Pemanfaatan Metode Pembangkitan Parameter RSA untuk Modifikasi SHA-1

Miftah Mizan – NIM : 13507064

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
E-mail : if17064@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang penggabungan metode pembangkitan parameter RSA dengan fungsi hash SHA-1. SHA-1 adalah salah satu fungsi hash yang cukup populer dan banyak digunakan. Sayangnya fungsi tersebut memiliki kelemahan karena bisa terkena collision attack dimana dua dokumen yang sangat berbeda bisa memiliki message digest yang sama. Hal tersebut juga menyebabkan SHA-1 mudah diserang oleh orang lain dan diubah isi pesannya. Untuk itu diperlukan suatu modifikasi agar SHA-1 memiliki algoritma yang lebih kompleks dan dapat mengurangi kemungkinan terjadinya collision. Salah satu cara yang coba diusulkan melalui makalah ini adalah dengan menambahkan variabel berupa parameter yang dihasilkan dari metode pembangkitan parameter RSA ke dalam algoritma SHA-1.

Dalam makalah ini akan dilakukan eksperimen dan implementasi dari hasil modifikasi SHA-1 tersebut dengan bantuan parameter yang digenerate dengan prinsip yang sama dalam pembangkitan parameter RSA. Eksperimen dan implementasi akan dilakukan dalam program kecil dengan bahasa *Java.* Selain itu juga akan dilakukan analisa dan perbandingan antara hasil modifikasi yang baru dengan SHA-1 yang sebelumnya. Pada analisa dan perbandingan tersebut, juga akan dilihat seberapa signifikan penambahan variabel baru tadi pada message digest yang dihasilkannya. Sensitivitas dari hasil modifikasi SHA-1 juga akan diuji dan dibandingkan dengan SHA-1 yang sebelumnya.

***Kata kunci**: SHA-1, collision,* pembangkitan parameter RSA, modifikasi, eksperimen

1. Pendahuluan

Seiring perkembangan ilmu kriptografi, orang mulai banyak menerapkannya untuk meningkatkan keamanan informasi pada pengiriman pesan. Tentunya hal tersebut diiringi dengan perkembangan teknologi untuk memecahkan kriptografi yang ada. Sehingga sekaranng perkembangan teknologi kriptografi semakin meningkat, baik dari segi enkripsi pesan maupun untuk memecahkan pesan tersebut.

Namun ternyata dalam ilmu kriptografi yang begitu luas, yang berkembang tidak hanya teknik untuk mengenkripsi atau mendekripsi pesan. Salah satu teknik yang banyak digunakan dalam kehidupan sekarang adalah teknik otentikasi dan verifikasi.

Teknik otentikasi dan verifikasi ini digunakan untuk memastikan keaslian dan keutuhan pesan yang dikirim oleh seseorang. Dengan kata lain, pesan tersebut tidak dimodifikasi oleh orang lain dan juga tidak mengalami perubahan apapun pada isi pesannya.

Salah satu fungsi yang ditawarkan untuk mengatasi masalah tersebut adalah fungsi *hash*. Fungsi *hash* adalah suatu fungsi satu arah (*one-way function*) yang menghasilkan suatu nilai unik dari pesan tertentu. Fungsi *hash* ini akan menghasilkan nilai dengan panjang yang sama namun bernilai unik untuk setiap pesan.

Fungsi *hash* banyak diterapkan pada berbagai algoritma, salah satunya adalah *Secure Hash Algorithm* (*SHA*). SHA adalah suatu algoritma yang dibuat *NIST* dan diimplementasikan bersama banyak teknik kriptografi lain seperti DSS atau RSA. SHA dikatakan aman karena jika pesan yang dikirim diubah sedikit saja, maka akan dihasilkan *message digest* yang berbeda dengan *message digest* dari pesan yang asli. Tentunya hal tersebut akan mengakibatkan kegagalan verifikasi jika kita akan mengecek keaslian suatu pesan.

