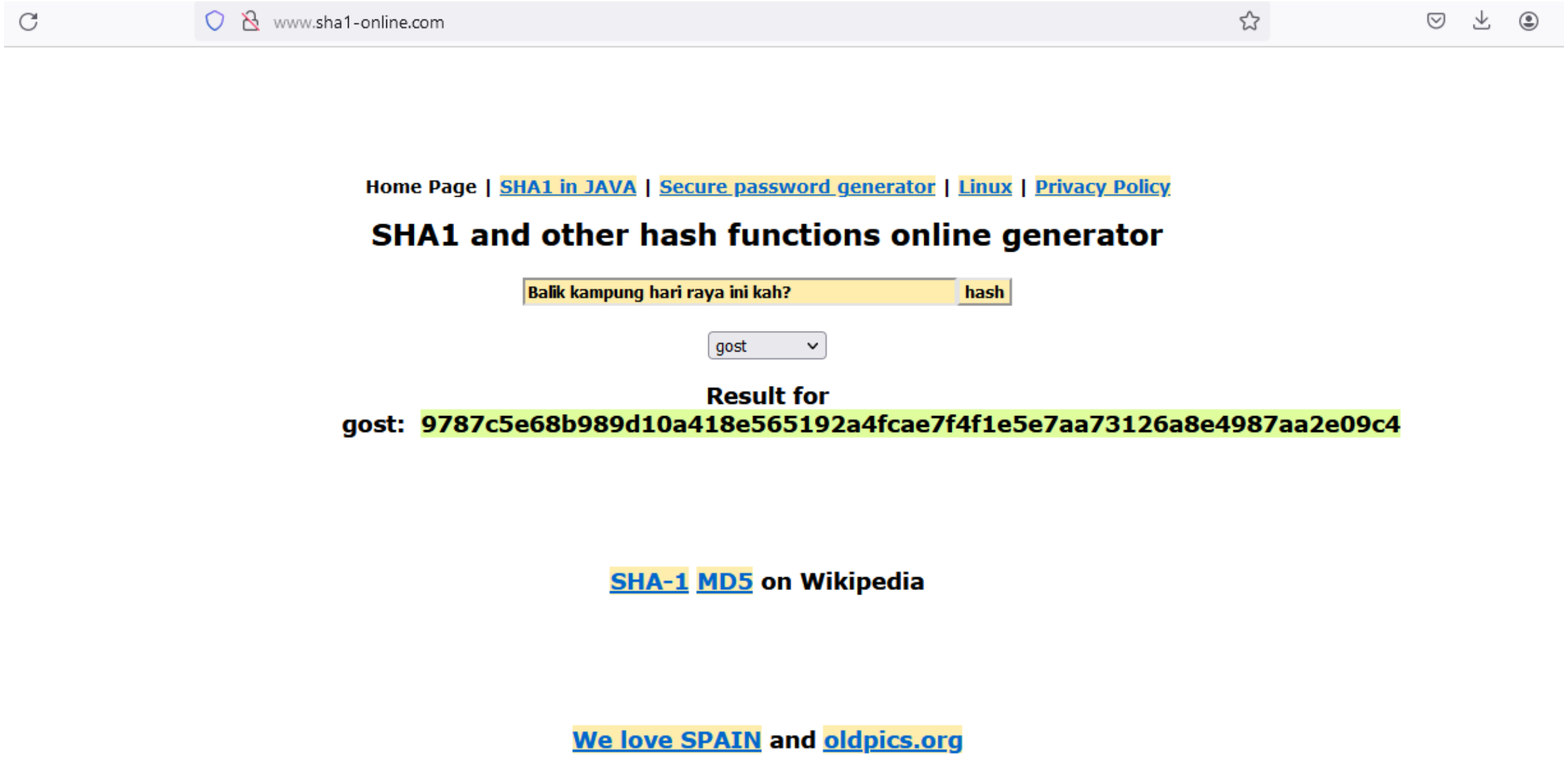
Nilai W_1 sampai W_{16} berasal dari 16 *word* pada blok yang sedang diproses (1 word = 32 bit), sedangkan nilai W_t berikutnya didapatkan dari persamaan

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

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



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



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 tahun 2005, Rijmen dan Oswald mempubliksikan serangan pada versi *SHA*-1 yang direduksi (hanya menggunakan 53 putaran dari 80 putaran) dan menemukan kolisi dengan kompleksitas sekitar 2^{80} operasi (lihat di http://eprint.iacr.org/2005/010).

