Bahan Kuliah ke-21

IF5054 Kriptografi

Message Authentication Code (*MAC*)
Pembangkit Bilangan Acak Semu

Disusun oleh:

Ir. Rinaldi Munir, M.T.

Departemen Teknik Informatika
Institut Teknologi Bandung
2004

21. *Message Authentication Code* (*MAC*) dan Pembangkit Bilangan Acak Semu

21.1 *MAC*

- *MAC* adalah fungsi *hash* satu-arah yang menggunakan kunci rahasia (*secret key*) dalam pembangkitan nilai hash. Dengan kata lain, nilai hash adalah fungsi dari pesan dan kunci. *MAC* disebut juga *keyed hash function* atau *key-dependent one-way hash function*.

- *MAC* memiliki sifat dan properti yang sama seperti fungsi hash satu-arah yang telah didiskusikan sebelumnya, hanya saja ada tambahan kunci. Kunci digunakan untuk memverifikasi nilai *hash*.

- Secara matematis, *MAC* dinyatakan sebagai

  $$MAC = C_K(M)$$

  yang dalam hal ini,

  MAC = nilai *hash*
  C = fungsi *hash* (atau algoritma *MAC*)
  K = kunci rahasia

  Fungsi *C* memampatkan pesan *M* yang berukuran sembarang dengan menggunakan kunci *K*.

  Fungsi *F* adalah fungsi *many-to-one*. Hal ini berarti potensial beberapa pesan memiliki *MAC* yang sama, tetapi menemukan pesan-pesan semacam itu sangat sulit.

21.2 Aplikasi *MAC*

- *MAC* ditambahkan (*embed*) pada pesan. Selanjutnya, *MAC* digunakan untuk otentikasi tanpa perlu merahasiakan pesan. Catatlah bahwa *MAC* bukanlah tanda-tangan digital.

- Aplikasi *MAC* lainnya:
  - otentikasi arsip yang digunakan oleh dua atau lebih pengguna
  - menjaga integritas (keaslian) isi arsip terhadap perubahan, misalnya karena serangan virus.
    Caranya sbb: hitung *MAC* dari arsip, simpan *MAC* di dalam sebuah tabel.

    Jika pengguna menggunakan fungsi *hash* satu-arah biasa (seperti *MD5*), maka virus dapat menghitung nilai *hash* yang baru dari arsip yang sudah diubah, lalu mengganti nilai *hash* yang lama di dalam tabel.

    Tetapi, jika digunakan *MAC*, virus tidak dapat melakukan hal ini karena ia tidak mengetahui kunci.

21.3 Algoritma *MAC*

(a) Algoritma MAC berbasis cipher blok (block cipher)

- *MAC* dibangkitkan dengan menggunakan algoritma *cipher* blok dengan mode *CBC* atau *CFB*. Nilai *hash*-nya (yang menjadi *MAC*) adalah hasil enkripsi blok terakhir.

- Misalkan *DES* digunakan sebagai *cipher* blok, maka ukuran blok adalah 64 bit, dan kunci rahasia *MAC* adalah kunci DES yang panjangnya 56 bit.

- *Data Authentication Algorithm* (*DAA*) adalah algoritma *MAC* berbasis *DES-CBC* yang digunakan secara luas:
  - *DAA* menggunakan *IV* (*Initialization Vector*) = 0 dan bit-bit *padding* yang seluruhnya 0.
  - *DAA* mengenkripsi pesan dengan menggunakan *DES* dalam mode *CBC*.
  - Hasil enkripsi blok terakhir menjadi *MAC*, atau hanya mengambil *n* bit paling kiri ($16 \le n \le 64$) dari hasil enkripsi blok terakhir sebagai *MAC*.

(b) Algoritma MAC berbasis fungsi hash satu-arah

- Fungsi *hash* satu-arah seperti *MD5* dapat digunakan sebagai *MAC*.

- Caranya adalah sebagai berikut:
  - Misalkan *A* dan *B* akan bertukar pesan. *A* dan *B* berbagi sebuah kunci rahasia *K*.
  - *A* menyambung (*concate*) pesan *M* dengan *K*, lalu menghitung nilai *hash* dari hasil penyambungan itu:

    $$H(M, K)$$

    Nilai *hash* ini adalah *MAC* dari pesan tersebut. *A* lalu mengirim *M* dan *MAC* kepada *B*.

