STUDI MENGENAI ALGORITMA PENANDATANGANAN DIGITAL DENGAN VERIFIKASI YANG CEPAT

Aldo Juwito Yahya – NIM : 13503085

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13085@students.if.itb.ac.id

Abstrak

Tanda tangan digital adalah tanda tangan yang terdapat pada suatu data digital, seperti pesan yang dikirim melalui saluran komunikasi dan dokumen elektronis yang disimpan dalam komputer. Fungsinya sebagai otentikasi data digital tersebut dan menjamin integritasnya.

Teknik yang lazim dilakukan adalah dengan menggunakan fungsi *hash* dari pesan kemudian membangkitkan tanda tangan berdasarkan fungsi *hash* tersebut. Proses pembangkitan ini dilakukan dengan algoritma kriptografi kunci publik. Algoritma yang banyak digunakan saat ini antara lain RSA, ElGamal, dan Rabin-Williams.

Algoritma yang akan dibahas pada makalah ini adalah algoritma varian dari Rabin-Williams, yang juga setipe dengan algoritma RSA. Algoritma ini ditemukan oleh Daniel J. Bernstein. Algoritma ini unggul dalam hal proses verifikasinya yang sangat cepat.

Makalah ini akan mencoba untuk memaparkan algoritma ini mulai dari pembangkitan kunci publik dan privatnya, proses penandatanganan, dan verifikasinya. Selain itu, perbandingan dengan algoritma lain akan dilakukan terutama dengan algoritma RSA dan Rabin-Williams.

Kata kunci : Tanda tangan digital, *digital signature,* RSA, Rabin-Williams, Daniel J. Bernstein, *fast verification*

1. Pendahuluan

Sejak berabad-abad lamanya, tanda tangan digunakan untuk membuktikan otentikasi dokumen kertas. 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 seperti pesan yang dikirim melalui saluran komunikasi dan dokumen elektronis yang disimpan di dalam memori komputer. Tanda tangan pada data digital inilah yang disebut dengan tanda tangan digital. Yang dimaksud dengan tanda-tangan digital adalah suatu nilai kriptografis yang bergantung pada pesan dan pengirim pesan (Hal ini kontras dengan tanda tangan pada dokumen kertas yang bergantung hanya pada pengirim dan selalu sama untuk semua dokumen). Dengan tanda tangan digital, maka integritas data dapat dijamin, di samping itu juga digunakan untuk membuktikan asal pesan (keabsahan pengirim), dan nirpenyangkalan.

Teknik yang lazim dilakukan dalam menandatangani suatu pesan digital adalah dengan menggunakan fungsi *hash* dari pesan kemudian membangkitkan tanda tangan

berdasarkan fungsi *hash* tersebut. Proses pembangkitan ini dilakukan dengan algoritma kriptografi kunci publik. Algoritma yang banyak digunakan saat ini antara lain RSA, ElGamal, dan Rabin-Williams.

Verifikasi akan tanda tangan digital merupakan salah satu proses penting yang diperlukan untuk menentukan keabsahan suatu pesan. Proses ini melibatkan pesan itu sendiri dan kunci privat yang akan digunakan untuk mendekripsi tangan tangan digital. Dengan proses verifikasi yang cepat, maka keabsahan pesan dapat dengan cepat diketahui. Selain itu, dalam kenyataannya, proses verifikasi lebih sering dilakukan dibanding dengan proses pemberian tanda tangannya itu sendiri. Kecepatan verifikasi tentunya akan sangat mempengaruhi keandalan suatu algoritma untuk membangkitkan tanda tangan digital.

2. Definisi

2.1 Tanda Tangan Digital

2.1.1 Latar Belakang

Seperti yang telah diketahui bahwa terdapat empat aspek keamanan yang disediakan oleh kriptografi. Keempat aspek tersebut adalah:
1. Kerahasiaan pesan (*confidentiality/secrecy*)
2. Otentikasi (*authentication*).
3. Keaslian pesan (*data integrity*).
4. Anti-penyangkalan (*nonrepudiation*).

Kerahasiaan pesan dapat dicapai dengan menggunakan metode enkripsi dan dekripsi. Pertanyaan sekarang adalah bagaimana dengan ketiga aspek yang lainnya. Metode enkripsi/dekripsi tidak mampu memenuhi aspek-aspek tersebut. Di sinilah kegunaan dari apa yang disebut dengan tanda tangan digital.

Berikut adalah contoh tanda tangan digital pada suatu pesan:

Kepada Anto

Besok ada kuis Kriptografi. Bahannya mulai dari sistem kriptografi kunci publik, fungsi *hash*, tanda tangan digital, MAC, dan pembangkit bilangan acak. Tolong beritahu Andi dan Geri.

Dari Beni.

----BEGIN PGP SIGNATURE----

Fisl;3ji[IJDL:3kjlKJG[93j;KLSAJ){3+)DJ{djk;k;afk;k_#UJKfd;_]eoj[dld;cvcn

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

Tanda tangan digital

Menandatangani pesan dapat dilakukan dengan salah satu dari kedua cara berikut:
1. Enkripsi pesan.
   Mengenkripsi pesan dengan sendirinya juga menyediakan ukuran otentikasi. Pesan yang terenkripsi sudah menyatakan bahwa pesan tersebut telah ditandatangani.
2. Tanda tangan digital dengan fungsi *hash (hash function)*

Tanda tangan digital dibangkitkan dari *hash* terhadap pesan. Nilai *hash* adalah kode ringkas dari pesan. Tanda tangan digital berlaku seperti tanda tangan pada dokumen kertas. Tanda tangan digital ditambahkan (*append*) pada pesan.

2.1.2 Tanda Tangan Menggunakan Fungsi *Hash*

Penandatanganan pesan dengan cara mengenkripsinya selalu memberikan dua fungsi berbeda: kerahasiaan pesan dan otentikasi. Pada beberapa kasus, seringkali otentikasi yang diperlukan, tetapi kerahasiaan pesan tidak.. Maksudnya, pesan tidak perlu dienkripsikan, sebab yang dibutuhkan hanya otentikasi saja.

Hanya sistem kriptografi kunci public yang cocok dan alami untuk pemberian tanda-tangan digital dengan menggunakan fungsi *hash.* Hal ini disebabkan karena skema tanda tangan digital berbasis sistem kunci-publik dapat menyelesaikan masalah *non-repudiation* (baik penerima dan pengirim pesan mempunyai pasangan kunci masing-masing).

2.1.3 Metode Pemberian Tanda Tangan Digital

Pengirim pesan mula-mula menghitung *message digest* dari pesan. *Message Digest* (MD) diperoleh dengan mentransformasikan pesan M dengan menggunakan fungsi *hash* satu-arah *H,*

MD = H(M)

Selanjutnya, *message digest MD* dienkripsi dengan algoritma kriptografi kunci-publik dan menggunakan kunci privat (*SK)* si pengirim. Hasil enkripsi inilah yang dinamakan dengan tanda-tangan digital S.

