KOMBINASI ALGORITMA SHA 1 DAN ALGORITMA RSA UNTUK MEMBENTUK TANDA TANGAN DIGITAL

SARI MUSTIKAWATI

JURUSAN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR
BOGOR
2003

RINGKASAN

SARI MUSTIKAWATI. Kombinasi Algoritma SHA 1 Dan Algoritma RSA Untuk Membentuk Tanda Tangan Digital. Dibimbing oleh BIB PARUHUM SILALAHI dan RINDANG KARYADIN.

Tanda tangan digital termasuk ke dalam teknik kriptografi yang memanfaatkan teknologi kunci publik (enkripsi asimetris). Tanda tangan digital dapat dibuat dengan mengkombinasikan teknik *hashing* (untuk membentuk *message digest* dari pesan asli) dan enkripsi asimetris (untuk membentuk kunci publik dan kunci privat). Teknik *hashing* yang digunakan adalah teknik *hashing* satu arah dengan menggunakan algoritma SHA 1, sedangkan algoritma RSA digunakan untuk enkripsi asimetris.

Tulisan ini bertujuan untuk mempelajari, menganalisis, dan mengimplementasikan pembentukan dan verifikasi tanda tangan digital menggunakan dua metode, yaitu dengan mengkombinasikan algoritma SHA 1 dan algoritma RSA dan dengan menggunakan algoritma RSA saja. Analisis yang dilakukan pada kedua algoritma ini meliputi analisis algoritma, dan analisis hasil implementasi.

Analisis terhadap algoritma SHA 1 menunjukkan bahwa algoritma ini memiliki kompleksitas $O(n)$ untuk kasus terburuk. Sedangkan untuk pembentukan kunci publik dan kunci privat dengan algoritma RSA diperlukan beberapa algoritma pendukung, yaitu algoritma Miller Rabin, Witness, Euclid, dan Extended Euclid. Semua algoritma pendukung tersebut memiliki kompleksitas waktu yang sama, yaitu $O(k)$, dengan k adalah panjang representasi bit bilangan bulat.

Kompleksitas waktu yang didapatkan untuk proses pembentukan tanda tangan digital dengan kedua metode ini adalah sama yaitu $O(n + k)$. Untuk proses verifikasi, kompleksitas waktu yang didapatkan dengan menggunakan kombinasi dua algoritma adalah $O(1 + k)$ dan untuk algoritma RSA adalah $O(n + k)$. Pada proses pembentukan tanda tangan digital menggunakan kombinasi dua algoritma yang ditandatangani adalah *message digest* yang dihasilkan oleh algoritma SHA 1, sedangkan pada algoritma RSA yang ditandatangani adalah pesan itu sendiri. Dari hasil implementasi didapatkan untuk proses pembentukan tanda tangan digital menggunakan kombinasi dua algoritma lebih cepat dibandingkan dengan menggunakan algoritma RSA saja untuk file-file yang berukuran besar.

Ucap syukur kehadirat Ilahi, karena atas izin-Nya
Karya kecil ini dapat kupersembahkan untuk
Bapak, Ibu, Kakak, dan keponakanku "Aulia" tersayang

KOMBINASI ALGORITMA SHA 1 DAN ALGORITMA RSA UNTUK MEMBENTUK TANDA TANGAN DIGITAL

SARI MUSTIKAWATI

Skripsi
Sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer
pada
Jurusan Ilmu Komputer

JURUSAN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR
BOGOR
2003

Judul Skripsi : Kombinasi Algoritma SHA 1 Dan Algoritma RSA Untuk Membentuk Tanda Tangan Digital
Nama : Sari Mustikawati
NRP : G06498024
Jurusan : Ilmu Komputer

Menyetujui,

Ir. Bib Paruhum Silalahi, M. Komp
Pembimbing I

Rindang Karyadin, S. T, M. Komp
Pembimbing II

Mengetahui,

Ir. Julio Adisantoso, M. Komp
Ketua Program Studi

Ir. Agus Buono, M. Si, M. Komp
Ketua Jurusan

Tanggal Lulus :

RIWAYAT HIDUP

Penulis dilahirkan di Jakarta pada tanggal 12 Januari 1980 dari orang tua yang bernama Samah dan Sri Irianti. Penulis merupakan anak kedua dari dua bersaudara.

Tahun 1998 penulis lulus dari SMU N 54 Jakarta dan pada tahun yang sama lulus seleksi masuk IPB melalui jalur Undangan Seleksi Masuk IPB. Penulis memilih Program Studi Ilmu Komputer, Jurusan Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam.

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Allah SWT atas segala rahmat dan karunia-Nya sehingga tugas akhir ini dapat diselesaikan. Tema yang dipilih dalam penelitian ini adalah kriptografi, dengan judul Kombinasi Algoritma SHA 1 Dan Algoritma RSA Untuk Membentuk Tanda Tangan Digital.

Terima kasih yang sebesar-besarnya penulis ucapkan kepada Bapak Ir. Bib Paruhum Silalahi, M. Komp selaku pembimbing I dan Bapak Rindang Karyadin, S. T, M. Komp selaku pembimbing II yang telah memberikan saran dan bimbingan selama pengerjaan tugas akhir ini. Selanjutnya, penulis juga ingin mengucapkan terima kasih kepada :

1. Bapak dan Ibu tercinta, terima kasih atas semua do'a, bimbingan, dukungan, kasih sayang, dan kesabarannya menunggu penulis lulus.
2. Mba Santi dan A' Maman, terima kasih atas segala bantuan dan dukungannya. Keponakanku tersayang, Aulia, yang selalu membuat keceriaan dan menghilangkan rasa bosan.
3. Temanku Ima dan Eep terima kasih atas semua bantuannya dalam segala bentuk, terutama untuk pinjaman komputernya.
4. Budhine, Fery, Bayu, Dian atas semua bantuannya, Cucu (terima kasih buat pinjaman bukunya).
5. Ina, Sisi, teman-teman seperjuangan yang selalu memberi keceriaan kepada penulis, dan semua rekan *ilkomerz ' 35* yang telah menorehkan kesan mendalam selama masa perkuliahan.
6. Seluruh staf pengajar Jurusan Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan, IPB. Terima kasih atas semua ilmu yang telah diberikan.
7. Seluruh staf pegawai Jurusan Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan, IPB. Terima kasih atas bantuannya.
8. Ibu Susi (perpustakaan matematika), terima kasih atas informasinya.

Dan semua pihak yang tidak dapat disebutkan satu persatu

Semoga tugas akhir ini dapat bermanfaat, dan untuk semua pihak yang terlibat dalam penyusunan tugas akhir ini, semoga mendapat balasan yang setimpal dari- Nya.

Jakarta, Agustus 2003

SARI MUSTIKAWATI

DAFTAR ISI

Halaman

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

PENDAHULUAN

Latar Belakang

Pengiriman pesan elektronis melalui internet sekarang ini sudah umum dilakukan karena efisien, cepat dan murah. Tapi mungkin ada satu hal yang dilupakan bahwa internet adalah media komunikasi umum yang sangat rawan terhadap penyadapan, pencurian dan pemalsuan informasi. Saat pengiriman pesan, seseorang bisa saja dengan ilegal mengubah isi pesan tanpa diketahui pengirim atau penerima. Pengubahan isi pesan tentu saja akan merugikan karena dapat digunakan untuk berbagai kepentingan. Tanpa fasilitas keamanan yang baik, penerima akan menerima pesan tersebut tanpa mencurigai adanya perubahan.

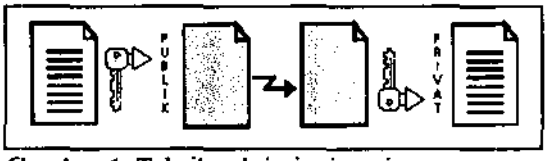
Namun jika pengirim membubuhkan tanda tangan digital pada pesan itu, penerima dapat merasa yakin bahwa setelah ditandatangani pengirim, pesan itu tidak ada yang memanipulasi saat dalam perjalanan. Teknologi tanda tangan digital memanfaatkan teknologi kunci publik. Sepasang kunci publik-privat dibuat untuk keperluan seseorang. Kunci privat disimpan oleh pemiliknya, dan dipergunakan untuk membuat tanda tangan digital. Sedangkan kunci publik dapat diserahkan kepada siapa saja yang ingin memeriksa tanda tangan digital yang bersangkutan pada suatu pesan. Proses pembuatan dan pemeriksaan tanda tangan digital ini melibatkan sejumlah teknik kriptografi seperti *hashing* (membuat *message digest* dari pesan asli) dan enkripsi asimetris (Wibowo, 1998).

Secara garis besar, teknik kriptografi dapat digunakan untuk menyamarkan pesan dan untuk autentikasi pesan. Teknik kriptografi untuk menyamarkan pesan menekankan pada *confidentiality*, yaitu pencegahan akan pengaksesan informasi (*passive attack*) yang dilakukan oleh pihak yang tidak berhak, sedangkan teknik kriptografi untuk autentikasi pesan menekankan pada pencegahan akan modifikasi informasi (*active attack*) yang dilakukan oleh pihak yang tidak berhak. Pada tulisan ini akan dibahas mengenai kombinasi algoritma SHA 1 dan algoritma RSA untuk menghasilkan tanda tangan digital untuk autentikasi pesan.

Teknik *hashing* yang digunakan untuk membentuk tanda tangan digital adalah teknik *hashing* satu arah. Fungsi *hash* satu arah dirancang sedemikian rupa sehingga sangat sulit untuk menentukan pesan yang berkaitan dengan *message digest* yang dihasilkan. Setiap pesan yang berbeda akan menghasilkan *message digest* yang berbeda pula.

Dengan menggunakan fungsi *hash* satu arah maka akan dihasilkan *message digest* dari pesan asli yang akan ditandatangani. *Secure Hash Algorithm* (SHA 1) merupakan salah satu algoritma *hash* yang dapat digunakan untuk menghasilkan *message digest* yang akan menjaga kerahasiaan isi pesan yang dikirimkan.

Salah satu teknik enkripsi asimetris adalah dengan menggunakan teknik penyandian RSA (Rivest, Shamir, Adleman). Teknik enkripsi asimetris menggunakan dua buah kunci untuk proses enkripsi dan dekripsi (**Gambar 1**). Sistem sandi asimetris seperti RSA bisa juga digunakan sebagai tanda tangan digital. Ini membuat aplikasi yang bisa dibuat menggunakan sistem sandi asimetris jauh lebih banyak. RSA adalah sistem sandi yang barangkali paling mudah dimengerti cara kerjanya karena hanya menggunakan operasi pemangkatan.



Gambar 1. Teknik enkripsi asimetris.

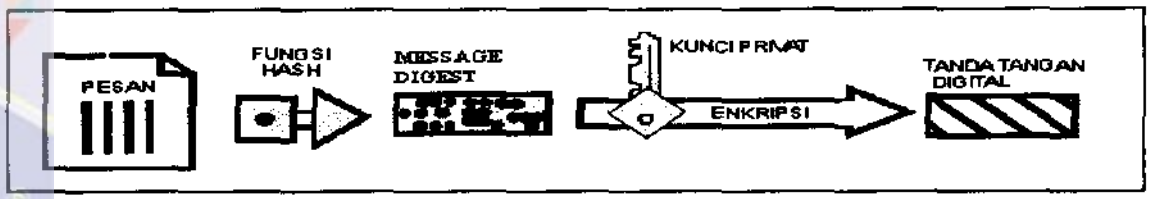
Tujuan

Tujuan dari penelitian ini adalah :

1. Mempelajari dan memberikan alternatif proses pembentukan tanda tangan digital menggunakan kombinasi algoritma SHA 1 dan algoritma RSA.
2. Melakukan analisis terhadap kedua algoritma tersebut.
3. Mengimplementasikan kedua algoritma tersebut menggunakan bahasa pemrograman Visual Basic 6.0.
4. Membandingkan kecepatan kombinasi kedua algoritma tersebut dengan algoritma RSA dalam membentuk tanda tangan digital.

