Implementasi Tanda-tangan Digital dengan Menggunakan Algoritma RSA dan Fungsi *Hash* SHA-1 pada Microsoft Office Word 2007

Sesdika Sansani -- 13507047[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*if17047@students.if.itb.ac.id*

***Abstraksi*— Microsoft Office Word adalah salah satu aplikasi pengolah kata yang banyak digunakan. Salah satu fitur yang disediakan adalah tanda tangan digital untuk menjamin integritas dokumen. Akan tetapi, fitur ini belum dapat menjamin bahwa pengirim atau pembuat pesan adalah orang yang seharusnya. Padahal di beberapa kasus, pembuat pesan pun perlu diotentikasi untuk mengetahui keaslian pesan, misalnya pada surat waris atau surat wasiat.**

Salah satu solusi yang dapat digunakan adalah dengan melakukan enkripsi dengan algoritma kunci publik pada message digest yang dihasilkan oleh fungsi hash sehingga penerima dokumen bisa memastikan bahwa pembuat dokumen adalah orang yang benar. Penambahan proses pada pembuatan tanda tangan digital ini akan diimplementasikan sebagai pada Microsoft Word 2007. Proses yang digunakan adalah algoritma SHA-1 dan RSA.

***Index Terms—add-in* Word 2007, otentikasi, RSA, SHA-1, tanda tangan digital**

I. PENDAHULUAN

Microsoft Word atau Microsoft Office Word adalah perangkat lunak pengolah kata (*word processor*) andalan Microsoft [1]. Pertama diterbitkan pada 1983 dengan nama Multi-Tool Word untuk Xenix, versi-versi lain kemudian dikembangkan untuk berbagai sistem operasi, misalnya DOS (1983), Apple Macintosh (1984), SCO UNIX, OS/2, dan Microsoft Windows (1989). Setelah menjadi bagian dari Microsoft Office System 2003 dan 2007 diberi nama Microsoft Office Word. Gambar 1 merupakan tampilan dari Microsoft Word 2007.

Salah satu fitur yang disediakan oleh Word adalah tanda tangan digital. Fitur ini digunakan untuk menjamin integritas dokumen. Tanda tangan digital akan terlihat dan tidak akan memungkinkan untuk mengedit dokumen sehingga terkunci. Dokumen terkunci dapat dibuka, tetapi melakukan hal ini akan membuat tanda tangan digital tidak valid. Jadi proses ini memastikan bahwa dokumen Anda tidak dirusak [2].

Akan tetapi, fitur ini hanya dapat menjamin integritas dari pesan. Fitur ini belum dapat menjamin bahwa pengirim atau pembuat pesan adalah orang yang seharusnya. Padahal di beberapa kasus, pembuat pesan pun perlu diotentikasi untuk mengetahui keaslian pesan, misalnya pada surat waris atau surat wasiat.

Gambar 1 Tampilan Aplikasi Microsoft Word 2007

Salah satu solusi yang dapat digunakan adalah dengan melakukan enkripsi dengan algoritma kunci publik pada message digest yang dihasilkan oleh fungsi hash sehingga penerima dokumen bisa memastikan bahwa pembuat dokumen adalah orang yang benar. Penambahan proses pada pembuatan tanda tangan digital ini akan diimplementasikan sebagai pada Microsoft Word 2007. Proses yang digunakan adalah algoritma SHA-1 dan RSA.

II. TANDA TANGAN DIGITAL (DIGITAL SIGNATURE) [3]

Aspek keamanan yang secara umum disediakan oleh kriptografi adalah sebagai berikut.

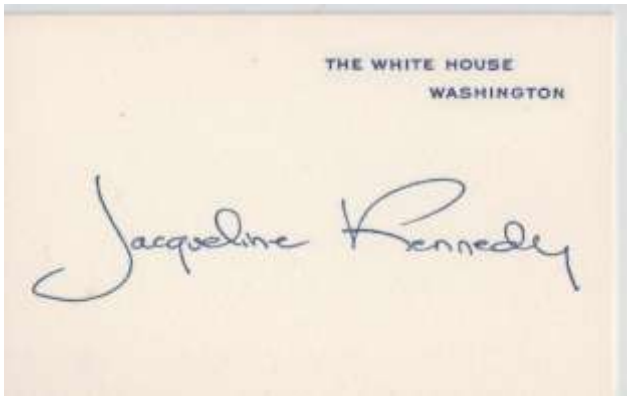
1. Kerahasiaan pesan (*confidentiality/secrecy*)
2. Otentikasi (*authentication*).
3. Keaslian pesan (*data integrity*).
4. Anti-penyangkalan (*nonrepudiation*).

Aspek satu diselesaikan dengan enkripsi/dekripsi. Aspek 2 sampai dengan 4 diselesaikan dengan tanda-

tangan digital (*digital signature*).

- Tanda-tangan mempunyai karakteristik sebagai berikut:
- Tanda-tangan adalah bukti yang otentik.
- Tanda tangan tidak dapat dilupakan.
- Tanda-tangan tidak dapat dipindah untuk digunakan ulang.
- Dokumen yang telah ditandatangani tidak dapat diubah.
- Tanda-tangan tidak dapat disangkal (*repudiation*).

