Bab 9

Cryptographically Secure Hashing

Cryptographically secure hashing adalah proses pembuatan suatu "sidik jari" (*fingerprint* atau kerap juga disebut *digest*) untuk suatu naskah. Sidik jari relatif tidak terlalu besar, antara 128 hingga 512 bit tergantung algoritma yang digunakan, sedangkan besar naskah tidak terbatas. Contoh penggunaan sidik jari adalah untuk *digital signature* (lihat bab 16). Agar proses pembuatan sidik jari aman dari segi kriptografi, kemungkinan *collision*, dimana dua naskah yang berbeda mempunyai sidik jari yang sama, harus sangat kecil. Kriteria yang sudah menjadi standard untuk keamanan algoritma *secure hashing* adalah:

- *Preimage resistance.* Untuk suatu nilai *hash* yang sembarang (tidak diketahui asal-usulnya), sangat sukar untuk mencari naskah yang mempunyai nilai *hash* tersebut.

- *Second preimage resistance.* Untuk suatu naskah m_1, sangat sukar untuk mencari naskah lain m_2 ($m_1 \neq m_2$) yang mempunyai nilai *hash* yang sama ($hash(m_1) = hash(m_2)$). Persyaratan ini kerap disebut juga *weak collision resistance.*

- *Collision resistance.* Sangat sukar untuk mencari dua naskah m_1 dan m_2 yang berbeda ($m_1 \neq m_2$) yang mempunyai nilai *hash* yang sama ($hash(m_1) = hash(m_2)$). Persyaratan ini kerap disebut juga *strong collision resistance.*

Algoritma untuk *secure hashing* biasanya membagi naskah sehingga terdiri dari beberapa blok, setiap blok biasanya 512 atau 1024 bit. Naskah diberi *padding* meskipun besarnya merupakan kelipatan dari besarnya blok dan

padding diahiri dengan *size* dari naskah. Algoritma biasanya terdiri dari dua tahap:

- *preprocessing* dan
- *hashing*.

Tahap *preprocessing* biasanya terdiri dari *padding* dan *parameter setup*. Tahap *hashing* membuat sidik jari dengan mengkompresi naskah yang sudah diberi *padding*. Kompresi dilakukan dengan setiap blok secara berurut diproses dan hasilnya dijadikan *feedback* untuk proses blok berikutnya. Konstruksi seperti ini dinamakan konstruksi Merkle-Damgård (lihat [mer79] bab II), dan digunakan oleh MD5 dan SHA, dua algoritma *secure hashing* yang akan dibahas.

Tentunya keamanan algoritma *secure hashing* sangat tergantung pada proses kompresi yang digunakan. Selain 3 kriteria *resistance* yang telah dibahas, dewasa ini *secure hashing* juga diharapkan memiliki resistensi terhadap *length extension attack*. *Length extension attack* adalah *attack* dimana dari mengetahui $hash(m_1)$ dan panjangnya naskah m_1, tanpa mengetahui m_1, seseorang dapat membuat naskah m_2 dan $hash(m_1 \circ m_2)$, dimana $\circ$ adalah operasi penyambungan naskah (*concatenation*). *Attack* ini digunakan terhadap *authentication* yang menggunakan *hashing* yang lemah. Sementara ini, untuk mengatasi masalah *length extension attack*, mekanisme HMAC (lihat bagian 9.3) kerap digunakan. Ditingkat yang lebih rinci, kriteria keamanan yang digunakan untuk merancang *block cipher* seperti *strict avalanche criterion*, dimana perbedaan input 1 bit menyebabkan setiap bit output mempunyai probabilitas 0.5 untuk berubah independen dari bit lainnya, sebaiknya juga berlaku untuk proses kompresi.

Penggunaan utama *secure hashing* memang untuk *digest*, contohnya untuk *digital signature*. Namun selain untuk keperluan *digest*, *secure hashing* kerap juga digunakan sebagai *pseudorandom function*. Salah satu contoh aplikasi *pseudorandom function* adalah untuk *key derivation*, seperti yang dilakukan oleh protokol Kerberos (lihat bagian 22.1) dan juga protokol IKE dalam IPsec (lihat bagian 20.3).