Ruang Lingkup

Ruang lingkup penelitian ini meliputi proses pembentukan dan verifikasi tanda tangan digital menggunakan dua metode. Metode yang pertama dengan mengkombinasikan algoritma SHA 1 dan algoritma RSA, sedangkan metode kedua dengan menggunakan algoritma RSA. Pada penelitian ini tidak dibahas mengenai mekanisme pendistribusian kunci publik.



Gambar 2. Proses pembentukan tanda tangan.

TINJAUAN PUSTAKA

Kriptografi

Kriptografi (*cryptography*) adalah merupakan suatu ilmu dan seni untuk menjaga data-data atau informasi agar aman. Kriptografi berasal dari kata "*Crypto*" yang berarti "*Secret*" (rahasia) dan "*Graphy*" yang berarti "*Writing*" (tulisan). Sebuah algoritma kriptografik (*Cryptographic Algorithm*), disebut *cipher* yaitu persamaan matematik yang digunakan untuk proses Enkripsi dan Dekripsi.

Metode Enkripsi ini digunakan untuk meyandikan data-data atau informasi sehingga tidak dapat dibuka atau dibaca oleh orang lain yang tidak berhak. Dengan enkripsi data akan disandikan (*encrypted*) dengan menggunakan sebuah kunci (*key*). Untuk membuka (*decrypt*) data tersebut dapat digunakan dua buah cara:

1. Menggunakan kunci yang sama dengan kunci yang digunakan untuk mengenkripsi (digunakan pada kasus *private key cryptography* / enkripsi simetris).
2. Menggunakan kunci yang berbeda (digunakan pada kasus *public key cryptography* / enkripsi asimetris).

Tanda Tangan Digital

Tanda tangan digital mirip dengan tanda tangan yang dihasilkan oleh tulisan tangan. Tanda tangan digital adalah cara yang tepat secara matematis untuk membubuhkan identitas seseorang pada suatu pesan. Dibandingkan tanda tangan yang dihasilkan oleh tulisan tangan maka tanda tangan digital lebih sulit untuk dipalsukan (Grabbe, 1998). Menurut Wibowo (1998) tanda tangan digital memiliki beberapa sifat, yaitu :

1. Otentik, tidak bisa/sulit ditulis/ditiru oleh orang lain. Pesan dan tanda tangan tersebut juga bisa menjadi barang bukti, sehingga penandatangan tidak bisa menyangkal bahwa dulu ia tidak pernah menandatanganinya.
2. Hanya sah untuk dokumen (pesan) itu saja atau kopinya yang sama persis. Tanda tangan itu tidak bisa dipindahkan ke pesan lainnya, meskipun pesan lain itu hanya berbeda sedikit. Ini juga berarti bahwa jika pesan itu diubah, maka tanda tangan digital dari pesan tersebut tidak lagi sah.
3. Dapat diperiksa dengan mudah, termasuk oleh pihak-pihak yang belum pernah bertatap muka langsung dengan penandatangan.

Algoritma Tanda Tangan Digital

Algoritma tanda tangan digital ada banyak macamnya. Semuanya adalah algoritma kunci publik dengan informasi rahasia untuk menandatangani pesan dan informasi yang dapat disebarkan ke publik untuk memeriksa keaslian tanda tangan digital. Terkadang proses penandatanganan disebut enkripsi dengan sebuah kunci privat dan proses pemeriksaan keaslian tanda tangan disebut dekripsi dengan sebuah kunci publik. Hal ini menimbulkan salah pengertian karena hal ini hanya berlaku untuk satu algoritma, yaitu algoritma RSA. Banyak algoritma yang dapat digunakan untuk tanda tangan digital, tetapi tidak dapat digunakan untuk enkripsi (Schneier, 1996).

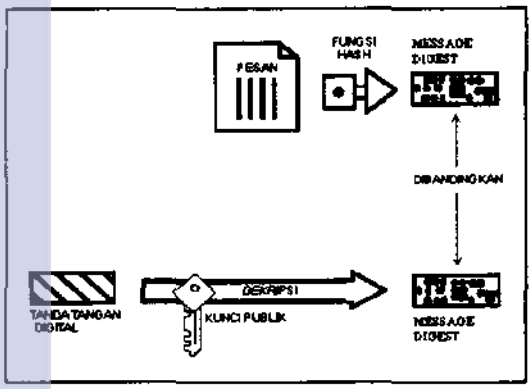
Tanda Tangan Digital dengan Kunci Publik dan Fungsi *Hash* Satu Arah

Pada imlementasinya, algoritma kunci publik terkadang tidak efisien untuk menandatangani pesan yang panjang. Untuk menghemat waktu, protokol tanda tangan digital biasanya diimplementasikan dengan fungsi *hash* satu arah. Pada protokol ini, baik fungsi *hash* satu arah dan algoritma tanda tangan digital telah disetujui bersama sebelumnya baik oleh pengirim maupun penerima pesan. Fungsi *hash* satu arah digunakan untuk menjamin bahwa tanda tangan itu hanya berlaku untuk pesan yang bersangkutan saja.

Proses yang terjadi dalam pembentukan tanda tangan digital adalah (**Gambar 2**) :

1. Pengirim membentuk *message digest* (*hash value*) dari pesan yang akan dikirimkan.
2. Pengirim menandatangani *message digest* dengan kunci privat yang dimilikinya.
3. Pengirim mengirimkan pesan dan *message digest* yang telah ditandatangani (tanda tangan digital).

Penerima pesan dapat memeriksa keabsahan tanda tangan digital itu dengan cara membuat lagi *message digest* dari pesan yang diterimanya. Kemudian penerima pesan mendekripsi tanda tangan digital pengirim menggunakan kunci publik untuk mendapatkan *message digest* yang asli. Penerima pesan kemudian membandingkan kedua *message digest* tersebut. Jika kedua *message digest* tersebut sama, maka dapat diyakini bahwa pesan tersebut memang ditanda tangani oleh pengirim pesan (**Gambar 3**).



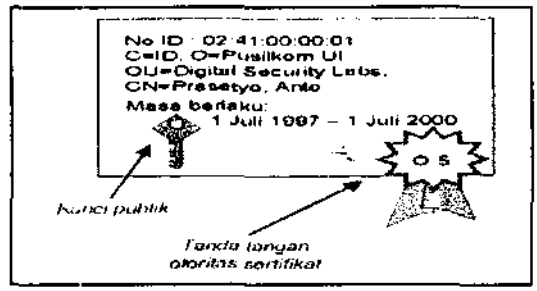
Gambar 3. Proses pemeriksaan keabsahan tanda tangan digital.

Namun sebenarnya pada proses pembentukan tanda tangan digital ini ada masalah dalam pendistribusian kunci publiknya. Hal ini dapat dijelaskan sebagai berikut :

1. Misalkan Anto ingin mengirimkan kunci publiknya kepada Badu. Tapi saat kunci itu dikirimkan lewat jaringan publik, Andi mencuri kunci publik Anto, kemudian Andi menyerahkan kunci publiknya kepada Badu sambil mengatakan bahwa kunci itu adalah kunci publik milik Anto.
2. Badu, karena tidak pernah memegang kunci publik milik Anto yang asli percaya saja saat menerima kunci publik Andi.
3. Saat Anto hendak mengirim pesan yang telah ditandatangani dengan kunci privatnya kepada Badu, sekali lagi Andi mencurinya. Tanda tangan Anto pada pesan itu lalu dihapus, dan kemudian Andi membubuhkan tanda tangan dengan kunci privatnya.
4. Andi mengirim pesan itu kepada Badu sambil mengatakan pesan itu berasal dari Anto dan ditandatangani oleh Anto. Badu kemudian memeriksa tanda tangan itu, dan mendapatkan bahwa tanda tangan itu sah dari Anto. Tentu saja tanda tangan itu kelihatan sah, karena Badu memeriksanya dengan kunci publik Andi, bukan dengan kunci publik Anto.

Untuk mengatasi masalah sekuriti pendistribusian kunci publik, maka kunci publik itu direkatkan pada suatu sertifikat digital. Sertifikat digital selain berisi kunci publik juga berisi informasi lengkap mengenai jati diri pemilik kunci tersebut, sebagaimana layaknya KTP, seperti nomor seri, nama pemilik, kode negara/perusahaan, masa berlaku, dan sebagainya (**Gambar 4**).

Sama halnya dengan KTP, sertifikat digital juga ditandatangani secara digital oleh lembaga yang mengeluarkannya, yaitu otoritas sertifikat (OS) atau *certificate authority* (CA). Dengan menggunakan kunci publik dari suatu sertifikat digital, pemeriksa tanda tangan dapat merasa yakin bahwa kunci publik itu memang berkorelasi dengan seseorang yang namanya tercantum dalam sertifikat digital itu (Wibowo, 1998).



Gambar 4. Contoh sertifikat digital.

Fungsi *Hash* Satu Arah

Fungsi *hash* satu arah memiliki banyak nama, diantaranya : fungsi kompresi, *message digest*, sidik jari, *manipulation detection code* (MDC), *message authentication code* (MAC), dan *data authentication code* (DAC). Fungsi *hash* satu arah, *H(M)* bekerja pada pesan, *M*, dengan panjang input yang bervariasi dan mengubahnya ke dalam ukuran yang tetap yang disebut *message digest* atau *hash value*, *h*. Karakteristik yang dimiliki fungsi *hash* satu arah adalah :

1. Jika diberikan pesan, *M*, maka akan mudah untuk menghitung *hash value*, *h*.

2. Jika diberikan *hash value*, *h*, maka akan sulit untuk menghitung pesan, *M*, dimana *H(M)* = *h*.
3. Jika diberikan pesan *M*, maka akan sulit untuk menemukan pesan lain, *M'* ,dimana *H(M)* = *H(M')*.

Fungsi *hash* satu arah dirancang sedemikian rupa sehingga sangat sulit untuk menentukan pesan yang berkaitan dengan *message digest* yang dihasilkan. Setiap pesan yang berbeda akan menghasilkan *message digest* yang berbeda pula.

Ada dua macam fungsi *hash*. Pertama adalah fungsi *hash* tanpa kunci, dimana *hash* nya adalah fungsi dari input. Fungsi *hash* semacam ini dapat dihitung oleh siapa saja. Kedua adalah fungsi *hash* dengan kunci yaitu fungsi dari input dan kunci, hanya orang yang memiliki kunci saja yang dapat menghitung *message digest* yang dihasilkan (Schneier, 1996).

Algoritma SHA 1

SHA 1 adalah salah satu algoritma *hash* yang umum digunakan. Algoritma ini dirancang oleh *The National Institute of Standards and Technology* (NIST) pada tahun 1993. SHA 1 menghasilkan *message digest* dengan panjang 160 bit. SHA aman karena didesain untuk tidak memungkinkan mendapatkan pesan yang berhubungan dengan *message digest*, atau untuk menemukan dua pesan yang berbeda yang menghasilkan *message digest* yang sama.

Algoritma SHA 1 merupakan perbaikan dari algoritma MD 4 yang diciptakan oleh Profesor Ronald L. Rivest. Proses untuk menghasilkan *message digest* pada algoritma ini meliputi lima tahapan, yaitu : *message padding*, penambahan panjang bit, inisialisasi buffer, memroses 16 subblok 32 bit, dan output yang dihasilkan.

Algoritma RSA

Algoritma RSA (kepanjangan dari nama penemu-penemunya yaitu Rivest, Shamir, Adleman yang membuatnya di tahun 1978) merupakan sistem kriptografi dengan kunci publik. Algoritma RSA merupakan algoritma yang paling populer diantara algoritma kunci publik lainnya karena algoritma tersebut relatif mudah untuk dipelajari dan diimplementasikan. Algoritma RSA menggunakan bilangan prima untuk membentuk kunci publik dan kunci privat. Kunci publik dan kunci privat dihubungkan melalui suatu formula matematika yang menggunakan bilangan prima.