- Pada bulan Februari 2005, Xiayoun Wang, Yiqun Lisa Yin, dan Hongbo Yo mempublikasikan serangan yang dapat menemukan kolisi pada versi penuh SHA-1, yang membutuhkan sekitar 2^{69} operasi (lihat beritanya di:

  http://www.schneier.com/blog/archives/2005/02/sha_1broken.html)

- 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

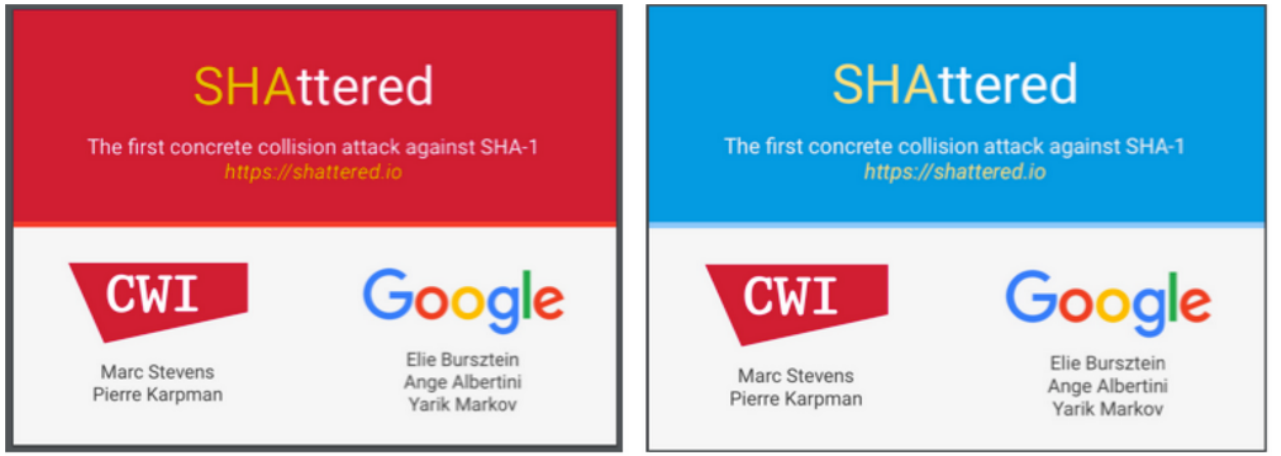
Published

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

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 :
cf2b7075c1c82eb83c5c1cf88d4818c02f6a9e7799bd111960f62fa247a28aa4893320364 5cda413eac702b8
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

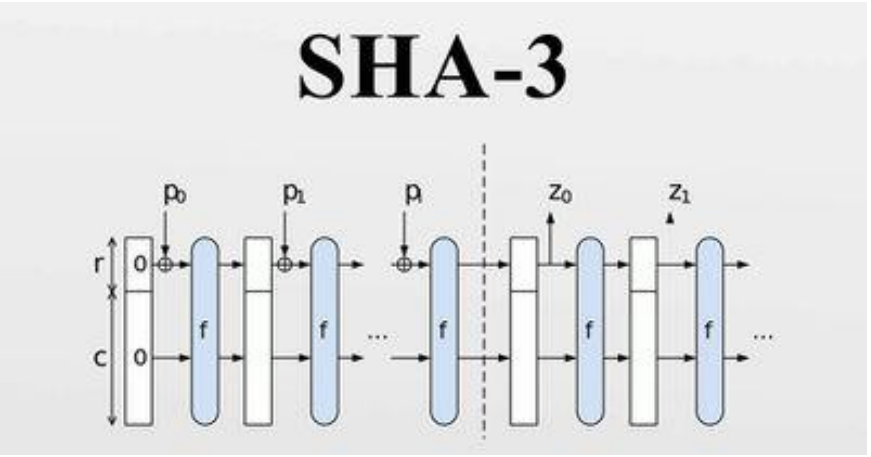
Kode Program Hash dengan Python (4)

