Bahan Kuliah ke-19

IF5054 Kriptografi

Digital Signature Standard (*DSS*)

Disusun oleh:

Ir. Rinaldi Munir, M.T.

Departemen Teknik Informatika
Institut Teknologi Bandung
2004

19. *Digital Signature Standard* (*DSS*)

19.1 Pendahuluan

- Pada bulan Agustus 1991, NIST (*The National Institute of Standard and Technology*) mengumumkan bakuan (standard) untuk tanda-tangan digital yang dinamakan *Digital Signature Standard* (DSS).

- *DSS* terdiri dari dua komponen:
  1. Algoritma tanda-tangan digital yang disebut *Digital Signature Algorithm* (*DSA*).
  2. Fungsi *hash* standard yang disebut *Secure Hash Algorithm* (*SHA*).

- *DSS* adalah standard, sedangkan *DSA* adalah algoritma. Standard tersebut menggunakan algoritma *DSA* untuk penandatanganan pesan dan *SHA* untuk membangkitkan message digest dari pesan.

19.2 *Digital Standard Algorithm* (*DSA*)

- *DSA* termasuk ke dalam algoritma kriptografi kunci-publik. *DSA* tidak dapat digunakan untuk enkripsi; *DSA* dispesifikasikan khusus untuk tanda-tanagn digital. *DSA* mempunyai dua fungsi utama:
  1. Pembentukan tanda-tangan (*signature generation*), dan
  2. Pemeriksaan keabsahan tanda-tangan (*signature verification*).

- Sebagaimana halnya pada algoritma kriptografi kunci-publik, *DSA* menggunakan dua buah kunci, yaitu kunci publik dan kunci privat. Pembentukan tanda-tangan menggunakan kunci rahasia privat, sedangkan verifikasi tanda-tangan menggunakan kunci publik pengirim.

- DSA menggunakan fungsi *hash SHA* (*Secure Hash Algorithm*) untuk mengubah pesan menjadi *message digest* yang berukuran 160 bit (SHA akan dijelaskan pada upa-bab 19.3).

19.3.1 Parameter DSA

- *DSA* dikembangkan dari algoritma ElGamal. *DSA* mempunyai properti berupa parameter sebagai berikut:

  1. *p*, adalah bilangan prima dengan panjang *L* bit, yang dalam hal ini $512 \le L \le 1024$ dan *L* harus kelipatan 64. Parameter *p* bersifat publik dan dapat digunakan bersama-sama oleh orang di dalam kelompok.
  2. *q*, bilangan prima 160 bit, merupakan faktor dari $p - 1$. Dengan kata lain, $(p - 1) \bmod q = 0$. Parameter *q* berisfat publik.
  3. $g = h^{(p-1)/q} \bmod p$, yang dalam hal ini $h < p - 1$ sedemikian sehingga $h^{(p-1)/q} \bmod p > 1$. Parameter *g* bersifat publik.
  4. *x*, adalah bilangan bulat kurang dari *q*. Parameter *x* adalah kunci privat.
  5. $y = g^x \bmod p$, adalah kunci publik.
  6. *m*, pesan yang akan diberi tanda-tangan.

19.3.2 Prosedur Pembangkitan Sepasang Kunci

1. Pilih bilangan prima p dan q, yang dalam hal ini $(p - 1) \bmod q = 0$.
2. Hitung $g = h^{(p-1)/q} \bmod p$, yang dalam hal ini $1 < h < p - 1$ dan $h^{(p-1)/q} \bmod p > 1$.
3. Tentukan kunci privat x, yang dalam hal ini $x < q$.
4. Hitung kunci publik $y = g^x \bmod p$.

Jadi, prosedur di atas menghasilkan:

kunci publik dinyatakan sebagai (p, q, g, y);
kunci privat dinyatakan sebagai (p, q, g, x).

19.3.3 Prosedur Pembangkitan Tanda-tangan (*Signing*)

1. Ubah pesan m menjadi *message digest* dengan fungsi *hash* SHA, H.
2. Tentukan bilangan acak $k < q$.
3. Tanda-tangan dari pesan m adalah bilangan r dan s. Hitung r dan s sebagai berikut:

   $r = (g^k \bmod p) \bmod q$

   $s = (k^{-1} (H(m) + x * r)) \bmod q$
4. Kirim pesan m beserta tanda-tangan r dan s.