Fungsi tanda tangan pada dokumen kertas juga diterapkan untuk otentikasi pada data digital (pesan, dokumen elektronik). Tanda-tangan untuk data digital dinamakan **tanda-tangan digital**. Tanda-tangan digital bukanlah tulisan tanda-tangan yang di-digitisasi (di-*scan*) seperti yang ada pada gbr 2. Akan tetapi, tanda-tangan digital adalah nilai kriptografis yang bergantung pada isi pesan dan kunci (contohnya ada pada gbr 3). Tanda-tangan pada dokumen cetak selalu sama, apa pun isi dokumennya. Tanda-tangan digital selalu berbeda-beda antara satu isi dokumen dengan dokumen lain.



Gambar 2 Tanda tangan yang di-*scan*

Kepada Yth.
Bapak Dekan
Di Tempat

Dengan hormat.
Bersama surat ini saya ingin mengabarkan bahwa nilai skripsi mahasiswa yang bernama Faisal Saleh dengan NIM 13902021 adalah 86,5 atau dalam nilai indeks A. Sidang skripsi sudah dilakukan pada Hari Rabu Tanggal 21 Januari 20 Juli 2005.

Atas perhatian Bapak saya ucapkan terima kasih.

Bandung, 25 Juli 2005

Dosen Pembimbing Skripsi

Ir. Ahmad Agus

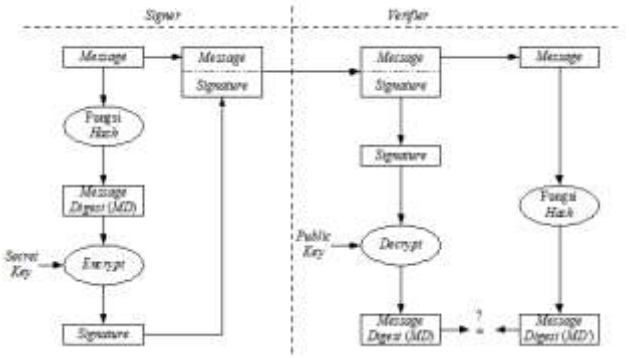
-----BEGIN PGP SIGNATURE-----

iQA/AwUAQnibsbPbxejK4Bb3EQJXvQCg8zN6UL0xnwBTPR5
FfWNt4uxh3AEAn2NC/G2VTUrLpcSyo2l/S/D/+rUl=pZeh

-----END PGP SIGNATURE-----

Gambar 3 Contoh tanda tangan digital

Salah satu cara yang dapat digunakan menandatangani pesan dengan tanda tangan digital adalah dengan menggunakan kombinasi fungsi *hash* (*hash function*) dan kriptografi kunci-publik. Pada umumnya, penandatangan-an pesan dengan cara mengenkripsinya selalu memberikan dua fungsi berbeda, yaitu kerahasiaan pesan dan otentikasi pesan. Namun pada beberapa kasus, seringkali otentikasi yang diperlukan, tetapi kerahasiaan pesan tidak. Maksudnya, pesan tidak perlu dienkripsikan, sebab yang dibutuhkan hanya keotentikan pesan saja. Perhatikan gbr 4 berikut.



Gambar 4 Skema penandatangan digital dan verifikasi

Keotentikan ini dijelaskan sebagai berikut:

a. Apabila pesan *M* (*Message*) yang diterima sudah berubah, maka *message digest MD*' yang dihasilkan dari fungsi hash berbeda dengan *message digest MD* semula. Ini berarti pesan tidak asli lagi.
b. Apabila pesan *M* tidak berasal dari orang yang sebenarnya, maka *MD* yang dihasilkan dari persamaan 3 berbeda dengan *MD'* yang dihasilkan pada proses verifikasi (hal ini karena kunci publik yang digunakan oleh penerima pesan tidak berkoresponden dengan kunci privat pengirim).
c. Bila *MD* = *MD'*, ini berarti pesan yang diterima adalah pesan yang asli (*message authentication*) dan orang yang mengirim adalah orang yang sebenarnya (*user authentication*).

Dua algoritma *signature* yang digunakan secara luas adalah *RSA* dan *ElGamal*. Pada *RSA*, algoritma enkripsi dan dekripsi identik, sehingga proses *signature* dan verifikasi juga identik. Langkah-langkah pemberian tanda-tangan dengan algoritma RSA adalah sebagai berikut.

1. Pengirim menghitung nilai *hash* dari pesan *M* yang akan dikirim, misalkan nilai *hash* dari *M* adalah *h*.

2. Pengirim mengenkripsi h dengan kunci privatnya menggunakan persamaan enkripsi *RSA* pada (1):

$$S = h^{SK} \textbf{ mod } n \quad (1)$$

yang dalam hal ini *SK* adalah kunci privat pengirim dan n adalah modulus ($n = pq$, p dan q adalah dua buah bilangan prima).
3. Pengirim mentransmisikan $M + S$ ke penerima

Sedangkan langkah-langkah untuk melakukan verifikasi tanda-tangan adalah sebagai berikut

1. Penerima menghitung nilai *hash* dari pesan M yang akan dikirim, misalkan nilai *hash* dari M adalah h'.
2. Penerima melakukan dekripsi terhadap tanda-tangan S dengan kunci publik si pengirim menggunakan persamaan dekripsi *RSA* pada (2);

$$h = S^{PK} \textbf{ mod } n \quad (2)$$

yang dalam hal ini *PK* adalah kunci privat pengirim dan n adalah modulus ($n = pq$, p dan q adalah dua buah bilangan prima).
3. Penerima membandingkan h dengan h'. Jika $h = h$' maka tanda-tangan digital adalah otentik. Jika tidak sama, maka tanda-tangan tidak otentik sehingga pesan dianggap tidak asli lagi atau pengirimnya