Bilangan Prima dan Bilangan Komposit

Bilangan *a*, dimana $a > 1$ disebut bilangan prima apabila bilangan itu hanya dapat dibagi oleh bilangan 1 dan bilangan *a* itu sendiri. Sedangkan bilangan yang bukan prima disebut bilangan komposit. Sebagai contoh bilangan 39 adalah bilangan komposit karena 39 dapat dibagi habis dengan bilangan 3. Bilangan 0, 1, dan semua bilangan bulat negatif tidak termasuk bilangan prima dan juga bilangan komposit (Cormen *et al*, 1990).

Faktor Persekutuan Terbesar

Faktor persekutuan terbesar (*Greatest Common Divisor* / GCD) dari dua bilangan bulat *a* dan *b*, dimana keduanya tidak sama dengan nol, adalah bilangan bulat terbesar yang membagi habis *a* dan *b*. Faktor persekutuan terbesar dari *a* dan *b* dinotasikan dengan GCD(*a*,*b*). Sebagai contoh : GCD(24,30) = 6, GCD(5,7) = 1 (Cormen *et al*, 1990).

Relatif Prima

Dua buah bilangan *a* dan *b* disebut relatif prima apabila GCD(*a*,*b*) = 1. Sebagai contoh bilangan 8 dan 15 adalah relatif prima, karena pembagi 8 adalah 1, 2, 4, dan 8, sedangkan pembagi 15 adalah 1, 3, dan 5, sehingga GCD(8, 15) = 1 (Cormen *et al*, 1990).

Analisis Algoritma

Algoritma adalah prosedur perhitungan yang terdefinisi dengan baik yang mengambil beberapa nilai, atau sekumpulan nilai sebagai input dan menghasilkan output. Algoritma dapat juga diartikan sebagai urutan dari langkah-langkah perhitungan yang merubah input menjadi output. Dengan menganalisis suatu algoritma dapat diduga besarnya sumber daya yang dibutuhkan oleh suatu algoritma. Sumber daya dapat berupa memori, *communication bandwidth*, atau gerbang logika (*logic gate*).

Kompleksitas komputasi, *T(n)* didefinisikan sebagai waktu yang dibutuhkan oleh suatu program dalam menyelesaikan masalah dengan input yang berukuran *n*. Berdasarkan waktu eksekusi program, *T(n)*, dapat ditentukan *growth rate* dari *T(n)*, yaitu laju pertumbuhan waktu terhadap variasi ukuran input. Sebagai contoh, analisis terhadap suatu algoritma menghasilkan $T(n) = an^2 + bn + c$, dengan *a*, *b*, *c* konstanta,

maka dikatakan growth rate dari $T(n)$ adalah n^2, yang merupakan bagian paling signifikan pada bentuk polinomial kuadrat $an^2 + bn + c$. Nilai-nilai konstanta a, b, c tergantung pada jenis komputer, dan bahasa pemrograman, yang digunakan untuk menjalankan program dan hanya dapat ditentukan melalui percobaan eksekusi program.

Kompleksitas komputasi dari suatu algoritma memberikan kemudahan dalam menentukan bagaimana perilaku $T(n)$ berubah-ubah terhadap n. Kompleksitas komputasi tidak dipengaruhi oleh implementasi menggunakan bahasa pemrograman tertentu. Suatu algoritma dengan *growth rate* yang rendah, misalnya *log n* atau *n.log n*, berjalan lebih cepat jika dibandingkan dengan algoritma yang memiliki *growth rate* lebih besar, misalnya n^2, n^3, 2^n, $n!$, dan n^n.

Pada tulisan ini, algoritma SHA 1 dan algoritma RSA dianalisis berdasarkan keadaan kompleksitas waktu komputasi untuk kasus terburuk (*worst case*). Waktu komputasi untuk kasus terburuk didefinisikan sebagai waktu komputasi terlama untuk sembarang input berukuran n. Menurut Cormen *et al* (1990) ada tiga alasan mengapa perlu menganalisa waktu komputasi untuk kasus terburuk, yaitu :

1. Kasus terburuk dari suatu algoritma adalah batas atas dari waktu komputasi untuk setiap input. Dengan mengetahui waktu komputasi terlama akan memberikan jaminan bahwa algoritma tersebut tidak akan lebih lama dari pada kasus terburuknya.
2. Untuk beberapa algoritma kasus terburuk cukup sering terjadi. Misalnya pada proses pencarian informasi di basis data, kasus terburuk pada algoritma pencarian akan terjadi ketika informasi yang dicari tidak terdapat pada basis data.
3. Terkadang *average case* sama buruknya dengan kasus terburuk.

Deskripsi Algoritma SHA 1

Langkah-langkah yang dilakukan dalam membentuk *message digest* dengan menggunakan algoritma SHA 1 terdiri dari lima langkah, yaitu :

1. Message padding

Input pesan pada algoritma SHA 1 dibagi menjadi blok-blok yang masing-masing panjangnya adalah 512 bit. Akibat pembagian ini maka jumlah bit pada blok terakhir akan lebih kecil atau sama dengan 512 bit. Selanjutnya blok terakhir ini akan mengalami *message padding*.

Proses pembentukan *message padding* adalah sebagai berikut :

- Input pesan yang masuk dalam bentuk *American Standard Code for Information Interchange* (ASCII) diubah ke dalam rangkaian bit, dan hitung panjang rangkaian bit ini (K).
- Selanjutnya rangkaian bit ini dibagi menjadi blok-blok yang panjangnya masing-masing 512 bit. Blok terakhir panjangnya akan lebih kecil atau sama dengan 512 bit.
- Pada blok terakhir pesan lakukan penambahan bit-bit isian (*padding*). Bit yang digunakan sebagai bit isian adalah bit "1" diikuti sejumlah bit "0" sehingga :
  - Jika panjang bit pesan asli lebih kecil dari 448 bit, maka tambahkan bit "1" pada posisi bit paling akhir, diikuti dengan beberapa bit "0" sehingga total panjang bit setelah proses tersebut adalah 448 bit.
  - Jika panjang bit pesan asli lebih besar atau sama dengan 448 bit, maka tambahkan bit "1" pada posisi bit paling akhir diikuti beberapa bit "0" sehingga total panjang bit setelah proses tersebut adalah 512 bit. Kemudian buat 448 bit baru yang isinya bit "0".
  - Jika panjang bit pesan asli sama dengan 512 bit, maka buat blok baru untuk menampung proses *message padding*. Bit pertama dari blok baru diisi bit "1", sedangkan bit-bit berikutnya diisi dengan bit "0"sampai dengan panjang bit 448.

Sebagai contoh misalkan isi pesan adalah "abc", maka ubah pesan tersebut ke dalam kode ASCII-nya, yaitu : a = 97, b = 98, c = 99. Kemudian ubah representasi desimal dari kode ASCII tersebut ke dalam representasi binernya, yaitu : 01100001 | 01100010 | 01100011, panjang pesan asli ini (K) adalah 24 bit. Karena panjang bit pesan asli kurang dari 448 bit, maka dilakukan proses *padding* dengan menambahkan "1" pada posisi bit paling akhir, dan tambahkan "0" sehingga panjang total bit setelah proses *padding* ini adalah 448 bit.

2. Penambahan panjang bit

Setelah proses *message padding*, jumlah bit pada blok terakhir adalah 448 bit. Representasikan K ke dalam bilangan biner untuk memperoleh 64

bit terakhir, agar total panjang blok terakhir 512 bit. Urutan byte paling kanan dijadikan *low order*.

Misalkan panjang pesan asli (K) = 24 bit dan direpresentasikan ke dalam bilangan biner 16 bit adalah 00000000 | 00011000. Nilai ini dijadikan 64 bit sehingga : 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00011000

Nilai di atas ditambahkan pada langkah (1), sehingga total panjang bit adalah 512.

3. Inisialisasi buffer

Untuk menyimpan nilai inisialisasi awal digunakan buffer H_0, H_1, H_2, H_3, H_4. Sedangkan untuk menyimpan nilai proses sementara digunakan buffer A, B, C, D, E. Nilai H_0, H_1, H_2, H_3, H_4 sebagai inisialisasi awal adalah :

H_0 = 67 45 23 01
H_1 = EF CD AB 89
H_2 = 98 BA DC FE
H_3 = 10 32 54 76
H_4 = C3 D2 E1 F0

4. Memproses 16 subblok 32 bit

Langkah ini terdiri dari empat *round*, masing-masing *round* terdiri dari 20 operasi sehingga untuk setiap satu blok akan dilakukan operasi sebanyak 80 kali.

- Blok pesan ditransformasi dari 16 subblok 32 bit (M_0 sampai M_{15}) menjadi 80 subblok 32 bit (W_0 sampai W_{79}) menggunakan langkah berikut :

$W_t = M_t$ untuk t = 0 sampai t = 15
$W_t = (W_{t-3}$ XOR W_{t-8} XOR W_{t-14} XOR $W_{t-16})<<<1$ untuk t = 16 sampai t = 79
A = H_0, B = H_1, C = H_2, D = H_3, E = H_4

- Untuk setiap *round* didefinisikan fungsi f_t :

f_t (B,C,D) = (B AND C) OR ((NOT B) AND D), untuk operasi 0-19
f_t (B,C,D) = B XOR C XOR D, untuk operasi 20-39
f_t (B,C,D)= (B AND C) OR (B AND D) OR (C AND D), untuk operasi 40-59
f_t (B,C,D) = B XOR C XOR D, untuk operasi 60-79

- Diperlukan deret konstanta K(0), K(1),..., K(79)

K_t = 5A827999, untuk operasi 0-19
K_t = 6ED9EBA1, untuk operasi 20-39
K_t = 8F1BBCDC, untuk operasi 40-59
K_t = CA62C1D1, untuk operasi 60-79

- Untuk setiap satu langkah operasi pada SHA 1, t=0 sampai t=79 lakukan :

TEMP=(A<<<5)+ f_t (A,B,C)+E+ W_t+K_t
E=D; D=C; C=(B<<<30); B=A; A=TEMP;

Hasil akhir adalah : $H_0 = H_0 + A$,
$H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$,
$H_4 = H_4 + E$.

5. Output

Setelah semua blok 512 bit diproses maka akan dihasilkan output. Blok terakhir menghasilkan *message digest* dari pesan yang dimasukkan yaitu buffer H_0, H_1, H_2, H_3, H_4. Panjang *message digest* yang dihasilkan adalah 160 bit dan masing-masing buffernya terdiri dari delapan digit heksadesimal.

Deskripsi Algoritma RSA

Algoritma RSA digunakan untuk membentuk kunci publik dan kunci privat. Langkah-langkah yang dilakukan adalah sebagai berikut :

1. Pilih dua bilangan prima p dan q secara acak
2. Hitung $n = p * q$ dan $PHI = (p-1) * (q-1)$
3. Pilih secara acak bilangan integer e, dimana $1<e<PHI$, sehingga gcd $(e, PHI) = 1$ (artinya e relatif prima terhadap PHI)
4. Kemudian hitung integer unik d, dimana $1<d<PHI$, sehingga $d = (e^{-1})$ mod PHI
5. (n, e) adalah kunci publik dan d adalah kunci privat.