Saat bab ini ditulis, NIST sedang mengadakan sayembara untuk pembuatan standard SHA-3. Salah satu sebab NIST merasa standard SHA baru perlu dibuat adalah ditemukannya *collision* untuk MD5 oleh Xiaoyun Wang dan koleganya (lihat [wan05] dan juga bagian 9.1). Waktu komputer yang dibutuhkan untuk membuat *collision* tidak terlalu lama, jadi MD5 tidak memenuhi kriteria diatas. Xiaoyun Wang dan koleganya juga berhasil menemukan kelemahan pada SHA-1. NIST mengharapkan bahwa SHA-3 akan memenuhi 4 kriteria *resistance* yang telah dibahas, ditambah dengan persyaratan bahwa jika hanya sebagian dari semua bit digunakan (tentunya banyaknya bit yang digunakan harus cukup besar), maka algoritma tetap memenuhi keempat kriteria.

9.1 MD5

Algoritma *secure hashing* MD5 dirancang oleh Ron Rivest dan penggunaannya sangat populer dikalangan komunitas *open source* sebagai *checksum* untuk *file* yang dapat di *download*. MD5 juga kerap digunakan untuk menyimpan password dan juga digunakan dalam *digital signature* dan *certificate*.

Spesifikasi lengkap untuk algoritma MD5 ada pada suatu RFC (*request for comment*, lihat [riv92]). Besarnya blok untuk MD5 adalah 512 bit sedangkan *digest size* adalah 128 bit. Karena *word size* ditentukan sebesar 32 bit, satu blok terdiri dari 16 *word* sedangkan *digest* terdiri dari 4 *word*. Indeks untuk bit dimulai dari 0. *Preprocessing* dimulai dengan *padding* sebagai berikut:

1. Bit dengan nilai 1 ditambahkan setelah ahir naskah.

2. Deretan bit dengan nilai 0 ditambahkan setelah itu sehingga besar dari naskah mencapai nilai 448 (mod 512) (sedikitnya 0 dan sebanyaknya 511 bit dengan nilai 0 ditambahkan sehingga tersisa 64 bit untuk diisi agar besar naskah menjadi kelipatan 512).

3. 64 bit yang tersisa diisi dengan besar naskah asli dalam bit. Jika besar naskah asli lebih dari 2^{64} bit maka hanya 64 *lower order* bit yang dimasukkan. *Lower order word* untuk besar naskah asli dimasukkan sebelum *high-order word*.

Setelah *padding*, naskah terdiri dari n *word* $M[0 \ldots n-1]$ dimana n adalah kelipatan 16. Langkah berikutnya dalam *preprocessing* adalah menyiapkan MD *buffer* sebesar 4 word:

$$(A, B, C, D)$$

dimana A merupakan *lower order word*. *Buffer* diberi nilai awal sebagai berikut (nilai dalam hexadecimal dimulai dengan *lower order byte*).

A:	01	23	45	67
B:	89	*ab*	*cd*	*ef*
C:	*fe*	*dc*	*ba*	98
D:	76	54	32	10.

Proses *hashing* dilakukan per blok, dengan setiap blok melalui 4 putaran. Proses *hashing* menggunakan 4 fungsi $F, G, H,$ dan I yang masing-masing mempunyai input 3 *word* dan output 1 *word*:

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ G(X,Y,Z) &= (X \wedge Z) \vee (Y \wedge \neg Z) \\ H(X,Y,Z) &= X \oplus Y \oplus Z \\ I(X,Y,Z) &= Y \oplus (X \vee \neg Z) \end{aligned}$$

dimana $\wedge$ adalah *bitwise and*, $\vee$ adalah *bitwise or*, $\oplus$ adalah *bitwise exclusive or*, dan $\neg$ adalah *bitwise not* (*one's complement*). Selain keempat fungsi diatas, proses *hashing* juga menggunakan tabel dengan 64 *word*, $T[1]$ sampai dengan $T[64]$, yang berada dalam tabel D.1 di appendix D.