Namun sayangnya SHA-1 yang merupakan fokus dari makalah ini memiliki kelemahan tersendiri karena dapat diserang sehingga memunculkan *collision.* Untuk itu diperlukan

suatu metode agar algoritma SHA-1 lebih kompleks dan tidak mudah diserang.

Pada sisi lain terdapat suatu metode yang menarik dalam membangkitkan suatu parameter acak pada algoritma pembangkitan parameter di RSA. Metode pembangkitan parameter ini menggunakan bilangan prima random yang nantinya menghasilkan *public key* dan *private key* yang dibuthkan dalam algoritma RSA. Metode ini merupakan salah satu metode yang efektif digunakan karena dengan algoritma ini, hasil dari enkripsi yang dilakukan cukup sulit untuk dipecahkan. Selain itu dengan metode pembangkitan parameter tersebut, hasil dari algoritma RSA tersebut dapat dipastikan selalu acak untuk setiap pesan bahkan untuk setiap perubahan pesan yang terjadi.

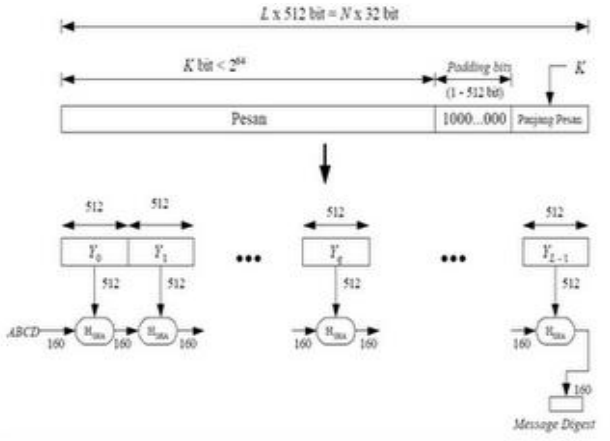
Dengan melihat dua fakta tersebut, muncul suatu ide untuk menggabungkan algoritma SHA-1 dengan metode pembangkitan parameter dalam RSA sehingga menghasilkan modifikasi SHA-1 dalam bentuk yang lebih kompleks. Dari hasil modifikasi tersebut juga diharapkan dapat membuat algoritma SHA-1 yang lebih kuat sehingga tidak mudah terkena *collision attack*. Selain itu hasil dari modifikasi ini juga perlu analisa lebih mendalam dibandingkan dengan algoritma SHA-1 sebelumnya.

2. Dasar Teori

SHA-1

Algoritma SHA-1 menerima masukan berupa pesan dengan ukuran maksimum 2^{64} bit dan menghasilkan *message digest* yang panjangnya 160 bit.

Secara keseluruhan proses SHA-1 dapat digambarkan sebagai berikut :



Langkah-langkah pembuatan message digest secara garis besar adalah sebagai berikut :

1. Padding bits
   Pesan yang masuk akan ditambah sejumlah bit sehingga panjang pesan dapat menjadi kongruen dengan 448 modulo 512. Dengan kata lain panjang pesan setelah ditambah bit nantinya adalah 64 bit kurang dari kelipatan 512. Hal ini disebabkan algoritma SHA-1 memproses pesan dalam blok-blok yang berukuran 512.

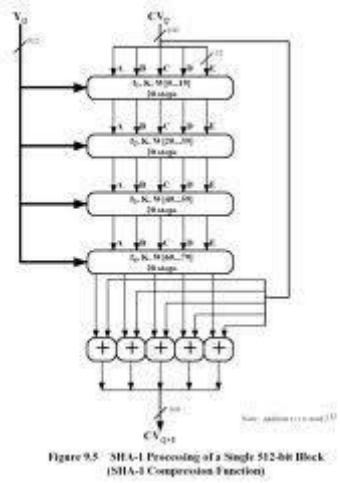
   Meskipun pesan yang dimasukkan sudah memiliki panjang 448 bit, maka akan tetap ditambah bit-bit sepanjang 512. Bit-bit ini terdiri dari sebuah bit 1 dan diikuti degnan sisanya bit 0.