Memeriksa integitas (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



3. SHA-3 (Keccak)

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

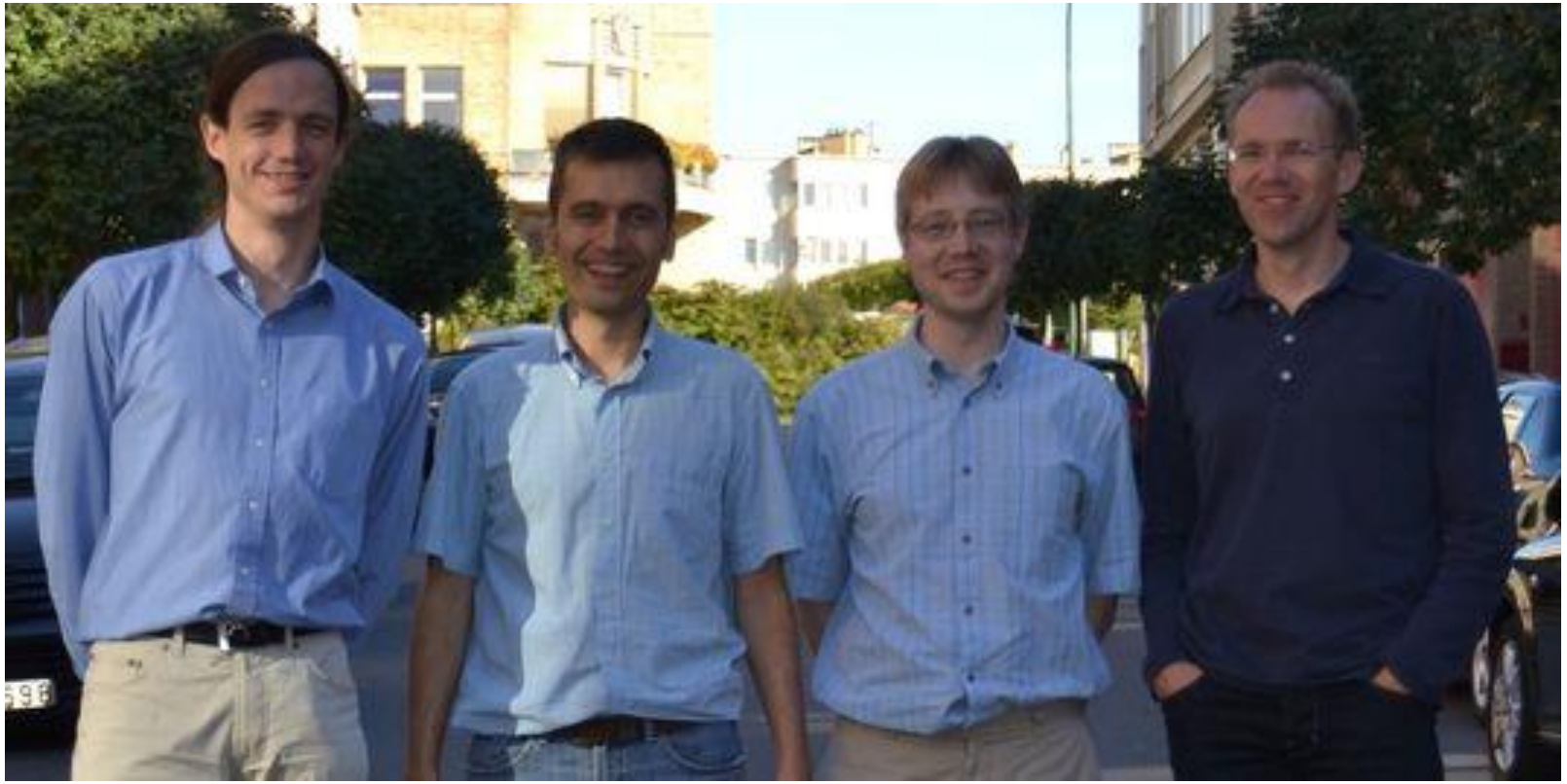
SKEIN

