PERBANDINGAN KINERJA ALGORITMA RSA DAN ECC PADA TANDA TANGAN DIGITAL

Muara P. Sipahutar – NIM : 13503064

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if113064@students.if.itb.ac.id

ABSTRAKSI

Makalah ini membandingkan karakteristik kinerja dua jenis algoritma kunci publik yaitu RSA(Rivest, Shamir, Adleman) dan ECC (*Elliptic Curve Cryptography*) yang digunakan di dalam tanda tangan digital (*Digital Signature*) untuk menentukan bagaimana baiknya kedua algoritma tersebut dalam teknologi kakas yang modern sekarang ini dan protokol yang menggunakan tanda tangan tersebut.

Tanda tangan digital digunakan dalam penyampaian pesan untuk mengidentifikasi sang pengirim dan meyakinkan bahwa pesan tersebut tidak diubah/dimodifikasi setelah ditanda-tangani. Efisiensi ruang dan waktu dari algoritma tanda tangan digital sangat penting karena algoritma tersebut akan digunakan oleh banyak orang dalam bidang sistem pengiriman pesan.

***Kata kunci**:* kunci publik, RSA, ECC, algoritma, efisiensi.

1. Pendahuluan

Makalah ini membandingkan karakteristik kinerja dari RSA dan *Elliptic Curve Digital Signature Algorithm* dengan mengimplementasi kedua algoritma dan membandingkan waktu *running*-nya dengan maksud mengetahui efisiensi waktu keduanya.

Tanda tangan digital digunakan pada pengiriman pesan untuk memeriksa identitas dari pengirim pesan dan untuk meyakinkan bahwa pesan tersebut tidak dimodifikasi setelah ditandatangani. Hal ini sangat penting untuk memeriksa otentikasi dari sebuah pesan. Aplikasi tanda tangan digital telah dipakai dalam dunia digital, menggantikan tanda tangan biasa. Karena tanda tangan digital adalah jenis dari tanda tangan biasa, maka tanda tangan digital digunakan di banyak aplikasi tanda tangan di internet(contohnya *e-voting*, *online banking*, aplikasi pendaftaran universitas *online* dll).

Pentingnya tanda tangan digital dalam komunikasi digital memberikan kesempatan pada penelitian untuk mengembangkan kriptosistem yang baru seperti *Elliptic Curve Cryptography* (ECC), khususnya kebutuhan untuk algoritma yang lebih mangkus yang seiring dengan berkembangnya perangkat elektronik *mobile* yang terbatas memorinya. Meningkatnya ukuran kunci yang dibutuhkan RSA untuk keamanan melawan serangan *brute force* oleh komputer canggih atau komputer terdistribusi juga membuat ECC lebih diunggulkan dengan kelebihannya dalam ukuran kunci yang lebih kecil namun lebih aman.

Setelah implementasi dan menjalankan tanda tangan digital ECC dan RSA dengan ukuran kunci yang bervariasi pada beberapa uji kasus, saya menyimpulkan bahwa hasilnya lebih konsisten dengan pengetahuan yang telah didapatkan melalui akademik membandingkan dua sistem. Pembangkitan kunci RSA cukup memakan waktu, kedua skema kriptografi dapat dibandingkan (sampai 7680 bit tanda tangan RSA) untuk tanda tangan pesan, dan nilai RSA lebih baik dari tanda tangan ECC dalam verifikasi tanda tangan.

2. Dasar Teori

Kriptografi kunci publik digunakan dalam tanda tangan digital untuk memeriksa identitas

pengirim dari sebuah pesan dan isi dari pesan tersebut. Ini harus dilakukan dalam cara yang khusus sehingga kunci privat dari pengirim tetap rahasia dan musuh yang tidak diketahui tidak bisa untuk memodifikasi tanda tangan tersebut atau menirunya. Sebuah kriptosistem mengharuskan kunci privat pengguna sulit didapatkan dari kunci publik dan kunci publik dapat disebarkan ke mana saja. Analogi daripada kunci tersebut dijelaskan di bawah ini.

Alice ingin mengirim sebuah pesan ke Bob dan Bob ingin untuk meyakinkan bahwa Alice adalah pengirim pesan sebenarnya dan isi dari pesan tidak dimodifikasi pada waktu pengiriman.

