Studi dan Implementasi *Digital Signature* Menggunakan Algoritma RSA dan Fungsi HAVAL

Gemita Ria The – NIM: 13507133
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if17133@students.if.itb.ac.id

Abstrak—Tanda tangan telah lama digunakan untuk membuktikan otentikasi dokumen dan fungsi tanda tangan pada dokumen kertas juga diterapkan untuk otentikasi data digital. Digital signature merupakan suatu nilai kriptografis dari sebuah pesan yang bergantung pada pesan dan pengirim pesan. Pemberian tanda tangan tersebut memanfaatkan kriptografi kunci publik dengan menggunakan fungsi hash. Pada makalah ini akan dikembangkan sebuah digital dignature menggunakan algoritma RSA dan fungsi HAVAL. Fungsi HAVAL sendiri merupakan fungsi hash yang dapat menghasilkan message digest dengan panjang yang berbeda – beda. Dengan adanya digital signature, integritas dan asal pesan dapat dijamin.

***Kata kunci—digital signature, hash,* RSA, HAVAL**

I. PENDAHULUAN

Teknik kriptografi digunakan untuk menyelesaikan masalah – masalah keamanan seperti berikut:

- Kerahasiaan pesan (*confidentiality/secrecy*)
- Keabsahan pengirim (*user authentication*)
- Keaslian pesan (*message integrity*)
- Anti-penyangkalan (*nonrepudiation*)

Ketiga masalah terakhir dapat diselesaikan dengan teknik otentikasi pesan (*message authentication*). Salah satu alternatif cara untuk melakukan otentikasi adalah dengan menandatangani pesan (*message signature*). Pemberian tanda tangan tersebut dilakukan secara digital dan pesan yang telah ditandatangani menunjukkan bahwa pesan tersebut otentik (baik isi maupun pengirimnya).

Kombinasi algoritma kunci publik dan fungsi *hash* satu arah dapat digunakan untuk membuat sebuah tanda tangan digital. RSA merupakan salah satu algoritma kunci publik yang paling sering digunakan dan fungsi HAVAL merupakan fungsi *hash* yang dapat menghasilkan *message digest* dengan panjang yang berbeda – beda dan telah terbukti sampai saat ini belum berhasil di-"bobol" sehingga kombinasi RSA dan HAVAL dapat menjadi sebuah alternatif untuk membuat *digital signature.*

Makalah ini akan membahas mengenai apa yang dimaksud dengan tanda tangan digital dan RSA, kemudian akan dilanjutkan dengan penjelasan mengenai fungsi HAVAL. Selanjutnya akan dibentuk algoritma tanda tangan digital berdasarkan kombinasi RSA dan fungsi HAVAL tersebut.

Hasil implementasi tanda tangan digital dengan algoritma RSA dan fungsi HAVAL selanjutnya akan diuji dan hasilnya akan diuraikan bersama dengan analisa yang dilakukan terhadap hasil pengujian.

II. TANDA TANGAN DIGITAL (*DIGITAL SIGNATURE*)

Tanda tangan digital dikembangkan untuk mendukung keamanan data dalam bentuk otentikasi pesan yang didasari pada tanda tangan di dokumen kertas. Tanda tangan digital merupakan sebuah tanda tangan elektronik yang dapat digunakan untuk mengotentikasi identitas pengirim pesan dan dapat digunakan untuk menjamin keaslian dokumen atau pesan yang dikirimkan. Tanda tangan digital dapat dengan mudah dipindahkan, tidak dapat dipalsukan, dan memiliki *time-stamp*.

Sebuah tanda tangan digital dapat digunakan untuk berbagai jenis pesan, tidak terbatas pada pesan yang dienkripsi saja. Menandatangani pesan dapat dilakukan dengan salah satu dari dua cara berikut ini:

- Enkripsi pesan
  Pesan yang terenkripsi telah menyatakan bahwa pesan tersebut telah ditandatangani.
- Tanda tangan digital dengan fungsi *hash* (*hash function*)
  Tanda-tangan digital dibangkitkan dari hasil *hash* terhadap pesan. Nilai *hash* adalah kode ringkas dari pesan. Tanda tangan digital berlaku seperti tanda-tangan pada dokumen kertas yang ditambahlan pada pesan.

Skema tanda tangan digital terdiri dari sebuah tuple (M, K, G, S, V) dengan

M – pesan
K – kunci privat dan publik (pk, sk)
G – PPT algortima pembangkit kunci
S – PPT algoritma untuk tanda tangan dengan masukan kunci privat dan pesan
V – verifikasi pesan dengan Vpk(m,Ssk(m)) = 1 jika pk, sk) adalah kunci yang valid

Pasangan kunci publik dan privat akan diperoleh melalui G, kemudian pesan yang akan diberikan tanda tangan dikenakan S dengan menggunakan kunci privat

sk. Dalam hal ini, S adalah algoritma enkripsi kunci publik. Pesan yang telah tertandatangani tersebut akan diberikan kepada penerima pesan. Untuk mengetahui asal pesan dan apakah pesan tersebut masih asli, dilakukan sebuah proses verifikasi dengan algoritma V yang menggunakan parameter kunci publik pengirim yaitu pk.

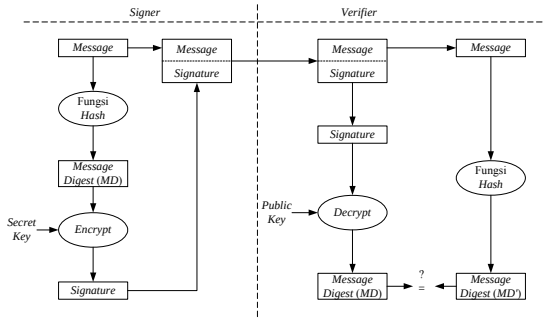
Skema tanda tangan digital tersebut harus mampu menangani keabsahan pengirim, keaslian pesan, dan anti penyanggahan. Pengecekan keabsahan pengirim dilakukan dengan melakukan proses verifikasi (pengecekan terhadap kunci publik dan kunci privat). Keaslian pesan berkaitan dengan keutuhan pesan. Jika terjadi modifikasi pesan ataupun modifikasi pada bagian tanda tangan digital akan menyebabkan kegagalan otentikasi. Anti-penyanggahan menjamin pengirim tidak dapat menyangkal tentang isi pesan atau fakta bahwa ia yang mengirimkan pesan. Jika keabsahan pengirim dan keaslian pesan telah berhasil diverifikasi, maka pengirim tidak dapat menyanggah pesan yang dikirim.

Proses pemberian tanda tangan digital dengan menggunakan gabungan algoritma kunci publik dan fungsi *hash* ialah sebagai berikut:

- Pesan yang hendak dikirim diubah menjadi bentuk yang lebih sederhana yaitu *message digest* dengan menggunakan fungsi *hash* satu-arah (*one-way*) H,
  $$MD = H(M)$$
- Selanjutnya *message digest* tersebut dienkripsi dengan menggunakan algoritma kunci publik menggunakan kunci privat (SK) pengirim menjadi tanda-tangan digital S,
  $$S = E_{SK}(MD)$$
- Pesan M tersebut digabungkan dengan tanda-tangan digital S kemudian pesan tersebut dikirim ke tujuan.

Di tempat penerima, pesan tersebut akan diverifikasi dengan cara:

- Tanda-tangan digital tersebut didekripsi dengan menggunakan kunci publik (PK) pengirim pesan dan menghasilkan *message digest* semula, MD, sebagai berikut:
  $MD = D_{PK}(S)$
- Pesan M yang diterima diubah menjadi *message digest*, MD', menggunakan fungsi *hash* yang sama dengan yang digunakan oleh pengirim.
- Jika MD = MD', berarti tanda tangan tersebut otentik dan berasal dari pengirim yang benar.