Pembentukan Dan Verifikasi Tanda Tangan Digital

Untuk membentuk tanda tangan digital suatu pesan, m, pertama pesan dibagi-bagi ke dalam sejumlah blok pesan, m_i. Persamaan yang digunakan untuk membentuk tanda tangan digital adalah :

$$s_i = m_i^d \bmod n$$

Sedangkan untuk proses verifikasi tanda tangan digital dilakukan proses pemangkatan pada tanda tangan digital yang dihasilkan menggunakan kunci publik, yaitu :

$$m_i = s_i^e \bmod n$$

Untuk pembentukan tanda tangan digital menggunakan kombinasi algoritma SHA 1 dan RSA, yang ditanda tangani bukan pesan yang diinput, melainkan *message digest* yang dihasilkan oleh algoritma SHA 1. Jika hasil verifikasi sama dengan *message digest* yang dihasilkan oleh pesan m, maka tanda tangan digital yang dihasilkan sah, sedangkan jika hasil verifikasi yang dihasilkan berbeda dengan *message digest* yang dihasilkan, maka tanda tangan digital yang dihasilkan tidak sah. Proses pembentukan dan verifikasi tanda tangan digital menggunakan algoritma modular exponen.

HASIL DAN PEMBAHASAN

A. Analisis Algoritma

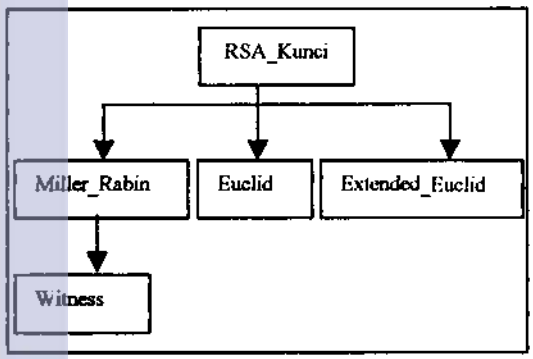
Analisis yang dilakukan terhadap algoritma didasarkan pada asumsi bahwa mesin yang digunakan adalah model *random access machine* (RAM), berprosesor tunggal. Pada mesin ini instruksi-instruksi program dieksekusi baris per baris secara berurutan. Operasi yang akan dianalisis adalah operasi aritmatika dasar, yaitu penjumlahan, pengurangan, perkalian, pembagian, dan operasi *assignment*, serta diasumsikan lamanya waktu eksekusi setiap operasi tersebut adalah satu satuan waktu.

A.1. Analisis Algoritma SHA 1

Proses untuk menghasilkan *message digest* dengan menggunakan algoritma SHA 1 meliputi lima tahapan, yaitu : *message padding*, penambahan panjang bit, inisialisasi buffer, memproses 16 subblok 32 bit, dan output yang dihasilkan. Tiga langkah yang pertama memiliki waktu eksekusi konstan karena tidak tergantung dari panjang input. Pada langkah yang keempat, setiap 512 bit blok pesan dioperasikan sebanyak 80 kali. Sehingga total waktu eksekusi algoritma ini adalah $80n + x$, dengan x adalah suatu konstanta. Jadi kompleksitas waktu algoritma SHA 1 sama dengan $O(n)$.

A.2. Analisis Algoritma RSA

Untuk menghasilkan kunci publik dan kunci privat dengan menggunakan algoritma RSA diperlukan subrutin-subrutin pendukung seperti yang terlihat pada hirarki algoritma RSA (**Gambar 5**).



Gambar 5. Hirarki algoritma RSA.

A.2.1. Analisis Subrutin Witness

Subrutin Witness berfungsi untuk menentukan apakah suatu bilangan bulat, a, merupakan saksi (*witness*) bagi kekompositan bilangan bulat n. Jika subrutin Witness mengembalikan nilai True, maka bilangan bulat n merupakan bilangan komposit, tetapi jika subrutin Witness mengembalikan nilai False, maka bilangan bulat n merupakan bilangan prima. Algoritma subrutin Witness adalah sebagai berikut :

Function Witness (Byval *a* as Long, Byval *n* as Long) As Boolean
[1] Witness ← True
[2] $d \leftarrow 1$
[3] Andaikan $(b_k, b_{k-1},\ldots,b_0)$ adalah representasi biner dari $n - 1$
[4] For $i \leftarrow k$ downto 0 do
[5] $x \leftarrow d$
[6] $d \leftarrow (d * d)$ Mod n
[7] If $(d = 1)$ And $(x <> 1)$ And $(x <> (n - 1))$ Then
[8] Exit Function
[9] If $b_i = 1$ Then
[10] $d = (d * a)$ Mod n
[11] If $d <> 1$ Then
[12] Exit Function
[13] Witness ← False

Waktu eksekusi subrutin Witness secara keseluruhan ditentukan oleh baris [4]. Total waktu eksekusi subrutin ini adalah $kv_1 + v_2$, dengan v_1 dan v_2 adalah suatu konstanta dan k adalah panjang representasi biner dari n-1. v_1 adalah waktu eksekusi di dalam kondisi perulangan baris [4], sedangkan v_2 adalah waktu eksekusi di luar kondisi perulangan baris [4]. Jadi subrutin Witness mempunyai kompleksitas waktu sama dengan $O(k)$.

A.2.2. Analisis Subrutin Miller_Rabin

Subrutin Miller_Rabin berfungsi untuk menguji apakah suatu bilangan bulat, n, adalah bilangan prima atau bukan dengan melakukan sejumlah s kali perulangan. Jika n adalah bilangan prima, maka subrutin ini akan mengembalikan nilai True, jika sebaliknya maka subrutin ini akan mengembalikan nilai False. Algoritma subrutin Miller_Rabin adalah sebagai berikut :

Function Miller_Rabin (Byval *n* as Long, Byval *s* as Byte) As Boolean
[1] Miller_Rabin ← False
[2] For $i \leftarrow 1$ to s do
[3] $a \leftarrow$ Random $(1, n-1)$
[4] If Witness(a, n) Then
[5] Exit Function

[6] Miller_Rabin ← True

Kasus terburuk terjadi pada saat subrutin Miller_Rabin mengembalikan nilai True, artinya bilangan bulat *n* yang diuji adalah bilangan prima. Waktu eksekusi subrutin Miller_Rabin secara keseluruhan ditentukan oleh baris program [2], sedangkan waktu eksekusi di luar baris itu memiliki nilai konstan. Menurut Cormen (1990) pemilihan nilai $s = 50$ sudah cukup baik, sehingga total waktu eksekusi subrutin Miller_Rabin adalah 50 * total waktu eksekusi subrutin Witness + konstanta. Jika *n* adalah bilangan bulat dengan jumlah bit *k*, maka kompleksitas waktu subrutin Miller_Rabin adalah $O(k)$.

A.2.3. Analisis Subrutin Euclid

Subrutin Euclid berfungsi untuk menentukan faktor persekutuan terbesar antara dua bilangan bulat *a* dan *b*. Algoritma subrutin Euclid adalah sebagai berikut :

Function Euclid (Byval *a* as Long, Byval *b* as Long) As Long
[1] If $b = 0$ Then
[2] Euclid ← a
[3] Else
[4] Euclid ← (b, a mod b)

Waktu eksekusi subrutin Euclid sebanding dengan jumlah pemanggilan rekursif yang dilakukannya yaitu sebesar $O(\lg b)$. Analisis terhadap subrutin ini dilakukan dengan menggunakan bilangan *Fibonacci*, F_k (Bukti : lihat **Lampiran 1**). Jika subrutin Euclid diberikan input dua buah bilangan bulat dengan panjang bit *k*, maka kompleksitas subrutin ini adalah $O(k)$.

A.2.4. Analisis Subrutin Extended_Euclid

Subrutin Extended_Euclid digunakan untuk mencari kunci privat *d*. Algoritma subrutin Extended_Euclid adalah sebagai berikut :

Function Extended_Euclid (Byval *a* as Long, Byval *b* as Long) As Long
[1] If $b = 0$ Then
[2] return (a, 1, 0)
[3] (d1, x1, y1) ← Extended_Euclid (b,a mod b)
[4] (d, x, y) ← (d1, y1, x1 - $\lfloor a/b \rfloor$ y1)
[5] return (d,x,y)

Oleh karena jumlah pemanggilan rekursif subrutin Extended_Euclid sama dengan subrutin Euclid, maka kompleksitas waktu untuk subrutin Extended_Euclid adalah sebesar $O(k)$.

Setelah menganalisis subrutin-subrutin yang digunakan oleh algoritma RSA dalam membentuk kunci publik dan kunci privat, maka analisis dilanjutkan pada algoritma rutin RSA_Kunci sebagai berikut :

Procedure RSA_Kunci
[1] a = INT (RND * 200)
[2] p ← 2 + a
[3] If (p mod 2 = 0) Then
[4] p ← p + 1
[5] prime ← False
[6] Do while prime = True
[7] If Miller_Rabin (p, 50) Then
[8] prime ← True
[9] else
[10] p ← p + 2
[11] Loop
[12] q ← 2 + a
[13] If (q mod 2 = 0) Then
[14] q ← q + 1
[15] prime ← False
[16] Do while prime = True
[17] If Miller_Rabin (q, 50) Then
[18] prime ← True
[19] else
[20] q ← q + 2
[21] Loop
[22] n ← p * q
[23] e ← 2
[24] Do while Euclid (e, (p-1) *(q-1)) <> 1
[25] e ← e + 1
[26] Loop
[27] Extended_Euclid (e, (p-1) * (q-1))

Baris yang paling dominan pada rutin ini adalah baris [6], [16], dan [24], selain itu waktu eksekusi baris lainnya adalah konstan. Maka kompleksitas rutin RSA_Kunci adalah $O(k)$.

A.3. Analisis Algoritma Modular Exponen

Algoritma modular exponen digunakan untuk melakukan operasi eksponensial modular, a^b mod *n*. Algoritmanya adalah sebagai berikut :

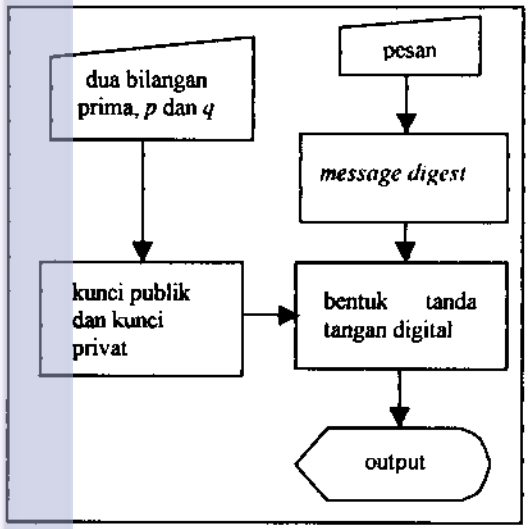
Function Modular_Exponen (Byval *a* as Long, Byval *b* as Long, Byval *n* as Long)
[1] d ← 1
[2] Andaikan (b_k, b_{k-1},....,b_0) adalah representasi biner dari b
[3] For i ← k downto 0 do

[4] $d \leftarrow (d * d) \bmod n$
[5] If $b_i = 1$ Then
[6] $d \leftarrow (d * a) \bmod n$
[7] Modular_Exponen $\leftarrow d$

Waktu eksekusi algoritma modular exponen secara keseluruhan ditentukan oleh kondisi perulangan baris [3], sedangkan baris lainnya memiliki waktu eksekusi konstan. Total waktu eksekusi subrutin ini adalah $kv_1 + v_2$, dengan v_1 dan v_2 adalah suatu konstanta dan k adalah panjang representasi biner dari bilangan bulat b. v_1 adalah waktu eksekusi di dalam kondisi perulangan baris [3], sedangkan v_2 adalah waktu eksekusi di luar kondisi perulangan baris [3]. Jadi algoritma modular exponen mempunyai kompleksitas waktu sama dengan $O(k)$.

A.4. Analisis Algoritma Pembentukan Tanda Tangan Digital

Pada tulisan ini proses pembentukan tanda tangan digital diimplementasikan dengan menggunakan dua metode. Metode yang pertama adalah dengan menggunakan kombinasi algoritma SHA 1 dan algoritma RSA, sedangkan metode yang kedua adalah dengan menggunakan algoritma RSA saja. Diagram alir proses pembentukan tanda tangan digital dengan kedua metode tersebut dapat dilihat pada **Gambar 6** dan **Gambar 7**.