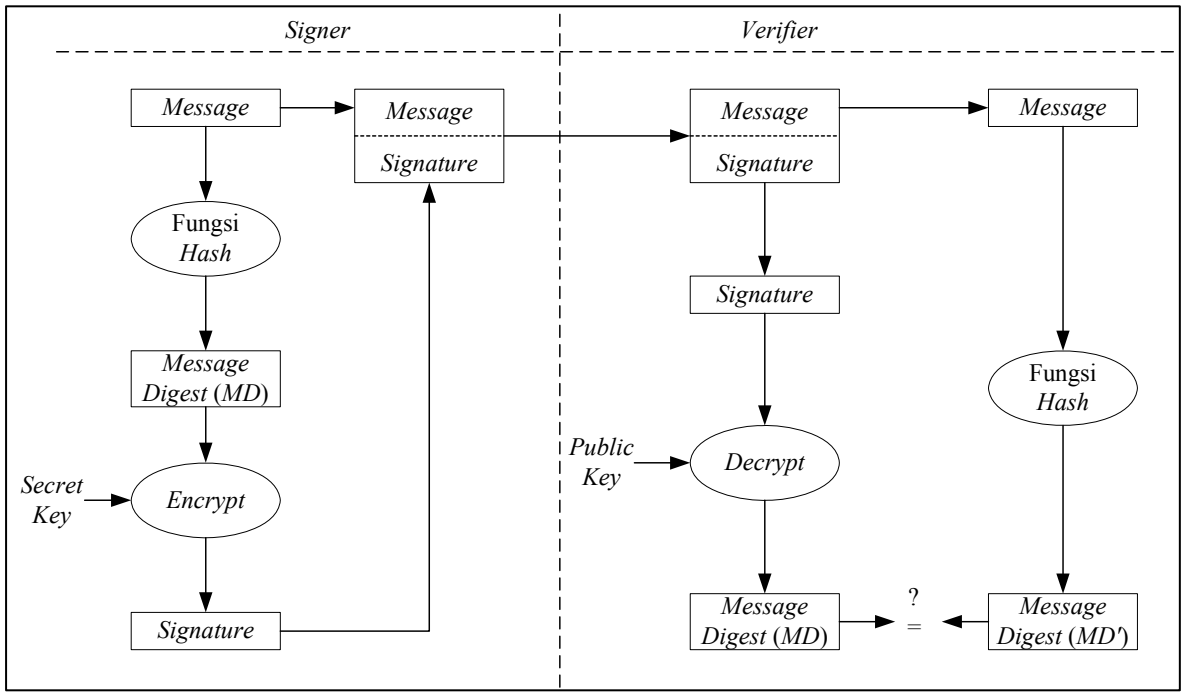
$S = E_{SK}(MD)$

Kemudian, tanda-tangan digital *S* dilekatkan ke pesan *M* (dengan cara menyambung/*append*) *S*, lalu keduanya dikirim melalui saluran komunikasi. Dalam hal ini, kita katakan bahwa pesan *M* sudah ditandatangani oleh pengirim dengan tanda-tangan digital *S*.

Di tempat penerima, tanda-tangan diverifikasi untuk dibuktikan keotentikannya dengan cara berikut:

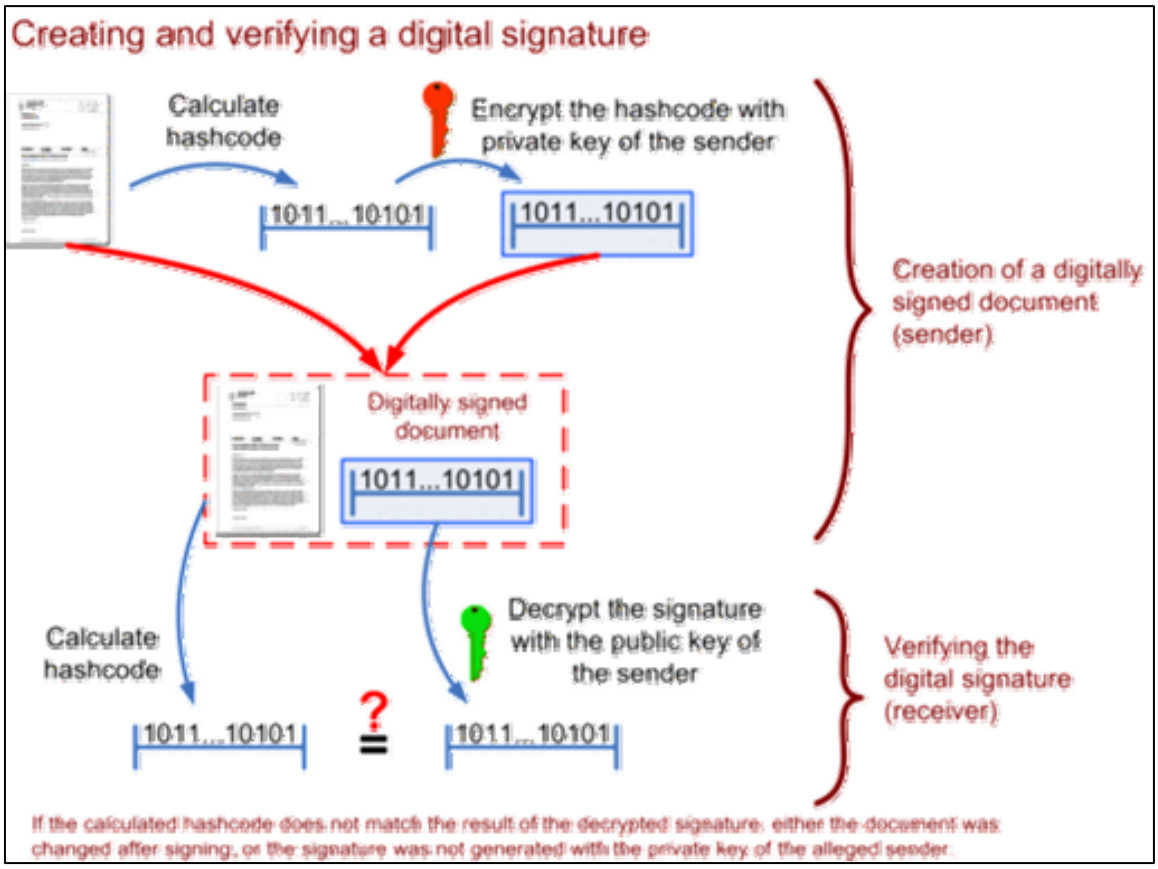
1. Tanda tangan digital *S* didekripsi dengan menggunakan kunci public (*PK*) pengirim pesan, menghasilkan *message digest* semula, *MD*, sebagai berikut:
   $MD = D_{PK}(S)$
2. Penerima kemudian mengubah pesan *M* menjadi *message digest MD'* menggunakan fungsi *hash* satu-arah yang sama dengan fungsi *hash* yang digunakan oleh pengirim.
3. Jika *MD' = MD*, berarti tanda tangan yang diterima otentik dan berasal dari pengirim yang benar.

Skema tanda tangan digital yang menggunakan fungsi *hash* ditunjukkan pada gambar di bawah ini.



Gambar 1 Skema tanda tangan digital

Sedangkan untuk visualisasi penandatanganan dan pemverifikasian tanda-tangan diperlihatkan pada gambar di bawah ini.



Gambar 2 Visualisasi penandatanganan dan verifikasi

Keotentikan pesan dapat dijelaskan sebagai berikut:

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

Pengirim pesan tidak dapat menyangkal pesan yang ia kirim, sebab tanda tangan digital dapat digunakan untuk melakukan nirpenyangkalan. Andaikan pengirim berbohong telah mengirim pesan. Sangkalan dari pengirim dapat dibantah dengan cara sebagai berikut: jika ia tidak mengirim pesan, berarti ia tidak mengenkripsi *message digest* dari pesan dengan kunci privatnya. Faktanya, kunci public yang berkoresponden dengan kunci privat pengirim menghasilkan *MD = MD'*. Ini berarti *message digest* memang benar dienkripsi oleh pengirim sebab hanya pengirim yang mengetahui kunci privatnya sendiri.

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. Selain *RSA,* terdapat algoritma yang dikhususkan untuk tanda tangan digital, yaitu *Digital Signature Algorithm (DSA)*, yang merupakan bakuan untuk *Digital Signature Standard (DSS*). Pada *DSA*, algoritma *signature* dan verifikasi berbeda.

2.2 RSA

2.2.1 Sejarah

Dari sekian banyak algoritma kriptografi kunci-publik yang pernah dibuat, algoritma yang paling populer adalah algoritma *RSA*. Algoritma ini dibuat oleh tiga orang peneliti dari *MIT (Massachussets Institute of Technology)* pada tahun 1976, yaitu Ron (R)ivest, Adi (S)hamir, dan Leonard (A)dleman. Keamanan algoritma *RSA* terletak pada sulitnya memfaktorkan bilangan yang besar menjadi factor-faktor prima. Pemfaktoran dilakukan untuk memperoleh kunci privat. Selama pemfaktoran bilangan besar menjadi factor-faktor prima belum ditemukan algoritma yang mangkus, maka selama itu pula keamanan algoritma *RSA* tetap terjamin.