Gambar 1 Skema otentikasi tanda-tangan digital menggunakan fungsi hash

III. ALGORITMA RSA

Algoritma RSA merupakan salah satu algoritma kunci publik yang dibuat oleh tiga orang peneliti dari MIT (*Massachussets Institute of Technology*) pada tahun 1976, yaitu: Ron Rivest, Adi Shamir, dan Leonard Adleman. Algoritma ini didasarkan pada teorema Euler.

A. Algoritma Pembangkitan Kunci

Berikut ini merupakan algoritma yang digunakan untuk membangkitkan kunci publik dan kunci privat yang akan digunakan pada algoritma enkripsi dan dekripsi RSA:

- Pilih dua bilangan prima p dan q dan hitung nilai n, dimana n = pq
- Hitunglah $\phi(n)$, dimana $\phi(n) = (p-1)(q-1)$
- Pilih kunci publik, E, yang relatif prima terhadap $\phi(n)$. Pasangan nilai (n, E) adalah kunci publik.
- Kunci privat didapatkan dengan cara:
  $$D = \frac{1 + k\phi(n)}{E}$$
  Pasangan nilai (n, D) adalah kunci privat.

Nilai n tidak bersifat rahasia namun nilai n tersebut diperlukan untuk proses enkripsi/dekripsi.

B. Algoritma Enkripsi/Dekripsi

Berikut ini merupakan algoritma enkripsi dan dekripsi RSA:

- Untuk proses enkripsi pesan dibagi – bagi menjadi blok - blok P_i, dimana setiap nilai $P_i < (p-1)(q-1) = \phi(n)$
- Setiap blok tersebut akan dienkripsi menjadi blok – blok cipher dengan menggunakan persamaan:
  $$C_i \equiv P_i^E \bmod (n)$$
- Untuk proses dekripsi, setiap blok cipher tersebut akan didekripsi dengan menggunakan persamaan:
  $P_i \equiv C_i^D \bmod (n)$

IV. HAVAL

HAVAL merupakan salah satu fungsi *hash* satu arah atau yang dikenal dengan nama *one-way hashing algorithm*. Fungsi *hash* merupakan fungsi yang menerima masukan *string* yang panjangnya bebas dan mengkonversikannya menjadi sebuah *string* yang panjangnya tetap. Fungsi *hash* satu arah sendiri merupakan fungsi *hash* yang bekerja dalam satu arah yaitu pesan yang telah diubah menjadi *message digest* tidak akan dapat dikembalikan menjadi pesan semula. Algoritma *hash* satu arah ini banyak digunakan untuk otentikasi informasi, sebagai contoh untuk *digital signature*. Sifat – sifat dari fungsi *hash* satu arah adalah sebagai berikut:

- Fungsi H dapat diterapkan pada blok data berukuran berapa saja
- H menghasilkan nilai (h) dengan panjang tetap
- H(x) mudah dihitung untuk setiap nilai x yang diberikan
- Untuk setiap h yang dihasilkan, tidak mungkin dikembalikan nilai x sedemikian sehingga H(x) = h. itulah sebabnya fungsi H dikatakan fungsi *hash* satu arah
- Untuk setiap x yang diberikan, tidak mungkin mencari y ≠ x sedemikian sehingga H(y) = H(x)
- Tidak mungkin mencari pasangan x dan y sedemikian sehingga H(x) = H(y)

A. Struktur HAVAL

HAVAL dipublikasikan pertama kali sebagai fungsi *hash*. HAVAL memiliki 15 level keamanan yang berbeda – beda, yang jumlah putarannya dapat dipilih di antara 3, 4, dan 5, dan panjang *hash* dapat dipilih di antara 128 bit dan 256 bit dengan peningkatan sebanyak 32 bit. Struktur HAVAL hampir menyerupai MD4.

Panjang blok HAVAL adalah 1024 bit. Metode *padding* dari MD4 telah diperluas. Setelah dilakukan *padding* dengan bit 1 atau 0, 16 bit tambahan dibuat yang merupakan gabungan versi yang digunakan (3 bit), jumlah putaran dari fungsi kompresi(3 bit), dan panjang *hash* (10 bit) yang kemudian ditambahkan sebelum 64 bit panjang pesan.

HAVAL akan melakukan *padding* terlebih dahulu terhadap pesan tersebut dan panjang pesan tersebut setelah dilakukan *padding* ialah kelipatan 1024 dan *padding* tersebut tetap dilakukan walaupun panjang pesan telah mencapai kelipatan 1024. Blok terakhir dari pesan yang telah di*padding* mengandung jumlah bit dari pesan asli yang belum di*padding*, jumlah bit yang dibutuhkan di dalam *digest*, dan jumlah putaran setiap blok pesan yang diproses. Versi HAVAL yang digunakan juga dicantumkan, biasanya versi yang digunakan adalah versi 1.

B. Proses HAVAL

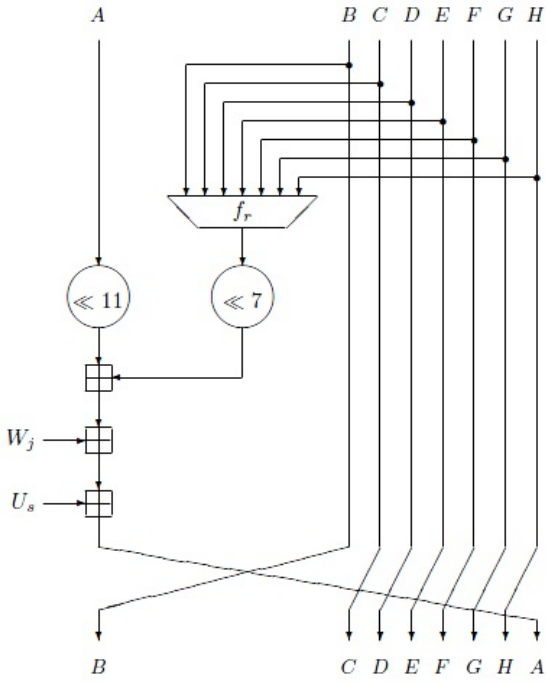
Misalkan pesan yang telah di*padding* itu adalah B_{n-1}, B_{n-2}, …, B_0, dimana setiap B_i merupakan sebuah blok 1024 bit. HAVAL dimulai dari blok B_0 dan sebuah 8-*word* (256 bit) string konstan D0 = $D_{0,7}D_{0,6}…D_{0,0}$, yang diambil dari hasil pecahan Π = 3.1415…, dan memproses pesan B_{n-1}, B_{n-2}, …, B_0 secara blok per blok. Lebih tepatnya, pesan tersebut dipadatkan dengan mengulangi perhitungan:

$D_{i+1} = H(D_i, B_i)$

dimana i merupakan angka yang berada pada range 0 sampai n-1 dan H disebut sebagai *updating algorithm* dari HAVAL.

Akhirnya, 256 bit terakhir dari string D_n disesuaikan dengan panjang *digest* yang dispesifikasikan di blok terakhir B_{n-1} dan *string* tersebut menjadi *message digest* dari pesan M. Proses HAVAL tersebut dapat dirangkum ke dalam tiga langkah, yaitu sebagai berikut:

1. *Padding* pesan M sehingga panjangnya menjadi kelipatan 1024. Blok terakhir dari pesan yang telah di*padding* mengindikasikan panjang dari pesan asli M sebelum di*padding*, panjang yang dibutuhkan dari *digest* M, jumlah putaran setiap blok yang diproses dan versi HAVAL yang digunakan.
2. Hitung secara berulang $D_{i+1} = H(D_i, B_i)$ dengan I dari 0 sampai n-1 dimana D_0 adalah sebuah *string* tetap 8-*word* (256 bit) dan n adalah jumlah total blok yang ada di dalam pesan yang telah di*padding*.
3. Sesuaikan nilai 256-bit D_n yang diperoleh berdasarkan perhitungan di atas berdasarkan panjang *digest* yang telah dispesifikasikan di blok terakhir B_{n-1} dan hasil dari penyesuaian nilai tersebut menjadi *message digest* dari pesan M.