1. Alice bisa membangkitkan sebuah kunci privat dan kunci publik kemudian mengirimkan kunci publiknya ke Bob.
2. Lalu Alice membuat sebuah *hash* dari pesan yang ia inginkan untuk dikirim ke Bob. Dia lalu mengenkripsi *hash* ini dengan menggunakan kunci privatnya. Dia menambahkan tanda tangan ni ke pesan yagn ia inginkan untuk dikirimkan ke Bob.
3. Bob sekarang bisa memeriksa bahwa Alice telah mengirim pesan dengan mengdekripsi tanda tangan menggunakan kunci publik Alice. Hasil dekripsinya adalah *hash* dari pesan awal yang Alice kirimkan. Bob bisa membuat *hash* kembali pesan dengan cara yang sama seperti yang Alice lakukan dan membandingkan kedua *hash*.

Dengan metode ini, Bob dapat membuktikan apakah Alice mengirim pesan atau tidak karena hanya kunci privat Alice yang dapat mengenkripsi tanda tangan. Dia juga bisa membuktikan bahwa pesan yang ia terima adalah pesan awal tanpa perubahan yang dikirimkan oleh Alice, selama *hashing* selalu unik, perubahan apa pun dapat mengubah *hash* pesan (disebut juga *message digest*).

Sistem kunci publik yang dipakai untuk membangkitkan tanda tangan digital dapat membuat perbedaan dalam kinerja proses tanda tangan digital. Dua kriptosistem kunci publik yang saya bandingkan dalam makalah ini adalah RSA dan ECDSA.

3. Rivest Shamir Adleman (RSA)

RSA adalah salah satu kriptografi kunci publik yang tertua dan banyak digunakan. Algoritma ini ditemukan di tahun 1977 oleh Ron Rivest, Adi Shamir dan Leonard Adleman.

Kriptosistem RSA didasarkan dari asumsi bahwa memfaktorkan adalah tugas perhitungan yang susah. Hal ini berarti bahwa dengan memberikan sumber dan waktu untuk melakukan perhitungan, seorang musuh seharusnya tidak bisa menjebol RSA (menemukan kunci privat) dengan memfaktorkan. Ini tidak berarti bahwa memfaktorkan adalah satu-satunya jalan untuk menjebol RSA (faktanya, menjebol RSA mungkin lebih mudah daripada memfaktorkan, lihat [2]), tetapi sekarang tidak ada metode yang dengan formal ditemukan untuk menjebol RSA secara efisien.

3.1 Pembangkitan Kunci RSA

Sebuah pasangan kunci publik dan kunci privat RSA dapat dibangkitkan dengan menggunakan algoritman di bawah [1]:

1. Pilih dua angka prima sembarang *p* dan *q*.
2. Hitung $n = p*q$.
3. Hitung $\phi(n) = (p-1)*(n-1)$.
4. Pilih kunci publik e, yang relatif prima terhadap $\phi(n)$.
5. Bangkitkan kunci privat dengan menggunakan persamaan $e*d = 1 \pmod{\phi(n)}$. Perhatikan bahwa $e*d = 1 \pmod{\phi(n)}$ ekivalen dengan $e*d = 1 + k\,\phi(n)$, sehingga secara sederhana d dapat dihitung dengan
   $d = (1 + k\,\phi(n))/e$

Hasil dari algoritma di atas adalah :

- Kunci publik adalah pasangan (e,n)
- Kunci privat adalah pasangan (d, n)

Catatan : *n* tidak bersifat rahasia, sebab ia diperlukan pada perhitungan enkripsi/dekripsi.

3.2 Pembangkitan Tanda Tangan RSA

Tanda tangan dari sebuah pesan *m* adalah modulus eksponensial langsung dari *hash* pesan dan kunci privat. Tanda tangan *s* bisa didapat dengan :

$s = hash(m)^d \pmod n$

Algoritma hash biasa yang digunakan adalah SHA-1.

3.3 Verifikasi Tanda Tangan RSA

Untuk memeriksa sebuah tanda tangan digital *s* dari pesan *m*, tanda tangan harus didekripsi dengan menggunakan kunci publik pemilik (*e*, *n*). Hash h didapatkan dengan :

$$h = s^e \pmod n$$

Jika *h* cocok dengan hash(*m*), maka tanda tangan adalah *valid*, dan pesan terbukti tidak diubah sejak ditanda-tangani.