III. ALGORITMA RSA

RSA merupakan sebuah algoritma kunci publik yang banyak digunakan [4]. Algoritma ini memiliki kelebihan dalam keamanannya yang disebabkan sulitnya memfaktorkan bilangan yang besar menjadi faktor prima. Algoritma RSA memiliki properti sebagai berikut:

- p dan q bilangan prima (rahasia)
- $n = p$ x q (tidak rahasia)
- $\phi(n) = (p - 1)(q - 1)$ (rahasia)
- e (kunci enkripsi) (tidak rahasia)
  Syarat: PBB(e, $\phi(n)$) = 1
- d (kunci dekripsi) (rahasia)
  d dihitung dari $d \equiv e^{-1} \bmod (\phi(n))$
- m (plainteks) (rahasia)
- c (cipherteks) (tidak rahasia)

Terdapat 3 proses dalam penggunaan algoritma ini, yaitu :

1. Pembangkitan sepasang kunci
   a. Pilih dua bilangan prima, p dan q (rahasia)
   b. Hitung $n = pq$.
   c. Hitung $\phi(n) = (p - 1)(q - 1)$.
   d. Pilih sebuah bilangan bulat e untuk kunci publik, sebut, e relatif prima terhadap $\phi(n)$
   e. Hitung kunci dekripsi, d, dengan persamaan (3)

$$ed \equiv 1 \pmod{\phi(n)} \text{ atau } d \equiv e^{-1} \bmod (\phi(n)) \quad (3)$$

   Hasil dari algoritma di atas adalah :
   - Kunci publik adalah pasangan (e, n)
   - Kunci privat adalah pasangan (d, n)
2. Proses enkripsi
   a. Nyatakan pesan menjadi blok-blok plainteks: m_1, m_2, m_3, … (syarat: $0 < m_i < n - 1$)
   b. Hitung blok cipherteks c_i untuk blok plainteks p_i dengan persamaan (4)

$$c_i = m_i^e \textbf{ mod } n \quad (4)$$

   yang dalam hal ini, e adalah kunci publik.
3. Proses dekripsi
   Proses dekripsi dilakukan dengan menggunakan persamaan (5)

$$m_i = c_i^d \textbf{ mod } n \quad (5)$$

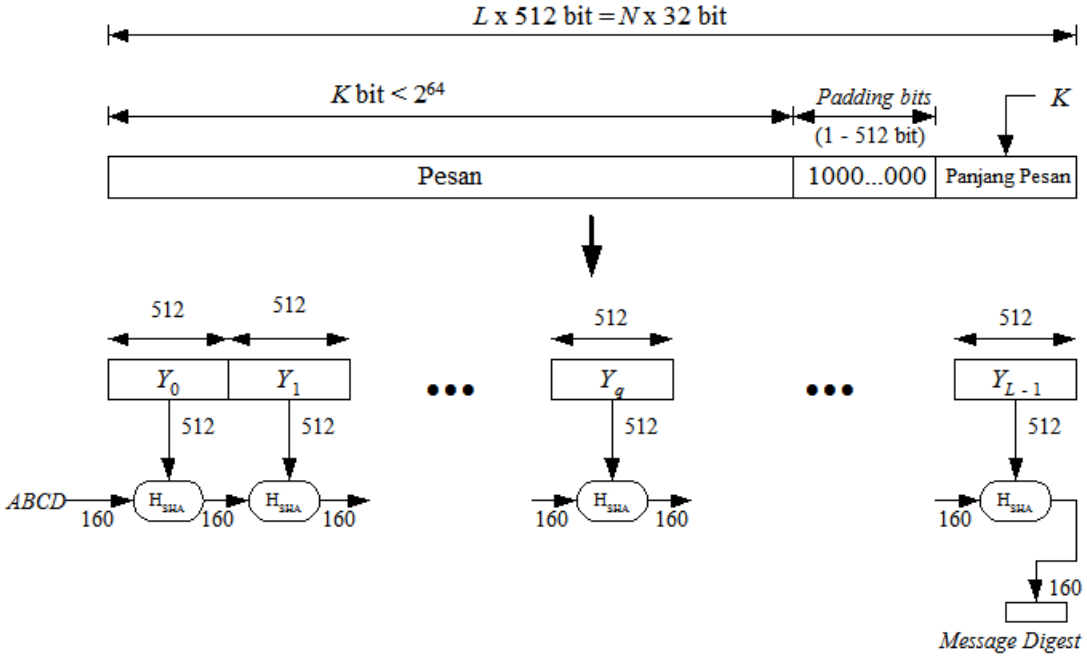
   yang dalam hal ini, d adalah kunci privat.