Secara garis besar, algoritma untuk *hashing* untuk satu blok adalah sebagai berikut (menggunakan array 16 *word* $X[0]$ sampai dengan $X[15]$ yang dapat menyimpan satu blok):

1. *Copy* satu blok (*word* $M[16i]$ sampai dengan $M[16i+15]$) ke $X[0]$ sampai dengan $X[15]$.
2. Simpan A, B, C, D dalam A', B', C', D'.
3. Lakukan putaran 1 pada A, B, C, D.
4. Lakukan putaran 2 pada A, B, C, D.
5. Lakukan putaran 3 pada A, B, C, D.
6. Lakukan putaran 4 pada A, B, C, D.
7. Tambahkan nilai simpanan pada A, B, C, D:

$$\begin{aligned} A &\leftarrow (A + A') \bmod 2^{32}, \\ B &\leftarrow (B + B') \bmod 2^{32}, \\ C &\leftarrow (C + C') \bmod 2^{32}, \\ D &\leftarrow (D + D') \bmod 2^{32}. \end{aligned}$$

Algoritma diatas diulang hingga semua blok dalam M terproses (dimulai dengan $i = 0$ sampai dengan $i = n/16 - 1$).

Sekarang kita jelaskan putaran 1 sampai dengan 4 yang dilakukan pada A, B, C, D. Setiap putaran menggunakan operasi berbeda. Untuk putaran 1, operasi $P_1(A, B, C, D, k, s, i)$ didefinisikan sebagai berikut:

$$A \leftarrow B + ((A + F(B, C, D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi X kekiri s bit. Putaran 1 terdiri dari:

$$\begin{aligned} P_1(A, B, C, D, 0, 7, 1), \quad & P_1(D, A, B, C, 1, 12, 2), \\ P_1(C, D, A, B, 2, 17, 3), \quad & P_1(B, C, D, A, 3, 22, 4), \\ P_1(A, B, C, D, 4, 7, 5), \quad & P_1(D, A, B, C, 5, 12, 6), \\ P_1(C, D, A, B, 6, 17, 7), \quad & P_1(B, C, D, A, 7, 22, 8), \\ P_1(A, B, C, D, 8, 7, 9), \quad & P_1(D, A, B, C, 9, 12, 10), \\ P_1(C, D, A, B, 10, 17, 11), \quad & P_1(B, C, D, A, 11, 22, 12), \\ P_1(A, B, C, D, 12, 7, 13), \quad & P_1(D, A, B, C, 13, 12, 14), \\ P_1(C, D, A, B, 14, 17, 15), \quad & P_1(B, C, D, A, 15, 22, 16). \end{aligned}$$

Untuk putaran 2, operasi $P_2(A, B, C, D, k, s, i)$ didefinisikan sebagai berikut:

$$A \leftarrow B + ((A + G(B, C, D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi X kekiri s bit. Putaran 2 terdiri dari:

$$\begin{array}{rr}
P_2(A, B, C, D, 1, 5, 17), & P_2(D, A, B, C, 6, 9, 18), \\
P_2(C, D, A, B, 11, 14, 19), & P_2(B, C, D, A, 0, 20, 20), \\
P_2(A, B, C, D, 5, 5, 21), & P_2(D, A, B, C, 10, 9, 22), \\
P_2(C, D, A, B, 15, 14, 23), & P_2(B, C, D, A, 4, 20, 24), \\
P_2(A, B, C, D, 9, 5, 25), & P_2(D, A, B, C, 14, 9, 26), \\
P_2(C, D, A, B, 3, 14, 27), & P_2(B, C, D, A, 8, 20, 28), \\
P_2(A, B, C, D, 13, 5, 29), & P_2(D, A, B, C, 2, 9, 30), \\
P_2(C, D, A, B, 7, 14, 31), & P_2(B, C, D, A, 12, 20, 32).
\end{array}$$

Untuk putaran 3, operasi $P_3(A, B, C, D, k, s, i)$ didefinisikan sebagai berikut:

$$A \leftarrow B + ((A + H(B, C, D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi X kekiri s bit. Putaran 3 terdiri dari:

$$\begin{array}{rr}
P_3(A, B, C, D, 5, 4, 33), & P_3(D, A, B, C, 8, 11, 34), \\
P_3(C, D, A, B, 11, 16, 35), & P_3(B, C, D, A, 14, 23, 36), \\
P_3(A, B, C, D, 1, 4, 37), & P_3(D, A, B, C, 4, 11, 38), \\
P_3(C, D, A, B, 7, 16, 39), & P_3(B, C, D, A, 10, 23, 40), \\
P_3(A, B, C, D, 13, 4, 41), & P_3(D, A, B, C, 0, 11, 42), \\
P_3(C, D, A, B, 3, 16, 43), & P_3(B, C, D, A, 6, 23, 44), \\
P_3(A, B, C, D, 9, 4, 45), & P_3(D, A, B, C, 12, 11, 46), \\
P_3(C, D, A, B, 15, 16, 47), & P_3(B, C, D, A, 2, 23, 48).
\end{array}$$