4. Elliptic Curve Cryptography

Sebuah kurva elips dinotasikan dengan sebuah persamaan sebagai berikut :

$$y^2 = x^3 + ax + b$$
di mana $4a^3 + 27b^2 \neq 0$

Banyak masalah yang menarik muncul dari sebuah set dari titik di kurva elips pada daerah terbatas di bawah operasi grup. Daerah terbatas yang digunakan secara umum adalah yang di atas prima (F_p) dan daerah biner (F_2^n). Keamanan ECC didasari oleh masalah logaritma diskrit kurva elips(Elliptic Curve Discrete Logarithm Problem/ECDLP). Masalah ini didefinisikan :

Diberikan titik X, Y pada kurva elips,
Temukan *z* dengan ketentuan :
$X = zY$

Masalah logaritma diskrit pada grup ini di daerah terbatas adalah suatu fungsi satu arah yang baik karena sekarang tidak dikenalnya serangan waktu polinomial untuk menyelesaikan masalah. Metode untuk menghitung solusi ECDPL lebih tidak efisien dari memgfaktorkan, jadi ECC dapat menyediakan keamanan yang sama seperti RSA dengan panjang kunci yang lebih pendek.

ECC dikembangkan secara independen oleh Neal Koblitz dan Victor Miller di tahun 1985.

4.1 Pembangkitan Kunci ECC

Untuk membangkitkan sebuah pasangan kunci publik dan privat untuk digunakan di dalam ECC, sebuah entitas harus melakukan langkah berikut ini :

1. Temukan kurva elips E(K), di mana K adalah sebuah daerah terbatas seperti F_p atau F_2^n dan temukan titik Q pada E(K). *n* adalah *order of* Q.
2. Pilih sebuah angka semu acak x sehingga $1 = x = (n-1)$.
3. Hitung titik $P = xQ$.
4. Pasangan kunci ECC adalah (*P*, *x*), di mana P adalah kunci publik dan *x* adalah kunci privat.

4.2 Tanda Tangan Digital ECC (ECDSA)

Elliptic Curve Digital Signature Algorithm(ECDSA) didefinisikan di FIPS 186-2 [4] sebagai sebuah standari untuk tanda tangan digital pemerintah, dan dijelaskan di ANSI X9 62. ECDSA pertama kali disebarkan oleh Scott Vanstone di tahun 1992.

4.2.1 Pembangkitan Tanda Tangan ECDSA

Untuk membangkitkan sebuah tanda tangan S untuk pesan M, dengan menggunakan pasangan kunci ECC (P,x) dengan E(K), sebuah entitas harus melalui langkah berikut ini [3]:

1. Bangkitkan sebuah bilangan acak *k* sehungga $1 = k = (n-1)$.
2. Hitung titik $kQ = (x_1, y_1)$.
3. Hitung $r = x_1 \pmod n$. Jika r = 0, kembali ke langkah 1.
4. Hitung $k^{-1} \pmod n$.
5. Hitung SHA-1(*m*)M dan ubah hasilnya ke integer e.
6. Hitung $s = k^{-1}(e+xr) \pmod n$. Jika *s* = 0, kembali ke langkah 1.
7. Tanda tangan untuk pesan *m* adalah S = (*r*,*s*).

4.2.2 Verifikasi Tanda Tangan ECDSA

Untuk memeriksa sebuah tanda tangan S = (r,s) untuk pesan m pada kurva E(K) dengan menggunakan kunci publik P pemilik, sebuah entitas melakukan langkah-langkah :

1. Periksa r dan s adalah integer pada interval [1, n-1] .
2. Hitung SHA-1(m) dan ubah ini ke sebuah integer e.
3. Hitung $w = s^{-1} (\text{mod } n)$.
4. Hitung $u_1 = ew (\text{mod } n)$ dan $u_2 = rw (\text{mod } n)$.
5. Hitung $X = u_1 Q + u_2 P$
6. Jika $X = 0$, tolak S. Sebaliknya, hitung $v = x_1 (\text{mod } n)$.
7. Terima jika dan hanya jika $v = r$.

5. Perbandingan Run-time

Untuk menguji dan membandungkan karakteristik kinerja algoritma tanda tangan digital RSA dan ECDSA, saya menguji tiga komponen dari keduanya yaitu : pembangkitan kunci, pembangkitan tanda tangan dan verifikasi tanda tangan. Karena ECC menawarkan keamanan sama seperti RSA tetapi dengan ukuran kunci yang lebih pendek, kinerja diuji berdasarkan atas tabel di bawah.

Tabel 1: Perbandungan ukuran kunci (dalam *bit*)

Simetri	ECC	RSA
80	163	1024
112	233	2240
128	283	3072
192	409	7680
256	571	15360

Uji coba dilakukan pada sebuah komputer berprosesor Intel Pentium 4 2GHz dan memori RAM 512 MB. Pesan yang digunakan untuk penandatanganan adalah sebuah *file* teks berukuran 100 Kb.