Algoritma *RSA* memiliki besaran-besaran sebagai berikut:

1. p dan q bilangan prima (rahasia)
2. n = p . q (tidak rahasia)
3. $\Phi(n) = (p-1)(q-1)$ (rahasia)
4. e (kunci enkripsi) (tidak rahasia)
5. d (kunci dekripsi) (rahasia)
6. m (plainteks) (rahasia)
7. c (cipherteks) (tidak rahasia)

2.2.2 Perumusan Algoritma *RSA*

Algoritma *RSA* didasarkan pada teorema Euler yang menyatakan bahwa

$$a^{\Phi(n)} \equiv 1 \pmod{n} \qquad \text{(persamaan 2.1)}$$

dengan syarat:

1. a harus relatif prima terhadap n
2. $\Phi(n) = n(1 - 1/p_1)(1 - 1/p_2)$, yang dalam hal ini $p_1, p_2, \ldots, p_r$ adalah faktor prima dari n. $\Phi(n)$ adalah fungsi yang menentukan berapa banyak dari bilangan-bilangan 1, 2, 3, …, n yang relatif prima terhadap n.

Berdasarkan sifat $a^k \equiv b^k \pmod{n}$ untuk k bilangan bulat ≥ 1 maka persamaan 2.1 dapat ditulis menjadi:

$$a^{k\Phi(n)} \equiv 1^k \pmod{n}$$

atau

$$a^{k\Phi(n)} \equiv 1 \pmod{n} \qquad \text{(persamaan 2.2)}$$

Bila a diganti dengan m, maka persamaan 2.2 dapat ditulis menjadi:

$$m^{k\Phi(n)} \equiv 1 \pmod{n} \qquad \text{(persamaan 2.3)}$$

Berdasarkan sifat $ac \equiv bc \pmod{n}$, maka bila persamaan 2.3 dikali dengan m menjadi:

$$m^{k\Phi(n)+1} \equiv m \pmod{n} \qquad \text{(persamaan 2.4)}$$

yang dalam hal ini m relatif prima terhadap n.
Misalkan e dan d dipilih sedemikian sehingga

$$e \,.\, d \equiv 1 \pmod{\Phi(n)} \qquad \text{(persamaan 2.5)}$$

atau

$$e \,.\, d = k\,\Phi(n) + 1 \qquad \text{(persamaan 2.6)}$$

Sulihkan persamaan 2.6 ke dalam persamaan 2.4 menjadi:

$$m^{e.d} \equiv m \pmod{n} \qquad \text{(persamaan 2.7)}$$

Persamaan 2.7 dapat ditulis kembali menjadi:

$$(m^e)^d \equiv m \pmod{n} \qquad \text{(persamaan 2.8)}$$

Yang artinya, perpangkatan m dengan e diikuti dengan perpangkatan dengan d menghasilkan kembali m semula. Berdasarkan persamaan 2.8, maka enkripsi dan dekripsi dirumuskan sebagai berikut:

$$E_e(m) = c \equiv m^e \bmod n \qquad \text{(persamaan 2.9)}$$
$$D_d(c) = m \equiv c^d \bmod n \qquad \text{(persamaan 2.10)}$$

Karena $e \,.\, d = d \,.\, e$, maka enkripsi diikuti dengan dekripsi ekivalen dengna dekripsi diikuti enkripsi:

$$D_d(E_e(m)) = E_e(D_d(m)) \equiv m^d \bmod n \qquad \text{(persamaan 2.11)}$$

Oleh karena $m^d \bmod n \equiv (m + jn)^d \bmod n$ untuk sembarang bilangan bulat j, maka tiap plainteks $m, m + n, m + 2n, \ldots,$ menghasilkan chiperteks yang sama. Dengan kata lain, transformasinya dari banyak ke satu. Agar transformasinya satu-ke-satu, maka m harus dibatasi dalam himpunan $\{0, 1, 2, \ldots, n-1\}$ sehingga enkripsi dan dekripsi tetap benar seperti pada persamaan 2.9 dan 2.10.

2.2.3 Algoritma Pembangkitan Kunci

1. Pilih dua buah bilangan prima sembarang, p dan q.
2. Hitung $n = p \,.\, q$ (sebaiknya $p \neq q$, sebab jika $p = q$ maka $n = p^2$ sehingga p dapat diperoleh dengan menarik akar pangkat dua dari n).
3. Hitung $\Phi(n) = (p-1)(q-1)$
4. Pilih kunci public, e, yang relatif prima terhadap $\Phi(n)$.
5. Bangkitkan kunci privat dengan menggunakan persamaan 2.5 di atas, yaitu $e \,.\, d \equiv 1 \pmod{\Phi(n)}$. Perhatikan bahwa $e \,.\, d \equiv 1 \pmod{\Phi(n)}$ ekivalen dengan $e \,.\, d = k\,\Phi(n)$

+ 1, sehingga secara sederhana *d* dapat dihitung dengan

$$d = (1 + k\Phi(n)) / e$$

Hasil dari algoritma di atas:
- Kunci public adalah pasangan (*e, n*)
- Kunci privat adalah pasangan (*d, n*)

Di mana *n* tidak bersifat rahasia, sebab ia diperlukan pada perhitungan enkripsi/dekripsi.

2.2.4 Metode Pemberian Tanda Tangan

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*:

   $S = hSK \bmod n$

   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 .

2.2.5 Metode Verifikasi Tanda Tangan

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*:

   $h = SPK \bmod n$

   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

2.3 Sistem Kriptografi Rabin

Sistem kriptografi Rabin merupakan sistem kriptografi asimetri yang faktor keamanannya ditentukan oleh kesulitan dalam pemfaktoran bilangan besar, sama seperti RSA. Namun, setiap hasil dari fungsi Rabin dapat dihasilkan dari empat kemungkinan masukan. Jika setiap hasil (*output*) merupakan chiperteks, maka dibutuhkan waktu ekstra dalam proses dekripsi untuk menentukan plainteks yang sebenarnya dari keempat kemungkinan masukan tadi.

2.3.1 Sejarah

Sesuai dengan namanya, sistem kriptografi Rabin pertama kali dipublikasikan oleh Michael O. Rabin pada tahun 1979. Sistem kriptografi Rabin merupakan salah satu sistem kriptografi asimetri pertama yang kemampuan sekuritasnya dapat dibuktikan secara matematik mengingat metode pemfaktoran bilangan secara cepat sampai saat ini belum terpecahkan.

2.3.2 Algoritma Pembangkitan Kunci

Sama halnya dengan sistem kriptografi asimetri yang lain, Rabin juga menggunakan kunci publik dan kunci privat. Kunci public nantinya akan digunakan pada proses enkripsi dan dapat diketahui oleh semua pihak (tidak rahasia), sementara kunci privat digunakan oleh pihak penerima pesan untuk dekripsi dan bersifat rahasia.

Algoritma pembangkitan kuncinya adalah sebagai berikut:
1. Pilih dua buah bilangan prima besar sembarang yang saling berbeda.
2. Hitung $n = p \cdot q$. *n* adalah kunci publik. Bilangan prima *p* dan *q* adalah kunci privat.

Untuk megenkripsi pesan hanya dibutuhkan kunci publik *n*. Sedangkan untuk dekripsi, dibutuhkan bilangan *p* dan *q* sebagai kunci privat.

Sebagai contoh, $p = 7$ dan $q = 11$, maka $n = 77$. Tentu saja ini merupakan contoh yang kurang baik karena bilangan prima yang dipilih kecil sehingga nilai *n* yang dihasilkan relatif mudah untuk difaktorkan.