Untuk putaran 4, operasi $P_4(A, B, C, D, k, s, i)$ didefinisikan sebagai berikut:

$$A \leftarrow B + ((A + I(B, C, D) + X[k] + T[i]) <<< s)$$

dimana $X <<< s$ adalah rotasi X kekiri s bit. Putaran 4 terdiri dari:

$$\begin{array}{rr}
P_4(A, B, C, D, 0, 6, 49), & P_4(D, A, B, C, 7, 10, 50), \\
P_4(C, D, A, B, 14, 15, 51), & P_4(B, C, D, A, 5, 21, 52), \\
P_4(A, B, C, D, 12, 6, 53), & P_4(D, A, B, C, 3, 10, 54), \\
P_4(C, D, A, B, 10, 15, 55), & P_4(B, C, D, A, 1, 21, 56), \\
P_4(A, B, C, D, 8, 6, 57), & P_4(D, A, B, C, 15, 10, 58), \\
P_4(C, D, A, B, 6, 15, 59), & P_4(B, C, D, A, 13, 21, 60), \\
P_4(A, B, C, D, 4, 6, 61), & P_4(D, A, B, C, 11, 10, 62), \\
P_4(C, D, A, B, 2, 15, 63), & P_4(B, C, D, A, 9, 21, 64).
\end{array}$$

Setelah semua blok diproses, maka hasil ahir A, B, C, D menjadi MD5 *digest* dari naskah asli. Urutan byte untuk *digest* dimulai dengan *lower order byte* dari A dan diahiri oleh *higher order byte* dari D.

Beberapa peneliti telah berhasil membuat *collision* untuk MD5. Xiaoyun Wang dan koleganya berhasil membuat *collision* untuk sepasang naskah yang masing-masing terdiri dari 2 blok (lihat [wan05]). Kita akan bahas esensi dari metode yang digunakan untuk membuat *collision* tersebut.

Wang menggunakan teknik *differential cryptanalysis* dengan dua macam perbedaan:

- perbedaan bit (*exclusive or*) dan
- selisih (menggunakan pengurangan).

Jika *differential cryptanalysis* terhadap DES berfokus pada perbedaan bit (lihat bagian 8.1), metode Wang menggunakan kombinasi selisih dan perbedaan bit, tetapi lebih berfokus pada selisih. Kombinasi ini menghasilkan perbedaan yang lebih rinci. Sebagai contoh, jika dua *word* X dan X', yang masing-masing besarnya 32 bit, mempunyai selisih $X' - X = 2^6$, perbedaan bit antara X dan X' bisa berupa

- perbedaan 1 bit ($X'_7 = 1$ dan $X_7 = 0$), atau
- perbedaan 2 bit ($X'_{8-7} = 10$ dan $X_{8-7} = 01$), atau
- perbedaan 3 bit ($X'_{9-7} = 100$ dan $X_{9-7} = 011$),
- dan seterusnya.

Differential dari X dan X' didefinisikan sebagai

$$\Delta X = X' - X.$$

Jika naskah $M = (M_0, M_1, \ldots, M_{k-1})$ dan naskah $M' = (M'_0, M'_1, \ldots, M'_{k-1})$, maka *full differential* dari fungsi hash H untuk M dan M' adalah

$$\Delta H_0 \overset{(M_0, M'_0)}{\longrightarrow} \Delta H_1 \overset{(M_1, M'_1)}{\longrightarrow} \ldots \Delta H_{k-1} \overset{(M_{k-1}, M'_{k-1})}{\longrightarrow} \Delta H,$$

dimana ΔH_0 adalah perbedaan awal (jadi sama dengan 0). ΔH adalah perbedaan output antara M dan M', dan $\Delta H_i = \Delta IV_i$ adalah perbedaan output untuk tahap i yang juga merupakan nilai awal untuk tahap berikutnya. Cukup jelas bahwa jika $\Delta H = 0$, maka kita dapatkan *collision*. *Differential* yang mengakibatkan *collision* disebut *collision differential*. Untuk lebih rinci, *differential* tahap i, $\Delta H_i \overset{(M_i, M'_i)}{\longrightarrow} \Delta H_{i+1}$, dapat diuraikan menjadi