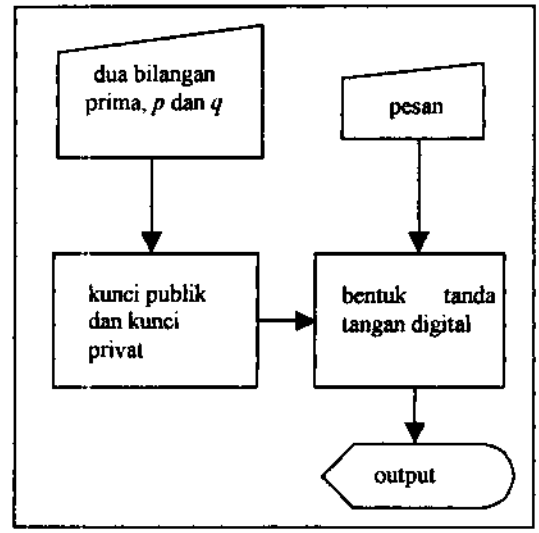


Gambar 6. Proses pembentukan tanda tangan digital dengan kombinasi algoritma SHA 1 dan RSA.

Langkah untuk membentuk tanda tangan digital dengan kombinasi dua algoritma adalah :

1. pembentukan *message digest*
2. pembentukan kunci publik dan privat
3. bentuk tanda tangan digital
4. output

Waktu eksekusi untuk langkah 2 dan 4 adalah konstan, misalkan t_1, sedangkan waktu langkah 1 dibentuk dengan menggunakan algoritma SHA 1 yang memiliki total waktu eksekusi sebesar $80n + x$, dimana x adalah suatu konstanta. Kompleksitas waktu untuk langkah 2 adalah $O(k)$, dengan k adalah panjang representasi bilangan biner. Sehingga total kompleksitas waktu proses pembentukan tanda tangan digital dengan kombinasi dua algoritma adalah $O(n + k)$.



Gambar 7. Proses pembentukan tanda tangan digital dengan algoritma RSA.

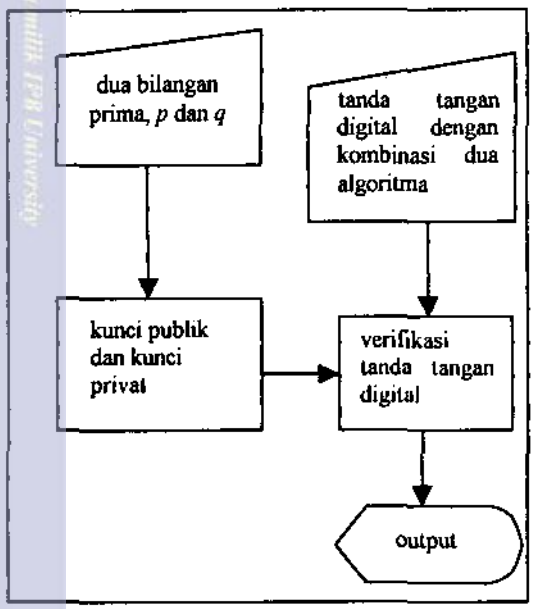
Langkah untuk membentuk tanda tangan digital dengan algoritma RSA adalah :

1. pembentukan kunci publik dan privat
2. bentuk tanda tangan digital
3. output

Kompleksitas waktu untuk langkah 1 adalah $O(k)$, dengan k adalah panjang representasi bilangan biner. Waktu eksekusi langkah 2 dipengaruhi oleh panjang pesan yang diinput untuk dibentuk tanda tangan digitalnya. Jadi untuk ukuran input sebesar n diperlukan waktu eksekusi sebesar $t_1 n$. Sedangkan waktu eksekusi untuk langkah 3 adalah konstan, misalkan t_2. Total kompleksitas waktu proses pembentukan tanda tangan digital dengan algoritma RSA adalah $O(n + k)$.

A.5. Analisis Algoritma Verifikasi Tanda Tangan Digital

Algoritma untuk proses verifikasi tanda tangan digital juga diimplementasikan dengan menggunakan dua metode yang sama dengan proses pembentukan tanda tangan digital. Diagram alir dari kedua metode tersebut dapat dilihat pada **Gambar 8** dan **Gambar 9**.

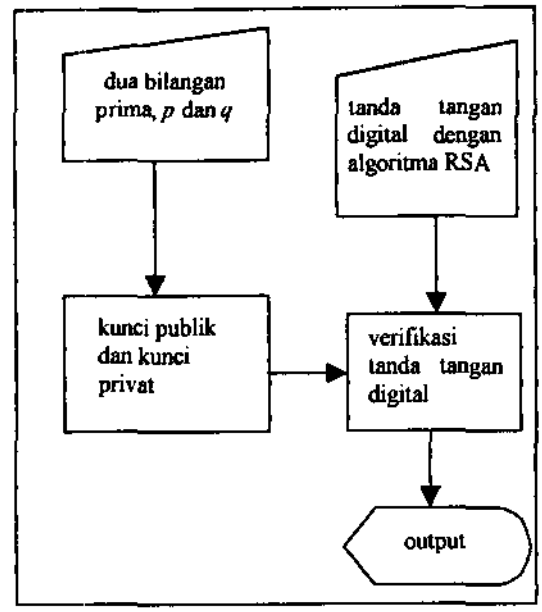


Gambar 8. Proses verifikasi tanda tangan digital dengan kombinasi algoritma SHA 1 dan RSA.

Langkah untuk verifikasi tanda tangan digital dengan kombinasi dua algoritma adalah :

1. pembentukan kunci publik dan privat
2. verifikasi tanda tangan digital
3. output

Kompleksitas waktu untuk langkah 1 adalah $O(k)$, dengan k adalah panjang representasi bilangan biner. Waktu eksekusi untuk langkah 2 dan 3 adalah konstan, misalkan v_1. Jadi total kompleksitas waktu proses pembentukan tanda tangan digital dengan kombinasi dua algoritma adalah $O(1 + k)$.



Gambar 9. Proses verifikasi tanda tangan digital dengan algoritma RSA.

Langkah untuk verifikasi tanda tangan digital dengan algoritma RSA **(Gambar 9)** adalah :

1. pembentukan kunci publik dan privat
2. verifikasi tanda tangan digital
3. output

Kompleksitas waktu untuk langkah 1 adalah $O(k)$, dengan k adalah panjang representasi bilangan biner. Waktu eksekusi langkah 2 ditentukan oleh panjang pesan, jadi untuk pesan dengan ukuran n, maka diperlukan waktu eksekusi sebesar $v_1 n$. Sedangkan waktu eksekusi untuk langkah 3 adalah konstan, misalkan v_2. Total kompleksitas waktu proses verifikasi tanda tangan digital dengan algoritma RSA adalah $O(n + k)$.

B. Hasil Implementasi

Implementasi proses pembentukan dan verifikasi tanda tangan digital dilakukan dengan dua metode. Metode yang pertama adalah dengan kombinasi algoritma SHA 1 dan RSA, sedangkan metode yang kedua dengan menggunakan algoritma RSA saja. Contoh proses keseluruhan pembentukan tanda tangan digital menggunakan kedua metode ini dapat dilihat pada **Lampiran 2** dan **Lampiran 3**. Spesifikasi komputer yang digunakan untuk mengimplementasikan kedua metode tersebut adalah sebagai berikut :

1. Perangkat Keras :
   - Prosesor Celeron 400 Mhz
   - RAM 196 MB

- *Harddisk* 10 GB

2. Perangkat Lunak :
   - Sistem Operasi Windows 98
   - Bahasa pemrograman Visual Basic 6.0

Dari hasil implementasi dilakukan pengukuran kecepatan proses pembentukan dan verifikasi tanda tangan digital. Pengukuran dilakukan sebanyak sepuluh kali dengan menggunakan ukuran input yang bervariasi. Hasil pengukuran disajikan pada **Tabel 1** dan **Tabel 2**.

Hasil pengukuran menunjukkan bahwa untuk proses pembentukan tanda tangan digital dengan kombinasi algoritma SHA 1 dan RSA memerlukan waktu yang lebih lama (**Gambar 10**). Hal ini disebabkan karena adanya proses pembentukan *message digest* terlebih dahulu sebelum dilakukan proses penandatanganan, sehingga secara rata-rata kecepatan proses pembentukan tanda tangan digital menggunakan kombinasi dua algoritma lebih lama dibandingkan dengan algoritma RSA (**Gambar 11**).

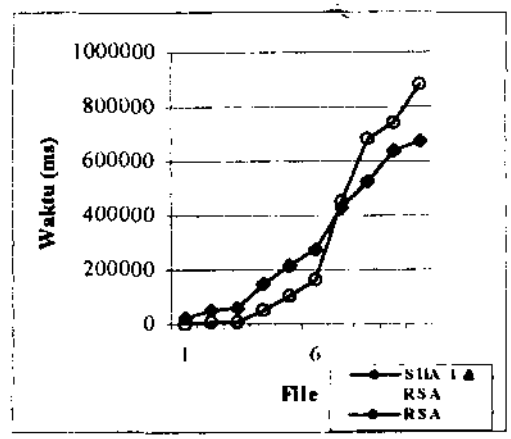
Tabel 1. Hasil pengukuran kecepatan proses pembentukan tanda tangan digital

File	Ukuran File (Bytes)	Waktu (ms)		Kecepatan rata-rata (Byte / ms)	
		SHA 1 & RSA	RSA	SHA 1 & RSA	RSA
1	2400	21424	1304	0,112	1,840
2	5630	54358	6150	0,104	0,915
3	7010	61472	8439	0,114	0,831
4	17200	148012	51793	0,116	0,332
5	24800	214297	105881	0,116	0,234
6	31600	273833	164950	0,115	0,192
7	49800	429016	452066	0,116	0,110
8	57600	525667	679953	0,110	0,085
9	65900	637693	742310	0,103	0,089
10	69800	670499	884054	0,104	0,079
Rata-rata (Byte/ms)				**0,111**	**0,471**

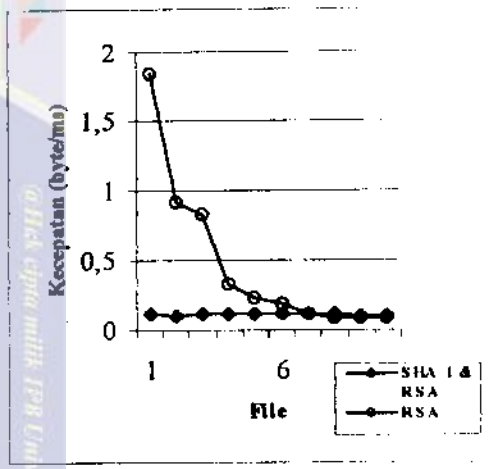
Tabel 2. Hasil pengukuran kecepatan proses verifikasi tanda tangan digital

File	Ukuran File (Bytes)	Waktu (ms)		Kecepatan rata-rata (Byte / ms)	
		SHA 1 & RSA	RSA	SHA 1 & RSA	RSA
1	2400	2	145	1200	16,552
2	5630	3	335	1876,667	16,806
3	7010	2	416	3505	16,851
4	17200	2	1030	8600	16,699
5	24800	3	1482	8266,667	16,734
6	31600	2	1893	15800	16,693
7	49800	3	3051	16600	16,323
8	57600	3	3997	19200	14,411
9	65900	3	5767	21966,667	11,427
10	69800	3	5988	23266,667	11,657
Rata-rata (Byte/ms)				**12028,17**	**15,415**

Untuk ukuran file mendekati 50 KB terlihat bahwa waktu yang diperlukan dengan menggunakan kombinasi dua algoritma lebih cepat dibandingkan dengan algoritma RSA. Hasil pengukuran juga menunjukkan bahwa kecepatan algoritma RSA dalam membentuk tanda tangan digital makin menurun seiring dengan bertambahnya ukuran input, sedangkan pada kombinasi dua algoritma kecepataannya relatif konstan walaupun ukuran inputnya bertambah besar.