2. Penambahan nilai panjang pesan
   Pesan yang telah diberikan bit-bit tambahan ditambah lagi dengan 64 bit yang menyatakan panjang pesan awal. Sehingga sekarang panjang pesan yang dipunyai merupakan kelipatan 512.

3. Initializing buffer
   SHA-1 membutuhkan 5 buah *buffer* yang memiliki panjang 32 bit. Setiap *buffer* dinisialisasi dengan nilai-nilai sebagai berikut :

   h0 = 0x67452301
   h1 = 0xEFCDAB89
   h2 = 0x98BADCFE
   h3 = 0x10325476
   h4 = 0xC3D2E1F0

4. Pemrosesan pesan per blok



a. Pertama kali, pesan dibagi menjadi sejumlah n blok dengan panjang masing-masing 512 bit.
b. Kemudian untuk setiap blok ke n akan dibagi kembali menjadi 16 (0 - 15) blok dengan ukuran 32 bit.
c. Setelah itu akan dilakukan *extend* hingga menjadi ada 80 blok dengan ukuran 32 bit dimana blok 16 – 79 dibuat dengan menggunakan rumus seperti berikut :

   w[i] = (w[i-3] **xor** w[i-8] **xor** w[i-14] **xor** w[i-16]) **leftrotate** 1

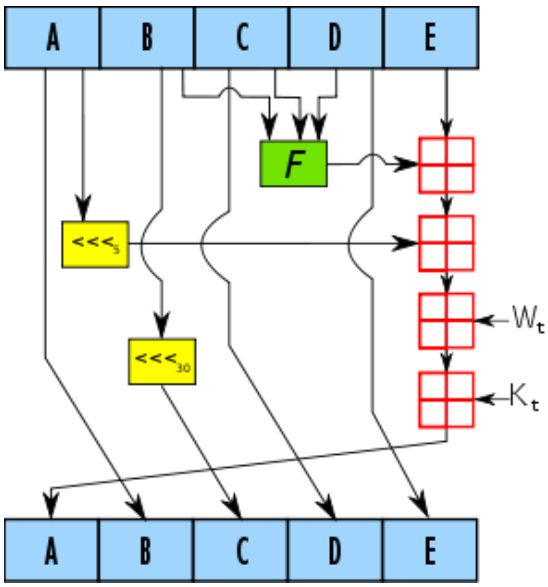
   dengan i adalah angka 16 – 79.
d. Setiap blok dari 0 – 79 akan mengalami pemrosesan fungsi logika untuk menghasilkan nilai f dengan rincian sebagai berikut :

Putaran	$f_t(b,c,d)$
0 .. 19	$(b \wedge c) \vee (\neg b \wedge d)$
20 .. 39	$b \oplus c \oplus d$
40 .. 59	$(b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
60 .. 79	$b \oplus c \oplus d$

dan juga menentukan bilangan tambahan yang akan dipakai di setiap putarannya, dengan rincian sebagai berikut :

Putaran $0 \le t \le 19$	$K_t = 5A827999$
Putaran $20 \le t \le 39$	$K_t = 6ED9EBA1$
Putaran $40 \le t \le 59$	$K_t = 8F1BBCDC$
Putaran $60 \le t \le 79$	$K_t = CA62C1D6$

e. Setelah itu setiap putaran akan mengalami proses yang sama yaitu seperti pada gambar di bawah :



Proses yang terjadi dalam gambar memiliki rincian sebagai berikut :

A = h0
B = h1
C = h2
D = h3
E = h4

Setelah diinisialisasi, maka akan dilanjutkan ke proses berikut :

temp = (A **leftrotate** 5) + f + E + k + w[i]
E = D
D = C
C = B **leftrotate** 30
B = A
A = temp

Kemudian nilai yang baru dari A, B, C, D, E akan ditambahkan masing-masing ke h0, h1, h2, h3, dan h4. Proses ini

akan terus diulang dari hingga blok ke 80.