2.3.3 Metode Enkripsi

Untuk enkripsi dibutuhkan kunci publik *n*. Metodenya adalah sebagai berikut:

Misalkan $P = \{0, ..., n\text{-}1\}$ merupakan jangkauan area plainteks yang terdiri dari angka dan $m \in P$ adalah plainteksnya. Chiperteks *c* dihitung dengan persamaan:

$$c = m^2 \bmod n$$

sedemikian sehingga *c* merupakan sisa dari plainteks *m* pangkat dua, modulo kunci publik *n*. Sebagai contoh, *P* = {0,...,76} adalah area plainteks. Misalkan *m* = 20 sebagai plainteks. Maka chiperteksnya adalah

$$c = m^2 \bmod n = 400 \bmod 77 = 15$$

Dari hasil *c* yang diperoleh, terdapat empat buah nilai *m* yang mungkin, yaitu $m \in \{13, 20, 57, 64\}$. Hasil ini benar berdasarkan chiperteks yang dihasilkan oleh algoritma Rabin, yang dapat dikatakan merupakan fungsi empat ke satu.

2.3.4 Metode Dekripsi

Untuk mendekripsi suatu chiperteks yang telah dihasilkan, dibutuhkan kunci privat. Prosesnya adalah sebagai berikut:

Jika *c* dan *r* diketahui, maka plainteksnya adalah $m \in \{0, ..., n-1\}$ dengan $m^2 \equiv c \bmod r$. Untuk bilangan komposit *r* (seperti algoritma Rabin *n* = *p* . *q*), belum ada metode efisien yang diketahui untuk menemukan nilai *m*. Jika $r \in \mathbb{P}$ (seperti *p* dan *q* pada algoritma Rabin), maka *Chinese remainder theorem* dapat digunakan untuk memecahkan nilai *m*.

Sedemikian sehingga

$$m_p = \sqrt{c} \bmod p$$

Dan

$$m_q = \sqrt{c} \bmod q$$

Harus dihitung terlebih dahulu. Dalam contoh di atas, didapatkan m_p = 1 dan m_q = 9. Dengan menggunakan *extended Euclidean algorithm*, y_p and y_q, dengan $y_p \cdot p + y_q \cdot q = 1$ dihitung. Dari contoh didapatkan nilai y_p = -3 dan y_q = 2.

Dengan menggunakan *Chinese remainder theorem*, hitung empat hasil akar kuadrat +*r*, -*r*, +*s* dan -*s* dari $c + n\mathbb{Z} \in \mathbb{Z}/n\mathbb{Z}$. $\mathbb{Z}/n\mathbb{Z}$ adalah set dari sisa modulo *n*. Keempat hasil akar kuadrat itu adalah set dari {0, …, *n*-1).

$$\begin{aligned} r &= (y_p \cdot p \cdot m_q + y_q \cdot q \cdot m_p) \bmod n \\ -r &= n - r \\ s &= (y_p \cdot p \cdot m_q - y_q \cdot q \cdot m_p) \bmod n \\ -s &= n - s \end{aligned}$$

Salah satu dari hasil di atas atas dimodulo oleh *n* merupakan plainteks *m* yang sebenarnya. Dalam contoh sebelumnya, $m \in \{64, \mathbf{20}, 13, 57\}$.

2.3.5 Menghitung Akar Kuadrat

Proses dekripsi membutuhkan perhitungan akan akar kuadrat dari chiperteks *c* modulo bilangan prima *p* dan *q*. Dengan memilih $p \equiv q \equiv 3 \pmod 4$, maka proses perhitungan akar kuadrat menjadi lebih mudah, yaitu:

$$m_p = c^{\frac{(p+1)}{4}} \bmod p$$

dan

$$m_q = c^{\frac{(q+1)}{4}} \bmod q$$

Dapat ditunjukkan bahwa metode ini berfungsi untuk *p* yang diberikan. $p \equiv 3 \pmod 4$ menyatakan bahwa (*p*+1)/4 adalah bilangan bulat. Asumsi ini mengindikasikan bahwa $c \equiv 0 \pmod p$ tidak berlaku. Sedemikian sehingga dapat diasumsikan bahwa *p* tidak dapat membagi habis *c*. Sehingga

$$m_p^2 \equiv c^{\frac{(p+1)}{2}} \equiv c \cdot c^{\frac{(p-1)}{2}} \equiv c \cdot \left(\frac{c}{p}\right) \pmod p,$$

Di mana $\left(\frac{c}{p}\right)$ adalah simbol *Legendre*. Dari $c \equiv m^2 \pmod{pq}$, diperoleh bahwa *c* merupakan sisa pangkat dua. Oleh karena itu, $\left(\frac{c}{p}\right) = 1$ dan

$$m_p^2 \equiv c \pmod p.$$

Relasi dari $p \equiv 3 \pmod 4$ tidak diperlukan karena modulo prima dari hasil pangkat dua dapat dihitung dengan metode di atas.

2.4 *Rabin-Williams*

Sistem kriptografi *Rabin-Williams* merupakan varian dari sistem kriptografi *Rabin* sehingga cukup banyak langkah-langkah algoritmanya yang serupa.

2.4.1 Algoritma Pembangkitan Kunci

Algoritma pembangkitan kuncinya adalah sebagai berikut:

1. Pihak penerima pesan memilih dua buah bilangan prima besar sembarang yang saling

berbeda, p dan q yang kongruen dengan 3 modulo 4.

2. Hitung $N = p \cdot q$. N adalah kunci publik. Nilai N ini kemudian dipublikasikan ke semua pihak karena sifatnya tidak rahasia.

2.4.2 Metode Enkripsi

Untuk enkripsi dibutuhkan kunci publik N. Pesan yang dalam bentuk $m \in \{0, 1, \ldots, N\}$ dienkripsi dengan menghitung

$$c_m \equiv m^2 \pmod{N}$$

Tentu saja sangat sulit untuk membalik operasi di atas tanpa pengetahuan akan nilai p dan q. Selain itu, telah dibuktikan juga bahwa membalik operasi ini sama sulitnya dengan memfaktorkan N untuk memperoleh nilai p dan q.

2.4.3 Metode Dekripsi

Dekripsi dilakukan berdasarkan hasil enkripsi yang menggunakan kunci privat. Penerima dari pesan akan memecahkan persamaan:

$$x^2 \equiv c \pmod{N}$$

Tentunya dengan pengetahuan akan nilai p dan q.

Persamaan di atas menghasilkan persoalan trivial karena jika

$$p = 4k_p - 1$$

maka

$$x^2 \equiv x^{p+1} \equiv x^{4k_p} \equiv (x^2)^{2k_p} \equiv (c^{k_p})^2 \pmod{p}$$

sesuai dengan *Fermat's little theorem*. Sedemikian sehingga $\{x_0, x_1\} = \{c^{k_p}, -c^{k_p}\}$ adalah satu set solusi dari modulo p.

Dengan cara yang sama didapatkan pula set solusi $\{y_0, y_1\} = \{c^{k_q}, -c^{k_q}\}$.

Kemudian, dengan *Chinese remainder theorem,* terdapat suatu solusi unik (modulo N)

$$\begin{cases} x \equiv x_i & \pmod{p} \\ x \equiv y_j & \pmod{q} \end{cases}$$

Untuk setiap $i = 0,1$ dan setiap $j = 0,1$. Dengan demikian semua solusi akan memenuhi persamaan

$$x^2 \equiv c \pmod{N}$$

di mana $x^2 \equiv x_i^2 \equiv c \pmod{p}$ mengindikasikan $p \mid (x^2 - c)$ dan $x^2 \equiv y_i^2 \equiv c \pmod{q}$ mengindikasikan $q \mid (x^2 - c)$ dan sedemikian sehingga $N = pq \mid (x^2 - c)$ (karena p relatif prima terhadap q)

Solusi eksplisit untuk setiap kemungkinan empat persamaan tersebut dapat diperoleh dengan mengimplementasikan *Euclid's algorithm.* Algoritma ini akan mencari bilangan bulat r dan s di mana

$$1 = pr + qs$$

dan selanjutnya

$$x = (pr)y_i + (qs)x_i$$

adalah suatu solusi eksplisit sebagaimana yang diinginkan untuk

$$\begin{cases} x \equiv x_i & \pmod{p} \\ x \equiv y_j & \pmod{q} \end{cases}$$

Pemasalahan yang timbul adalah memilih solusi yang benar di antara $\{x_0,x_1,y_0,y_1\}$. Masalah ini dapat dengan mudah diatasi apabila si pengirim pesan menyertakan ruang yang cukup untuk redundansi pada pesan sehingga hanya satu dari solusi yang memenuhi.