Gambar 2Langkah – langkah untuk HAVAL

C. Padding

Tujuan dari melakukan *padding* pada pesan ialah untuk membuat panjang pesan menjadi *kelipatan* 1024 dan membiarkan pesan mengindikasikan panjang dari pesan asli, jumlah bit *digest*, jumlah putaran dan versi dari HAVAL yang digunakan. HAVAL menggunakan 64-bit *field* MSGLENG untuk mendeskripsikan panjang pesan yang belum di*padding*, 10-bit *field* DGSTLENG untuk mendeskripsikan jumlah bit yang dibutuhkan di dalam sebuah *digest*, 3-bit *field* PASS untuk mendeskripsikan jumlah putaran dari setiap blok pesan yang diproses dan 3-bit *field* VERSION untuk mengindikasikan versi HAVAL. Jumlah bit di dalam *digest* dapat berupa 128, 160, 192, 224, dan 256

sedangkan jumlah putaran dapat berupa 3, 4, dan 5. Untuk versi HAVAL diisi dengan angka 1.

HAVAL melakukan *padding* terhadap pesan dengan menambahkan bit 1 setelah *most significant bit* dari pesan, yang kemudian diikuti dengan bit 0 sampai panjang pesan yang baru mencapai 944 mod 1024. Kemudian HAVAL akan melakukan *padding* dengan bit VERSION, PASS, DGSTLENG, dan MSGLENG.

D. Updating Algorithm H

Algoritma ini memproses sebuah blok ke dalam 3, 4, atau 5 putaran, sesuai dengan yang didefinisikan di dalam *field* PASS pada blok terakhir dari pesan. Kelima putaran tersebut ditunjukkan dengan H_1, H_2, H_3, H_4, dan H_5.

Misalkan masukan untuk H ialah (D*in*, B) dimana D_{in} adalah *string* berukuran 8-*word* dan B adalah blok dengan ukuran 32-*word* (1024 bit) maka D_{out} merupakan hasil dari fungsi H terhadap masukan (D*in*, B). Proses H dapat digambarkan sebagai berikut:

$E_0 = D_{in}$;
$E_1 = H_1(E_0, B)$;
$E_2 = H_2(E_1, B)$;
$E_3 = H_3(E_2, B)$;
$E_4 = H_4(E_3, B)$; (jika PASS = 4,5)
$E_5 = H_5(E_4, B)$; (jika PASS = 5)

$$D_{out} = \begin{cases} E_3 + E_0 \text{ jika PASS} = 3 \\ E_4 + E_0 \text{ jika PASS} = 4 \\ E_5 + E_0 \text{ jika PASS} = 5 \end{cases}$$

Setiap H_1, H_2, H_3, H_4, dan H_5 memiliki 32 putaran operasi dan setiap putaran akan memproses *word* yang berbeda dari B. Urutan *word* yang akan diproses oleh B berbeda – beda tergantung oleh putarannya.

Fungsi yang ada di dalam H_1, H_2, H_3, H_4, dan H_5 ialah sebagai berikut:

$$f_1(x_6, x_5, x_4, x_3, x_2, x_1, x_0) = x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_1 \oplus x_0$$

$$f_2(x_6, x_5, x_4, x_3, x_2, x_1, x_0) = x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_1x_2 \oplus x_1x_4 \oplus x_2x_6 \oplus x_3x_5 \oplus x_4x_5 \oplus x_0x_2 \oplus x_0$$

$$f_3(x_6, x_5, x_4, x_3, x_2, x_1, x_0) = x_1x_2x_3 \oplus x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_3 \oplus x_0$$

$$f_4(x_6, x_5, x_4, x_3, x_2, x_1, x_0) = x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_3x_4x_6 \oplus x_1x_4 \oplus x_2x_6 \oplus x_3x_4 \oplus x_3x_5 \oplus x_3x_6 \oplus x_4x_5 \oplus x_4x_6 \oplus x_0x_4 \oplus x_0$$

$$f_5(x_6, x_5, x_4, x_3, x_2, x_1, x_0) = x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_1x_2x_3 \oplus x_0x_5 \oplus x_0$$

E. Mengubah Hasil Terakhir dari H

$D_n = D_{n,7}D_{n,6}\ldots D_{n,}0$ merupakan hasil dari H yang berukuran 256 bit. D_n akan langsung menjadi *message digest* jika panjang *message digest* yang diminta adalah 256 bit. Jika ternyata, panjang yang diminta berbeda, maka D_n tersebut akan diubah menjadi *string* dengan panjang yang sesuai. Ada empat kasus yang dapat ditinjau yaitu 128 bit, 160 bit, 192 bit, dan 224 bit.

Untuk kasus 128 bit maka $D_{n,7}$,$D_{n,6}$,$D_{n,5}$ dan $D_{n,4}$ dibagi sebagai berikut:

$$D_{n,7} = X^{[8]}_{7,3}X^{[8]}_{7,2}X^{[8]}_{7,1}X^{[8]}_{7,0},$$
$$D_{n,6} = X^{[8]}_{6,3}X^{[8]}_{6,2}X^{[8]}_{6,1}X^{[8]}_{6,0},$$
$$D_{n,5} = X^{[8]}_{5,3}X^{[8]}_{5,2}X^{[8]}_{5,1}X^{[8]}_{5,0},$$
$$D_{n,4} = X^{[8]}_{4,3}X^{[8]}_{4,2}X^{[8]}_{4,1}X^{[8]}_{4,0}.$$

Hasil *digest* dengan ukuran 128 bit dinyatakan dalam $Y_3Y_2Y_1Y_0$ dimana

$$Y_3 = D_{n,3} \boxplus (X^{[8]}_{7,3}X^{[8]}_{6,2}X^{[8]}_{5,1}X^{[8]}_{4,0}),$$
$$Y_2 = D_{n,2} \boxplus (X^{[8]}_{7,2}X^{[8]}_{6,1}X^{[8]}_{5,0}X^{[8]}_{4,3}),$$
$$Y_1 = D_{n,1} \boxplus (X^{[8]}_{7,1}X^{[8]}_{6,0}X^{[8]}_{5,3}X^{[8]}_{4,2}),$$
$$Y_0 = D_{n,0} \boxplus (X^{[8]}_{7,0}X^{[8]}_{6,3}X^{[8]}_{5,2}X^{[8]}_{4,1}).$$

Untuk kasus 160 bit, $D_{n,7}$,$D_{n,6}$, dan $D_{n,5}$ dibagi sesuai persamaan berikut:

$$D_{n,7} = X^{[7]}_{7,4}X^{[6]}_{7,3}X^{[7]}_{7,2}X^{[6]}_{7,1}X^{[6]}_{7,0},$$
$$D_{n,6} = X^{[7]}_{6,4}X^{[6]}_{6,3}X^{[7]}_{6,2}X^{[6]}_{6,1}X^{[6]}_{6,0},$$
$$D_{n,5} = X^{[7]}_{5,4}X^{[6]}_{5,3}X^{[7]}_{5,2}X^{[6]}_{5,1}X^{[6]}_{5,0}.$$