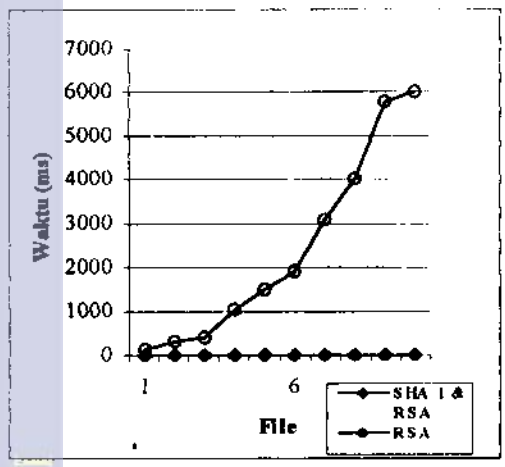


Gambar 10. Perbandingan waktu pembentukan tanda tangan digital antara dua metode.

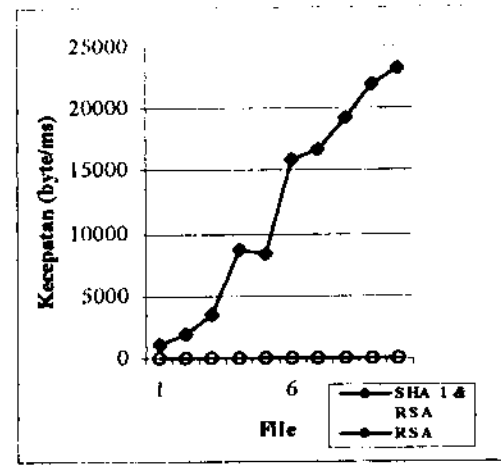


Gambar 11. Perbandingan kecepatan proses pembentukan tanda tangan digital antara dua metode.

Hasil yang didapatkan dari pengukuran proses verifikasi menunjukkan bahwa waktu yang diperlukan untuk proses verifikasi dengan kombinasi algoritma SHA 1 dan RSA lebih cepat dibandingkan dengan menggunakan algoritma RSA saja (**Gambar 12**). Hal ini disebabkan karena pada kombinasi dua algoritma proses verifikasi tidak dipengaruhi oleh ukuran pesan, sedangkan pada algoritma RSA dipengaruhi oleh ukuran pesannya, sehingga secara rata-rata kecepatan proses verifikasi tanda tangan digital dengan kombinasi dua algoritma lebih cepat (**Gambar 13**).



Gambar 12. Perbandingan waktu verifikasi tanda tangan digital antara dua metode.



Gambar 13. Perbandingan kecepatan proses verifikasi tanda tangan digital dengan kedua metode.

KESIMPULAN DAN SARAN

Kesimpulan

Analisis terhadap algoritma SHA 1 menunjukkan bahwa pada empat bagian, yaitu *message padding*, penambahan panjang bit, inisialisasi buffer, dan output memiliki waktu eksekusi konstan karena tidak dipengaruhi oleh ukuran input. Sedangkan pada bagian pemrosesan blok waktu eksekusinya dipengaruhi oleh ukuran inputnya. Setiap satu blok input 512 bit dilakukan operasi sebanyak 80 kali, sehingga total waktu eksekusi algoritma ini adalah $80n + x$, dengan x adalah suatu konstanta. Jadi kompleksitas waktu algoritma SHA 1 sama dengan $O(n)$.

Untuk pembentukan kunci publik dan kunci privat dengan algoritma RSA digunakan beberapa algoritma pendukung, yaitu algoritma Miller Rabin, algoritma Witness, algoritma Euclid, dan Extended Euclid. Semua algoritma pendukung tersebut memiliki kompleksitas waktu yang sama , yaitu $O(k)$, dengan k adalah panjang representasi bit bilangan bulat.

Pada penelitian ini proses pembentukan dan verifikasi tanda tangan digital dilakukan dengan dua metode. Metode yang pertama adalah dengan mengkombinasikan algoritma SHA 1 dan algoritma RSA, sedangkan metode kedua adalah dengan menggunakan algoritma RSA saja. Kompleksitas waktu yang didapatkan untuk proses pembentukan tanda tangan digital dengan kedua metode ini adalah sama yaitu $O(n + k)$. Untuk proses verifikasi, kompleksitas waktu yang

didapatkan dengan menggunakan kombinasi dua algoritma adalah $O(1 + k)$ dan untuk algoritma RSA adalah $O(n + k)$. Pada proses pembentukan tanda tangan digital menggunakan kombinasi dua algoritma yang ditandatangani adalah *message digest* yang dihasilkan oleh algoritma SHA 1, sedangkan pada algoritma RSA yang ditandatangani adalah pesan itu sendiri.

Untuk ukuran file mendekati 50 KB terlihat bahwa waktu yang diperlukan dengan menggunakan kombinasi dua algoritma lebih cepat dibandingkan dengan algoritma RSA, sehingga kombinasi dua algoritma lebih cocok digunakan untuk file-file yang berukuran besar. Hasil pengukuran juga menunjukkan bahwa kecepatan algoritma RSA dalam membentuk tanda tangan digital makin menurun seiring dengan bertambahnya ukuran input, sedangkan pada kombinasi dua algoritma kecepataannya relatif konstan walaupun ukuran inputnya bertambah besar.

Saran

Proses pembentukan tanda tangan digital dapat dikembangkan lebih lanjut dengan melakukan proses enkripsi pada pesan yang akan dikirimkan, sehingga dapat lebih meningkatkan keamanan dan kerahasiaan pesan tersebut. Bilangan prima yang digunakan pada implementasi algoritma RSA dalam penelitian ini relatif kecil, karena tidak lebih dari 200. Dalam kenyataannya algoritma RSA memerlukan bilangan prima yang sangat besar. Untuk itu perlu pengembangan lebih lanjut agar algoritma RSA dapat diimplementasikan dengan bilangan integer yang sangat besar.

DAFTAR PUSTAKA

Cormen, T.H., C.E. Leiserson & R.L. Rivest. 1990. *Introduction To Algorithms*. The MIT Press, Massachussetts-London.

Fadilah, M. 2000. Analisis Kompleksitas Waktu Komputasi Algoritma *Data Encryption Standard (DES)* dan *Rivest Shamir Adleman (RSA)*. Skripsi. Jurusan Matematika FMIPA IPB, Bogor.

National Institute of Standards and Technology (NIST). FIPS 180-1. 1995. *Specifications for SECURE HASH STANDARD.*

Grabbe, J. O. 1998. *Digital Signatures Illustrated*. http://www.aci.net/kalliste/digsig.htm. [5 Juli 2002].

Prasetya, M. 2001. *Message Digest 5 (MD 5) Dan Secure Hash Algorithm 1 (SHA 1)* Untuk Autentikasi Pesan. Skripsi. Jurusan Ilmu Komputer FMIPA IPB, Bogor.

Schneier, B. 1996. *Applied Crytography : Protocols, Algorithms, and Source Code in C*. second edition. John Wiley & Sons, Inc, New York.

Wibowo, A. M. 1998. *Tanda Tangan Digital dan Sertifikat Digital : Apa itu ?*. http://www.geocities.com/amwibowo/resource/sertifik/sertifik.html. [15 Mei 2002].

http://www.elektroindonesia.com/elektro/ut33a.html. [12 Agustus 2002].

LAMPIRAN

Lampiran 1. Pemanggilan rekursif pada algoritma Euclid

Lemma : Jika $a > b \geq 0$ dan subrutin Euclid (a,b) melakukan $k \geq 1$ pemanggilan rekursif, maka $a \geq F_{k+2}$ dan $b \geq F_{k+1}$.

Bukti : Pembuktian dilakukan dengan menggunakan induksi pada k. Jika $k = 1$, maka $b \geq 1 = F_2$, oleh karena $a > b$, maka $a \geq 2 = F_3$. Jadi lemma benar untuk $k = 1$.
Andaikan lemma benar untuk k-1 pemanggilan rekursif. Kemudian akan dibuktikan untuk k pemanggilan rekursif lemma juga benar. Jika $k > 0$, maka $b > 0$ dan subrutin Euclid (a, b) asumsikan melakukan pemanggilan subrutin Euclid $(b, a \bmod b)$ sebanyak k-1 kali. Menurut hipotesis induksi : $b \geq F_{k+1}$ dan (a mod b) $\geq F_k$. Akan dibuktikan bahwa $a \geq F_{k+2}$. Karena $a > b > 0$, maka $\lfloor a / b \rfloor \geq 1$ atau $1 - \lfloor a / b \rfloor \leq 0$, sehingga diperoleh :

$$\begin{aligned} b + (a \bmod b) &= b + (a - \lfloor a / b \rfloor b) \\ &= a + b (1 - \lfloor a / b \rfloor) \\ &\leq a \end{aligned}$$

Atau

$$\begin{aligned} a &\geq b + (a \bmod b) \\ &\geq F_{k+1} + F_k \\ &= F_{k+2} \end{aligned}$$ ■

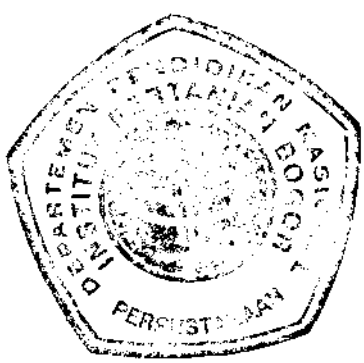
Teorema : Untuk setiap bilangan bulat $k \geq 1$, jika $a > b \geq 0$ dan $b < F_{k+1}$, maka subrutin Euclid (a,b) melakukan pemanggilan rekursif kurang dari k kali.

Bilangan *Fibonacci* yang berurutan merupakan input yang akan menghasilkan kasus terburuk bagi subrutin Euclid. Sebagai contoh : Euclid (F_3, F_2) melakukan tepat satu kali pemanggilan rekursif, dan untuk k ≥ 2 diperoleh $F_{k+1} \bmod F_k = F_{k-1}$, sehingga :

$$\begin{aligned} \text{GCD}(F_{k+1}, F_k) &= \text{GCD}(F_k, (F_{k+1} \bmod F_k)) \\ &= \text{GCD}(F_k, F_{k-1}) \end{aligned}$$

Jadi Euclid (F_{k+1}, F_k) melakukan pemanggilan rekursif sebanyak tepat $k - 1$ kali, yang memenuhi batas atas dari teorema.

Berdasarkan lemma : $b = F_{k+1} < F_{k+2}$. Diketahui bahwa nilai F_k dapat didekati oleh $\phi^k / \sqrt{5}$, dengan ϕ adalah *golden ratio* yang bernilai $(1+\sqrt{5})/2$. Karena pemanggilan rekursif yang dilakukan sebanyak k kali, maka $b < F_{k+1}$, sehingga $b \approx \phi^k / \sqrt{5}$ atau $k = O(\lg b)$. ■



Lampiran 2. Proses pembentukan dan verifikasi tanda tangan digital dengan kombinasi dua algoritma

I. Proses pembentukan *message digest*

1. Misalkan isi pesan adalah "abc", maka ubah pesan tersebut ke dalam kode ASCII-nya, yaitu : a = 97, b = 98, c = 99. Kemudian ubah representasi desimal dari kode ASCII tersebut ke dalam representasi binernya, yaitu : 01100001 | 01100010 | 01100011, panjang pesan asli ini (K) adalah 24 bit. Karena panjang bit pesan asli kurang dari 448 bit, maka dilakukan proses *padding* dengan menambahkan "1" pada posisi bit paling akhir, dan tambahkan "0" sehingga panjang total bit setelah proses *padding* ini adalah 448 bit.