$$\Delta H_i \xrightarrow{P_1} \Delta R_{i+1,1} \xrightarrow{P_2} \Delta R_{i+1,2} \xrightarrow{P_3} \Delta R_{i+1,3} \xrightarrow{P_4} \Delta R_{i+1,4} = \Delta H_{i+1},$$

dimana $\Delta R_{i+1,j}$ adalah perbedaan output untuk putaran j. Untuk lebih rinci lagi, *differential* putaran j, $\Delta R_{j-1} \xrightarrow{P_j} \Delta R_j$ dengan probabilitas P_j, dimana $j = 1, 2, 3, 4$, dapat diuraikan menjadi

$$\Delta R_{j-1} \xrightarrow{P_{j1}} \Delta X_1 \xrightarrow{P_{j2}} \ldots \xrightarrow{P_{j16}} \Delta X_{16} = \Delta R_j,$$

dimana $\Delta_{t-1} \xrightarrow{P_{jt}} \Delta X_t$ merupakan *differential characteristic* untuk langkah t dalam putaran j, $t = 1, 2, \ldots, 16$. Jika P adalah probabilitas untuk *differential* $\Delta H_i \xrightarrow{(M_i, M_i')} \Delta H_{i+1}$, maka

$$P \geq \prod_{i=1}^{4} P_i \text{ dan } \forall j \in \{1, 2, 3, 4\} : P_j \geq \prod_{t=1}^{16} P_{jt}.$$

Metode Wang menggunakan *collision differential* dua tahap

$$\Delta H_0 \xrightarrow{(M_0, M_0')} \Delta H_1 \xrightarrow{(M_1, M_1')} \Delta H,$$

dimana

$$\begin{aligned}
\Delta M_0 = M_0' - M_0 &= (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, 2^{15}, 0, 0, 2^{31}, 0), \\
\Delta M_1 = M_1' - M_1 &= (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, -2^{15}, 0, 0, 2^{31}, 0), \\
\Delta H_1 &= (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25}).
\end{aligned}$$

Nilai awal buffer, yaitu (a_0, b_0, c_0, d_0), digunakan untuk IV_0, dimana

$$\begin{aligned}
a_0 &= 0x67452301, \\
b_0 &= 0xefcdab89, \\
c_0 &= 0x98badcfe, \\
d_0 &= 0x10325476.
\end{aligned}$$

Untuk meningkatkan probabilitas *differential characteristic* dalam penggunaannya, naskah yang dibuat secara acak dapat dimodifikasi. Untuk itu Wang pertama membuat persyaratan yang cukup (*sufficient condition*) agar berbagai *characteristic* dijamin berlaku dengan probabilitas 1. Kemudian teknik untuk melakukan modifikasi naskah agar memenuhi berbagai persyaratan dibuat.

Untuk membuat persyaratan yang cukup agar suatu *characteristic* dijamin berlaku dengan probabilitas 1, diperlukan *data flow analysis*. Sebagai contoh, *differential characteristic* yang digunakan Wang untuk langkah 8 pada putaran pertama tahap pertama adalah

$$(\Delta a_2, \Delta b_1, \Delta c_2, \Delta d_2) \longrightarrow (\Delta a_2, \Delta b_2, \Delta c_2, \Delta d_2),$$

dimana perbedaan variabel mematuhi persamaan-persamaan sebagai berikut:

$$\begin{aligned} b'_1 &= b_1, \\ a'_2 &= a_2[7, \ldots, 22, -23], \\ d'_2 &= d_2[-7, 24, 32], \\ c'_2 &= c_2[7, 8, 9, 10, -12, -24, -25, -26, 27, 28, 29, 30, 31, 32, 1, 2, 3, 4, 5, -6], \\ b'_2 &= b_2[1, 16, -17, 18, 19, 20, -21, -24]. \end{aligned}$$