Hasil *digest* dengan ukuran 160-bit dinyatakan dalam $Y_4Y_3Y_2Y_1Y_0$ dimana

$$Y_4 = D_{n,4} \boxplus (X^{[7]}_{7,4}X^{[6]}_{6,3}X^{[7]}_{5,2}),$$
$$Y_3 = D_{n,3} \boxplus (X^{[6]}_{7,3}X^{[7]}_{6,2}X^{[6]}_{5,1}),$$
$$Y_2 = D_{n,2} \boxplus (X^{[7]}_{7,2}X^{[6]}_{6,1}X^{[6]}_{5,0}),$$
$$Y_1 = D_{n,1} \boxplus (X^{[6]}_{7,1}X^{[6]}_{6,0}X^{[7]}_{5,4}),$$
$$Y_0 = D_{n,0} \boxplus (X^{[6]}_{7,0}X^{[7]}_{6,4}X^{[6]}_{5,3}).$$

Untuk kasus 192 bit, $D_{n,7}$ dan $D_{n,6}$ dibagi menjadi:

$$D_{n,7} = X^{[6]}_{7,5}X^{[5]}_{7,4}X^{[5]}_{7,3}X^{[6]}_{7,2}X^{[5]}_{7,1}X^{[5]}_{7,0},$$
$$D_{n,6} = X^{[6]}_{6,5}X^{[5]}_{6,4}X^{[5]}_{6,3}X^{[6]}_{6,2}X^{[5]}_{6,1}X^{[5]}_{6,0}.$$

Hasil *digest*nya akan dinyatakan dalam $Y_5Y_4Y_3Y_2Y_1Y_0$ dimana

$$Y_5 = D_{n,5} \boxplus (X^{[6]}_{7,5}X^{[5]}_{6,4}),$$
$$Y_4 = D_{n,4} \boxplus (X^{[5]}_{7,4}X^{[5]}_{6,3}),$$
$$Y_3 = D_{n,3} \boxplus (X^{[5]}_{7,3}X^{[6]}_{6,2}),$$
$$Y_2 = D_{n,2} \boxplus (X^{[6]}_{7,2}X^{[5]}_{6,1}),$$
$$Y_1 = D_{n,1} \boxplus (X^{[5]}_{7,1}X^{[5]}_{6,0}),$$
$$Y_0 = D_{n,0} \boxplus (X^{[5]}_{7,0}X^{[6]}_{6,5}).$$

Untuk kasus 224 bit, $D_{n,7}$ dibagi menjadi:

$$D_{n,7} = X^{[5]}_{7,6}X^{[5]}_{7,5}X^{[4]}_{7,4}X^{[5]}_{7,3}X^{[4]}_{7,2}X^{[5]}_{7,1}X^{[4]}_{7,0}.$$

Hasil *digest*nya dinyatakan dalam $Y_6Y_5Y_4Y_3Y_2Y_1Y_0$ dimana

$$Y_6 = D_{n,6} \boxplus X_{7,0}^{[4]},$$
$$Y_5 = D_{n,5} \boxplus X_{7,1}^{[5]},$$
$$Y_4 = D_{n,4} \boxplus X_{7,2}^{[4]},$$
$$Y_3 = D_{n,3} \boxplus X_{7,3}^{[5]},$$
$$Y_2 = D_{n,2} \boxplus X_{7,4}^{[4]},$$
$$Y_1 = D_{n,1} \boxplus X_{7,5}^{[5]},$$
$$Y_0 = D_{n,0} \boxplus X_{7,6}^{[5]}.$$

V. SKEMA *DIGITAL SIGNATURE* MENGGUNAKAN RSA DAN HAVAL

Sebuah tanda tangan digital (*digital signature*) pada dokumen digital dapat diterapkan dengan menggunakan kombinasi algoritma kunci publik dan fungsi *hash*. Berdasarkan hal tersebut maka algoritma RSA dan fungsi *hash* HAVAL juga dapat dikombinasikan untuk membentuk sebuah skema tanda tangan digital. RSA sendiri telah banyak digunakan dan merupakan salah satu algoritma yang direkomendasikan untuk membuat tanda tangan digital. Karena itu, selanjutnya akan dibahas bagaimana skema tanda tangan yang akan dibuat dengan mengombinasikan RSA dan fungsi HAVAL.

A. Skema tanda tangan digital

Skema untuk tanda tangan digital dengan kombinasi algoritma RSA dan fungsi HAVAL ialah sebagai berikut:

- Pesan yang akan diberikan tanda tangan digital (M) dimasukkan ke dalam fungsi *hash* HAVAL untuk mendapatkan *message digest*
- Lakukan pembangkitan kunci publik dan kunci privat secara acak
- *Message digest* tersebut kemudian akan dienkripsi dengan menggunakan algoritma RSA beserta kunci privat yang diperoleh dari pembangkitan kunci secara acak dan hasil enkripsi dari *message digest* tersebut akan menjadi tanda tangan digital dari pesan (M)

B. Pembangkitan kunci

Untuk dapat menjalankan algoritma RSA tentunya dibutuhkan sepasang kunci publik dan kunci privat. Pasangan kunci publik dan kunci privat tersebut diperoleh dari proses berikut:

- Bilangan prima yang dipakai di algoritma ini hanyalah bilangan prima yang terletak di antara 1 sampai 500
- Seluruh bilangan prima yang terletak di antara 1 sampai 500 diletakkan ke dalam sebuah *list integer*
- Selanjutnya sebuah nilai akan dibangkitkan secara acak dari *range* nilai 1 sampai jumlah elemen yang terdapat di dalam *list* bilangan prima tersebut. Nilai ini akan menjadi indeks bilangan prima yang akan diambil dari *list* bilangan prima.
- Pembangkitan indeks *list* tersebut dilakukan sebanyak dua kali dan kedua nilai tersebut akan menjadi nilai p dan nilai q yang kemudian digunakan untuk mendapatkan nilai n dimana n = pq
- Menentukan nilai phi = (p-1)(q-1)
- Melakukan pembangkitan kunci publik (pk) dengan ketentuan membangkitkan sebuah bilangan prima yang berada di antara 1 sampai phi
- Melakukan pembangkitan kunci privat (sk) dengan ketentuan nilai (sk*pk) mod phi = 1

C. Pemberian tanda tangan

Pesan atau dokumen yang akan diberikan tanda tangan tersebut dimasukkan ke dalam fungsi *hash* HAVAL untuk mendapatkan *message digest*. Proses *hashing* yang dilakukan ialah sebagai berikut:

HAVAL menerima masukan sebuah pesan (M), panjang message *digest* yang diinginkan, dan jumlah putaran yang akan dilewati oleh setiap blok. Pesan tersebut kemudian akan di*padding* sampai panjang pesannya menjadi kelipatan 1024. Kemudian pesan dibagi ke dalam blok – blok dengan ukuran 1024 bit. Blok – blok tersebut akan diproses ke dalam fungsi – fungsi sesuai dengan jumlah putaran yang dimasukkan oleh pengguna.

Hasil dari pengolahan blok – blok tersebut pasti akan berukuran 256 bit. Selanjutnya hasil tersebut disesuaikan dengan panjang *message digest* yang dimasukkan oleh pengguna. Message *digest* kemudian dienkripsi dengan menggunakan algoritma RSA dengan menggunakan kunci privat pengirim. Namun sebelum dienkripsi, message *digest* tersebut diubah terlebih dahulu menjadi *BigInteger* dan dibagi – bagi ke dalam blok dengan ukuran 1 blok terdiri dari 2 karakter message *digest*. Jika ternyata panjang message *digest* tersebut bukan kelipatan dua maka message *digest* tersebut akan di*padding* dengan angka 0.

Blok – blok yang telah terbentuk tersebut akan dienkripsi dengan menggunakan algoritma enkripsi RSA dan hasil dari algoritma tersebutlah yang akan menjadi tanda tangan digital dari pesan yang akan dikirimkan.