2.5 Fungsi *Hash* Satu Arah

Fungsi *hash* adalah fungsi yang menerima masukan *string* yang panjangnya sembarang dan mengkonversinya menjadi *string* keluaran (*message digest*) yang panjangnya tetap (*fixed*) dan biasanya dengan ukuran yang jauh lebih kecil dari ukuran *string* semula. Fungsi *hash* dapat digunakan untuk melakukan verifikasi atas sebuah salinan arsip. Jika *message digest* suatu salinan arsip serupa dengan *message digest* arsip aslinya maka salinan arsip tersebut sama dengan arsip aslinya.

Fungsi *hash* satu-arah adalah fungsi *hash* yang bekerja dalam satu arah. Dengan demikian, arsip yang sudah diubah menjadi sebuah *message digest* tidak dapat dikembalikan ke bentuk arsip semula. Terdapat beberapa fungsi *hash* yang

sudah banyak dikenal, misalnya MD2, MD4, MD5, *Secure Hash Function* (SHA), Snefru, dan masih banyak lagi yang lainnya. Namun demikian, algoritma yang banyak dipakai dalah aplikasi kriptografi adalah MD5 dan SHA.

Sembarang pesan *M* berukuran bebas dikompresi oleh fungsi hash *H* melalui persamaan

$$h = H(M)$$

Sifat-sifat fungsi *hash* adalah sebagai berikut:

1. Fungsi *H* dapat diterapkan pada blok data berukuran berapa saja.
2. *H* menghasilkan nilai (*h*) dengan panjang tetap (*fixed-length output*).
3. $H(x)$ mudah dihitung untuk setiap nilai *x* yang diberikan.
4. Untuk setiap *h* yang dihasilkan, tidak mungkin dikembalikan nilai x sedemikian sehingga $H(x) = h$. Itulah sebabnya fungsi *H* dikatakan fungsi *hash* satu-arah (*one-way hash function*).
5. Untuk setiap *x* yang diberikan, tidak mungkin mencari $y \neq x$ sedemikian sehingga $H(y) = H(x)$.
6. Tidak mungkin mencari pasangan *x* dan *y* sedemikian sehingga $H(x) = H(y)$.

2.6 SHA

SHA adalah fungsi *hash* satu-arah yang dibuat oleh *NIST* dan digunakan bersama *DSS* (*Digital Signature Standard*). Oleh *NSA*, *SHA* dinyatakan sebagai standard fungsi *hash* satu-arah. *SHA* didasarkan pada *MD4* yang dibuat oleh Ronald L. Rivest dari *MIT*. Algoritma *SHA* menerima masukan berupa pesan dengan ukuran maksimum 264 bit (2.147.483.648 *gigabyte*) dan menghasilkan *message digest* yang panjangnya 160 bit, lebih panjang dari *message digest* yang dihasilkan oleh *MD5*. *SHA* mengacu pada keluarga fungsi *hash* satu-arah. Enam varian *SHA*: SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512. SHA-0 sering diacu sebagai *SHA* saja

Langkah-langkah pembuatan *message digest* dengan *SHA*-1 adalah sebagai berikut:

1. Penambahan bit-bit pengganjal (padding bits)
   Panjang bit pengganjal adalah antara 1-512 yang terdiri dari sebuah bit 1 diikuti dengan bit 0. Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Hal ini berarti panjang pesan selalu ditambah dengan bit-bit pengganjal 64 bit kurangnya dari kelipatan 512. Jika panjang pesan 448 maka pesan tersebut ditambah dengan 512 bit menjadi 960 bit.
2. Penambahan nilai panjang pesan semula.
   Pesan yang telah diberi bit-bit pengganjal selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Setelah ditambah dengan 64 bit panjang pesan semula sekarang menjadi kelipatan 512.
3. Inisialisasi penyangga (*buffer*) MD.
   Algoritma SHA membutuhkan 5 buah penyangga (buffer) yang masing-masing panjangnya 32 bit. Total panjang penyangga adalah 160 bit. Kelima penyangga ini menampung hasil antara dan hasil akhir. Nilai inisialisasi setiap penyangga adalah sebagai berikut:
   A = 1732584193; //0x67452301;
   B = 4023233417; //0xEFCDAB89;
   C = 2562383102; //0x98BADCFE;
   D = 271733878; //0x10325476;
   E = 3285377520; //0xC3D2E1F0;
4. Pengolahan pesan dalam blok berukuran 512 bit.
   Pesan dibagi menjadi L blok yang masing-masing panjangnya 512 bit. Setiap blok 512 bit diproses bersama dengan penyangga MD menjadi keluaran 128-bit (HSHA).
   Proses HSHA terdiri dari 80 putaran dan masing-masing putaran menggunakan bilangan penambah Ki, yaitu:
   long K1 = 1518500249;
   //0x5A827999
   long K2 = 1859775393;
   //0x6ED9EBA1
   long K3 = 2400959708;
   //0x8F1BBCDC
   long K4 = 3395469782;
   //0xCA62C1D6

Pada tahun 2005, Rijmen dan Oswald mempublikasikan serangan pada versi *SHA*-1 yang direduksi (hanya menggunakan 53 putaran dari 80 putaran) dan menemukan kolisi dengan kompleksitas sekitar 280 operasi.

Pada bulan Februari 2005, Xiayoun Wang, Yiqun Lisa Yin, dan Hongbo Yo mempublikasikan serangan yang dapat menemukan kolisi pada versi penuh SHA-1, yang membutuhkan sekitar 269 operasi.

3. Tanda Tangan Digital dengan Verifikasi Cepat

Sistem penandatanganan digital dengan verifikasi cepat ini diperkenalkan oleh Daniel J. Bernstein pada tahun 2000. Sistem ini dikembangkan berdasarkan sistem *Rabin-Williams* dengan ukuran kunci, keamanan, dan kecepatan penandatanganan yang sama dengan sistem *Rabin-Williams.*

3.1 Kunci

Sama halnya dengan *Rabin-Williams,* kunci privat adalah sepasang bilangan prima p dan q di mana $p \bmod 40 = 3$, dan $q \bmod 40 = 7$. Nilai p memenuhi $4.2^{765} < p < 5.2^{765}$, dan $2^{800}\pi < pq < 2^{800}(\pi+1)$. Di mana $\pi \approx 2{,}16 \cdot 10^{221}$ adalah bilangan bulat yang diperoleh dengan menuliskan 222 digit pertama dari bilangan π dalam urutan terbalik. Dalam bentuk heksadesimal akan seperti berikut ini:

```
9949AB0B 00D505ED 84F87BF4 6E10D628 5C86E943 7729912B DDD99955
B3877272 B6D65463 1772C6BE F62755DC 39DBB3A3 47D512C4 71838883 72BE4B91
59883460 B03ECBF4 88DAFE8B 6E49454C 010C23FE C6F682E4 9E241C1A 28E0B6F5
```

Prosedur di atas kelihatannya akan menghasilkan kunci privat secara acak yang mendekati distribusi seragam. Proses diawali dengan meng-*generate* bilangan acak yang seragam antara 4.2^{765} dan 5.2^{765} yang memenuhi $p \bmod 40 = 3$. Proses ini diulangi hingga diperoleh p yang prima. Hal yang sama dilakukan untuk q antara $\left\lfloor 2^{800}\pi / p \right\rfloor$ dan $\left\lfloor 2^{800}(\pi+1)/p \right\rfloor$ yang memenuhi $q \bmod 40 = 7$. Proses diulang hingga diperoleh q yang prima.