IV. SECURE HASH ALGORITHM 1 (SHA-1)

Dalam kriptografi, SHA-1 adalah salah satu fungsi *hash* kriptografi yang dirancang oleh *National Security Agency* dan diterbitkan oleh NIST sebagai *Federal Information Processing Standard* US. SHA merupakan singkatan dari Secure Hash Algorithm. Tiga algoritma SHA memiliki struktur yang berbeda sehingga dibedakan menjadi SHA-0, SHA-1, dan SHA-2. SHA-1 sangat mirip dengan SHA-0, namun memperbaiki kekurangan dari spesifikasi hash SHA asli yang menyebabkan kelemahan yang signifikan.
SHA-1 adalah yang paling banyak digunakan dibandingkan fungsi hash SHA lainnya, dan banyak digunakan dalam beberapa aplikasi keamanan dan protokol secara luas. Pada tahun 2005, ditemukan cacat keamanan pada SHA-1, yaitu bahwa kelemahan dari sisi matematika mungkin ada, mengindikasikan bahwa diperlukan fungsi hash yang lebih kuat [5]. Skema pembuatan hasil SHA (*message digest*) dengan SHA-1 ada pada gbr 5 [6].

SHA-1 menghasilkan nilai *hash* (disebut *message digest*) yang panjangnya 160 bit. Cara kerja SHA-1 adalah sebagai berikut. Plainteks dihitung panjangnya, lalu ditambah bit-bit pengganjal (*padding bits*) supaya panjangnya ≡ 448 (mod 512). Bila panjang pesan tepat sama dengan 448 bit, maka tetap ditambahi bit pengganjal sebanyak 512 bit sehingga panjang pesan 960 bit. Format bit pengganjal adalah satu buah bit 1 diikuti banyak bit 0 seperlunya. Contoh 1000000000000… . Setelah itu, panjang keseluruhan pesan dihitung. Hasil perhitungan ini sejumlah 64-bit ditambahkan ke ekor rangkaian bit pengganjal tadi. Sampai di sini, panjang pesan sudah

menjadi kelipatan 512 bit.



Gambar 5 Skema pembuatan message digest dengan SHA-1

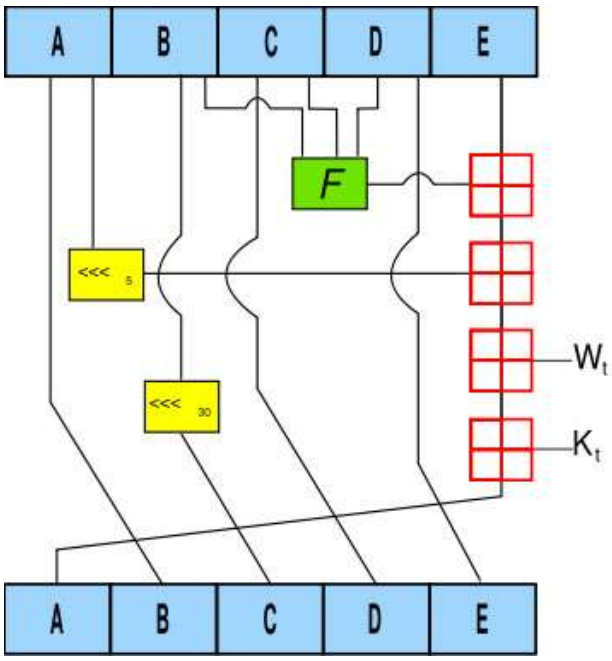
Pesan sepanjang tadi dibagi-bagi menjadi blok-blok 512 bit. Setiap blok dilewatkan ke fungsi H_{SHA} untuk dihitung *hash*-nya. Fungsi H_{SHA} menerima masukan blok 512 bit itu dan nilai *hash* dari H_{SHA} blok sebelumnya. Nilai *hash* yang final, yaitu *message digest* dari plainteks, adalah hasil keluaran dari H_{SHA} blok terakhir. Untuk blok pertama, digunakan 5 buah *buffer*, diberi nama A B C D E yang panjangnya masing-masing 32-bit sebagai nilai *hash dummy*. Isi dari *buffer* sebelum SHA-1 dihitung sudah ditetapkan sebagai berikut:

A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476
E = 0xC3D2E1F0

Gbr 6 merupakan gambar satu iterasi dalam fungsi kompresi SHA-1. A, B, C, D dan E adalah nilai buffer 32-bit. *F* adalah fungsi nonlinear yang bervariasi. Simbol $\lll n$ menunjukkan rotasi bit ke kiri sebanyak *n* tempat dan nilai *n* bervariasi untuk setiap operasi. W_t adalah pesan 2 byte pada putaran *t*. K_t adalah bilangan konstan pada putaran t. Simbol $\boxplus$ menunjukkan penambahan modulo 32 bit**.** Iterasi di atas diulang sebanyak 80 putaran.

Nilai *W* dihitung dari blok pesan 512-bit yang sedang diproses oleh H_{SHA} saat ini. Untuk putaran pertama, W_1 adalah 32 bit pertama dari blok 512 bit. Putaran kedua, W_2 adalah 32 bit berikutnya. Begitu seterusnya sampai W_{16}. Mulai putaran ke-17, nilai W_i dihitung dari rumus (6):

$$W_t = W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3} \qquad (6)$$



Gambar 6 Skema fungsi *hash* SHA-1

Nilai *K* adalah konstanta penambah. Nilainya sudah ditetapkan sebagai berikut:

Putaran ke-00 ≤ x ≤ 19; K = 0x5A827999
Putaran ke-20 ≤ x ≤ 39; K = 0x6ED9EBA1
Putaran ke-40 ≤ x ≤ 59; K = 0x8F1BBCDC
Putaran ke-60 ≤ x ≤ 79; K = 0xCA62C1D6

Terakhir ada fungsi *F*, apa yang dikerjakan juga bergantung pada putaran ke berapa saat itu. Isi fungsi *F* ada pada tabel i :

Tabel i 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	$b \oplus c \oplus d$

Setelah semua 80 putaran ini selesai, isi dari *buffer* A, B, C, D, dan E menjadi input bagi H_{SHA} blok berikutnya. Nilai *hash* final adalah gabungan rangkaian bit di dalam A-B-C-D-E hasil H_{SHA} blok terakhir.