D. Otentikasi tanda tangan

Untuk melakukan verifikasi tanda tangan, pesan beserta tanda tangannya harus dimasukkan ke dalam aplikasi. Pesan tanpa tanda tangan akan diubah menjadi message *digest* kembali dengan menggunakan fungsi HAVAL dan diubah ke dalam *BigInteger* kemudian tanda tangan yang diberikan ke pesan tersebut didekripsi dengan algoritma RSA dan dibandingkan dengan hasil message *digest* dalam bentuk *BigInteger* tersebut. Apabila hasil dekripsi tersebut sama dengan *message digest* berarti pesan tersebut telah lulus otentikasi.

VI. IMPLEMENTASI SKEMA TANDA TANGAN DIGITAL DAN HASILNYA

Implementasi akan dilakukan untuk menguji skema tanda tangan digital yang telah dibuat dengan menggunakan kombinasi algoritma kriptografi kunci publik RSA dan fungsi *hash* HAVAL. Implementasi dilakukan dengan menggunakan bahasa pemrograman C# dan diuji pada lingkungan Windows.

A. Spesifikasi Implementasi

Aplikasi diberi nama "RSA-HAVAL Digital Signature" ini memiliki spesifikasi sebagai berikut:

- Dapat melakukan pembangkitan kunci publik dan privat secara acak dan menyimpannya dalam *file* eksternal
- Dapat menandatangani dokumen teks
- Dapat melakukan otentikasi terhadap dokumen yang telah ditandatangani

Berikut ini daftar nama kelas yang telah diimplementasikan untuk membangun aplikasi tersebut:

- BigInteger : kelas BigInteger dan operasinya
- GenAlgo: kelas untuk membangkitkan kunci publik dan kunci privat
- RSA: kelas yang mengimplementasikan algoritma RSA
- Haval: kelas yang mengimplementasikan algoritma fungsi *hash* HAVAL
- AuthSignature: merupakan kelas interface untuk melakukan otentikasi dokumen
- DigitalSignature: merupakan kelas interface utama dari aplikasi
- KeyGen: merupakan kelas interface untuk pembangkitan kunci acak
- GiveSignature: merupakan kelas interface untuk memberikan tanda tangan

KeyGen akan secara otomatis membangkitkan kunci publik dan privat beserta nilai n yang kemudian kedua kunci tersebut disimpan ke dalam *file* eksternal terpisah dengan ekstensi .pukey (kunci publik) dan .pikey (kunci privat). Algoritma RSA sendiri membutuhkan parameter yang cukup besar sehingga tidak memungkinkan untuk menggunakan integer. Oleh karena itu, aplikasi ini menggunakan kelas BigInteger yang telah ada dikembangkan untuk C#.

B. Pembangkitan kunci

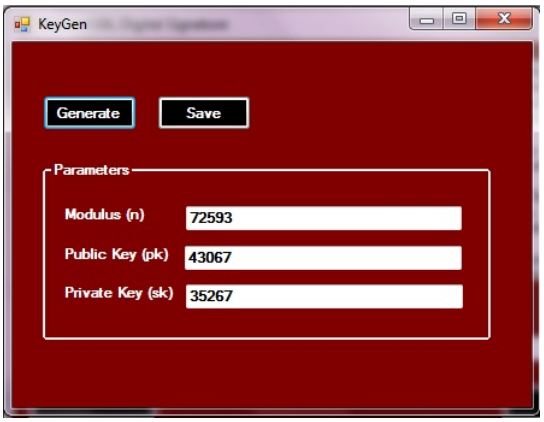
Kode program untuk pembangkitan kunci yang akan digunakan di dalam algoritma RSA ialah sebagai berikut:

```
class GenAlgo{
  List<int> primes = new List<int>();
  public int p;
  public int q;
  public int n;
  public int phi;
  public int pubkey;
  public int prikey;
  public int x;

  public GenAlgo() {
    for (int i = 1; i <= 500; i++){
      bool isPrime = true;
      for (int j = 2; j < i; j++){
        if (i % j == 0){
          isPrime = false;
          break;
        }
      }
       if (isPrime){
        primes.Add(i);
       }
    }
    p = 0;
    q = 0;
    n = 0;
    phi = 0;
    pubkey = 0;
    prikey = 0;
  }

  public void gen() {
    while (p == q){
      p = this.genprime();
      q = this.genprime();
    }
    n = p * q;
    phi = (p - 1) * (q - 1);
      pubkey = genpub();
      prikey = genpri();
   }

   public int genprime() {
     int i = 0;
     Random r = new Random();
     i = r.Next(1, primes.Count);
       x = i;
       return primes[i];
     }

   public int genpub(){
     int i = 0;
     int j;
     Random r = new Random();
     bool isPrime = false;
     while(!isPrime){
       i = r.Next(1, phi);
       for (j = 2; j < i; j++) {
         if (i % j == 0) {
            break;
         }
       }
       if (j == i) {
         isPrime = true;
       }
     }
     return i;
    }

    public int genpri() {
      int i = 0;
      Random r = new Random();
      while ((i * pubkey) % phi != 1){
        i = r.Next(phi);
      }
      return i;
```

```
    }
  }
```

Fungsi genprime digunakan untuk membangkitkan bilangan prima acak yang nilainya berkisar di antara 1-500. Fungsi genpub digunakan untuk membangkitkan kunci publik. Fungsi genpri digunakan untuk membangkitkan kunci privat.

Hasil penggunaan algoritma tersebut dalam membangkitkan kunci publik dan kunci privat dapat dilihat pada gambar berikut:



Gambar 3Hasil pembangkitan kunci publik dan kunci privat secara acak

Dari gambar tersebut diperoleh hasil:

n = 72593
kunci publik (pk) = 43067
kunci privat (sk) = 35267

C. Tanda tangan

Sebelum dilakukan pemberian tanda tangan, pesan/dokumen akan dikenakan fungsi HAVAL untuk mendapatkan message *digest*. Berikut ini konstanta yang digunakan di dalam algoritma HAVAL:

- Fungsi untuk menginisialisasi 8-*word* register internal

```
private void InitializeState(){
  state[0] = 0x243f6a88;
  state[1] = 0x85a308d3;
  state[2] = 0x13198a2e;
  state[3] = 0x03707344;
  state[4] = 0xa4093822;
  state[5] = 0x299f31d0;
  state[6] = 0x082efa98;
  state[7] = 0xec4e6c89;
}
```

- Urutan *word* tergantung pada putarannya

```
private    static    readonly    uint[]
  wordOrders = new uint[]{
    // pass 3
     19,  09,  04,  20,  28,  17,  08,  22,
     29,  14,  25,  12,  24,  30,  16,  26,
     31,  15,  07,  03,  01,  00,  18,  27,
     13,  06,  21,  10,  23,  11,  05,  02,
     // pass 4
     24,  04,  00,  14,  02,  07,  28,  23,
     26,  06,  30,  20,  18,  25,  19,  03,
     22,  11,  31,  21,  08,  27,  12,  09,
     01,  29,  05,  15,  17,  10,  16,  13,
     // pass 5
     27,  03,  21,  26,  17,  11,  20,  29,
     19,  00,  12,  07,  13,  08,  31,  10,
     05,  09,  14,  30,  18,  06,  28,  24,
     02,  23,  16,  22,  04,  01,  25,  15
};
```