Kunci publik merupakan hasil dari $p.q$. Hasil perkalian ini berada antara 2^{1535} dan 2^{1536}. Tentu saja sampai saat ini belum ditemukan algoritma yang sanggup memfaktorkan hasil perkalian ini dengan cepat.

3.2 Metode Pemberian Tanda Tangan

Setiap pesan mempunyai banyak tanda tangan. Penanda tangan harus mengikuti sejumlah aturan dalam menghasilkan tanda tangan dari suatu pesan. Beberapa tanda tangan bahkan tidak boleh di-*generate* walaupun sebenarnya disetujui oleh pihak yang memverifikasi.

Sama halnya dengan *RSA,* algoritma ini membutuhkan set X dari pesan dan fungsi *hash*
$H : \{r \in \mathbf{Z} : 0 \le r < 2^{256}\} \times \mathbf{X} \rightarrow$
$\{h \in \mathbf{Z} : 0 < h < 2^{1531}, h \bmod 8 = 1\}$
Misalkan n adalah kunci publik. Tanda tangan digital untuk pesan m dengan kunci publik n merupakan vektor bilangan bulat (r, h, f, s, t) di mana $0 \le r < 2^{256}$, $h = H(r,m)$, $f \in \{-2,-1,1,2\}$, $0 \le s < 2^{1536}$, $0 \le t < 2^{1536}$, dan $s^2 = tn + fh$.

Misalkan m adalah pesan yang akan ditandatangani, n adalah kunci publik yang telah dibuat. Tanda tangan digital untuk m dengan menggunakan kunci publik n adalah suatu vector bilangan bulat (r, h, f, s, t) di mana:

- $0 \le r < 2^{256}$,
- $h = H(r,m)$,
- $s^2 = tn + fh$,
- $n/2 < s \le n$,
- salah satu dari s atau $-s$ adalah kuadrat modulo n, namun bukan keduanya, dan
- f adalah bilangan bulat pertama dalam sekuens 1, 2, -1, -2 di mana fh adalah kuadrat modulo n.

Perhatikan bahwa jika (r, h, f, s, t) adalah tanda tangan digital standar dari m dengan kunci publik n, maka itu juga merupakan tanda tangan digital dari m dengan kunci publik n.

Algoritma S yang akan digunakan untuk proses tanda tangan pada pesan m mengasumsikan bahwa penandatangan akan selalu meng-*generate* tanda tangan standar, dan memilih sebuah bilangan random r yang seragam untuk setiap pesan yang akan ditandatangani. Asumsi yang pertama harus dipenuhi karena tidak ada jaminan yang disediakan bagi penandatangan yang membuat tanda tangan digital non-standar. Untuk asumsi yang kedua, sebenarnya bilangan r yang tidak seragam (*non-uniform*) pun tidak terlalu bermasalah. Pemilihan r yang seragam akan lebih menguatkan pembuktian jaminan akan keamanan algoritma.

Algoritma S. Diberikan pesan *m,* kunci privat (*p,q*), kunci publik $n = pq$, dan bilangan bulat *x, y* dengan $xp + yq = 1$, maka proses penandatangan adalah sebagai berikut:

1. Pilih bilangan random yang seragam $r \in \{0,1,...,2^{256} - 1\}$.
2. Hitung $h \leftarrow H(r,m)$.
3. Jika simbol *Legendre* dari *h* modulo *q* adalah -1, maka $e \leftarrow -1$; sebaliknya e $\leftarrow 1$.
4. Jika simbol *Legendre* dari *eh* modulo *p* adalah -1, maka $f \leftarrow 2e$; sebaliknya $f \leftarrow e$.
5. (Sekarang *fh* adalah kuadrat modulo *n*). Set nilai $a \leftarrow fh$.
6. Hitung $c \leftarrow yq(a^{(p+1)/4} \bmod p) + xp(a^{(q+1)/4} \bmod q) \bmod n$.
7. Jika $c > n/2$, set nilai $s \leftarrow c$; sebaliknya set $s \leftarrow n - c$. (Sekarang $s^2 \equiv a \pmod n$, dan $n/2 < s \leq n$.)
8. Hitung $t \leftarrow (s^2 - a)/n$.
9. Akhirnya diperoleh nilai (*h, f, s, t*). Nilai ini yang kemudian dicetak sebagai tanda tangan digital.

3.3 Metode Verifikasi Tanda Tangan

Misalkan *m* adalah pesan (sama seperti metode pemberian tanda tangan), dan *r, h, f, s, t* adalah bilangan bulat dengan $0 \le r < 2^{256}$, $0 \le h < 2^{1531}$, $f \in \{-2,-1,1,2\}$, $0 \le s < 2^{1536}$, dan $0 \le t < 2^{1536}$. Definisikan $c = s^2 - tn - fh$. Periksa apakah (*r, h, f, s, t*) adalah benar tanda tangan dari pesan *m* dengan kunci publik *n*. Caranya dengan memeriksa $h = H(r,m)$ dan $c = 0$.

Salah satu cara untuk memeriksa apakah $c = 0$ adalah dengan memeriksa apakah *c* habis dibagi oleh suatu bilangan prima kecil yang bersifat rahasia. Sebagai contoh, terdapat lebih dari $40 \cdot 2^{100}$ bilangan prima antara 2^{114} dan 2^{115} kongruen dengan 2 modulo 5. Jika $c \neq 0$ maka *c* tidak dapat habis dibagi dengan lebih dari 26 dari sejumlah bilangan prima tadi karena $|c| < 2^{3072}$, sehingga kesempatan suatu prima acak yang seragam yang dapat membagi habis *c* adalah di bawah 2^{-100}. Kemungkinan ini tentu saja dapat dengan aman diabaikan.

Dengan demikian, pilih bilangan prima *u*, kemudian hitung $s' = s \bmod u$, $t' = t \bmod u$, $n' = n \bmod u$, dan $h' = h \bmod u$. Terakhir hitung $((s')^2 - t'n' - fh') \bmod u$. Hasilnya adalah *c* mod *u*, bernilai 0 apabila $c = 0$, dan bukan 0 apabila $c \neq 0$.

Bilangan prima *u* yang sama dapat digunakan untuk memeriksa banyak tanda tangan, tidak cuma satu. Apabila bagian dari *h* dapat ditentukan berdasarkan fungsi *hash H* maka bagian tersebut dapat dihitung ulang kemudian diberikan operasi mod *u*. Sama halnya dengan *n'* dapat disimpan jika terdapat beberapa tanda tangan yang akan diperiksa dengan kunci publik *n* yang sama.

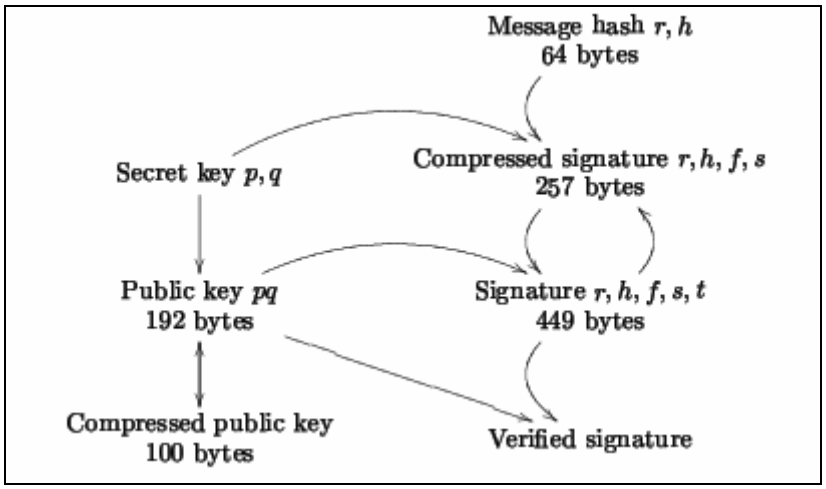
Beberapa kendala yang menjadi isu adalah ketidaknyamanan pihak pemverifikasi untuk menyimpan bilangan prima acak yang rahasia. Salah satu alternatif adalah cukup dengan menghitung *c* dan memverifikasi bahwa $c = 0$, atau memverifikasi bahwa *c* dapat habis dibagi oleh suatu set moduli yang mana bilangan pengali terkecil melebihi perhitungan batas dari $|c|$. Moduli itu dapat dipilih untuk membantu pembagian secara cepat.