- 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

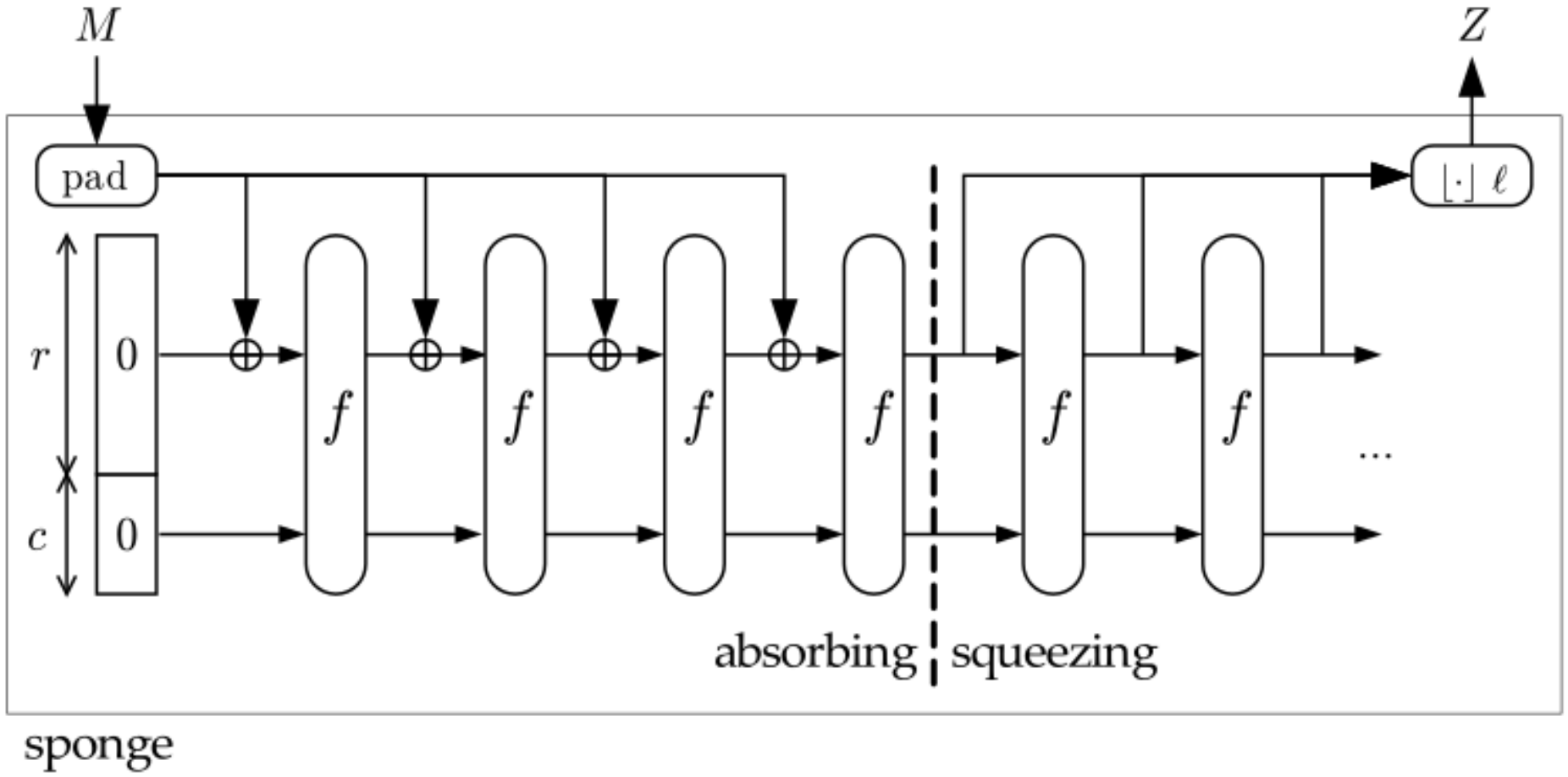
Sekilas Keccak

- 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 merasa bahwa dalam kasus ini, yang berbeda adalah lebih baik.

