Tanda Tangan Digital Dengan Menggunakan SHA-256 Dan Algoritma Knapsack Kunci-Publik

Bhimantyo Pamungkas - 13504016
Program Studi Teknik Informatika ITB, Bandung 40132, email: btyo_pamungkas@yahoo.co.id

Abstract – *Di zaman serba digital dewasa ini banyak hal yang memiliki bentuk baru, salah satunya adalah dokumen tertulis yang sekarang dengan mudah dibuat dalam bentuk digital. Untuk menjamin keotetikasian dan keutuhan dari dokumen dalam bentuk digital tersebut ada metode yang disebut sebagai tanda tangan digital dengan kriptografi. Algoritma enkripsi yang biasa dipakai untuk tanda tangan digital ini adalah RSA yang merupakan salah satu algoritma enkripsi kunci publik. Dalam makalah ini penulis mencoba untuk mengimplementasikan tanda tangan digital menggunakan algoritma kunci publik selain RSA, yaitu Knapsack kunci publik dengan menggunakan fungsi hash SHA-256. Makalah ini akan membahas garis besar implementasi yang dilakukan serta uji coba terhadap tanda tangan digital menggunakan algoritma knapsack kunci publik dan SHA-256.*

Kata Kunci *tanda tangan digital, SHA-256, knapsack*

1. PENDAHULUAN

Tanda tangan digital merupakan sebuah metode yang digunakan untuk memberikan tiga aspek keamanan terhadap dokumen dalam bentuk digital, yaitu integritas data, otentikasi pembuat, dan nirpenyangkalan. Pembuatan tanda tangan digital biasanya dilakukan dengan melakukan fungsi *hash* terhadap dokumen kemudian mengenkripsi hasil dari fungsi *hash* tersebut dengan algoritma enkripsi kunci publik yang kuat seperti RSA dan ElGamal. Selain kedua algoritma tersebut, terdapat beberapa algoritma enkripsi kunci publik yang mungkin dapat digunakan dalam pembuatan tanda tangan digital, salah satunya adalah algoritma knapsack kunci-publik.

Pada proses pembuatan tanda tangan digital, dokumen akan menghasilkan *message digest* dengan ukuran bergantung pada fungsi *hash* yang digunakan.Fungsi *hash* SHA-256 merupakan salah satu varian fungsi *hash* SHA yang menghasilkan *message digest* dengan panjang 256 bit. Hasil dari fungsi SHA-256 tersebut kemudian dienkripsi dengan menggunakan algoritma knapsack kunci-publik. Dengan panjang *message digest* 256 bit dapat memberikan besar jumlah elemen dari algoritma knapsack yang lebih bervariasi yang meliputi seluruh faktor pembagi dari bilangan 256 termasuk nilai 256 itu sendiri, sehingga knapsack kunci-publik akan memiliki kunci dengan jumlah elemen paling besar adalah 256 dan hal tersebut dapat memperkuat algoritma knapsack kunci-publik.

Pada makalah ini akan dibahas apakah metode tanda tangan digital dengan menggunakan SHA-256 dan algoritma knapsack kunci-publik dapat digunakan sebagai salah satu metode alternatif dalam pembuatan tanda tangan digital. Pengujian akan dilakukan dengan berbagai kasus perubahan pada dokumen yang mempengaruhi proses penandatanganan dan verifikasi. Pengujian juga akan dilakukan untuk mengetahui apakah kecepatan proses penandatanganan dan verifikasi masih dapat diterima dengan baik

2. KONSEP DASAR

2.1. Knapsack kunci-publik

Ide dasar dari algoritma kriptografi *knapsack* adalah mengkodekan pesan sebagai rangkaian solusi dari persoalan *knapsack* dimana setiap bobot di dalam persoalan *knapsack* dikalikan dengan blok-blok plainteks yang diubah dalam bentuk bit. Misalkan sebuah *knapsack* dengan jumlah elemen 6 dengan bobot masing-masing 1, 5, 6, 11, 14, 20 dan dengan plainteks 111001010110000000011000. Plainteks dibagi menjadi blok yang panjangnya sama dengan panjang persoalan *knapsack*, kemudian setiap bit di dalam blok dikalikan dengan bobot yang berkoresponden, contoh:

Blok ke-1 : 111001
Knapsack : 1, 5, 6, 11, 14, 20
Kriptogram : (1x1)+(1x5)+(1x6)+(1x20) = 32

Jika diteruskan hingga seluruh blok plainteks dilakukan hal yang sama, maka cipherteks yang akan dihasilkan adalah 32-30-0-11.