19.3.4 Prosedur Verifikasi Keabsahan Tanda-tangan (*Verifying*)

1. Hitung
   $w = s^{-1} \bmod q$
   $u_1 = (H(m) * w) \bmod q$
   $u_2 = (r * w) \bmod q$
   $v = ((g^{u1} * y^{u2}) \bmod p) \bmod q)$
2. Jika $v = r$, maka tanda-tangan sah, yang berarti bahwa pesan masih asli dan dikirim oleh pengirim yang benar.

19.3.5 Contoh Perhitungan *DSA*

a. Prosedur Pembangkitan Sepasang Kunci

1. Pilih bilangan prima p dan q, yang dalam hal ini $(p - 1) \bmod q = 0$.
   $p = 59419$
   $q = 3301$ (memenuhi $3301 * 18 = 59419 - 1$)

2. Hitung $g = h^{(p-1)/q} \bmod p$, yang dalam hal ini $1 < h < p - 1$ dan $h^{(p-1)/q} \bmod p > 1$.
   $g = 18870$ (dengan h = 100)

3. Tentukan kunci rahasia x, yang dalam hal ini $x < q$.
   $x = 3223$

4. Hitung kunci publik $y = g^x \bmod p$.
   $y = 29245$

b. Prosedur Pembangkitan Tanda-tangan (Signing)

1. Hitung nilai *hash* dari pesan, misalkan $H(m) = 4321$
2. Tentukan bilangan acak $k < q$.

   $k = 997$

   $k^{-1} = 2907 \pmod{3301}$
3. Hitung r dan s sebagai berikut:

   $r = (g^k \bmod p) \bmod q = 848$

   $s = (k^{-1} (H(m) + x * r)) \bmod q$

   $= 7957694475 \bmod 3301 = 183$
4. Kirim pesan m dan tanda-tangan r dan s.

c. Prosedur Verifikasi Keabsahan Tanda-tangan

1. Hitung

   $s^{-1} = 469 \pmod{3301}$

   $w = s^{-1} \bmod q = 469$

   $u_1 = (H(m) * w) \bmod q \; 2026549 \bmod 3301 = 3036$

   $u_2 = (r * w) \bmod q = 397712 \bmod 3301 = 1592$

   $v = ((g^{u1} * y^{u2}) \bmod p) \bmod q) = 848 \bmod 3301 = 848$
2. Karena $v = r$, maka tanda-tangan sah.

19.3.6 Implementasi DSA

- Adanya batasan bahwa nilai *p* mempunyai panjang 512 sampai 1024 bit dan *q* 160-bit, menyebabkan *DSA* hampir tidak mungkin diimplementasikan dalam perangkat lunak. Panjang bit yang besar ini dimaksudkan agar upaya untuk memecahkan parameter yang lain sangat sulit dilakukan

- *Compiler C* hanya sanggup menyatakan bilangan bulat hingga 2^{32}. Oleh karena itu, bila *DSA* diimplementasikan dalam perangkat lunak, batasan panjang bit *p* dan *q* diubah hingga maksimum nilai *p* dan *q* adalah 2^{32}.