5. *Message digest*
   Message digest yang didapat merupakan hasil gabungan dari h0, h1, h2, h3, dan h4 yang telah melewati keseluruhan proses tersebut. Hasilnya akan kembali ditulis dalam bentuk notasi hexa.

RSA

Dalam kriptografi, RSA adalah sebuah algoritma untuk kriptografi kunci publik yang pertama dikenal. RSA juga merupakan algoritma yang cocok digunakan untuk tanda tangan digital dan juga enkripsi. Algoritma RSA meliputi tiga tahap yaitu *key generation*, enkripsi and dekripsi. Tahap yang menjadi fokus dalam makalah ini adalah tahap *key generation*.

Key Generation	
Select p, q	p, q both prime, p≠q
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1) \times (q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$
Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod n$
Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod n$

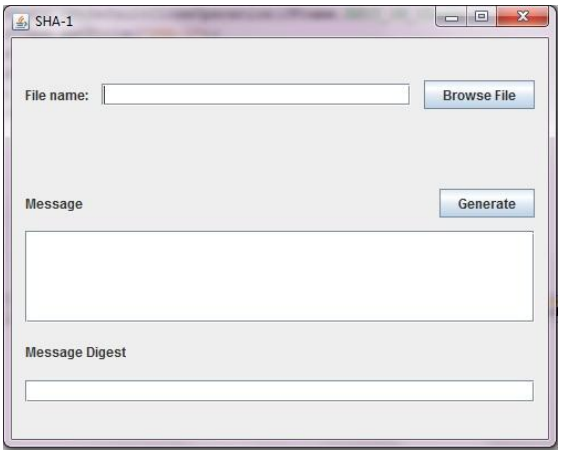
Tahap *key generation* meliputi langkah-langkah berikut :

- Tentukan bilangan prima (p dan q)
- Hitung modulus (*n*)
  n = pq; n akan digunakan sebagai modulus untuk *public key* dan privatee key
- Hitung φ (pq) = (p - 1)(q - 1)
- Tentukan nilai *e*
  Nilai e harus memenuhi persyaratan $1 < e < \varphi$ dan juga nilai φ harus *coprime* dengan *e*. Nilai *e* ini akan digunakan sebagai eksponen *public key*.
- Tentukan nilai *d*
  Nilai *d* harus memenuhi persyaratan *de* mod $\varphi = 1$. Nilai *d* ini akan digunakan sebagai eksponen *public key*. Sehingga akhirnya didapatkan *public key* (*n*, *e*) dan *private* key (*n*, *d*).

3. Eksperimen dan Implementasi

Pada tahap berikut ini, akan dilakukan suatu eksperimen untuk memodifikasi algoritma SHA-1 yang sudah ada dengan metode pembangkitan parameter yang dilakukan di algoritma RSA.

Untuk pertama kali, akan dibuat sebuah program dengan bahasa Java untuk membuat algoritma SHA-1 yang normal. Program ini dapat menerima masukan dari keyboard ataupun dari file teks. Kemudian program akan memproses dan menampilkan *message digest* yang dihasilkan. Berikut adalah interface dari program yang dibuat :



Ide eksperimen yang akan dilakukan disini adalah dengan melakukan modifikasi pada tahap ke empat dalam SHA-1 yaitu pada pemrosesan data per blok. Bagian yang akan dimodifikasi adalah bagian ketika setiap blok dari 0-79 mengalami pemrosesan fungsi logika untuk menghasilkan nilai f. Pada bagian tersebut akan dibuat suatu variable baru bernama x yang merupakan hasil dari penggunaan metode pembangkitan parameter RSA tersebut.