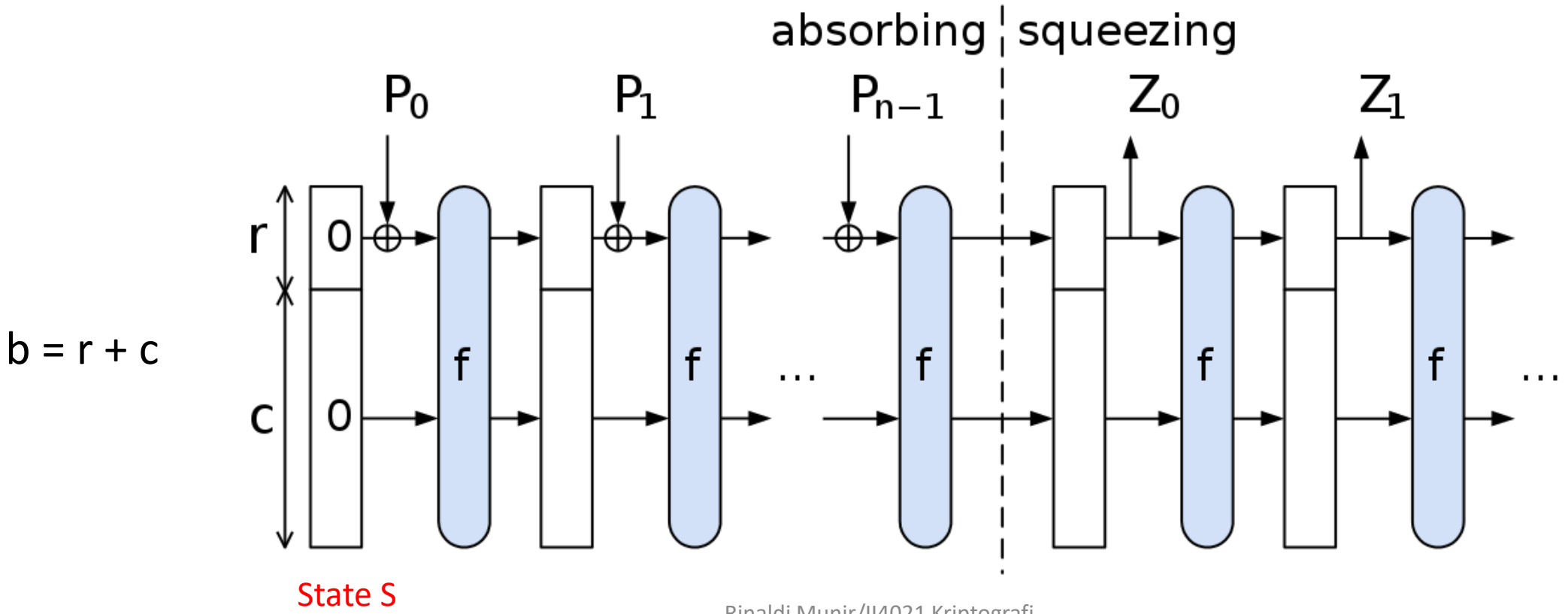
Konstruksi spons



Praproses:

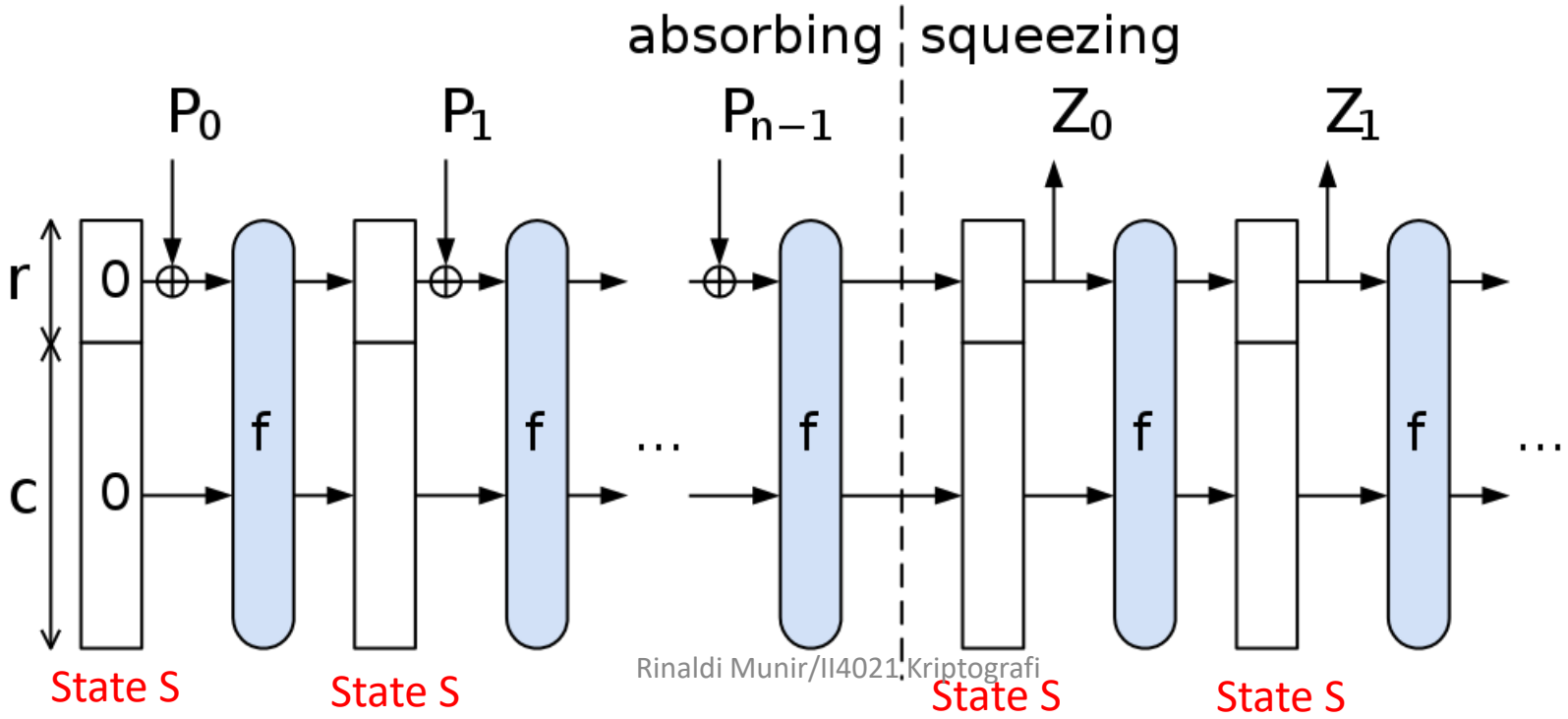
- Misalkan panjang *digest* yang dinginkan adalah *d* bit, Panjang setiap blok *r* bit.
- Pertama, pesan *M* ditambah dengan bit-bit pengganjal (*padding*) menjadi string *P* sehingga habis dibagi dengan *r* atau *n* = length(*P*)/*r*

- 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:



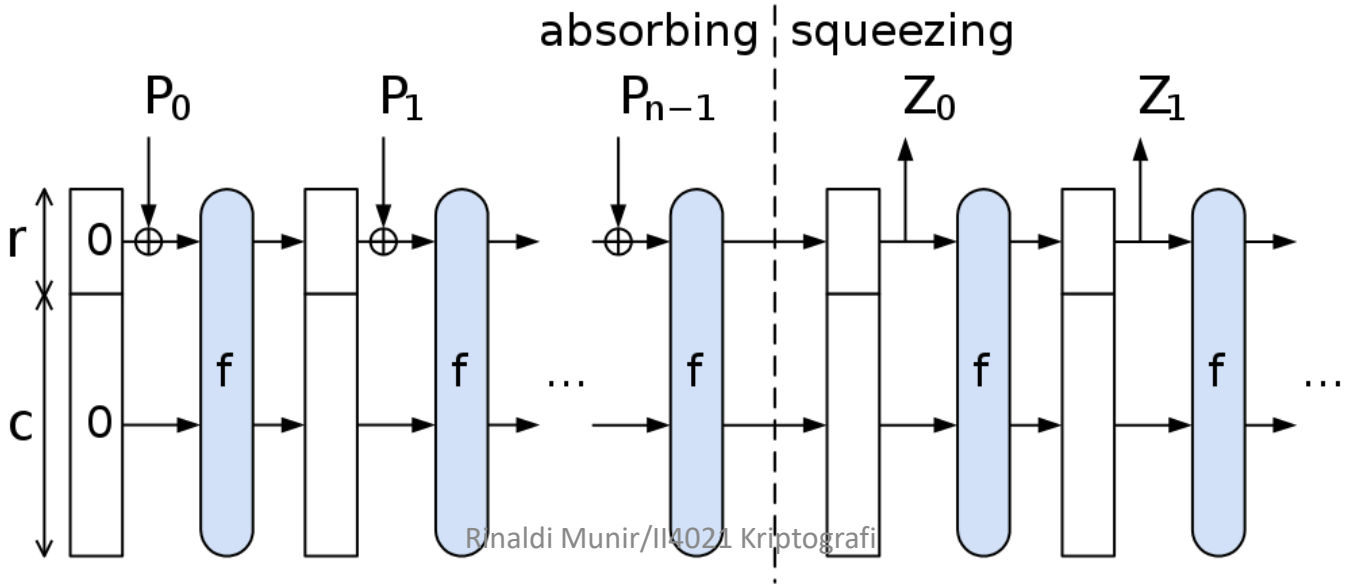
Fase penyerapan (*absorbing*)