Algoritma *knapsack* kunci-publik menggunakan *superincreasing knapsack* sebagai kunci privat yang merupakan persoalan *knapsack* dimana setiap elemen pada *knapsack* lebih besar daripada jumlah semua nilai sebelumnya, Sebagai contoh, *knapsack* dengan nilai-nilai elemennya adalah {1, 3, 6, 13, 27, 52} adalah *superincreasing*, tetapi {1, 3, 4, 9, 15, 25} bukan karena 15 lebih kecil dari pada 9+4+3+1. Untuk kunci publiknya, *superincreasing knapsack* dimodifikasi menjadi *non-superincreasing knapsack* dengan cara mengkalikan setiap elemen didalam barisan *superincreasing knapsack* dengan *n* modulo *m* dimana modulus *m* seharusnya angka yang lebih besar daripada jumlah semua elemen di dalam barisan dan pengali *n* seharusnya tidak mempunyai faktor persekutuan dengan *m* (relatif prima). Misalkan barisan *superincreasing* adalah {2, 3, 6, 13, 27, 52}

dengan m = 105 dan n = 31, maka barisan *non-superincreasing knapsack* dapat dihitung sebagai berikut:

2 x 31 mod 105 = 62
3 x 31 mod 105 = 93
6 x 31 mod 105 = 81
13 x 31 mod 105 = 88
27 x 31 mod 105 = 102
52 x 31 mod 105 = 37

Dengan demikian kunci publik yang dihasilkan adalah {62, 93, 81, 88, 102, 37}. Enkripsi dilakukan dengan cara yang sama seperti yang sudah dijelaskan sebelumnya. Sebagai contoh, misal plainteks adalah 011000110101101110, proses enkripsi untuk blok pertama dari plainteks adalah:

Blok ke-1 : 011000
Kunci : 62, 93, 81, 88, 102, 37
Kritogram : (1x93) + (1x81) = 174

Jika diteruskan cipherteks yang dihasilkan adalah 174, 280, 333. Untuk dekripsi pertama-tama perlu dihitung inversi n modulo m sedemikian sehingga $n \cdot n^{-1} = 1 \pmod{m}$ yang dapat dihitung dengan cara sederhana yaitu $n^{-1} = (1 + km) / n$ dengan k adalah sembarang bilangan bulat yang menghasilkan n^{-1} bilangan bulat. Dengan menggunakan contoh yang sebelumnya, nilai n^{-1} diperoleh dengan cara:

$n^{-1} = (1 + 105k) / 31$

Dengan mencoba berbagai nilai k, diperoleh nilai k = 18 sehingga nilai n^{-1} adalah 61. Cipherteks yang telah didapat sebelumnya akan didekripsi dengan cara:

174 x 61 mod 105 = 9 = 3+6 (011000)
280 x 61 mod 105 = 70 = 2+3+13+52 (110101)
333 x 61 mod 105 = 48 = 2+6+13+27 (101110)

Sehingga plainteks 011000110101101110 kembali dihasilkan melalui proses dekrispsi tersebut.

2.2. Tanda tangan digital

Tanda tangan digunakan untuk membuktikan otentikasi dokumen kertas. Tanda tangan memiliki karakteristik sebagai berikut:

- Tanda tangan adalah bukti yang otentik.
- Tanda tangan tidak dapat dilupakan.
- Tanda tangan tidak dapat dipindahtangankan.
- Dokumen yang telah ditandatangani tidak dapat berubah.
- Tanda tangan tidak dapat disangkal.

Fungsi tanda tangan pada dokumen kertas diterapkan untuk otentikasi pada data digital seperti pesan yang dikirim melalui saluran komunikasi dan dokumen elektronis yang disimpan di dalam memori komputer. Tanda tangan pada data digital ini dinamakan tanda tangan digital (*digital signature*). Yang dimaksud tanda tangan digital bukanlah tanda tangan yang didigitasi dengan alat *scanner*, tetapi suatu nilai kriptografis yang bergantung pada pesan dan pengirim pesan.