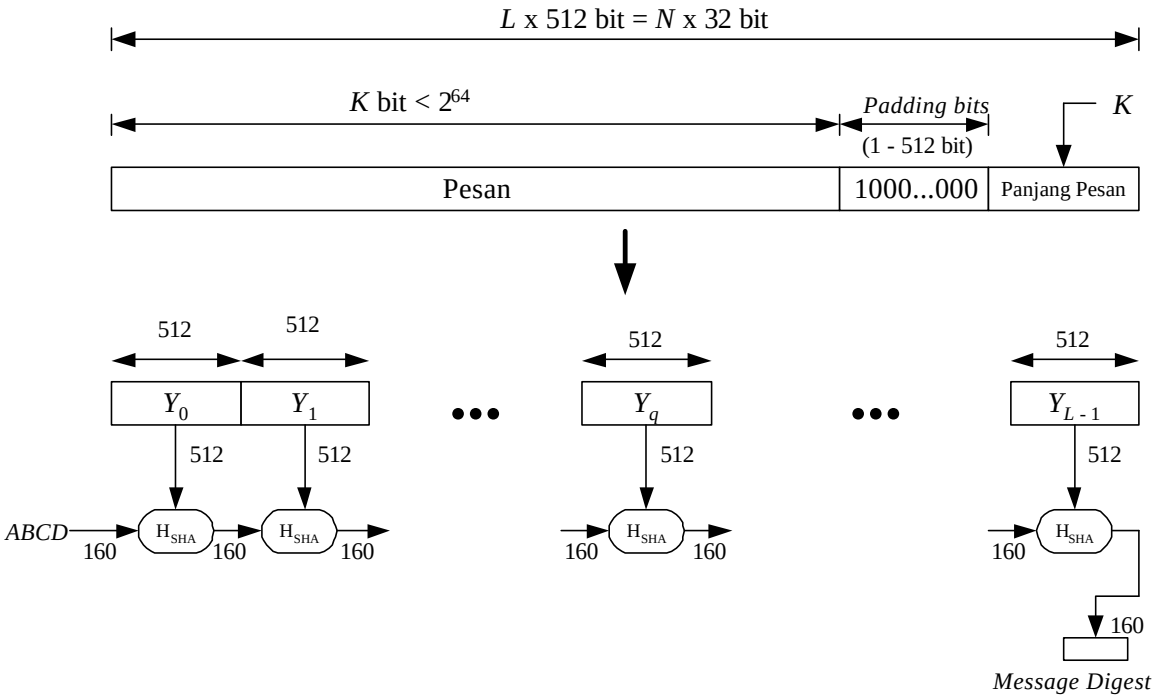
19.4 *Secure Hash Algorithm* (*SHA*)

- *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*.

- *SHA* disebut aman (*secure*) karena ia dirancang sedemikian rupa sehingga secara komputasi tidak mungkin menemukan pesan yang berkoresponden dengan *message digest* yang diberikan.

- Algoritma SHA menerima masukan berupa pesan dengan ukuran maksimum 2^{64} bit (2.147.483.648 *gigabyte*) dan menghasilkan *message digest* yang panjangnya 160 bit, lebih panjang dari *message digest* yang dihasilkan oleh *MD5*

- Gambaran pembuatan *message digest* dengan algoritma *SHA* diperlihatkan pada Gambar 19.1.



Gambar 19.1. Pembuatan *message digest* dengan algoritma SHA

- Langkah-langkah pembuatan *message digest* secara garis besar adalah sebagai berikut:
  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.

19.4.1. Penambahan Bit-bit Pengganjal

- Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Ini berarti panjang pesan setelah ditambahi bit-bit pengganjal adalah 64 bit kurang dari kelipatan 512. Angka 512 ini muncul karena *SHA* memperoses pesan dalam blok-blok yang berukuran 512.

- Pesan dengan panjang 448 bit pun tetap ditambah dengan bit-bit pengganjal. Jika panjang pesan 448 bit, maka pesan tersebut ditambah dengan 512 bit menjadi 960 bit. Jadi, panjang bit-bit pengganjal adalah antara 1 sampai 512.

- Bit-bit pengganjal terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.