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

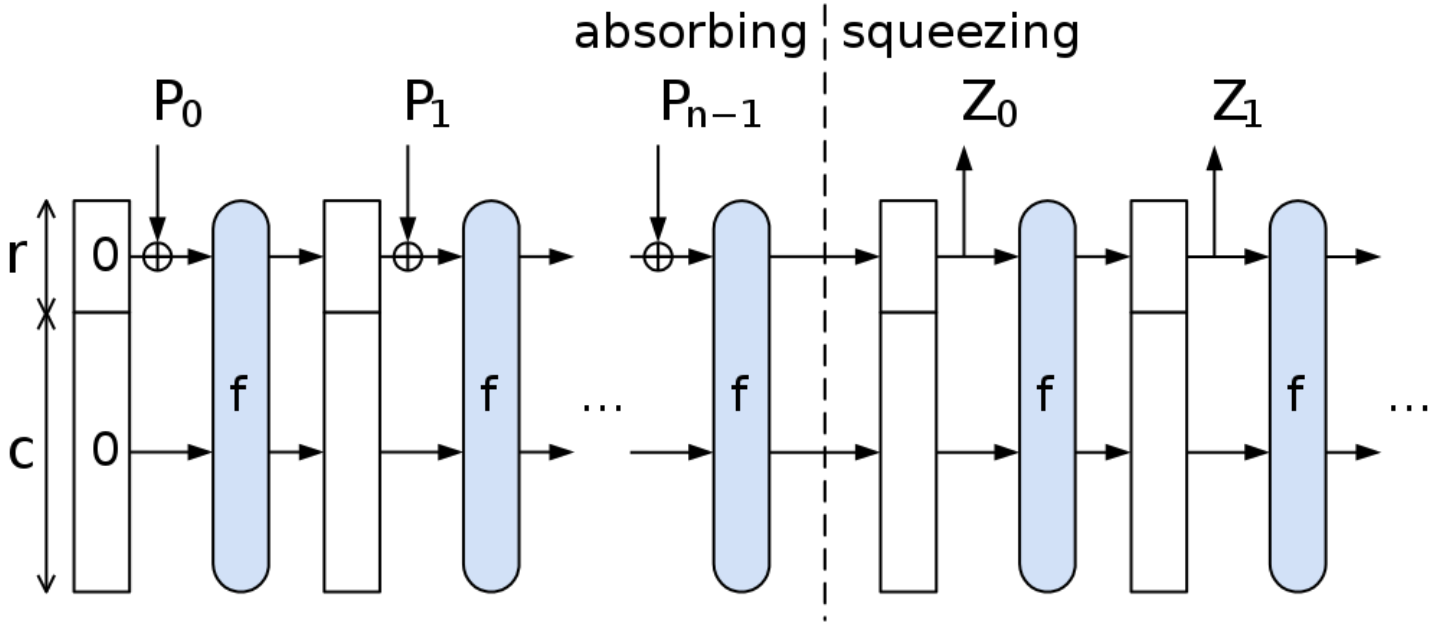


Fase pemerasan (*squeezing*)

- *Message digest* akan disimpan di dalam *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 ke dalam fungsi permutasi *f* menghasilkan *state* baru *S*.