V. Implementasi Skema Tanda Tangan Digital sebagai pada Microsoft Word 2007

Untuk menyelesaikan masalah yang telah dipaparkan sebelumnya, penulis akan mencoba mengimplementasikan tanda tangan digital yang merupakan gabungan dari algoritma RSA dan fungsi hash SHA sebagai pada Microsoft Word 2007. Program akan dikembangkan dengan bahasa pemrograman C# dan menggunakan Visual Studio 2008.

A. Spesifikasi Implementasi

yang dikembangkan diberi nama "Tanda Tangan Digital" dan memiliki spesifikasi sebagai berikut:

- Dapat melakukan pembangkitan kunci publik dan privat secara acak dan menyimpannya dalam file eksternal serta memuatnya dari file eksternal
- Dapat memberikan tanda tangan digital pada dokumen Microsoft Word 2007
- Dapat melakukan verifikasi terhadap dokumen yang telah ditandatangani

Berikut adalah daftar nama kelas yang telah dikembangkan untuk mendukung implementasi dari ini:

- BigInteger : kelas yang mengimplementasikan BigInteger dan operasinya
- RSA: kelas yang mengimplementasikan algoritma RSA
- SHA1: kelas yang mengimplementasikan algoritma SHA-1
- KeyGenerator: kelas yang mengimplementasikan pembangkit kunci pada algoritma RSA
- Display: kelas antarmuka yang menampilkan kunci yang dibangkitkan
- DigitalSignature: kelas antarmuka yang merupakan tampilan dari tanda tangan digital

B. Pembangkitan Kunci

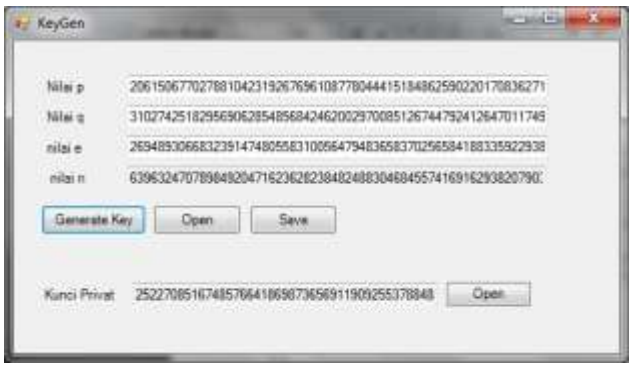
Kode program untuk pembangkitan kunci yang akan digunakan di dalam algoritma RSA ialah sebagai berikut:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace DigitalSignatureWord
{
 class KeyGenerator
 {
  public BigInteger p;
  public BigInteger q;
  public BigInteger e;
  public BigInteger d;
  public BigInteger n;

  public void GenerateKey()
  {
   Random R = new Random();
   p = gen48BytePrime();
   q = gen48BytePrime();
   n = p * q;
   BigInteger T = (p - 1) * (q - 1);
   e = gen48BytePrime();
   while (e.gcd(T) != 1)
   {
    e = gen48BytePrime();
   }
   d = e.modInverse(T);
  }

  public BigInteger gen48BytePrime()
  {
   Random R = new Random();
   BigInteger b = new BigInteger();
   while (!b.isProbablePrime())
   {
    b.genRandomBits(384, R);
   }
   return b;
  }

  public void saveKey()
  {
   String namaFile;
   SaveFileDialog fd = new
SaveFileDialog();
   fd.Filter = "Public Key Files
(*.pub)|*.pub";
   fd.ShowDialog();
   namaFile = fd.FileName;
   String isi = e.ToString(10) + " " +
n.ToString(10);
   ASCIIEncoding enc = new ASCIIEncoding();
   File.WriteAllBytes(namaFile,
enc.GetBytes(isi));

   fd.Filter = "Private Key Files
(*.pri)|*.pri";
   fd.ShowDialog();
   namaFile = fd.FileName;
   isi = "";
   isi = n.ToString(10) + " " +
d.ToString(10);
   File.WriteAllBytes(namaFile,
enc.GetBytes(isi));
  }

  public void openKey(int jenis)
  {
   p = new BigInteger(0);
```

```
    q = new BigInteger(0);
    e = new BigInteger(0);
    d = new BigInteger(0);
    n = new BigInteger(0);
    String namaFile;
    String isi;
    ASCIIEncoding enc = new ASCIIEncoding();