Nilai x tersebut didapat dengan cara melakukan tahap-tahap pembangkitan parameter seperti dalam algoritma RSA. Kemudian pada akhir tahap tersebut dimana nilai *private key* telah didapat, maka nilai tersebut akan dijadikan nilai x.

Nilai p tersebut didapat melalui proses sebagai berikut :

```
public int genp() {
            while (p == q){
                p                    =
this.genprime();
                q                    =
this.genprime();
            }
```

```
            n = p * q;
            phi = (p - 1) * (q -
1);
            pubkey = genpub();
            prikey = genpri();
           return prikey;
        }

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;
}
```

Sehingga pada tahap terjadinya penentuan nilai f yang pada awalnya sebagai berikut :

```
for i from 0 to 79
        if 0 ≤ i ≤ 19 then
            f = (b and c) or
((not b) and d)
            k = 0x5A827999
        else if 20 ≤ i ≤ 39
            f = b xor c xor d
            k = 0x6ED9EBA1
        else if 40 ≤ i ≤ 59
            f = (b and c) or (b
and d) or (c and d)
            k = 0x8F1BBCDC
        else if 60 ≤ i ≤ 79
            f = b xor c xor d
            k = 0xCA62C1D6
```

menjadi seperti berikut :

```
for i from 0 to 79
        if 0 ≤ i ≤ 19 then
            f = (b and c) or
((not b) and d)
            k = 0x5A827999
            x = genp();
        else if 20 ≤ i ≤ 39
            f = b xor c xor d
            k = 0x6ED9EBA1
            x = genp();
        else if 40 ≤ i ≤ 59
            f = (b and c) or (b
and d) or (c and d)
            k = 0x8F1BBCDC
            x = genp();
        else if 60 ≤ i ≤ 79
            f = b xor c xor d
            k = 0xCA62C1D6
            x = genp();
```

Kemudian pada bagian proses utama yang dilakukan pada setiap putaran yang pada awalnya berbentuk sebagai berikut :

```
temp = (a leftrotate 5) + f + e +
k + w[i]
e = d
d = c
c = b leftrotate 30
b = a
a = temp
```

menjadi seperti berikut :

```
temp = (a leftrotate 5) + f + e +
k + w[i] + x
e = d
d = c
c = b leftrotate 30
b = a
a = temp
```

Setelah melalui proses tersebut, tidak ada lagi bagian yang dimodifikasi. Metode modifikasi ini dipilih karena diharapkan dapat menambahkan kompleksitas dari algoritma SHA-1 sendiri. Kompleksitas tersebut didapat karena ditambahkan satu variable baru yaitu x yang memiliki nilai yang selalu berubah dan juga random pada setiap putarannya.

Berikut ini adalah contoh file teks yang digunakan dalam implementasi dari modifikasi SHA-1 :

> In cryptography, SHA-1 is a cryptographic hash function designed by the National Security Agency (NSA) and published by the NIST as a U.S. Federal Information Processing Standard. SHA stands for Secure Hash Algorithm. The three SHA algorithms are structured differently and are distinguished as SHA-0, SHA-1, and SHA-2. SHA-1 is very similar to SHA-0, but corrects an error in the original SHA hash specification that led to significant weaknesses. The SHA-0 algorithm was not adopted by many applications. SHA-2 on the other hand significantly differs from the SHA-1 hash function.

Kemudian ini adalah message digest yang didapat apabila pesan dimasukkan ke algoritma SHA-1 yang belum dimodifikasi :

```
48a4c591e36321ef9ab0ea8b0e506d57bb4d5202
```

Sedangkan ini adalah message digest yang didapat apabila pesan dimasukkan ke algoritma SHA-1 yang telah dimodifikasi dengan metode di atas :

```
921ab683910aa59ed1c05e7c8da147d8a17e4871
```

4. Analisis

Pada tahap ini akan dilakukan analisis terhadap algoritma SHA-1 yang telah dimodifikasi. Dari algoritma SHA-1 yang telah dimodifikasi, dapat kita lihat ada dua perubahan yang sangat besar dibandingkan algoritma SHA-1 yang normal.