19.4.2. Penambahan Nilai Panjang Pesan Semula

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

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

19.4.3. Inisialisai Penyangga *MD*

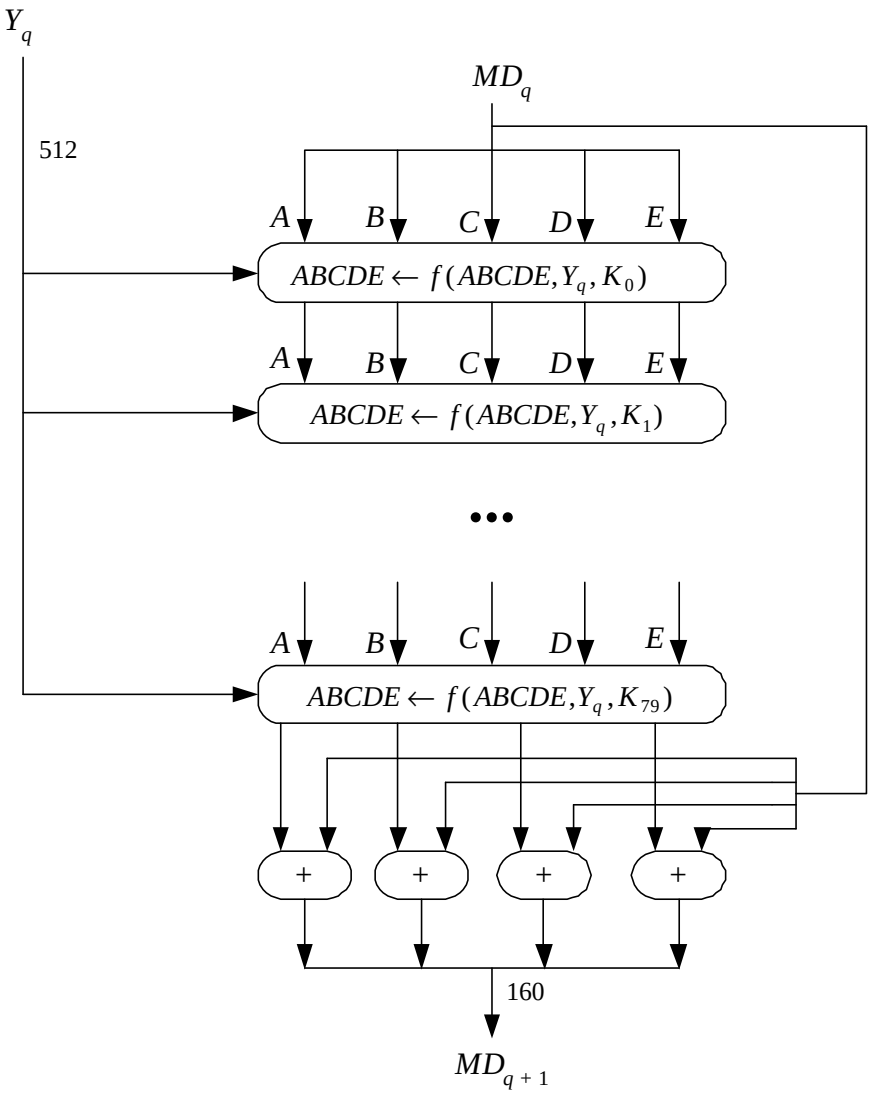
- *SHA* membutuhkan 5 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit (*MD5* hanya mempunyai 4 buah penyangga). Total panjang penyangga adalah $5 \times 32 = 160$ bit. Keempat penyangga ini menampung hasil antara dan hasil akhir.

- 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

19.4.4. 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_{SHA}. Gambaran proses H_{SHA} diperlihatkan pada Gambar 19.2.



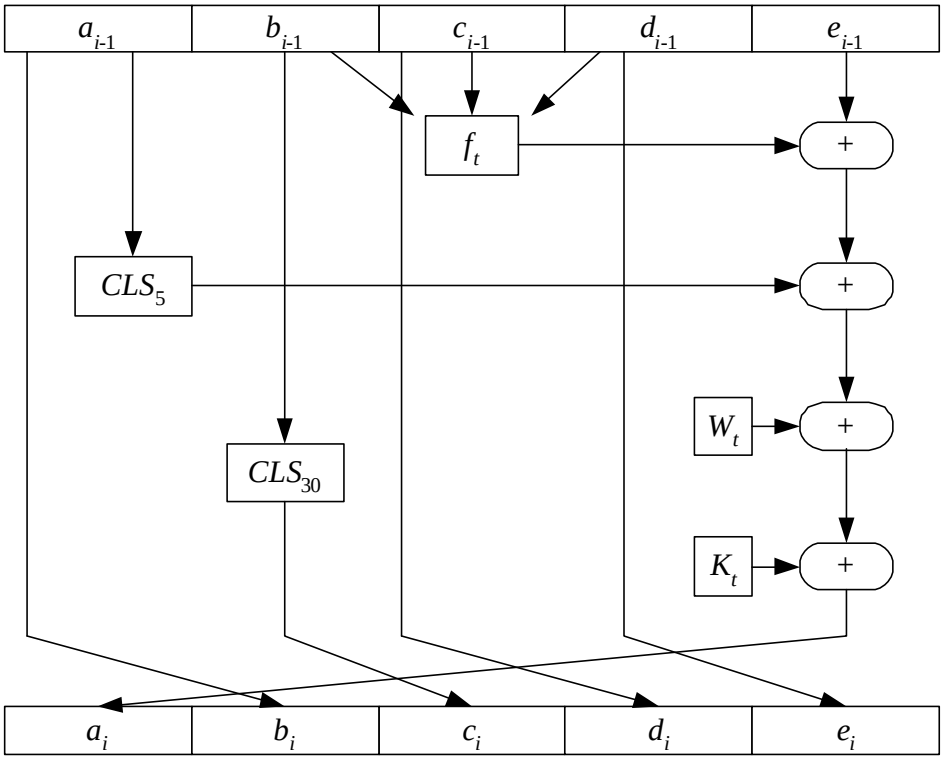
Gambar 19.2. Pengolahan blok 512 bit (Proses H_{SHA})