Dalam pemberian tanda tangan digital menggunakan *hash*, Pesan yang hendak dikirim diubah terlebih dahulu menjadi bentuk yang ringkas yang disebut *message digest*. *Message digest* (*MD*) diperoleh dengan mentransformasikan pesan M dengan menggunakan fungsi *hash* satu-arah (*one-way*) H,

$MD = H(M)$

Pesan yang sudah diubah menjadi *message digest* oleh fungsi *hash* tidak dapat dikembalikan lagi menjadi bentuk semula walaupun digunakan algoritma dan kunci yang sama (itulah sebabnya dinamakan fungsi *hash* satu-arah). Sembarang pesan yang berukuran apapun diubah oleh fungsi *hash* menjadi *message digest* yang berukuran tetap (umumnya 128 bit).
Selanjutnya, *message digest* MD dienkripsikan dengan algoritma kunci-publik menggunakan kunci rahasia (SK) pengirim menjadi sidik dijital S,

$S = E_{SK}(MD)$

Pesan M disambung (*append*) dengan sidik dijital S, lalu keduanya dikirim melalui saluran komunikasi . Dalam hal ini, kita katakan bahwa pesan M sudah ditandatangani oleh pengirim dengan sidik dijital S. Di tempat penerima, pesan diverifikasi untuk dibuktikan keotentikannya dengan cara berikut:
Sidik dijital S didekripsi dengan menggunakan kunci publik (*PK*) pengirim pesan, menghasilkan *message digest* semula, *MD*, sebagai berikut:

$MD = D_{PK}(S)$

Pengirim kemudian mengubah pesan M menjadi *message digest MD'* menggunakan fungsi *hash* satu-arah yang sama dengan fungsi *hash* yang digunakan oleh pengirim. Jika *MD' = MD*, berarti pesan yang diterima otentik dan berasal dari pengirim yang benar.

Keotentikan ini dijelaskan sebagai berikut:

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

dan orang yang mengirim adalah orang yang sebenarnya (*user authentication*).

3. IMPLEMENTASI

3.1. Lingkungan pengembangan

Spesifikasi komputer yang digunakan dalam melakukan implementasi tanda tangan digital ini adalah:

- Prosesor Intel Pentium 4 CPU 2.40GHz
- RAM 1.00GB

Lingkungan perangkat lunak yang digunakan adalah sebagai berikut:

- Sistem operasi Windows XP SP2
- .Net Framework 2.0
- Microsoft Visual Studio 2005
- Bahasa pemograman Visual Basic .NET

Visual Basic .NET dipilih sebagai bahasa pemograman karena memiliki fungsi bawaan yang memudahkan untuk konversi data karakter ke dalam bentuk byte dan bit atau sebaliknya.

3.2. Batasan implementasi

Batasan-batasan implementasi pembuatan tanda tangan digital ini adalah sebagai berikut:

- Fungsi *hash* SHA-256 tidak dibuat sendiri melainkan memakai bawaan dari library .NET
- Tanda tangan digital tidak melibatkan dokumen secara langsung, plainteks hanya berupa input yang bisa di salin dari dokumen teks.

3.3. Fungsi-fungsi pendukung

Dalam implementasi penandatanganan dan pengesahan tanda tangan digital, terdapat fungsi-fungsi kecil yang membantu dan tidak dapat dijabarkan dengan detil. Oleh karena itu, nama fungsi serta apa yang dilakukan oleh fungsi-fungsi tersebut akan dijelaskan dengan singkat sebagai berikut:

- GenInvN => menghitung n^{-1} berdasarkan nilai *n* dan *m*
- ValToKnap => membentuk array bit berdasarkan kunci dan nilai cipherteks
- KnapToString => membentuk string berisi bit-bit dari array bit
- StringToUlong => mengubah string cipherteks menjadi array of unsigned long

3.4. Fungsi pembentuk kunci privat

Kunci privat pada *knapsack* kunci-publik adalah sebuah *superincreasing*. Pembentukan kunci privat cukup mudah yaitu dengan mengiterasi hingga panjang yang diinginkan dimana setiap iterasi membentuk elemen kunci dengan jumlah elemen sebelumnya ditambah dengan 1(satu) untuk memudahkan. Fungsi ini membutuhkan nilai awal sebagai nilai dari elemen pertama dari kunci serta panjangnya kunci.

```
function MakePriKey(
  ival:long,
  jml:integer
)-> array of unsigned long

Deklarasi:
  pkey:array of unsigned long;
  cumul:unsigned long;
  i:integer;

Algoritma:
  begin
    cumul := ival;
    pkey[0] := ival;
    for i:=1 to jml-1 do
    begin
      pkey[i]:= cumul+1;
      cumul:= cumul + pkey[i];
    end;
    {endfor}
    return pkey;
  end.
```