3.4 Fungsi *Hash*

Dalam aplikasi yang dibuat oleh Daniel J. Bernstein, pesan dibuat berupa *strings of bytes,* dan fungsi *hash H* didefinisikan sebagai berikut. Berikan (*r,m*) melalui Merkle's Snefru-8/256 [5], *little-endian*, untuk memperoleh sebuah bilangan bulat *z* dengan $0 \le z < 2^{256}$. Kemudian $H(r,m) = 2^{320}\bar{e} + 2^{64}z + 1$, di mana $\bar{e} \approx 9{,}07.10^{363}$ adalah bilangan bulat yang diperoleh dengan menuliskan 364 digit pertama dari exp 1 dalam urutan terbalik. Dalam heksadesimal akan berupa seperti:

```
                                  20EC2CF9A875185 23FE3A9D 2F0146A5 91B19A5B 1D9B5799
AB577BA7 D2CF9561 A90606F1 48BC9CDB 98BFBAD3 27663C51 56C5EC8F DAF79AB5
7BC4FA99 5083FB9D EEF66C84 10F54365 DA041484 817305DD 16A6294E 28980AD7
C3D172C9 D0454CAD A219FC7B A0A5E47F B16FDAD7 7A4E33A3 9E673D5B F932E1F1
210405D4 CC46B9E3 EEDFDA4E 043D1196 81144131 4ECE679B 97D28C87 6467D34C
```

Kita dapat juga mengompres $H(r,m)$ sehingga nilai *z* menjadi berukuran 32-*byte*. Gambar di bawah ini menunjukkan secara implisit bagaimana proses ini dilakukan.



Gambar 3 Aliran data pada sistem kriptografi

4. Pengujian

Kasus uji yang menjadi perhatian adalah menyangkut keamanan akan algoritma yang telah dikembangkan. Di sini akan diperlihatkan bagaimana cara penyerang dapat memperoleh beberapa parameter penting dalam algoritma hanya dengan menggunakan nilai publik key n yang tidak rahasia, tanpa mengetahui hasil faktorisasinya.

Nilai *hash* adalah suatu bilangan bulat h dengan $0 < h < 2^{1531}$ dan $h \bmod 8 = 1$. Beberapa yang patut dicatat mengenai kunci publik n yang merupakan vektor dari bilangan bulat (h, f, s, t) adalah:

- h adalah nilai *hash*,
- $s^2 = tn + fh$,
- $n/2 < s \leq n$,
- salah satu dari s atau $-s$ adalah kuadrat modulo n, dan
- f adalah bilangan bulat pertama dalam sekuens 1, 2, -1, -2 di mana fh adalah kuadrat modulo n.

Dengan demikian (r, h, f, s, t) adalah tangan tangan digital standar dari m dengan kunci publik n jika dan hanya jika $0 \leq r < 2^{256}$, $h = H(r,m)$, dan (h, f, s, t) adalah *scribble* di bawah n.

Algoritma H. Diberikan sebuah kunci publik n dan kuadrat a modulo n dengan $0 \leq a < n$:

1. Hitung sebuah bilangan bulat z yang memenuhi $2z \equiv a \pmod{n}$.
2. Cari $f \in \{-2,-1,1,2\}$ di mana $(2/f)z \bmod n$ adalah nilai *hash*. Apabila tidak terdapat f yang memenuhi, maka proklaim gagal dan berhenti.
3. Set nilai $h \leftarrow (2/f)z \bmod n$. (sekarang fh adalah kuadrat modulo n).
4. Jika gcd $\{h, n) = 1$, langsung ke nomor 6.
5. (Karena h adalah nilai *hash*, maka tidak akan habis dibagi dengan n. Jadi h tidak mempunyai faktor nontrivial untuk kunci publik n). Gunakan h untuk memfaktorkan n. Periksa sesuai dengan algoritma S (halaman 9), apakah f adalah bilangan bulat pertama dalam sekuens 1, 2, -1, -2 di mana fh adalah kuadrat modulo n. Jika tidak, proklaim gagal dan berhenti.
6. Cetak nilai (h, f).

Teorema 1. Algoritma H akan mencetak (h, f) jika dan hanya jika (1) h adalah nilai *hash*, (2) f adalah bilangan bulat pertama dalam sekuens 1, 2, -1, -2 di mana fh adalah kuadrat modulo n, dan (3) $a = fh \bmod n$.

Pembuktian dari teorema di atas adalah sebagai berikut. Katakanlah algoritma H tersebut mencetak (h, f). Langkah nomor 2 akan memilih f sedemikian sehingga h adalah nilai *hash* pada langkah ke-3. Juga $fh \equiv 2z \equiv a \pmod{n}$, sehingga fh adalah kuadrat modulo n. Jika gcd $\{h, n\} = 1$ maka f adalah bilangan bulat pertama dalam {1, 2, -1, -2} di mana fh adalah kuadrat modulo n. Jika gcd $\{h, n\} \neq 1$ maka langkah ke-7 mengecek

apakah f adalah bilangan bulat pertama dalam {1, 2, -1, -2} sedemikian sehingga fh adalah kuadrat modulo n.

Sebaliknya, katakanlah (h, f) adalah pasangan bilangan yang memenuhi kondisi. Maka f akan ditemukan pada langkah ke-2 dari algoritma H, karena tidak mungkin terdapat dua buah nilai *hash* di antara z mod n, $2z$ mod n, $-z$ mod n, $-2z$ mod n, dan h diperoleh pada langkah ke-3. Algoritma H tidak akan gagal pada langkah ke-5, mengingat f adalah bilangan bulat pertama dalam {1, 2, -1, -2} sedemikian sehingga fh adalah kuadrat modulo n.

Algoritma U. Diberikan suatu kunci publik n, untuk mencetak suatu bilangan seragam acak dari kuadrat modulo n.

1. *Generate* sebuah bilangan bulat seragam acak $x \in \{0,1,...,2^{1536}-1\}$.
2. Jika $x \geq n$, kembali ke langkah ke-1.
3. Jika $x = 0$, cetak 0 dan berhenti.
4. *Generate* secara seragam acak suatu bit. Jika 0, maka kembali ke langkah ke-1.
5. Jika gcd $\{x, n\} > 1$, cetak x^2 mod n dan berhenti.
6. *Generate* secara seragam acak suatu bit. Jika 0, maka kembali ke langkah ke-1.
7. Cetak x^2 mod n dan berhenti.

Teorema 2. Algoritma U mencetak suatu bilangan seragam acak dari kuadrat modulo n.

Pembuktian dari teorema di atas adalah misalkan sebuah c kuadrat modulo n. Jika $c \equiv 0$ maka c mempunyai sebuah akar kuadrat, namakan 0. Langkah 1 mengeset $x \leftarrow 0$ dengan probabilitas $1/2^{1536}$, di mana langkah ke-3 akan mencetak c. Jika c tidak nol namun dapat habis dibagi oleh p atau q maka c mempunyai dua akar kuadrat.Langkah pertama akan mengeset x menjadi akar kuadrat dari c dengan probabilitas $2/2^{1536}$, dan langkah ke-4 menjadikan probabilitas dari x mencapai 1/2, yang mana langkah ke-5 akan mencetak c. Apabila c relatif prima dengan n maka c mempunyai empat akar kuadrat. Langkah ke-1 akan mengeset x menjadi akar kuadrat dari c dengan probabilitas $4/2^{1536}$, dan langkah 4 dan 6 menjadikan probabilitas x 1/4, di mana langkah 7 akan mencetak c. Singkatnya, setiap langkah ke-1 dilakukan akan menyebabkan algoritma U untuk mencetak c dengan probabilitas $1/2^{1536}$, tidak peduli berapa pun x.