Sedikit penjelasan mengenai notasi diatas, $a'_2 = a_2[7, \ldots, 22, -23]$ berarti bit 7 sampai dengan 22 dari a_2 adalah 0 sedangkan bit 7 sampai dengan 22 dari a'_2 adalah 1, dan bit 23 dari a_2 adalah 1 sedangkan bit 23 dari a'_2 adalah 0. Bit yang tidak disebut, nilainya sama pada a'_2 dan a_2. Efek dari langkah ke 8 adalah sebagai berikut;

$$\begin{aligned} b_2 &= c_2 + ((b_1 + F(c_2, d_2, a_2) + m_7 + T[8]) <<< 22), \\ b'_2 &= c'_2 + ((b_1 + F(c'_2, d'_2, a'_2) + m'_7 + T[8]) <<< 22), \end{aligned}$$

dimana m_7 dan m'_7 adalah *word* ke 8 untuk masing-masing blok. Kita gunakan notasi ϕ_7:

$$\phi_7 = F(c_2, d_2, a_2) = (c_2 \wedge d_2) \vee (\neg c_2 \wedge a_2).$$

Pencarian persyaratan didasarkan pada fakta bahwa $\Delta b_1 = 0$ dan $\Delta m_7 = 0$ (berdasarkan *differential characteristic* ΔM_0 dan ΔM_1, hanya Δm_4, Δm_{11} dan Δm_{14} yang tidak sama dengan nol). Jadi

$$\Delta b_2 = \Delta c_2 + (\Delta \phi_7 <<< 22).$$

Persyaratan untuk bit pertama dari $\Delta b2$ ($\Delta b_{2,1}$) adalah $d_{2,11} = \overline{a_{2,11}} = 1$ dan $b_{2,1} = 0$ dikarenakan

1. Jika $d_{2,11} = \overline{a_{2,11}} = 1$, maka $\Delta \phi_{7,11} = 1$.
2. $\Delta \phi_{7,11} = (\Delta \phi_7 <<< 22)_{11}$.
3. Karena $\Delta c_{2,1} = 0$, maka $\Delta b_{2,1} = 0 + 1 = 1$.

Persyaratan untuk semua bit Δb_2, baik yang 0 maupun 1 dirumuskan oleh Wang. Wang merumuskan semua persyaratan agar

$$\Delta H_0 \xrightarrow{1} \Delta R_{1,1}$$

dan

$$\Delta H_1 \xrightarrow{1} \Delta R_{2,1},$$

dengan kata lain *characteristic* untuk 1 putaran MD5 dijamin dengan probabilitas 1.

Agar persyaratan *characteristic* dapat dipenuhi, Wang melakukan modifikasi terhadap naskah yang dibuat secara acak. Sebagai contoh, salah satu persyaratan agar *characteristic* 1 putaran terjamin adalah untuk c_1 yaitu

$$c_{1,7} = 0, c_{1,12} = 0, c_{1,20} = 0.$$

Untuk memenuhi persyaratan tersebut, m_2 dimodifikasi sebagai berikut:

$$\begin{aligned} c_1^{new} &\leftarrow c_1^{old} - c_{1,7}^{old} - c_{1,12}^{old} \cdot 2^{11} - c_{1,20}^{old} \cdot 2^{19}, \\ m_2^{new} &\leftarrow ((c_1^{new} - c_1^{old}) >>> 17) + m_2^{old}, \end{aligned}$$

dimana m_2^{old} adalah nilai m_2 sebelum modifikasi, m_2^{new} adalah nilai m_2 setelah modifikasi, c_1^{old} adalah nilai c_1 sebelum modifikasi, dan c_1^{new} adalah nilai c_1 setelah modifikasi.

Modifikasi naskah dilakukan agar *characteristic* putaran pertama dijamin mempunyai probabilitas 1, baik untuk tahap 1 maupun tahap 2. Modifikasi naskah juga dilakukan agar sebagian dari persyaratan untuk putaran kedua terpenuhi. Dengan melakukan berbagai modifikasi tersebut, probabilitas *characteristic* tahap pertama ditingkatkan menjadi 2^{-37} sedangkan probabilitas *characteristic* tahap kedua ditingkatkan menjadi 2^{-30}. Menggunakan metode ini, Wang berhasil menemukan *collision* untuk MD5 dalam waktu kurang dari 1 jam dengan komputer.

Bersama Arjen Lenstra dan Benne de Weger, Xiaoyun Wang berhasil membuat *collision* untuk sepasang X.509 *certificate* dimana bagian *certificate* yang ditanda-tangan berbeda tetapi mempunyai MD5 *digest* yang sama [len05]. Sebagai konsekuensinya, kebenaran dari suatu *certificate* dapat diragukan karena bagian yang ditanda-tangan dapat ditukar dengan nilai lain (termasuk kunci publik lain) tanpa perubahan pada nilai *digital signature*. Jadi sebaiknya *digital signature* menggunakan algoritma *secure hashing* lain seperti SHA-1[1].