    if (jenis == 0)
    {
     OpenFileDialog fd = new
OpenFileDialog();
     fd.Filter = "Public Key Files
(*.pub)|*.pub";
     fd.ShowDialog();
     namaFile = fd.FileName;
     isi = File.ReadAllText(namaFile);
     String[] st = isi.Split(' ');
     Console.WriteLine(st.Length);
     e = new BigInteger(st[0], 10);
     n = new BigInteger(st[1], 10);
     Console.WriteLine(e.ToString());
     Console.WriteLine(n.ToString());
    }
    else if (jenis == 1)
    {
     OpenFileDialog fd = new
OpenFileDialog();
     fd.Filter = "Private Key Files
(*.pri)|*.pri";
     fd.ShowDialog();
     namaFile = fd.FileName;
     isi = File.ReadAllText(namaFile);
     String[] st = isi.Split(' ');
     Console.WriteLine(st.Length);
     n = new BigInteger(st[0], 10);
     d = new BigInteger(st[1], 10);
     Console.WriteLine(n.ToString());
     Console.WriteLine(d.ToString());
    }
   }

   public bool isPrime(int number)
   {
    BigInteger b = new BigInteger(number);
    return b.isProbablePrime();
   }

   public bool isCoprime(int e, int d)
   {
    BigInteger s = new BigInteger(e);
    BigInteger t = new BigInteger(d);

    return (s.gcd(d) == 1);
   }
  }
}
```

Fungsi `GenerateKey()` melakukan pembangkitan nilai-nilai untuk kunci publik (*e*), kunci privat, dan *n*. Kemudian nilai-nilai tersebut disimpan pada parameter yang ada dalam kelas KeyGenerator.

Antarmuka dari pembangkitan kunci ditampilkan oleh kelas Display dengan tampilan seperti pada gbr 7.



Gambar 7 Antarmuka pembangkit kunci RSA

C. Tanda Tangan

Fungsi yang digunakan untuk memberikan tanda tangan pada dokumen adalah fungsi `giveTTD`. Fungsi ini menerima masukan string isi dokumen dan menampilkan hasil verifikasi dengan menampilkan pesan MessageBox. Berikut adalah kode dari fungsi tersebut.

```
private String giveTTD(String pesan)
{
 //-----------------------------------
Membuat ttd
 //----- melakukan SHA
 //String pesan = "Sesdika Sansani";
 SHA1 sha1 = new SHA1(pesan);
 sha1.doSHA1();
 BigInteger hasilSHA = sha1.hasilAkhir;
 MessageBox.Show("hasil SHA: " +
hasilSHA.ToHexString());

 //----- melakukan RSA
 RSA R = new RSA(null);
 String strE = textKunciTTD.Text;
 String strN = textNttd.Text;
 BigInteger nilaiE = new BigInteger(strE,
10);
 BigInteger nilaiN = new BigInteger(strN,
10);

 String printhasil =
R.EnkripsiString(hasilSHA.ToHexString(),
nilaiE, nilaiN);
 debug("enc md", printhasil);

 return printhasil;
}
```

Proses yang dilakukan dalam fungsi ini adalah melakukan fungsi SHA-1 terlebih dahulu pada pesan. Setelah didapatkan hasilnya yaitu berupa message digest (MD), MD tersebut dienkripsi dengan algoritma RSA dan menghasilkan tanda tangan digital.

D. Verifikasi Dokumen

Fungsi yang digunakan untuk melakukan verifikasi pada dokumen adalah fungsi `doVerifikasi`. Fungsi ini menerima masukan string isi dokumen dan mengembalikan hasil tanda tangan digital sebagai tipe string. Berikut adalah kode dari fungsi tersebut.

```
private void doVerifikasi(String
hasilAkhir)
{
 if (!hasilAkhir.Contains("<ds>"))
 {
  MessageBox.Show("Tidak ada tanda tangan
digital pada email ini");
  return;
 }

 // menghitung md dari pesan -> SHA
 String pesan =
getPesanFromDigSig(hasilAkhir);
 debug("pesan ver:", pesan);
 SHA1 sha1 = new SHA1(pesan);
 sha1.doSHA1();
 BigInteger hasilSHA2 = sha1.hasilAkhir;
 String strHasilSHA =
hasilSHA2.ToHexString();
 debug("hasil SHA Ver:", strHasilSHA);

 // dekripsi MD pesan -> RSA
 String mdPesan =
getMDFromDigSig(hasilAkhir);
 debug("md pesan ver:", mdPesan);

 RSA R = new RSA(null);
 String strD = textKunciVer.Text;
 String strN = textNver.Text;
 BigInteger nilaiD = new BigInteger(strD,
10);
 BigInteger nilaiN = new BigInteger(strN,
10);
 String hasilDecMD =
R.DekripsiString(mdPesan, nilaiD, nilaiN);
 debug("dec md", hasilDecMD);