Algoritma V. Diberikan suatu kunci publik n, untuk mencetak sebuah *scribble* seragam acak.

1. *Generate* suatu bilangan seragam acak c kuadrat modulo n menggunakan algoritma U.
2. Jika $c > n/2$, set nilai $s \leftarrow c$; sebaliknya set $s \leftarrow n - c$.
3. Hitung (h,f) dari s^2 mod n menggunakan algoritma H. Jika algoritma H gagal, maka kembali ke langkah ke-1.
4. Hitung $t \leftarrow (s^2 - fh)/n$.
5. Cetak (h, f, s, t).

Teorema 3. Algoritma v mencetak bilangan sebuah *scribble* seragam acak dengan kunci publik n.

Pembuktian dari teorema di atas adalah sebagai berikut. Berdasarkan teorema 1, h adalah nilai *hash*, f adalah bilangan bulat pertama pada {1, 2, -1, -2) sedemikian sehingga fh adalah kuadrat modulo n, dan $s^2 \equiv fh$ mod n. Jadi t adalah bilangan bulat dan $s^2 = tn + fh$. Tepat salah satu dari s dan $-s$ adalah kuadrat modulo n. Sedemikian sehingga keluaran dari algoritma V adalah *scribble* dengan kunci publik n.

Sekarang misalkan sembarang *scribble* (h', f', s', t'). Berdasarkan definisi bahwa tepat satu dari s' dan $-s'$ adalah kuadrat modulo n. Algoritma V akan mencetak (h', f', s', t') jika dan hanya jika pada langkah ke-1, c merupakan bilangan kuadrat. Setiap kuadrat modulo n akan dihasilkan dengan probabilitas yang sama, sehingga setiap *scribble* yang dihasilkan juga berdasarkan probabilitas yang sama. Pernyataan ini dapat dibuktikan dengan cara berikut. Jika $c \equiv \pm s'$, maka s haruslah sama dengan s' pada langkah 2. Sedemikian sehingga $f'h' \equiv s^2$ (mod n), sehingga (h, f) = (h', f') oleh teorema 1, dan $t = t'$ pada langkah 4. Sebaliknya, jika $s = s'$, maka $c \equiv \pm s'$.

Algoritma W. Diberikan kunci publik n, dan sebuah bilangan bulat y yang relatif prima dengan n yang merupakan kuadrat modulo n.

1. *Generate* suatu bilangan seragam acak dari w kuadrat modulo n oleh algoritma U.
2. Hitung (h, f) dari w^2y mod n menggunakan algoritma H. Jika algoritma H gagal, maka kembali ke langkah 1.

3. Cetak (*h, f, w*).

Teorema 4. Algoritma W mencetak (*h, f, w*) di mana *h* adalah nilai *hash* yang seragam acak, *f* adalah bilangan bulat pertama dari sekuens 1, 2, -1, -2 sedemikian sehingga *fh* adalah kuadrat modulo *n*, dan $fh \equiv w^2 y \pmod{n}$.

Pembuktian dari teorema di atas adalah sebagai berikut. Berdasarkan teorema 1, *h* adalah nilai *hash, f* adalah bilangan bulat yang benar, dan $fh \equiv w^2 y \pmod{n}$. Terdapat bilangan akar kuadrat *x* dari *y* modulo *n* sedemikian sehingga *x* adalah kuadrat modulo *n*. Dituliskan $c = wx \bmod n$. Sehingga *c* adalah bilangan seragam acak dari kuadrat modulo *n*, dengan $c^2 \bmod n = w^2 y \bmod n$, sehingga distribusi dari (*h,f*) pada langkah ke-2 dari algoritma W sama dengan distribusi (*h, f*) pada langkah ke-3 algoritma V. Berdasarkan teorema 3, *h* terdisribusi secara seragam atau merata.

Sebagai catatan, algoritma V bekerja cepat. Terdapat 2^{1528} nilai *hash*, dengan kata lain terdapat 2^{1528} pilihan untuk *c* yang memenuhi langkah 1, jauh dari sekitar *n*/4 kuadrat modulo *n*. Sedemikian sehingga langkah 1 diulang sekitar $n/2^{1530} < 64$ kali. Hal yang sama berlaku untuk algoritma W. Pada prakteknya akan lebih aman untuk mengasumsikan bahwa bilangan acak 1536-bit tidak akan mempunyai faktor yang mengikuti *n.* Sehingga kita dapat melewati langkah 4 dan 5 pada algoritma H, dan langkah 3 sampai 6 pada algoritma U.

Pengujian keamanan dilakukan dengan cara *generic attack*. *Generic adaptive chosen-message attack* adalah suatu algoritma A yang menggunakan dua *oracle,* yaitu *hashing oracle* dan *signing oracle*. Masukan untuk A adalah kunci publik *n*.

Misalkan *H* adalah fungsi *hash*. Bilangan *hashing oracle* yang telah dipilih untuk *H* mencetak *H*(*x*) dengan nilai *x* diketahui. Bilangan *signing oracle* yang telah dipilih untuk *H* adalah algoritma S: diberikan suatu pesan *m*, algoritma ini akan mencetak suatu tanda tangan digital standar (*r, h, f, s, t*) untuk *m* dengan kunci publik *n,* di mana *r* adalah elemen acak dari {0, 1, ..., $2^{256} - 1$} yang dibuat secara terpisah untuk setiap antrian *oracle*.

Dari definisi A akan berhasil jika mencetak suatu vektor (*r, m, h, f, s*) sedemikian sehingga:

- *r* adalah {0, 1, ..., $2^{256} - 1$}
- *m* adalah pesan yang tidak terdapat pada antrian untuk *signing oracle*,
- (*r,m*) adalah antrian untuk *hashing oracle*, dengan keluaran berupa *h*,
- *f* adalah {-2, -1, 1, 2}, dan
- *s* adalah bilangan bulat sedemikian sehingga $s^2 \equiv fh \pmod{n}$

Probabilitas keberhasilan dari A adalah probabilitas keberhasilan A untuk setiap kunci publik acak *n*, dengan diberikan suatu nilai *hashing* dan *signing oracle* untuk fungsi *hash H* yang seragam acak.

5. Kesimpulan

Kesimpulan yang dapat diambil dari studi dan analisis akan algoritma penandatanganan digital oleh Daniel J. Bernstein ini adalah:

1. Karena merupakan varian dari algoritma *Rabin-Williams,* maka tingkat keamanannya pun dapat dikatakan sama dengan *Rabin-Williams* yang berdasarkan pada sulitnya memfaktorkan bilangan besar menjadi faktor-faktor primanya.
2. Pengujian yang dilakukan berdasarkan pada beberapa algoritma berhasilkan dibuktikan dengan baik sehingga melahirkan teorema-teorema yang semakin memperkuat hasil pengujian.
3. Kecepatan verifikasi algoritma ini lebih baik dibandingkan dengan algoritma RSA ataupun *Rabin-Williams* disebabkan karena perancangan algoritma yang efisien sehingga dapat dihasilkan suatu bilangan prima *u* yang dapat digunakan untuk memverifikasi banyak tanda tangan sekaligus.

.

DAFTAR PUSTAKA

[1] Scheneier, Bruce. (1996). *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc. New York.

[2] Munir, Rinaldi. (2006). *Diktat Kuliah IF5054 Kriptografi.* Program Studi

Teknik Informatika, Institut Teknologi Bandung

[3] Http://cr.yp.to

[4] Http://en.wikipedia.com

[5] Merkle, Ralph. (1990). A Fast Software One-Way Hash Function. Journal of Cryptology 3.

[6] Schneier, Bruce. (1996). Applied Cryptography 2nd. John Wiley & Sons.