2. Panjang pesan asli (K) = 24 bit di atas direpresentasikan ke dalam bilangan heksadesimal 16 bit yaitu 00000000 | 00000018. Nilai ini dijadikan 64 bit sehingga : 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00011000. Kemudian nilai 64 bit ini ditambahkan pada langkah (1), sehingga total panjang bit adalah 512. Setelah proses penambahan bit ini jumlah blok pesan (n) adalah 1.

3. Inisialisasi buffer awal H0, H1, H2, H3, H4
   H0 = 67452301
   H1 = EFCDAB89
   H2 = 98BADCFE
   H3 = 10325476
   H4 = C3D2E1F0.

4. Memproses 16 subblok 32 bit
   W[0] = 61626380
   W[1] = 00000000
   W[2] = 00000000
   W[3] = 00000000
   W[4] = 00000000
   W[5] = 00000000
   W[6] = 00000000
   W[7] = 00000000
   W[8] = 00000000
   W[9] = 00000000
   W[10] = 00000000
   W[11] = 00000000
   W[12] = 00000000
   W[13] = 00000000
   W[14] = 00000000
   W[15] = 00000018.

5. Hasil yang didapatkan setelah dilakukan proses *looping* sebanyak 80 kali adalah :

	A	B	C	D	E
t = 0:	0116FC33	67452301	7BF36AE2	98BADCFE	10325476
t = 1:	8990536D	0116FC33	59D148C0	7BF36AE2	98BADCFE
t = 2:	A1390F08	8990536D	C045BF0C	59D148C0	7BF36AE2
t = 3:	CDD8E11B	A1390F08	626414DB	C045BF0C	59D148C0
t = 4:	CFD499DE	CDD8E11B	284E43C2	626414DB	C045BF0C
t = 5:	3FC7CA40	CFD499DE	F3763846	284E43C2	626414DB
t = 6:	993E30C1	3FC7CA40	B3F52677	F3763846	284E43C2
t = 7:	9E8C07D4	993E30C1	0FF1F290	B3F52677	F3763846

t = 8:	4B6AE328	9E8C07D4	664F8C30	0FF1F290	B3F52677
t = 9:	8351F929	4B6AE328	27A301F5	664F8C30	0FF1F290
t = 10:	FBDA9E89	8351F929	12DAB8CA	27A301F5	664F8C30
t = 11:	63188FE4	FBDA9E89	60D47E4A	12DAB8CA	27A301F5
t = 12:	4607B664	63188FE4	7EF6A7A2	60D47E4A	12DAB8CA
t = 13:	9128F695	4607B664	18C623F9	7EF6A7A2	60D47E4A
t = 14:	196BEE77	9128F695	1181ED99	18C623F9	7EF6A7A2
t = 15:	20BDD62F	196BEE77	644A3DA5	1181ED99	18C623F9
t = 16:	4E925823	20BDD62F	C65AFB9D	644A3DA5	1181ED99
t = 17:	82AA6728	4E925823	C82F758B	C65AFB9D	644A3DA5
t = 18:	DC64901D	82AA6728	D3A49608	C82F758B	C65AFB9D
t = 19:	FD9E1D7D	DC64901D	20AA99CA	D3A49608	C82F758B
t = 20:	1A37B0CA	FD9E1D7D	77192407	20AA99CA	D3A49608
t = 21:	33A23BFC	1A37B0CA	7F67875F	77192407	20AA99CA
t = 22:	21283486	33A23BFC	868DEC32	7F67875F	77192407
t = 23:	D541F12D	21283486	0CE88EFF	868DEC32	7F67875F
t = 24:	C7567DC6	D541F12D	884A0D21	0CE88EFF	868DEC32
t = 25:	48413BA4	C7567DC6	75507C4B	884A0D21	0CE88EFF
t = 26:	BE35FBD5	48413BA4	B1D59F71	75507C4B	884A0D21
t = 27:	4AA84D97	BE35FBD5	12104EE9	B1D59F71	75507C4B
t = 28:	8370B52E	4AA84D97	6F8D7EF5	12104EE9	B1D59F71
t = 29:	C5FBAF5D	8370B52E	D2AA1365	6F8D7EF5	12104EE9
t = 30:	1267B407	C5FBAF5D	A0DC2D4B	D2AA1365	6F8D7EF5
t = 31:	3B845D33	1267B407	717EEBD7	A0DC2D4B	D2AA1365
t = 32:	046FAA0A	3B845D33	C499ED01	717EEBD7	A0DC2D4B
t = 33:	2C0EBC11	046FAA0A	CEE1174C	C499ED01	717EEBD7
t = 34:	21796AD4	2C0EBC11	811BEA82	CEE1174C	C499ED01
t = 35:	DCBBB0CB	21796AD4	4B03AF04	811BEA82	CEE1174C
t = 36:	0F511FD8	DCBBB0CB	085E5AB5	4B03AF04	811BEA82
t = 37:	DC63973F	0F511FD8	F72EEC32	085E5AB5	4B03AF04
t = 38:	4C986405	DC63973F	03D447F6	F72EEC32	085E5AB5
t = 39:	32DE1CBA	4C986405	F718E5CF	03D447F6	F72EEC32
t = 40:	FC87DEDF	32DE1CBA	53261901	F718E5CF	03D447F6
t = 41:	970A0D5C	FC87DEDF	8CB7872E	53261901	F718E5CF
t = 42:	7F193DC5	970A0D5C	FF21F7B7	8CB7872E	53261901
t = 43:	EE1B1AAF	7F193DC5	25C28357	FF21F7B7	8CB7872E
t = 44:	40F28E09	EE1B1AAF	5FC64F71	25C28357	FF21F7B7
t = 45:	1C51E1F2	40F28E09	FB86C6AB	5FC64F71	25C28357
t = 46:	A01B846C	1C51E1F2	503CA382	FB86C6AB	5FC64F71
t = 47:	BEAD02CA	A01B846C	8714787C	503CA382	FB86C6AB
t = 48:	BAF39337	BEAD02CA	2806E11B	8714787C	503CA382
t = 49:	120731C5	BAF39337	AFAB40B2	2806E11B	8714787C
t = 50:	641DB2CE	120731C5	EEBCE4CD	AFAB40B2	2806E11B
t = 51:	3847AD66	641DB2CE	4481CC71	EEBCE4CD	AFAB40B2
t = 52:	E490436D	3847AD66	99076CB3	4481CC71	EEBCE4CD
t = 53:	27E9F1D8	E490436D	8E11EB59	99076CB3	4481CC71
t = 54:	7B71F76D	27E9F1D8	792410DB	8E11EB59	99076CB3
t = 55:	5E6456AF	7B71F76D	09FA7C76	792410DB	8E11EB59
t = 56:	C846093F	5E6456AF	5EDC7DDB	09FA7C76	792410DB
t = 57:	D262FF50	C846093F	D79915AB	5EDC7DDB	09FA7C76
t = 58:	09D785FD	D262FF50	F211824F	D79915AB	5EDC7DDB
t = 59:	3F52DE5A	09D785FD	3498BFD4	F211824F	D79915AB
t = 60:	D756C147	3F52DE5A	4275E17F	3498BFD4	F211824F
t = 61:	548C9CB2	D756C147	8FD4B796	4275E17F	3498BFD4

t = 62:	B66C020B	548C9CB2	F5D5B051	8FD4B796	4275E17F
t = 63:	6B61C9E1	B66C020B	9523272C	F5D5B051	8FD4B796
t = 64:	19DFA7AC	6B61C9E1	ED9B0082	9523272C	F5D5B051
t = 65:	101655F9	19DFA7AC	5AD87278	ED9B0082	9523272C
t = 66:	0C3DF2B4	101655F9	0677E9EB	5AD87278	ED9B0082
t = 67:	78DD4D2B	0C3DF2B4	4405957E	0677E9EB	5AD87278
t = 68:	497093C0	78DD4D2B	030F7CAD	4405957E	0677E9EB
t = 69:	3F2588C2	497093C0	DE37534A	030F7CAD	4405957E
t = 70:	C199F8C7	3F2588C2	125C24F0	DE37534A	030F7CAD
t = 71:	39859DE7	C199F8C7	8FC96230	125C24F0	DE37534A
t = 72:	EDB42DE4	39859DE7	F0667E31	8FC96230	125C24F0
t = 73:	11793F6F	EDB42DE4	CE616779	F0667E31	8FC96230
t = 74:	5EE76897	11793F6F	3B6D0B79	CE616779	F0667E31
t = 75:	63F7DAB7	5EE76897	C45E4FDB	3B6D0B79	CE616779
t = 76:	A079B7D9	63F7DAB7	D7B9DA25	C45E4FDB	3B6D0B79
t = 77:	860D21CC	A079B7D9	D8FDF6AD	D7B9DA25	C45E4FDB
t = 78:	5738D5E1	860D21CC	681E6DF6	D8FDF6AD	D7B9DA25
t = 79:	42541B35	5738D5E1	21834873	681E6DF6	D8FDF6AD.

Setelah keseluruhan blok diproses hasil akhir adalah :
H0 = 67452301 + 42541B35 = A9993E36
H1 = EFCDAB89 + 5738D5E1 = 4706816A
H2 = 98BADCFE + 21834873 = BA3E2571
H3 = 10325476 + 681E6DF6 = 7850C26C
H4 = C3D2E1F0 + D8FDF6AD = 9CD0D89D.

Message digest = A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

II. Proses pembentukan kunci publik dan kunci privat

1. Tentukan dua buah bilangan $p = 149$ dan $q = 197$ secara acak. Untuk menguji keprimaan kedua bilangan tersebut digunakan algoritma Miller Rabin.
2. Hitung bilangan $n = p * q = 149 * 197 = 29353$
3. Tentukan $e = 3$ secara acak, sedemikian sehingga GCD(e, $(p - 1) * (q - 1)$) = 1. Algoritma Euclid digunakan untuk mencari nilai GCD.
4. Terakhir gunakan algoritma Extended Euclid untuk menghitung kunci privat, $d = 19339$, sehingga $d = e^{-1} \bmod ((p - 1) * (q - 1))$. Untuk mendapatkan nilai $d = 19339$, algoritma Extended Euclid dimodifikasi sehingga menjadi :

```
Function Extended_Euclid(ByVal e As Long, ByVal PHI As Long) As Double
u(0) = 1, u(1) = 0, u(2) = PHI
v(0) = 0, v(1) = 1, v(2) = e
While (v(2) <> 0)
        q = Int(u(2) / v(2))
        temp1 = u(0) - q * v(0)
        temp2 = u(1) - q * v(1)
        temp3 = u(2) - q * v(2)
        u(0) = v(0)
        u(1) = v(1)
        u(2) = v(2)
        v(0) = temp1
        v(1) = temp2
        v(2) = temp3
Wend
```

```
If (u(1) < 0) Then
        Extended_Euclid = (u(1) + PHI)
Else
        Extended Euclid  = u(1)
```

III. Proses pembentukan dan verifikasi tanda tangan digital

Message digest yang telah terbentuk pada langkah I akan ditandatangani menggunakan kunci privat dan kunci publik. Dari contoh di atas, *message digest* yang terbentuk adalah A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D. Kemudian masing- masing bilangan heksadesimal pada *message digest* tersebut diubah ke representasi desimalnya. Setiap representasi desimal ditandatangani satu persatu menggunakan persamaan : $s_i = m_i^d \bmod n$, dengan i = 1, 2, 3,..., 40. Untuk menghitung persamaan tersebut digunakan algoritma Modular Exponen. Hasil dari proses penandatanganan tersebut adalah :