 // compare
 if (strHasilSHA == hasilDecMD)
  MessageBox.Show("Dokumen ini
terverifikasi");
 else
  MessageBox.Show("Verifikasi gagal,
dokumen telah diubah atau salah memasukkan
kunci");
}
```

Proses yang dilakukan dalam fungsi ini adalah mengecek terlebih dahulu apakah terdapat tanda tangan digital pada isi dokumen. Jika tidak ditemukan adanya tanda tangan digital pada dokumen tersebut, program akan menampilkan pesan bahwa tidak ada tanda tangan digital pada dokumen yang bersangkutan. Jika sebaliknya, program akan melakukan fungsi SHA-1 terlebih dahulu pada isi dokumen tanpa tanda tangan digital. Kemudian melakukan dekripsi pada tanda tangan digital yang ditempelkan pada dokumen. Kemudian hasil SHA-1 isi dokumen dan dekripsi tanda tangan digital yang ditempelkan dibandingkan. Jika sama, maka program akan menampilkan bahwa dokumen terverifikasi. Jika sebaliknya, program akan menampilkan pesan bahwa verifikasi gagal dan penyebab kegagalan yang mungkin terjadi.

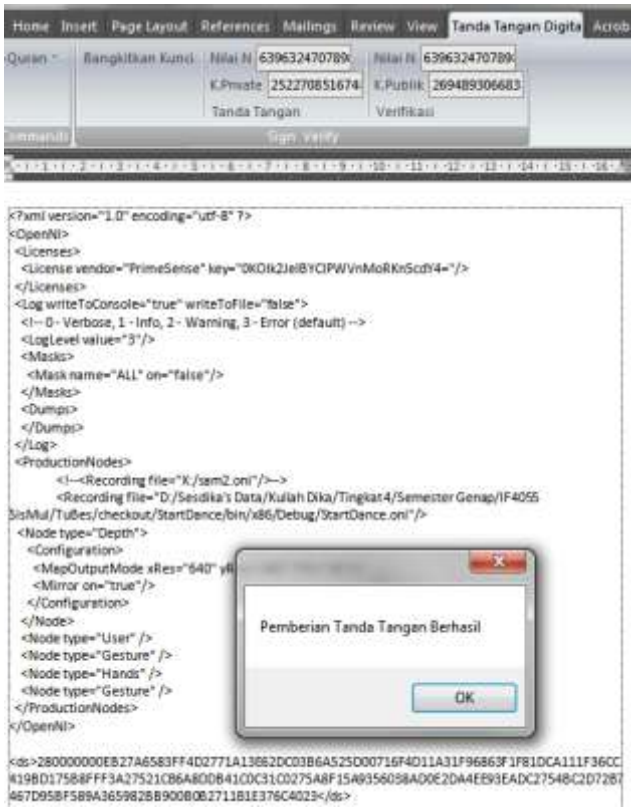
VI. PENGUJIAN DAN ANALISIS

Pada bab ini, penulis mencoba melakukan pengujian aplikasi yang mengimplementasikan tanda tangan digital gabungan antara algoritma SHA dan RSA. Kemudian penulis akan mencoba melakukan analisis terhadap hasil yang diberikan.

Untuk memberikan tanda tangan digital, hal pertama yang perlu dilakukan (jika belum memiliki pasangan kunci publik dan privat) adalah membangkitkan kunci, yaitu dengan menekan tombol ‘Bangkitkan Kunci’ pada Tanda Tangan Digital. Setelah itu, tekan tombol ‘Generate Key’ (lihat gbr 7) untuk mendapatkan pasangan kunci secara acak. Pengguna juga dapat menyimpan pasangan kunci yang dibangkitkan atau memuat kunci yang telah disimpan sebelumnya. Berikut adalah nilai-nilai kunci yang terlibat dalam algoritma RSA pada pengujian kali ini

Nilai *p*	20615067702788104231926769610877804441518486259022017083627117273712857660634346708664308141289301647342826939500911
Nilai *q*	31027425182956906285485684246200297008512674479241264701174900474883139924520407277278488594840011626934815089127041
Nilai Kunci Privat	25227085167485766418698736569119092553788483360930284177591902370732529198568656282003971552804821981952898104227072797270053404833720542277881551810550148096831542497202223632400026632861672325028855209135356826277640307898641190 3
Nilai Kunci Publik	26948930668323914748055831005647948365837025658418833592293d83167688295854786032361919572316650002461824415163522956 7
Nilai *n*	63963247078984920471623628238482488304684557416916293820790329581543357610695988226918445207119811312036462230919155578011420849458501533045228845002206579323299581258686494463634448946005888252341086778206342713250780926721423435 1

Program akan menerima masukan teks pada dokumen dan memberikan tanda tangan digital di akhir dokumen dengan diapit tag <ds></ds>. Pengguna perlu memasukkan terlebih dahulu nilai *n* dan kunci privat miliknya untuk memastikan bahwa pemberi tanda tangan adalah pemilik dokumen. Setelah itu pengguna perlu menekan tombol ‘Tanda Tangan’. Gambar 8 merupakan gambar dokumen yang telah diberikan tanda tangan digital.