9.2 SHA

Algoritma *secure hashing* SHA dirancang oleh National Security Agency (NSA) dan dijadikan standard FIPS (lihat [sha02]). Ada 4 varian SHA dalam standard FIPS-180-2 dengan parameter yang berbeda (lihat tabel 9.1).

Keamanan algoritma didasarkan pada fakta bahwa *birthday attack* pada *digest* sebesar n bit menghasilkan *collision* dengan faktor kerja sekitar $2^{n/2}$. Kita hanya akan membahas SHA-1 disini karena SHA-256, SHA-384 dan SHA-512 algoritmanya mirip dengan SHA-1 dan dijelaskan oleh [sha02]. SHA-1

[1] X.509 memperbolehkan kita untuk memilih algoritma SHA-1 untuk *secure hashing*.

Algoritma	Naskah (bit)	Blok (bit)	Word (bit)	Digest (bit)	Keamanan (bit)
SHA-1	$< 2^{64}$	512	32	160	80
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1024	64	384	192
SHA-512	$< 2^{128}$	1024	64	512	256

Tabel 9.1: 4 Varian SHA

menggunakan fungsi f_t, dimana $0 \leq t < 79$, dengan input 3 *word* masing-masing sebesar 32 bit dan menghasilkan output 1 *word*:

$$f_t = \begin{cases} Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) & 0 \leq t \leq 19 \\ Parity(x,y,z) = x \oplus y \oplus z & 20 \leq t \leq 39 \\ Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & 40 \leq t \leq 59 \\ Parity(x,y,z) = x \oplus y \oplus z & 60 \leq t \leq 79. \end{cases}$$

SHA-1 menggunakan konstan k_t sebagai berikut (dengan nilai dalam hexadecimal):

$$k_t = \begin{cases} 5a827999 & 0 \leq t \leq 19 \\ 6ed9eba1 & 20 \leq t \leq 39 \\ 8f1bbcdc & 40 \leq t \leq 59 \\ ca62c1d6 & 60 \leq t \leq 79. \end{cases}$$

Seperti halnya dengan MD5, SHA-1 terdiri dari dua tahap yaitu *preprocessing* dan *hashing*. *Preprocessing* dimulai dengan *padding* yang prosesnya persis sama dengan MD5 (lihat bagian 9.1), yaitu setelah ahir naskah, 1 bit dengan nilai 1 ditambahkan, disusul oleh bit dengan nilai 0 sebanyak 0 sampai dengan 511 tergantung panjang naskah, dan diahiri dengan 64 bit yang merepresentasikan besar naskah asli. Setelah *padding*, naskah terdiri dari n *word* $M[0 \ldots n-1]$ dimana n adalah kelipatan 16. Langkah berikutnya dalam *preprocessing* adalah menyiapkan SHA-1 *buffer* sebesar 5 word:

$$(H_0^{(0)}, H_1^{(0)}, H_2^{(0)}, H_3^{(0)}, H_4^{(0)}).$$

Buffer diberi nilai awal sebagai berikut (nilai dalam hexadecimal):

$$\begin{aligned} H_0^{(0)} &\leftarrow 67452301 \\ H_1^{(0)} &\leftarrow efcdab89 \\ H_2^{(0)} &\leftarrow 98badcfe \\ H_3^{(0)} &\leftarrow 10325476 \\ H_4^{(0)} &\leftarrow c3d2e1f0. \end{aligned}$$

Setelah *buffer* diberi nilai awal, tahap kedua yaitu *hashing* dilakukan terhadap setiap blok $(M^{(1)}, M^{(2)}, \ldots, M^{(n)})$ sebagai berikut $(i = 1, 2, \ldots, n)$:

1. Siapkan *message schedule* $\{W_t\}$:

$$W_t \leftarrow \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1 & 16 \le t \le 79. \end{cases}$$

2. Berikan nilai awal untuk variable a, b, c, d, dan e:

$$\begin{aligned} a &\leftarrow H_0^{(i-1)} \\ b &\leftarrow H_1^{(i-1)} \\ c &\leftarrow H_2^{(i-1)} \\ d &\leftarrow H_3^{(i-1)} \\ e &\leftarrow H_4^{(i-1)}. \end{aligned}$$