- Konstanta yang digunakan di setiap putaran

```
private    static    readonly    uint[]
  constants = new uint[]{
    // pass 3
    0x9c30d539,             0x2af26013,
    0xc5d1b023,             0x286085f0,
    0xca417918,             0xb8db38ef,
    0x8e79dcb0,             0x603a180e,
    0x6c9e0e8b,             0xb01e8a3e,
    0xd71577c1,             0xbd314b27,
    0x78af2fda,             0x55605c60,
    0xe65525f3,             0xaa55ab94,
    0x57489862,             0x63e81440,
    0x55ca396a,             0x2aab10b6,
    0xb4cc5c34,             0x1141e8ce,
    0xa15486af, 0x7c72e993,
    0xb3ee1411,             0x636fbc2a,
    0x2ba9c55d,             0x741831f6,
    0xce5c3e16,             0x9b87931e,
    0xafd6ba33, 0x6c24cf5c,
    // pass 4
    0x7a325381,             0x28958677,
    0x3b8f4898,             0x6b4bb9af,
    0xc4bfe81b,             0x66282193,
    0x61d809cc,             0xfb21a991,
    0x487cac60,             0x5dec8032,
    0xef845d5d,             0xe98575b1,
    0xdc262302,             0xeb651b88,
    0x23893e81,             0xd396acc5,
    0x0f6d6ff3,             0x83f44239,
    0x2e0b4482,             0xa4842004,
    0x69c8f04a,             0x9e1f9b5e,
    0x21c66842, 0xf6e96c9a,
    0x670c9c61,             0xabd388f0,
    0x6a51a0d2,             0xd8542f68,
    0x960fa728,             0xab5133a3,
    0x6eef0b6c, 0x137a3be4,
    // pass 5
    0xba3bf050,             0x7efb2a98,
    0xa1f1651d,             0x39af0176,
    0x66ca593e,             0x82430e88,
    0x8cee8619,             0x456f9fb4,
    0x7d84a5c3,             0x3b8b5ebe,
    0xe06f75d8,             0x85c12073,
    0x401a449f,             0x56c16aa6,
    0x4ed3aa62,             0x363f7706,
    0x1bfedf72,             0x429b023d,
    0x37d0d724,             0xd00a1248,
    0xdb0fead3,             0x49f1c09b,
    0x075372c9, 0x80991b7b,
    0x25d479d8,             0xf6e8def7,
    0xe3fe501a,             0xb6794c3b,
    0x976ce0bd,             0x04c006ba,
    0xc1a94fb6, 0x409f60c4
};
```

Fungsi lainnya tidak dicantumkan karena kode yang sangat panjang. Contoh pesan yang akan diberikan tanda tangan digital adalah:

> Anda diminta untuk menggunakan program sniffer (penyadap) seperti wireshark, tcpdump, atau sejenisnya untuk melakukan salah satu dari hal di bawah ini:
>
> 1. mengukur jumlah data yang digunakan untuk membuka satu sesi gmail atau facebook;
> 2. memantau serangan dari port scanning yang dilakukan dengan menggunakan program nmap.
>
> Untuk yang (1) yang disebut satu sesi adalah login, membaca sebuah email (atau menulis sebuah email), kemudian logout. Jangan lupa program sniffer diset agar *semua* data ditangkap. Biasanya default dari program sniffer hanya menangkap header saja. Tujuan utamanya adalah untuk menghitung jumlah data yang ditangkap.
>
> Untuk (2) tujuannya adalah untuk melihat pola "serangan" port scanning yang dilakukan oleh nmap. Apakah dia melakukan scanning dengan menguji port secara berurutan?

Hasil *message digest* yang diperoleh dari fungsi *hash* HAVAL tersebut ialah sebagai berikut:

- Untuk pasangan haval128, 3
  ```
  54c7b3e622923704cc79f2897b123faa
  ```
- Untuk pasangan haval160, 3
  ```
  aa48228118c7a40e366e455b5d098e1da52fd1f4
  ```
- Untuk pasangan haval192, 3
  ```
  241d113110d57b726b7c769fa60a5a786f7240ea13d7aff3
  ```
- Untuk pasangan haval224, 3
  ```
  6ecf536ab689d265794dde00537a090596cbfc31b5a136ddc5d7b543
  ```
- Untuk pasangan haval256, 3
  ```
  b1b94914dba092fb3db31f4a5d8235df2785c377364cc0414e873eeb68e0c34d
  ```
- Untuk pasangan haval128, 4
  ```
  b02751f171d799921aca9955c5d73bd3
  ```
- Untuk pasangan haval160, 4
  ```
  27551ce1187ce6953f1597afc161a6cc02d44861
  ```
- Untuk pasangan haval192, 4
  ```
  54e29f651d5c7160a6375f9ee4e8ac90fa6706fa7f128a48
  ```
- Untuk pasangan haval224, 4
  ```
  af52f769047f2ae1b69611968cb7a3b06a908eb60b0d02e273b74d50
  ```
- Untuk pasangan haval256, 4
  ```
  d3dbad92a767895dd4ab3e8cd0d0be261aab52d1bee84653f0d0eab02a673410
  ```
- Untuk pasangan haval128, 5
  ```
  c7036816c838ed3c4aac18b20e726c70
  ```
- Untuk pasangan haval160, 5
  ```
  d918dabebc4a001a1b333371b7128fafa5ebb7a3
  ```
- Untuk pasangan haval192, 5
  ```
  e159873878823e6a92881b3b55224cc77296a9c85da96f12
  ```
- Untuk pasangan haval224, 5
  ```
  1227d627305a768a3200a3ab327df5c202b6fb91d685992e0e9dd774
  ```
- Untuk pasangan haval256, 5
  ```
  1b6b601a3eadc74bf28123f810d6bb48741f503178989a0ca6ead78be093f026
  ```

Untuk mendapatkan hasil tanda tangan digital dari *message digest* yang telah dihasilkan oleh fungsi HAVAL, digunakan fungsi berikut:

```
public string cryptDS(BigInteger[] pBlokMD, BigInteger pExp, BigInteger pN){
   BigInteger[] tHasilEncrypt = new BigInteger[pBlokMD.Length];
   for (int i = 0; i < tHasilEncrypt.Length; i++){
      tHasilEncrypt[i] = pBlokMD[i].modPow(pExp, pN);
    }
    StringBuilder tTemp = new StringBuilder();
    for (int i = 0; i < tHasilEncrypt.Length; i++){
     tTemp.Append(tHasilEncrypt[i].ToHexString());
      tTemp.Append(" ");
    }
    return tTemp.ToString();
}
```

Tanda tangan digital hasil dari enkripsi tersebut dengan menggunakan kunci privat 35267 adalah sebagai berikut:

```
107B8B728B13A127C9056A9323043F6ADA18E3043E947BB3AFC8E84285BF61CEBC0815AEC84286AB3
```

D. Otentikasi Dokumen

Kode yang digunakan untuk mengotentikasi dokumen yang telah diberikan tanda tangan digital ialah sebagai berikut:

```
public string decryptDS(BigInteger[] pBlokMD, BigInteger pExp, BigInteger pN){
   BigInteger[] tHasilEncrypt = new BigInteger[pBlokMD.Length];
    for (int i = 0; i < tHasilEncrypt.Length; i++){
       tHasilEncrypt[i] = pBlokMD[i].modPow(pExp, pN);
    }
    StringBuilder tTemp = new StringBuilder();
    for (int i = 0; i < tHasilEncrypt.Length; i++){
```

```
        if((tHasilEncrypt[i].ToString()
.Length       ==       1)&&(i       !=
tHasilEncrypt.Length-1)){
           tTemp.Append("0");}
        tTemp.Append(tHasilEncrypt[i].T
oString());
        tTemp.Append(" ");
      }
      return tTemp.ToString();
}
```

Keluaran dari fungsi tersebut ialah hasil dekripsi dalam bentuk *BigInteger* (md'). Untuk melakukan verifikasi maka selain mendekripsi kembali tanda tangan yang telah dibuat, perlu juga dipanggil kembali fungsi *hash* HAVAL untuk membuat *message digest* dari pesan asli tanpa tanda tangan digital. *Message digest* tersebut juga akan bertipe *BigInteger* (md).