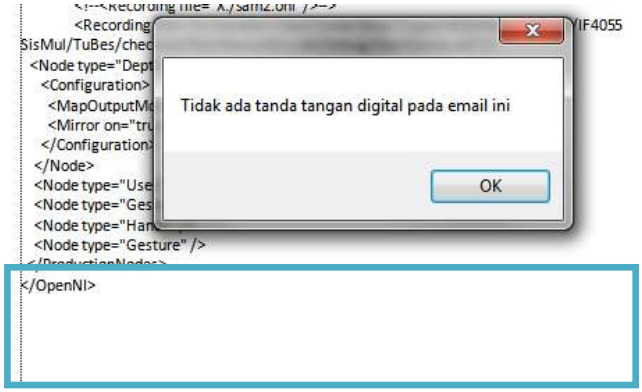


Gambar 8 Pemberian tanda tangan digital berhasil dan tanda tangan digital diletakkan di akhir dokumen

Setelah itu, dilakukan pengujian verifikasi pada dokumen asli. Pengguna cukup memasukkan nilai *n* dan kunci publik yang dimiliki oleh pembuat dokumen. Kemudian pengguna perlu menekan tombol 'Verifikasi' Gambar 9 merupakan tampilan jika dokumen masih asli.

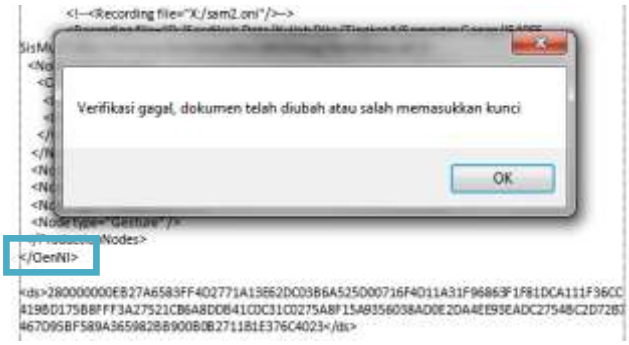
Gambar 9 Tampilan dokumen terverifikasi

Jika verifikasi dilakukan pada dokumen yang tidak memiliki tanda tangan digital, maka akan muncul pesan peringatan (gambar 10).

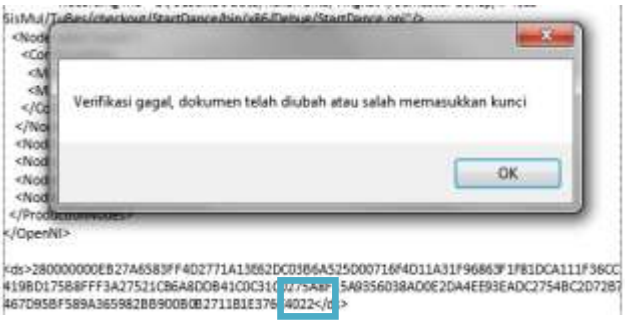


Gambar 10 Verifikasi pada dokumen yang tidak memiliki tanda tangan digital

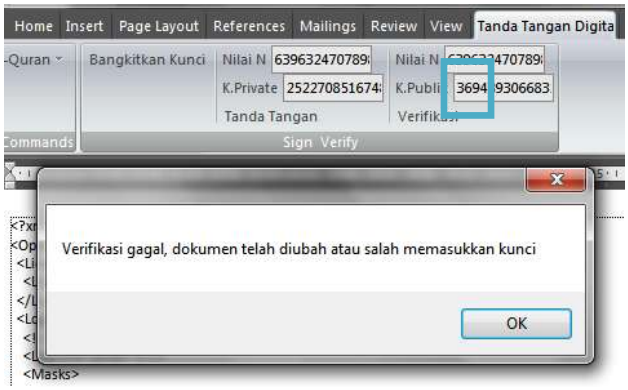
Kemudian pengujian dilakukan dengan melakukan pengubahan pada beberapa hal, yaitu mengubah isi dari dokumen, mengubah tanda tangan digital, dan mengubah kunci publik untuk melakukan verifikasi.



Gambar 11 Karakter di dalam dokumen diubah (dihapus, ditambah)



Gambar 12 Karakter di dalam tanda-tangan digital diubah



Gambar 13 Kunci publik diubah

Pada aplikasi ini, setelah tanda tangan dibuat, maka jika terdapat sedikit saja perubahan, baik perubahan di data (pada gbr 11) atau pada tanda tangan digital (pada gbr 12) dan juga jika kunci yang digunakan tidak berpadanan dengan kunci privatnya (pada gbr 13), maka akan terdeteksi bahwa terdapat perubahan pada pesan sehingga verifikasi gagal.

VII. KESIMPULAN

Kesimpulan dari makalah ini adalah sebagai berikut:

1. Untuk beberapa kasus, diperlukan otentikasi pembuat dokumen pada Microsoft Word untuk menjamin bahwa pembuat dokumen adalah pihak yang benar dan pesan tidak diubah oleh pihak manapun, misalnya pada surat waris atau surat wasiat.
2. Salah satu metode yang dapat digunakan untuk melakukan otentikasi pengguna dan validasi kebenaran isi pesan adalah dengan menggunakan tanda tangan digital, yaitu dengan gabungan algoritma RSA dan SHA-1.
3. Dengan menggunakan tanda tangan digital gabungan algoritma RSA dan SHA-1, pengguna dapat mengecek apakah suatu dokumen masih valid atau sudah diubah. Jika terjadi perubahan pada isi dokumen atau isi pesan atau kunci publik yang digunakan tidak sesuai dengan kunci privat pengirim (atau sebaliknya), program akan menampilkan bahwa verifikasi gagal. Jika tidak ada perubahan dan kuncinya sesuai, maka program memberikan pesan bahwa verifikasi berhasil.

REFERENSI

[1] http://id.wikipedia.org/wiki/Microsoft_Word
[2] http://www.nirmaltv.com/2008/10/11/how-to-add-a-digital-signature-in-word-2007/
[3] http://webmail.informatika.org/~rinaldi/Kriptografi/2010-2011/Tandatangan%20Digital.ppt
[4] http://webmail.informatika.org/~rinaldi/Kriptografi/2010-2011/Algoritma%20RSA.ppt
[5] http://en.wikipedia.org/wiki/SHA-1
[6] http://webmail.informatika.org/~rinaldi/Kriptografi/2010-2011/SHA.ppt

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, 9 Mei 2011

Sesdika Sansani
13507047