1. Bilangan A = 10 = m_1, jadi $10^{19339} \bmod 29353 = 27405 = s_1$
2. Bilangan 9 = 9 = m_2, jadi $9^{19339} \bmod 29353 = 20978 = s_2$
3. Bilangan 9 = 9 = m_3, jadi $9^{19339} \bmod 29353 = 20978 = s_3$
4. Bilangan 9 = 9 = m_4, jadi $9^{19339} \bmod 29353 = 20978 = s_4$
5. Bilangan 3 = 3 = m_5, jadi $3^{19339} \bmod 29353 = 5436 = s_5$
6. Bilangan E = 14 = m_6, jadi $14^{19339} \bmod 29353 = 28551 = s_6$
7. Bilangan 3 = 3 = m_7, jadi $3^{19339} \bmod 29353 = 5436 = s_7$
8. Bilangan 6 = 6 = m_8, jadi $6^{19339} \bmod 29353 = 25900 = s_8$
9. Bilangan 4 = 4 = m_9, jadi $4^{19339} \bmod 29353 = 19353 = s_9$
10. Bilangan 7 = 7 = m_{10}, jadi $7^{19339} \bmod 29353 = 17992 = s_{10}$
11. Bilangan 0 = 0 = m_{11}, jadi $0^{19339} \bmod 29353 = 0 = s_{11}$
12. Bilangan 6 = 6 = m_{12}, jadi $6^{19339} \bmod 29353 = 25900 = s_{12}$
13. Bilangan 8 = 8 = m_{13}, jadi $8^{19339} \bmod 29353 = 2 = s_{13}$
14. Bilangan 1 = 1 = m_{14}, jadi $1^{19339} \bmod 29353 = 1 = s_{14}$
15. Bilangan 6 = 6 = m_{15}, jadi $6^{19339} \bmod 29353 = 25900 = s_{15}$
16. Bilangan A = 10 = m_{16}, jadi $10^{19339} \bmod 29353 = 27405 = s_{16}$
17. Bilangan B = 11 = m_{17}, jadi $11^{19339} \bmod 29353 = 9486 = s_{17}$
18. Bilangan A = 10 = m_{18}, jadi $10^{19339} \bmod 29353 = 27405 = s_{18}$
19. Bilangan 3 = 3 = m_{19}, jadi $3^{19339} \bmod 29353 = 5436 = s_{19}$
20. Bilangan E = 14 = m_{20}, jadi $14^{19339} \bmod 29353 = 28551 = s_{20}$
21. Bilangan 2 = 2 = m_{21}, jadi $2^{19339} \bmod 29353 = 11841 = s_{21}$
22. Bilangan 5 = 5 = m_{22}, jadi $5^{19339} \bmod 29353 = 24157 = s_{22}$
23. Bilangan 7 = 7 = m_{23}, jadi $7^{19339} \bmod 29353 = 17992 = s_{23}$
24. Bilangan 1 = 1 = m_{24}, jadi $1^{19339} \bmod 29353 = 1 = s_{24}$
25. Bilangan 7 = 7 = m_{25}, jadi $7^{19339} \bmod 29353 = 17992 = s_{25}$
26. Bilangan 8 = 8 = m_{26}, jadi $8^{19339} \bmod 29353 = 2 = s_{26}$
27. Bilangan 5 = 5 = m_{27}, jadi $5^{19339} \bmod 29353 = 24157 = s_{27}$
28. Bilangan 0 = 0 = m_{28}, jadi $0^{19339} \bmod 29353 = 0 = s_{28}$
29. Bilangan C = 12 = m_{29}, jadi $12^{19339} \bmod 29353 = 1756 = s_{29}$
30. Bilangan 2 = 2 = m_{30}, jadi $2^{19339} \bmod 29353 = 11841 = s_{30}$
31. Bilangan 6 = 6 = m_{31}, jadi $6^{19339} \bmod 29353 = 25900 = s_{31}$
32. Bilangan C = 12 = m_{32}, jadi $12^{19339} \bmod 29353 = 1756 = s_{32}$
33. Bilangan 9 = 9 = m_{33}, jadi $9^{19339} \bmod 29353 = 20978 = s_{33}$
34. Bilangan C = 12 = m_{34}, jadi $12^{19339} \bmod 29353 = 1756 = s_{34}$
35. Bilangan D = 13 = m_{35}, jadi $13^{19339} \bmod 29353 = 16014 = s_{35}$
36. Bilangan 0 = 0 = m_{36}, jadi $0^{19339} \bmod 29353 = 0 = s_{36}$
37. Bilangan D = 13 = m_{37}, jadi $13^{19339} \bmod 29353 = 16014 = s_{37}$
38. Bilangan 8 = 8 = m_{38}, jadi $8^{19339} \bmod 29353 = 2 = s_{38}$
39. Bilangan 9 = 9 = m_{39}, jadi $9^{19339} \bmod 29353 = 20978 = s_{39}$
40. Bilangan D = 13 = m_{40}, jadi $13^{19339} \bmod 29353 = 16014 = s_{40}$

Blok output di atas merupakan hasil tanda tangan digital dengan input pesan "abc" dan kunci publik $e = 3$, $n = 29353$, serta kunci privat $d = 19339$.

Proses verifikasi tanda tangan digital dihitung dengan menggunakan persamaan $m_i = s_i^{\ e} \bmod n$, dengan $i = 1, 2, 3, \ldots, 40$. Hasil dari verifikasi blok tanda tangan digital adalah blok *message digest*, sebagai berikut:

1. $27405 = s_1$, jadi $27405^{\ 3} \bmod 29353 = 10 = m_1$
2. $20978 = s_2$, jadi $20978^{\ 3} \bmod 29353 = 9 = m_2$
3. $20978 = s_3$, jadi $20978^{\ 3} \bmod 29353 = 9 = m_3$
4. $20978 = s_4$, jadi $20978^{\ 3} \bmod 29353 = 9 = m_4$
5. $5436 = s_5$, jadi $5436^{\ 3} \bmod 29353 = 3 = m_5$
6. $28511 = s_6$, jadi $28511^{\ 3} \bmod 29353 = 14 = m_6$
7. $5436 = s_7$, jadi $5436^{\ 3} \bmod 29353 = 3 = m_7$
8. $25900 = s_8$, jadi $25900^{\ 3} \bmod 29353 = 6 = m_8$
9. $19353 = s_9$, jadi $19353^{3} \bmod 29353 = 4 = m_9$
10. $17992 = s_{10}$, jadi $17992^{\ 3} \bmod 29353 = 7 = m_{10}$
11. $0 = s_{11}$, jadi $0^{\ 3} \bmod 29353 = 0 = m_{11}$
12. $25900 = s_{12}$, jadi $25900^{\ 3} \bmod 29353 = 6 = m_{12}$
13. $2 = s_{13}$, jadi $2^{\ 3} \bmod 29353 = 8 = m_{13}$
14. $1 = s_{14}$, jadi $1^{\ 3} \bmod 29353 = 1 = m_{14}$
15. $25900 = s_{15}$, jadi $25900^{\ 3} \bmod 29353 = 6 = m_{15}$
16. $27405 = s_{16}$, jadi $27405^{\ 3} \bmod 29353 = 10 = m_{16}$
17. $9486 = s_{17}$, jadi $9486^{\ 3} \bmod 29353 = 11 = m_{17}$
18. $27405 = s_{18}$, jadi $27405^{\ 3} \bmod 29353 = 10 = m_{18}$
19. $5436 = s_{19}$, jadi $5436^{\ 3} \bmod 29353 = 3 = m_{19}$
20. $28551 = s_{20}$, jadi $28551^{\ 3} \bmod 29353 = 14 = m_{20}$
21. $11841 = s_{21}$, jadi $11841^{\ 3} \bmod 29353 = 2 = m_{21}$
22. $24157 = s_{22}$, jadi $24157^{\ 3} \bmod 29353 = 5 = m_{22}$
23. $17992 = s_{23}$, jadi $17992^{\ 3} \bmod 29353 = 7 = m_{23}$
24. $1 = s_{24}$, jadi $1^{\ 3} \bmod 29353 = 1 = m_{24}$
25. $17992 = s_{25}$, jadi $17992^{\ 3} \bmod 29353 = 7 = m_{25}$
26. $2 = s_{26}$, jadi $2^{\ 3} \bmod 29353 = 8 = m_{26}$
27. $24157 = s_{27}$, jadi $24157^{\ 3} \bmod 29353 = 5 = m_{27}$
28. $0 = s_{28}$, jadi $0^{\ 3} \bmod 29353 = 0 = m_{28}$
29. $1756 = s_{29}$, jadi $1756^{\ 3} \bmod 29353 = 12 = m_{29}$
30. $11841 = s_{30}$, jadi $11841^{\ 3} \bmod 29353 = 2 = m_{30}$
31. $25900 = s_{31}$, jadi $25900^{\ 3} \bmod 29353 = 6 = m_{31}$
32. $1756 = s_{32}$, jadi $1756^{\ 3} \bmod 29353 = 12 = m_{32}$
33. $20978 = s_{33}$, jadi $20978^{\ 3} \bmod 29353 = 9 = m_{33}$
34. $1756 = s_{34}$, jadi $1756^{\ 3} \bmod 29353 = 12 = m_{34}$
35. $16014 = s_{35}$, jadi $16014^{\ 3} \bmod 29353 = 13 = m_{35}$
36. $0 = s_{36}$, jadi $0^{\ 3} \bmod 29353 = 0 = m_{36}$
37. $16014 = s_{37}$, jadi $16014^{\ 3} \bmod 29353 = 13 = m_{37}$
38. $2 = s_{38}$, jadi $2^{\ 3} \bmod 29353 = 8 = m_{38}$
39. $20978 = s_{39}$, jadi $20978^{\ 3} \bmod 29353 = 9 = m_{39}$
40. $16014 = s_{40}$, jadi $16014^{\ 3} \bmod 29353 = 13 = m_{40}$

Selanjutnya blok output $m_1, m_2, m_3, \ldots, m_{40}$ dari langkah di atas diubah ke bentuk reperesentasi heksadesimal, sehingga menghasilkan output yang sama dengan *message digest* yang dihasilkan dari pesan "abc".

Lampiran 3. Proses pembentukan dan verifikasi tanda tangan digital menggunakan algoritma RSA

Misalkan isi pesan adalah "abc", maka ubah masing-masing karakter dari pesan tersebut ke dalam kode ASCII-nya, yaitu : a = 97, b = 98, c = 99. Setelah ketiga kode ASCII tersebut diperoleh, proses pembentukan tanda tangan digital dilanjutkan dengan cara sebagai berikut :

Setiap kode ASCII ditandatangani satu persatu, dengan menggunakan persamaan : $s_i = m_i^d \bmod n$, dengan i = 1, 2, 3. Algoritma untuk menghitung persamaan di atas adalah algoritma Modular Exponen. Hasil dari tanda tangan digital blok pesan adalah :

1. Karakter 'a' = 97 = m_1, jadi $97^{19339} \bmod 29353 = 16852 = s_1$
2. Karakter 'b' = 98 = m_2, jadi $98^{19339} \bmod 29353 = 12092 = s_2$
3. Karakter 'c' = 99 = m_3, jadi $99^{19339} \bmod 29353 = 13321 = s_3$

Blok output di atas merupakan hasil tanda tangan digital dengan input pesan "abc" dan kunci publik $e = 3$, $n = 29353$, serta kunci privat $d = 19339$.

Proses verifikasi tanda tangan digital dihitung dengan menggunakan persamaan $m_i = s_i^e \bmod n$ dengan i = 1, 2, 3. Hasil dari verifikasi blok tanda tangan digital adalah blok pesan, sebagai berikut :

1. $16852 = s_1$, jadi $16852^3 \bmod 29353 = 97 = m_1$
2. $12092 = s_2$, jadi $12092^3 \bmod 29353 = 98 = m_2$
3. $13321 = s_3$, jadi $13321^3 \bmod 29353 = 99 = m_3$

Selanjutnya blok output m_1, m_2, m_3 dari langkah di atas diubah ke bentuk reperesentasi karakternya, sehingga menghasilkan output yang sama dengan pesan yang dimasukkan, yaitu 'abc'.