- Proses H_{SHA} terdiri dari 80 buah putaran (*MD5* hanya 4 putaran), dan masing-masing putaran menggunakan bilangan penambah K_t, yaitu:

Putaran $0 \le t \le 19$	K_t = 5A827999
Putaran $20 \le t \le 39$	K_t = 6ED9EBA1
Putaran $40 \le t \le 59$	K_t = 8F1BBCDC
Putaran $60 \le t \le 79$	K_t = CA62C1D6

- Pada Gambar 19.2, Y_q menyatakan blok 512-bit ke-q dari pesan yang telah ditambah bit-bit pengganjal dan tambahan 64 bit nilai panjang pesan semula. MD_q adalah nilai *message digest* 160-bit dari proses H_{SHA} ke-q. Pada awal proses, MD_q berisi nilai inisialisasi penyangga MD.

- Setiap putaran menggunakan operasi dasar yang sama (dinyatakan sebagai fungsi *f*). Operasi dasar SHA diperlihatkan pada Gambar 19.3.



Gambar 19.3. Operasi dasar *SHA* dalam satu putaran (fungsi *f*)

- Operasi dasar *SHA* yang diperlihatkan pada Gambar 19.3 dapat ditulis dengan persamaan sebagai berikut:

$$a, b, c, d, e \leftarrow (\mathrm{CLS}_5(a) + f_t(b, c, d) + \mathrm{e} + W_t + K_t),\ a, \mathrm{CLS}_{30}(b), c, d$$

yang dalam hal ini,

a, b, c, d, e = lima buah peubah penyangga 32-bit (berisi nilai penyangga *A*, *B*, *C*, *D*, *E*)

t = putaran, $0 \le t \le 79$

f_t = fungsi logika

CLS_s = *circular left shift* sebanyak *s* bit

W_t = *word* 32-bit yang diturunkan dari blok 512 bit yang sedang diproses

K_t = konstanta penambah

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

atau dapat dinyatakan dalam kode program berikut:

```
for t ← 0 to 79 do
    TEMP ← (a <<< 5) + ft(b, c, d) + e +
    Wt + Kt)
    e ← d
    d ← c
    c ← b <<< 30
    b ← a
    a ← TEMP
endfor
```

yang dalam hal ini, <<< menyatakan operasi pergeseran *circular left shift.*

- Fungsi f_t adalah fungsi logika yang melakukan operasi *bitwise*. Operasi *bitwise* yang dilakukan dapat dilihat pada Tabel 1.

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	$b \oplus c \oplus d$

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

- Nilai W_1 sampai W_{16} berasal dari 16 *word* pada blok yang sedang diproses, sedangkan nilai W_t berikutnya didapatkan dari persamaan

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

- Setelah putaran ke-79, *a*, *b*, *c*, *d*, dan *e* ditambahkan ke *A*, *B*, *C*, *D*, dan *E* dan selanjutnya algoritma memproses untuk blok data berikutnya (Y_{q+1}). Keluaran akhir dari algoritma *SHA* adalah hasil penyambungan bit-bit di *A*, *B*, *C*, *D*, dan *E*.