3.5. Fungsi pembentuk kunci publik

Kunci publik dibuat dengan kunci privat, nilai *m*, dan nilai *n* sebagai masukan terhadap fungsi. Proses pembentukan kunci publik juga cukup sederhana, melakukan iterasi setiap elemen kunci privat dimana setiap iterasi elemen dari kunci privat dikalikan dengan nilai *n* lalu di modulo dengan nilai *m*.

```
function MakePubKey(
  prikey:array of unsigned long,
  mval:unsigned long,
  nval:unsigned long
)-> array of unsigned long

Deklarasi:
  pkey:array of unsigned long;
  i:integer;

Algoritma:
  begin
    for i:= 0 to prikey.length-1 do
    begin
     pkey[i]:= (prikey[i] * nval) % mval;
    end;
    {endfor}
    return pkey;
  end.
```

3.6. Fungsi enkripsi

Fungsi untuk proses enkripsi membutuhkan kunci publik dan array bit hasil konversi *hash* yang dihasilkan oleh SHA-256 menjadi array bit yang dapat dilakukan dengan menggunakan fungsi bawaan dari .NET. Proses enkripsi adalah menghitung jumlah elemen cipherteks yang akan dihasilkan dengan membagi panjang array bit dengan panjang kunci. Setelah itu melakukan iterasi sebanyak panjang cipherteks dimana setiap iterasinya nilai array bit dikalikan dengan nilai elemen pada kunci lalu dijumlahkan. Iterasi untuk elemen terakhir cipherteks dilakukan penanganan khusus terhadap batas akhir iterasi untuk jika panjang kunci tidak habis membagi panjang array bit.

```
function Encrypt(
  key:array of unsigned long,
  bb:array of bit
)-> array of unsigned long
```

```
Deklarasi:
  cipher:array of unsigned long;
  i,j,e:integer;

Algoritma:
  begin
    cipher.length:=(bb.length/key.length)
    for i:=0 to cipher.length-1 do
    begin
      if i=cipher.length-1 then
      begin
        e:=(bb.length-1-(i*key.length));
      end;
      else
      begin
        e:= key.length-1;
      end;
      {endif}
      for j:=0 to e do
      begin
        cipher[i]:= key[j] * bb[(i*
key.length) + j];
      end;
      {endfor}
    end;
    {endfor}
    return cipher;
  end.
```

3.7. Fungsi dekripsi

Fungsi untuk melakukan dekripsi memerlukan banyak masukan yaitu string cipherteks, kunci, nilai *m*, dan nilai *n*. Yang pertama dilakukan adalah menghitung nilai n^{-1} kemudian mengubah bentuk string cipher menjadi array of unsigned long untuk mempermudah proses dekripsi. Kemudian dilakuan iterasi terhadap elemen cipherteks untuk dihitung solusi *knapsack* dalam bentuk array bit lalu array bit tersebut diubah dalam bentuk string untuk mempermudah proses penggabungan dengan nilai-nilai bit pada elemen cipherteks lainnya. Karena nilai *hash* dari SHA-256 selalu tetap yaitu 256 bit, maka hasil dekripsi akan dihilangkan bit-bit berlebih dimulai pada index 256. Penghilangan bit berlebih tersebut aman dilakukan karena bit-bit berlebih tersebut pasti bernilai 0 dan jika tidak dihilangkan akan mengacaukan perbandingan hasil dekripsi dengan hasil *hash* plainteks pada saat verifikasi. Hasil akhir dekripsi sengaja dibuat menjadi bentuk string karena memudahkan perbandingan dengan hasil *hash* yang telah disinggung sebelumnya.

```
function Decrypt(
  cipher:string,
  prikey:array of unsigned long,
  mval:unsigned long,
  nval:unsigned long
)-> string

Deklarasi:
  inval:unsigned long;
  cp:array of unsigned long;
  sval:string;
  i:integer;

Algoritma:
  begin
    inval:= GenInvN(mval,nval);
    cp:= StringToUlong(cipher);
    for i:=0 to cp.length-1 do
    begin
      sval:= sval + KnapToString(
ValToKnap((cp[i] * ival) % mval, prikey)) ;
    end;
    {endfor}
    sval:= remove(sval,256);
    return sval;
  end.
```