Message digest tersebut akan dibandingkan dengan hasil dekripsi dari tanda tangan digital yang ada pada dokumen. Jika bernilai sama, maka dokumen tersebut telah lulus otentikasi.

Hasil md' dari proses dekripsi tersebut ialah sebagai berikut:

112692067505277049274067971555835658154

Sedangkan md hasil dari *hash*ing pesan asli adalah:

112692067505277049274067971555835658154

Dapat dilihat bahwa md' = md sehingga pesan telah lulus otentikasi.

VII. PENGUJIAN

Pengujian yang akan dilakukan terhadap skema tanda tangan digital yang telah dibuat berkisar pada serangan modifikasi yang terdiri dari tujuh macam yaitu:

A. Penambahan satu karakter pada tanda tangan digital

Sebuah karakter 'A' akan ditambahkan sehingga tanda tangan digital menjadi:

107B8B728B13A127C9**A**056A9323043F6ADA18E3043E947BB3AFC8E84285BF61CEBC0815AEC84286AB3

Hasil md' dari tanda tangan digital tersebut ialah:

112692064927105277049274067971555835658154

Sedangkan md hasil dari *hash*ing pesan asli ialah:

112692067505277049274067971555835658154

Dapat dilihat bahwa jika dilakukan penambahan sebuah karakter pada tanda tangan digital sebuah dokumen maka hasil md' ≠ md yang menyebabkan dokumen tersebut tidak lulus otentikasi. Hal ini juga menunjukkan bahwa skema tanda tangan digital RSA-HAVAL sensitif terhadap penambahan sebuah karakter di dalam tanda tangan digital.

B. Pengurangan satu karakter pada tanda tangan digital

Sebuah karakter 'B' akan dihilangkan sehingga tanda tangan digital menjadi:

107B8B72813A127C9056A9323043F6ADA18E3043E947BB3AFC8E84285BF61CEBC0815AEC84286AB3

Hasil md' dari tanda tangan digital tersebut ialah:

112629032067505277049274067971555835658154

Nilai md hasil dari *hash*ing pesan asli ialah:

112692067505277049274067971555835658154

Dapat dilihat bahwa jika dilakukan pengurangan sebuah karakter pada tanda tangan digital, hasil md' ≠ md yang menyebabkan dokumen tersebut tidak lulus otentikasi. Hal ini juga menunjukkan bahwa skema tanda tangan digital RSA-HAVAL sensitif terhadap pengurangan sebuah karakter di dalam tanda tangan digital.

C. Pengubahan satu karakter pada tanda tangan digital

Sebuah karakter '8' akan diubah menjadi karakter 'C' sehingga tanda tangan digital menjadi:

107B8B728B13A127C9056A9323043F6ADA18E3043E947BB3AFC8E84285BF61CEBC0**C**15AEC84286AB3

Hasil md' dari tanda tangan digital tersebut ialah:

112692067505277049274067971555835590058154

Nilai md hasil dari *hashing* pesan asli ialah:

112692067505277049274067971555835658154

Dapat dilihat bahwa jika dilakukan pengubahan sebuah karakter pada tanda tangan digital, hasil md' ≠ md yang menyebabkan dokumen tersebut tidak lulus otentikasi. Hal ini juga menunjukkan bahwa skema tanda tangan digital RSA-HAVAL sensitif terhadap pengubahan sebuah karakter di dalam tanda tangan digital.

D. Penambahan satu karakter pada isi dokumen

Sebuah karakter 'l' akan ditambahkan sehingga isi dokumen menjadi:

Anda**l** diminta untuk menggunakan program sniffer (penyadap) seperti wireshark, tcpdump, atau sejenisnya untuk melakukan salah satu dari hal di bawah ini: [dst]

Hasil md dari dokumen tersebut ialah:

253808454407696937738143557398268979851

Nilai md' hasil dekripsi tanda tangan digital ialah:

112692067505277049274067971555835658154

Dapat dilihat bahwa jika dilakukan penambahan sebuah karakter pada isi dokumen, hasil md ≠ md' yang menyebabkan dokumen tersebut tidak lulus otentikasi. Hal ini juga menunjukkan bahwa skema tanda tangan digital RSA-HAVAL sensitif terhadap penambahan sebuah karakter di dalam isi dokumen.

E. Pengurangan satu karakter pada isi dokumen

Sebuah karakter 'n' akan dihilangkan sehingga isi dokumen menjadi:

Ada diminta untuk menggunakan program sniffer (penyadap) seperti wireshark, tcpdump, atau sejenisnya untuk melakukan salah satu dari hal di bawah ini: [dst]

Hasil md dari dokumen tersebut ialah:

27808116082798405750351289258284096876 7

Nilai md' hasil dekripsi tanda tangan digital ialah:

1126920675052770492740679715558356581 54

Dapat dilihat bahwa jika dilakukan pengurangan sebuah karakter pada isi dokumen, hasil md ≠ md' yang menyebabkan dokumen tersebut tidak lulus otentikasi. Hal ini juga menunjukkan bahwa skema tanda tangan digital RSA-HAVAL sensitif terhadap pengurangan sebuah karakter di dalam isi dokumen.

F. Pengubahan satu karakter pada isi dokumen

Sebuah karakter 'a' akan diubah menjadi karakter 'i' sehingga isi dokumen menjadi:

Andi diminta untuk menggunakan program sniffer (penyadap) seperti wireshark, tcpdump, atau sejenisnya untuk melakukan salah satu dari hal di bawah ini: [dst]

Hasil md dari dokumen tersebut ialah:

13816284498063669974156617900757107730 7

Nilai md' hasil dekripsi tanda tangan digital ialah:

1126920675052770492740679715558356581 54

Dapat dilihat bahwa jika dilakukan pengubahan sebuah karakter pada isi dokumen, hasil md' ≠ md yang menyebabkan dokumen tersebut tidak lulus otentikasi. Hal ini juga menunjukkan bahwa skema tanda tangan digital RSA-HAVAL sensitif terhadap pengubahan sebuah karakter di dalam isi dokumen.

G. Menggunakan kunci publik yang berbeda

Pengujian juga akan dilakukan terhadap kasus jika penerima menggunakan kunci publik yang tidak berpadanan dengan kunci privat yang digunakan untuk membuat tanda tangan digital.

Misalkan kunci yang digunakan adalah:

n = 72593 kunci publik (pk) = 12345 kunci privat (sk) = 35267

Hasil md dari dokumen tersebut ialah:

1126920675052770492740679715558356581 54

Nilai md' hasil dekripsi tanda tangan digital ialah:

6207429649614563712818212395753289281243033653289675650359133031357446995752117752536631357560 78

Dapat dilihat bahwa jika penerima menggunakan kunci publik yang tidak berpadanan dengan kunci privat yang digunakan oleh pengirim untuk membuat tanda tangan digital, dokumen tidak akan lulus otentikasi. Hal ini juga menunjukkan bahwa skema tanda tangan digital RSA-HAVAL sensitif terhadap padanan kunci yang digunakan.

H. Kesimpulan Pengujian

Ketujuh macam pengujian yang telah dilakukan di atas memberikan hasil sesuai dengan yang diharapkan yaitu skema tanda tangan digital dengan menggunakan kombinasi algoritma kunci publik RSA dan fungsi *hash* HAVAL sensitif terhadap modifikasi yang terjadi baik modifikasi yang dilakukan pada tanda tangan digital maupun modifikasi yang dilakukan pada isi dokumen.