  - *B* dapat melakukan otentikasi terhadap pesan karena ia mengetahui kunci *K*.

21.2 Pembangkit Bilangan Acak Semu

- Bilangan acak (*random*) banyak digunakan di dalam kriptografi, misalnya untuk pembangkitan kunci (contohnya pada algoritma *OTP*), pembangkitan *initialization vector* (*IV*), dan sebagainya.

- Tidak ada komputasi yang benar-benar menghasilkan deret bilangan acak secara sempurna. Bilangan acak yang dihasilkan dengan rumus-rumus matematika adalah bilangan acak semu (*pseudo*), karena pembangkitan bilangannya dapat diulang kembali. Pembangkit deret bilangan acak semacam itu disebut *pseudo-random number generator* (*PRNG*).

Linear Congruential Generator (*LCG*)

- Pembangkit bilangan acak kongruen-lanjar (*linear congruential generator* atau *LCG*) adalah *PRNG* yang berbentuk:

$$X_n = (aX_{n-1} + b) \bmod m$$

yang dalam hal ini,

X_n = bilangan acak ke-n dari deretnya
X_{n-1} = bilangan acak sebelumnya
a = faktor pengali
b = *increment*
m = modulus
(a, b, dan m semuanya konstanta)

Kunci pembangkit adalah X_0 yang disebut **umpan** (*seed*).

LCG mempunyai periode tidak lebih besar dari *m*. Jika *a*, *b*, dan m dipilih secara tepat (misalnya b seharusnya relatif prima terhadap *m*), maka *LCG* akan mempunyai periode maksimal, yaitu $m - 1$.

Contoh: $X_n = (7X_{n-1} + 11) \bmod 17$, dan $X_0 = 0$

n	X_n
0	0
1	11
2	3
3	15
4	14
5	7
6	9
7	6
8	2
9	8
10	16
11	4
12	5
13	12
14	10
15	13
16	*0*
17	*11*
18	*3*
19	*15*
20	*14*
21	*7*
22	*9*
23	*6*
24	*2*

Periodenya hanya 15 (tidak maksimal; jika maksimal maka periodenya adalah 17 – 1 = 16)

- Keunggulan *LCG*: cepat dan hanya membutuhkan sedikit operasi bit.

- Sayangnya, *LCG* tidak dapat digunakan untuk kriptografi karena bilangan acaknya dapat diprediksi urutan kemunculannya.

- *LCG* tetap berguna untuk aplikasi non-kriptografi seperti simulasi, sebab *LCG* mangkus dan memperlihatkan sifat statistik yang bagus dan sangat tepat untuk uji-uji empirik.

- Tabel bebrapa nlai konstanta *a*, *m*, dan *m* yang bagus untuk *LCG*:

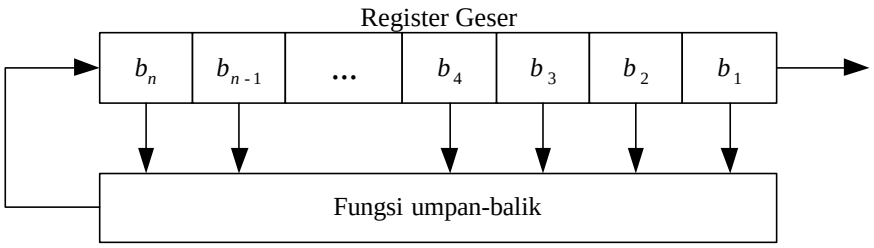
a	*b*	*m*
106	1283	6075
211	1663	7875
421	1663	7875
430	2351	11979
936	1399	6655
1366	1283	6075
171	11213	53125
859	2531	11979
419	6173	29282
967	3041	14406
141	28411	134456
625	6571	31104
1541	2957	14000
1741	2731	12960
1291	4621	21870
205	29573	139968
421	17117	81000
1255	6173	29282
281	28411	134456

Linear Feedback Shift Register (*LFSR*)

- *PRNG* yang digunakan di dalam kriptografi adalah *LFSR*. Konsep register geser (*shift register*) banyak diterapkan pada bidang kriptografi maupun teori pengkodean. Algoritma *cipher* aliran (*stream cipher*), misalnya, didasarkan pada register geser.