4. PENGUJIAN

4.1. Uji tanda tangan digital

Pengujian dilakukan dengan menggunakan abstrak dari makalah ini sebagai plainteks dengan hasil *hash* sebagai berikut:

m*·Ä÷i× m
b§ëó×7 gÚW ¼m , øaô

Dengan nilai bit-nya adalah:

00110110010101001110110100100011111110001110111110010110111010110110011011011001010000011100010100011011100101111100111111110101111110110011001001111001101111111010100111100100111101101101100100100000110100100100011111100100011111100001100010111

Kunci privat yang digunakan adalah:

3-4-8-16-32-64-128

Kunci publik yang dihasilkan dengan nilai *m* = 255 dan nilai *n* = 13 adalah:

39-52-104-208-161-67-134

Menghasilkan tanda tangan digital seperti dibawah ini:

513-399-607-210-674-292-713-332-490-371-384-386-201-334-503-386-500-396-698-592-319-414-408-552-490-540-414-552-223-52-433-238-564-401-195-158-403

Tanda tangan digital diatas terverifikasi sebagai valid. Dengan menggunakan hasil diatas sebagai kontrol, dilakukan beberapa kasus uji sebagai berikut beserta hasil uji.

Jenis uji	Hasil uji
Penghilangan satu huruf pada plainteks	Tidak valid
Penghilangan satu kalimat pada plainteks	Tidak valid
Pengubahan satu huruf pada plainteks	Tidak valid
Pengubahan satu kalimat pada plainteks	Tidak valid
Penghilagnan satu elemen pada tanda tangan digital	Tidak valid
Pengubahan satu elemen pada tanda tangan digital	Tidak valid
Penggunaan kunci yang tidak sesuai	Tidak valid
Pengubahan nilai *m*	Tidak valid
Pengubahan nilai *n*	Tidak valid
Pengubahan satu bit pada bit array	Tidak valid
Menukar kunci publik dengan kunci privat	Tidak valid

Dari hasil pengujian dapat dilihat bahwa algoritma *knapsack* kunci publik jika digunakan sebagai tanda tangan digital tahan pada perubahan yang terjadi pada berbagai kasus. Tetapi sayangnya jika tanda tangan digital dilakukan dengan kunci privat lalu diverifikasi dengan menggunakan kunci publik akan menghasilkan tidak valid. Hal tersebut menandakan bahwa penandatanganan harus menggunakan kunci publik dan verifikasi harus menggunakan kunci privat. Karena kunci publik dapat diketahui jika mengetahui kunci privat, maka tidak dapat dilakukan penyebaran kunci seperti jika menggunakan algoritma seperti RSA.

4.2. Uji kapasitas

Pengujian kapasitas ini dilakukan untuk mengetahui keterbatasan dari lingkungan implementasi dan program yang telah dibangun. Dari berbagai pengujian yang dilakukan ditemukan beberapa batasan teknis yang ditemukan sebagai berikut:

- Panjang kunci paling panjang yang dapat dihasilkan adalah 64 elemen
- Jika panjang kunci terlalu panjang, terkadang terjadi overflow

5. KESIMPULAN

Berdasarkan pengujian yang telah dilakukan, dapat ditarik beberapa kesimpulan terkait dengan penggunaan algoritma *knapsack* kunci-publik dan SHA-256 sebagai tanda tangan digital:

- Algoritma tersebut dapat menjaga integritas dokumen karena tahan pada berbagai macam perubahan yagn dilakukan pada dokumen maupun pada tanda tangan digital itu sendiri.
- Penggunaan tanda tangan digital dengan algoritma ini tidak cocok untuk dipakai secara luas karena keterbatasan pada sifat algoritma dan kunci yang tidak dapat melakukan tanda tangan dan verifikasi dengan baik jika penggunaan kunci ditukar (kunci privat untuk tanda tangan dan kunci publik untuk verifikasi)
- Tidak cocoknya penggunaan *knapsack* kunci-publik juga karena sudah ada rumus transformasi untuk merekonstruksi *superincreasing knapsack* dari *normal knapsack*.
- Keterbatasan tipe dasar unsigned long pada Visual Basic .NET hanya memungkinkan pembentukan dan penanganan kunci untuk proses enkripsi sepanjang 64 elemen., sedangkan panjang yang disarankan untuk memperkuat algoritma *knapsack* adalah paling sedikit 250 elemen.

DAFTAR REFERENSI

[1] Munir, Rinaldi, *Diktat Kuliah IF5054 Kriptografi*, Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2006.