Dengan adanya hasil pengujian tersebut maka dapat disimpulkan bahwa tanda tangan digital dengan menggunakan RSA-HAVAL ini merupakan skema tanda tangan digital yang aman dari serangan modifikasi. Hal ini disebabkan algoritma enkripsi/dekripsi RSA bergantung pada kunci publik, kunci privat, isi dokumen maupun tanda tangan digital. Sehingga jika salah satunya mengalami perubahan, message *digest* yang dihasilkan tidak akan sama lagi. Fungsi HAVAL juga bergantung pada isi dokumen sehingga jika isi dokumen saja yang dimodifikasi, dokumen tetap tidak akan lulus otentikasi.

VIII. ANALISIS

A. Perbandingan Performansi dengan MD4 dan MD5

Bila dibandingkan dengan algoritma MD4, perbedaan utama dari HAVAL ialah variabel 256 bit (bukan 128 bit) sehingga jumlah *register*nya dua kali lebih banyak daripada yang dimiliki oleh MD4. Ukuran blok pesan juga menjadi dua kali lipat lebih besar (32 *word* dan bukan 16 lagi) dan tentunya hal tersebut juga berpengaruh pada jumlah langkah di setiap putaran dari fungsi kompresi.

Algoritma HAVAL lebih cepat 60% bila dibandingkan dengan MD5 ketika menggunakan 3 putaran dan sama cepatnya dengan MD5 ketika menggunakan 5 putaran.

B. Analisis Keamanan

Keamanan dari skema tanda tangan digital ini pertama – tama dapat ditinjau dari segi fungsi HAVAL yang terletak pada jumlah putaran untuk fungsi kompresi pesan. Jumlah putaran yang dapat dipilih untuk mengkompresi sebuah pesan ialah sebanyak 3, 4, atau 5. Semakin besar jumlah putaran yang dipilih maka semakin aman sebuah skema tanda tangan digital tersebut namun hal tersebut berbanding terbalik dengan efisiensi algoritmanya.

Setiap putaran yang ada akan terdiri dari 32 langkah sehingga total langkah yang ada di dalam fungsi kompresi tersebut dapat berjumlah 96, 128, atau 160. Setiap langkah akan memperbaharui isi salah satu dari 8 buah register yang ada.

C. Perbandingan Kecepatan antara fungsi HAVAL3, HAVAL4, dan HAVAL5

Ketiga fungsi HAVAL yang ada hanya berbeda pada jumlah putaran. Pada umumnya, performansi HAVAL3 lebih cepat 30% dibandingkan dengan HAVAL4 dan 60% lebih cepat daripada HAVAL5. Hal ini berkaitan erat dengan jumlah putaran yang dilakukan.

D. Kelemahan HAVAL

Fungsi ini memiliki beberapa kekurangan yaitu:

- Dapat dengan mudah terjadi *collision* pada HAVAL dengan dua putaran

- HAVAL3 tidak disarankan untuk digunakan pada aplikasi yang membutuhkan perlindungan terhadap *collision* karena rentan terhadap *collision*
- Kelebihan fungsi *Boolean* yang tidak linear masih belum diterapkan ke seluruh proses *hashing* sehingga fungsi ini masih dapat diserang dengan serangan yang sama untuk MD4 atau MD5.
- Semakin besarnya jumlah putaran mengakibatkan semakin tidak efisiennya algoritma fungsi ini yang menyebabkan waktu pemrosesan akan semakin lama

E. Keamanan RSA

Keamanan dari algoritma ini didasarkan pada sulitnya memfaktorkan bilangan besar menjadi faktor – faktor primanya. Selagi belum ditemukan algoritma yang mangkus untuk memfaktorkan bilangan bulat menjadi faktor prima, maka algoritma RSA tetap direkomendasikan untuk mengenkripsi pesan.

RSA juga cocok digunakan untuk aplikasi yang lebih mengutamakan otentikasi dokumen karena proses otentikasi lebih cepat daripada proses pemberian tanda tangan jika kunci publiknya merupakan angka yang kecil.

F. Kelemahan RSA

Selain kelebihan yang telah dijelaskan di atas, RSA juga memiliki kelemahan yaitu:

- Penggunaan bilangan eksponensial yang kecil untuk enkripsi
- Menggunakan kunci yang sama untuk enkripsi dan untuk memberikan tanda tangan

Solusi untuk mengatasi kelemahan tersebut ialah sebagai berikut:

- Jangan menggunakan kunci RSA yang sama untuk melakukan enkripsi dan pemberian tanda tangan
- Selalu buat format tertentu pada *input* sebelum dilakukan enkripsi atau diberikan tanda tangan
- Selalu tambahnya bilangan acak minimal 8 *byte* ke dalam pesan sebelum mulai di enkripsi
- Ketika akan melakukan dekripsi, periksa format dari blok *cipher* yang ada. Jika formatnya tidak sesuai, maka berikan pesan kesalahan dan pesan tersebut tidak perlu didekripsi
- Ketika memberikan tanda tangan dan ternyata terjadi kesalahan, berikan pesan kesalahan dan jangan melanjutkan proses pemberian tanda tangan

IX. KESIMPULAN

Kesimpulan yang dapat ditarik dari pembahasan dan hasil implementasi di atas ialah:

- Tanda tangan digital merupakan salah satu cara untuk memastikan keabsahan pengirim (*user authentication*), keaslian pesan (*message integrity*), dan anti-penyangkalan (*nonrepudiation*)
- Kombinasi algoritma kriptografi kunci publik RSA dan fungsi hash HAVAL dapat dijadikan sebuah skema tanda tangan digital yang aman terhadap serangan modifikasi isi dokumen maupun modifikasi tanda tangan digital
- Skema tanda tangan digital ini lebih sulit untuk di-"bobol" karena *message digest* yang dihasilkan bervariasi dan jumlah putaran juga bervariasi sehingga tanda tangan yang dihasilkan pun akan bervariasi panjangnya
- Skema tanda tangan ini lebih efektif dibandingkan dengan penggunaan skema tanda tangan digital yang mengombinasikan algoritma RSA dan MD4/MD5 karena performansi dari RSA-HAVAL lebih cepat dibandingkan dengan RSA-MD4/MD5
- Semakin banyak jumlah putaran yang dilakukan di dalam fungsi HAVAL akan membuat tingkat keamanannya meningkat namun waktu yang dibutuhkan untuk melakukan komputasi juga semakin lama
- Skema tanda tangan ini dapat dibuat lebih efisien lagi dengan menyederhanakan fungsi hash terutama untuk fungsi *hash* yang menggunakan lima buah putaran

REFERENSI

[1] Munir, Rinaldi. "Diktat Kuliah IF5054 Kriptografi Departemen Teknik Informatika Institut Teknologi Bandung". 2005. Bandung: Institut Teknologi Bandung.

[2] Zheng, Yuliang, Josef Pieprzyk, dan Jennifer Seberry. *"HAVAL – A One-Way Hashing Algorithm with Variable Length of Output"*. 1993. Department of Computer Science. University of Wollongong, Australia.

[3] Knopf, Diplomarbeit von Christian. *"Cryptographic Hash Functions"*. 2007. Institut fur Theoretische Informatik.

[4] Rompay, Bart Van. *"Analysis Design of Cryptographic Hash Functions, Mac Algorithns and Block Ciphers"*. 2004. Faculteit Toegepaste Wetenschappen Arenbergkasteel. Katholieke Universiteit Leuven. Belgium.

[5] Shah, Kartik. *"RSA Algorithm"*. Lay Networks.

[6] Menezes, A., P. van Oorschot, S. Vanstone. *"Handbook of Applied Cryptography"*. 1996. CRC Press.

[7] http://www.di-mgt.com.au/rsa_alg.html#weaknesses

[8] http://www.c3.hu/docs/oreilly/tcpip/puis/ch06_05.htm

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 17 Mei 2010

Gemita Ria The
13507133