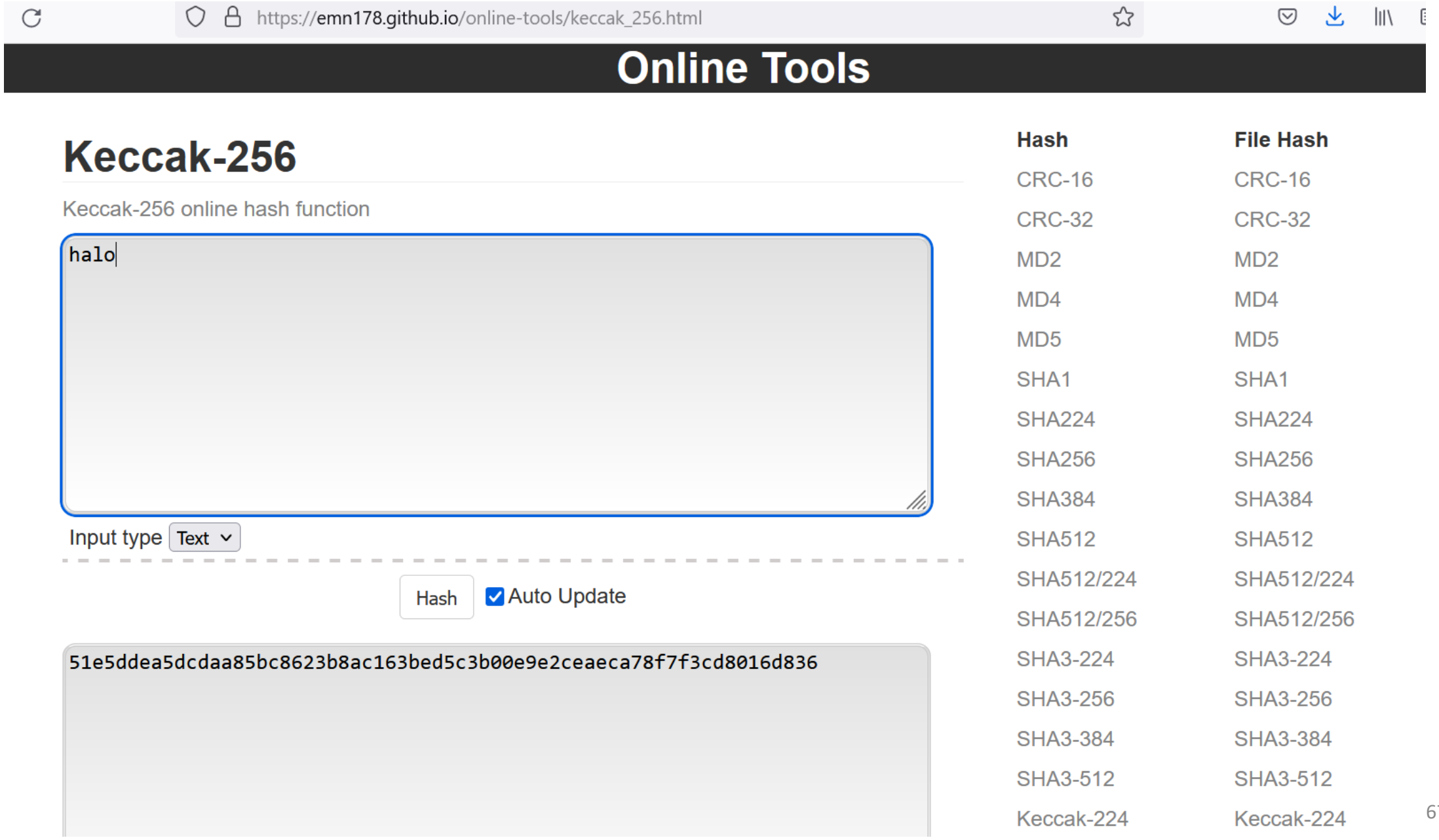


- Perhatikan bahwa *c* bit terakhir dari *state* tidak pernah terpengaruh secara langsung oleh blok masukan dan tidak pernah mengeluarkan luaran selama fase pemerasan. Hal ini untuk mencegah terjadinya kolisi.



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

Demo online: https://emn178.github.io/online-tools/keccak_256.html



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

SELAMAT BELAJAR