- Register geser umpan-balik (*feedback shift register*) atau *FSR* terdiri dari dua bagian (Gambar 21.1):
  1. Register geser
     yaitu barisan bit-bit ($b_n b_{n-1} \ldots b_4 b_3 b_2 b_1$) yang panjangnya n (disebut juga register geser n-bit)

  2. Fungsi umpan-balik
     yaitu fungsi yang menerima masukan dari register geser dan mengembalikan nilai fungsi ke register geser.

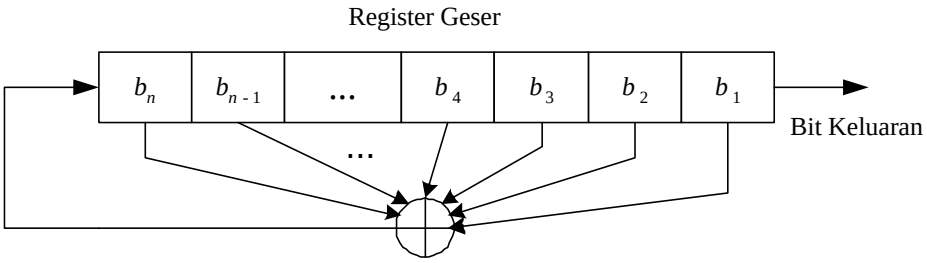


Gambar 21.1 Bagian-bagian *FSR*

- Tiap kali sebuah bit dibutuhkan, semua bit di dalam register digeser 1 bit ke kanan. Bit paling kiri (b_n) dihitung sebagai fungsi bit-bit lain di dalam register tersebut. Keluaran dari register geser adalah 1 bit (yaitu bit b_1 yang tergeser).

  Periode register geser adalah panjang barisan keluaran sebelum ia berulang kembali.

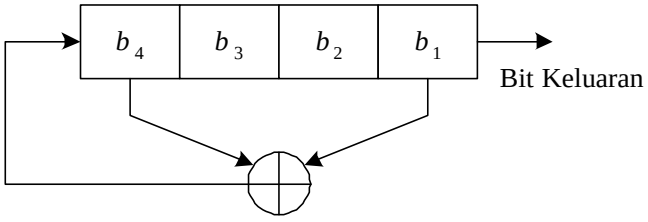
- Contoh register geser umpan-balik (*feedback shift register*) adalah *linear feedback shift register* (*LFSR*) – ditunjukkan pada Gambar 21.2. Fungsi umpan-balik adalah peng-XOR-an bit-bit tertentu di dalam register.



Gambar 21.2 *LFSR* sederhana

- Gambar 21.3 adalah contoh *LFSR* 4-bit, yang dalam hal ini fungsi umpan-balik meng-*XOR*-kan b_4 dengan b_1 dan menyimpan hasilnya di b_4:

$$b_4 = f(b_1, b_4) = b_1 \oplus b_4$$



Gambar 21.3 *LFSR* 4-bit

Jika register diinisialisasi dengan 1111, maka isi register (menyatakan status atau *state*) dan bit keluaran sebelum berulang kembali adalah:

i	Isi Register	Bit Keluaran
0	1 1 1 1	
1	0 1 1 1	1
2	1 0 1 1	1
3	0 1 0 1	1
4	1 0 1 0	1
5	1 1 0 1	0
6	0 1 1 0	1
7	0 0 1 1	0
8	1 0 0 1	1
9	0 1 0 0	1
10	0 0 1 0	0
11	0 0 0 1	0
12	1 0 0 0	1
13	1 1 0 0	0
14	1 1 1 0	0

Barisan bit-bit keluaran (yang merupakan bit-bit acak) adalah:

```
1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 …
```

yang bila dikelompokkan dalam 4-bit menjadi:

```
1111 0101 1001 000…
```

atau dalam notasi HEX:

```
F    5    9 …
```

- *LFSR n*-bit mempunyai $2^n - 1$ status internal (keadaan isi register). Ini berarti, secara teoritis *LFSR* dapat membangkitkan $2^n - 1$ barisan bit acak-semu sebelum perulangan. Jadi, periode (maksimal) *LFSR* adalah $2^n - 1$.

  (Disebutkan $2^n - 1$, bukan 2^n karena isi register `0000` tidak berguna, sebab ia menghasilkan barisan bit 0 yang tidak pernah berakhir)