3. Untuk $t = 0, 1, 2, \ldots, 79$:

$$\begin{aligned} T &\leftarrow (a <<< 5) + f_t(b, c, d) + e + K_t + W_t \pmod{2^{32}} \\ e &\leftarrow d \\ d &\leftarrow c \\ c &\leftarrow b <<< 30 \\ b &\leftarrow a \\ a &\leftarrow T. \end{aligned}$$

4. Lakukan kalkulasi *hash value* tahap i:

$$\begin{aligned} H_0^i &\leftarrow a + H_0^i \\ H_1^i &\leftarrow b + H_1^i \\ H_2^i &\leftarrow c + H_2^i \\ H_3^i &\leftarrow d + H_3^i \\ H_4^i &\leftarrow e + H_4^i. \end{aligned}$$

Setelah *hashing* dilakukan pada semua blok $(M^{(1)}, M^{(2)}, \ldots, M^{(n)})$, kita dapatkan *digest* sebagai berikut:

$$(H_0^{(n)}, H_1^{(n)}, H_2^{(n)}, H_3^{(n)}, H_4^{(n)}).$$

Beberapa ahli kriptografi telah mencoba mencari kelemahan SHA-1. Xiaoyun Wang dan koleganya berhasil memperkecil ruang pencarian untuk *collision*

SHA-1 dari 2^{80} operasi SHA-1 (yang merupakan "kekuatan" teoritis SHA-1 jika SHA-1 tidak memiliki kelemahan, berdasarkan *birthday attack*) menjadi 2^{69} operasi (lihat [wyy05]). Walaupun demikian, penggunaan SHA-1 masih dianggap cukup aman, dan jika ingin lebih aman lagi, maka SHA-256, SHA-384 atau SHA-512 dapat digunakan.

SHA-256, SHA-384 dan SHA-512, bersama dengan SHA-224 secara kolektif masuk dalam standard SHA-2. Saat bab ini ditulis, NIST sedang mengadakan sayembara pembuatan standard SHA-3 yang diharapkan akan lebih tangguh dari SHA-2.

9.3 Hash Message Authentication Code

Hash message authentication code atau HMAC adalah metode *authentication* untuk pesan atau naskah menggunakan *secure hashing* dan kunci rahasia. Jika k adalah kunci rahasia dan m adalah pesan atau naskah, maka rumus yang digunakan untuk HMAC adalah

$$HMAC(k, m) = H((k \oplus p_o) \circ H((k \oplus p_i) \circ m)).$$

dimana H adalah fungsi *secure hashing* (contohnya MD5 atau SHA-1), $\circ$ adalah operasi penyambungan (*concatenation*), p_o dan p_i masing-masing merupakan *padding* sebesar blok yang digunakan H. *Padding* p_o berisi byte $0x5c$ (hexadecimal) yang diulang untuk memenuhi blok, dan *padding* p_i berisi byte $0x36$ (hexadecimal) yang diulang untuk memenuhi blok. Jika kunci k lebih kecil dari blok, maka k di*padding* dengan 0 sampai memenuhi blok. Jika k lebih besar dari blok, maka k dipotong belakangnya hingga besarnya sama dengan blok. Metode HMAC digunakan karena metode yang menggunakan rumus berikut:

$$MAC(k, m) = H(k \circ m)$$

mempunyai kelemahan yaitu kelemahan terhadap *length extension attack*. Tergantung dari fungsi H, seseorang dapat menambahkan sesuatu (misalnya m_a) ke m:

$$m' = m \circ m_a$$

dan tanpa mengetahui k dapat membuat

$$MAC(k, m').$$

HMAC digunakan oleh SSL/TLS (lihat bagian 20.1) dan IPsec (lihat bagian 20.3). Metode HMAC menggunakan MD5 dinamakan HMAC-MD5. Demikian juga, metode HMAC menggunakan SHA-1 dinamakan HMAC-SHA-1.

9.4 Ringkasan

Bab ini telah membahas *secure hashing* yaitu proses pembuatan *fingerprint* atau *digest* untuk suatu naskah. Dua contoh algoritma *secure hashing* dijelaskan: MD5 dan SHA-1. MD5 telah dianggap tidak aman penggunaannya untuk *digital signature*. SHA-1, meskipun memiliki kelemahan, masih dianggap cukup aman. Untuk lebih aman lagi, SHA-256, SHA-384 atau SHA-512 dapat digunakan. Metode *authentication* menggunakan kunci, HMAC, juga dijelaskan.