Perubahan pertama adalah ditambahkannya suatu variable baru di dalam algoritma proses utama yang dilakukan di setiap putaran. Dengan ditambahkannya satu variable baru tersebut, tentunya sangat berperan besar dalam mengubah keseluruhan hasil pesan. Hal tersebut dikarenakan variable tersebut diletakkan di proses utama dan juga ikut di dalam setiap putaran (0-79).

Perubahan yang kedua adalah variable itu sendiri. Variable itu memiliki nilai yang random dan juga perlu melewati tahap yang cukup kompleks seperti dalam pembangkitan parameter di algoritma RSA. Selain itu nilai dari variable tersebut juga memiliki nilai yang cukup besar sehingga juga akan sulit untuk dipecahkan dengan cepat.

Pada algoritma SHA-1 yang belum dimodifikasi, dapat dengan mudah terkena *collision attack*. Namun dengan adanya variable baru ini juga diperlukan suatu metode untuk memecahkannya. Untuk memecahkan variable tersebut sangatlah sulit, karena nilai variable tersebut sangat random dan tidak terkait dengan isi pesan yang ada. Selain itu nilai yang ada juga bisa sangat besar sehingga semakin menyulitkan untuk dipecahkan.

Namun algoritma yang baru ini masih memiliki kelemahan tersendiri, yaitu pada proses penentuan bilangan prima p dan q yang akan dipakai pada proses penentuan variable baru tersebut. Pada algoritma yang sekarang, bilangan prima yang dipakai adalah bilangan prima antara 1-500. Untuk lebih memperkuat, algoritma yang baru ini diperlukan penentuan bilangan prima p dan q yang bisa mencapai enam digit atau lebih. Sehingga nilai variable yang baru juga akan lebih besar dan semakin sulit dipecahkan.

5. Kesimpulan

Dari hasil eksperimen dan implementasi yang telah dilakukan didapatkan kesimpulan sebagai berikut :

- Algoritma SHA-1 masih memiliki kelemahan karena dapat diserang dengan *collision attack*
- Metode pembangkitan parameter RSA memiliki keunggulan karena nilai yang dihasilkan sangat random dan dapat bernilai sangat besar
- Modifikasi pada algoritma SHA-1 dilakukan dengan penambahan variable baru di proses utama sehingga bisa mengubah keseluruhan hasil pesan secara signifikan
- Nilai variable yang ditambahkan dihasilkan dari metode pembangkitan parameter yang digunakan dalam algoritma RSA
- Algoritma modifikasi SHA-1 yang dibuat masih kurang kuat karena bilangan prima yang digunakan dalam metode pembangkitan variable baru merupakan bilangan prima antara 1-500 saja sehingga nilai variable baru juga kurang kompleks
- Secara keseluruhan, penambahan variable baru ini mampu memperkuat algoritma SHA-1 karena nilai yang dihasilkan tidak berhubungan dengan pesan sama sekali dan sangat random

- Algoritma baru ini masih belum diujikan ke berbagai jenis serangan apakah bisa dipecahkan dengan gabungan metode tertentu atau tidak

Untuk pengembangan lebih lanjut dapat digunakan list yang berisi bilangan prima yang memiliki nilai lebih besar sehingga dapat memperkuat nilai variable baru yang digunakan dalam algoritma modifikasi SHA-1 yang baru.

6. Daftar Pustaka

[1] Munir, Rinaldi. (2009). Bahan Kuliah IF3058 Kriptografi. Program Studi Informatika, Institut Teknologi Bandung.

[2] www.di-mgt.com.au/**rsa**_alg.html

[3]www.**rsa**.com/**rsa**labs/node.asp?id=2146

[4] www.itl.nist.gov/fipspubs/fip180-**1**.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, 29 April 2010

Miftah Mizan-13507064