5.1 Pembangkitan Kunci

Tabel 2: Kinerja Pembangkitan Kunci

Pembangkitan Kunci	Panjang Kunci		Waktu (detik)	
	ECC	RSA	ECC	RSA
	163	1024	0,08	0,16
	233	2240	0,18	7,47
	283	3072	0,27	9,80
	409	7680	0,64	133,90
	571	15360	1,44	679,06

Pembangkitan kunci untuk ECC tentu saja lebih baik dari RSA pada segala panjang kunci, dan terlihat khususnya ketika panjang kunci meningkat. Karena ECC tidak harus mengolah sumber data ke pembangkitan perhitungan bilangan prima secara intensif, ECC dapat membuat pasangan kunci publik dan kunci privat dengan kecepatan yang luar biasa dibandingkan dengan RSA.

Waktu pembangkitan kunci ECC berkembang secara linear sesuai dengan ukuran kunci, sedangkan RSA berkembang secara eksponensial.

5.2 Pembangkitan Tanda Tangan

Tabel 3: Kinerja Pembangkitan Tanda Tangan

Penandatanganan	Panjang Kunci		Waktu (detik)	
	ECC	RSA	ECC	RSA
	163	1024	0,15	0,01
	233	2240	0,34	0,15
	283	3072	0,59	0,21
	409	7680	1,18	1,53
	571	15360	3,07	9,20

Kinerja dari kedua algoritma tidak berbeda sampai ukuran kunci yang lebih besar, di mana ECC lebih baik daripada RSA. Satu pertimbangan yang sangat penting dari prioses pembangkitan tanda tangan adalah beberapa waktu digunakan oleh kedua algoritma adalah untuk menghitung *hash* SHA-1 dari pesan.

5.3 Verifikasi Tanda Tangan

Tabel 4: Kinerja Verifikasi Tanda Tangan

Verifikasi	Panjang Kunci		Waktu (detik)	
	ECC	RSA	ECC	RSA
	163	1024	0,23	0,01
	233	2240	0,51	0,01
	283	3072	0,86	0,01
	409	7680	1,80	0,01
	571	15360	4,53	0,01

RSA lebih unggul daripada ECC untuk proses verifikasi tanda tangan. Waktu untuk memeriksa pesan yang ditandatangani dengan RSA dapat diabaikan untuk segala panjang kunci, dan tidak menunjukkan perbedaan sampai panjang kunci 7680 ke 15360 *bits*. Kinerja ECC ketinggalan di setiap panjang kunci, menunjukkan pertumbuhan hampir linear untuk pertambahan ukuran kunci.

6. Implementasi Tanda Tangan Digital RSA

Implementasi tanda tangan RSA digunakan pada perbandingan waktu jalan (*run-time*) dengan menggunakan versi 5.1 dari Crypto++ TM Library[5]. Untuk kenyamanan dan *timing*, proses tanda tangan RSA dibagi menjadi tiga program dijelaskan di bawah.

6.1. rsaKeys.cpp(lihat Lampiran A1 untuk *source code*)

File rsaKeys bertanggung jawab untuk pembangkitan kunci publik dan kunci privat. *File* ini membangkitkan kunci publik dan kunci privat dengan diberikan ukuran kunci dalam *bits* dan lokasi *file* untuk menyimpan kunci ini.

6.2. rsaSign.cpp(lihat Lampiran A2 untuk *source code*)

File rsaSign bertanggung jawab untuk proses penandatanganan pesan dengan menggunakan kunci privat RSA yang disediakan oleh pengguna. *File* ini juga membutuhkan lokasi dari *file* pesan dan lokasi ke mana tanda tangan akan disimpan.

Saya memilih untuk mengimplementasi tanda tangan RSA dengan menggunakan PKCS #1 v2.1 *Signature Scheme* dengan versi *appendix* 1.5 (seperti dijelaskan di RSA *Cryptography Standard* [6]). Algoritma hash yang digunakan untuk meng-*hash* pesan pada tanda tangan adalah SHA1 (dijelaskan di FIPS 180-2[7]).

6.3. rsaVerify.cpp(lihat Lampiran A3 untuk *source code*)

File rsaVerify memeriksa sebuah tanda tangan yang dibuat dengan menggunakan sebuah kunci privat RSA dengan mendekripsikannya dengan kunci publik. *File* ini membutuhkan input dari pengguna berupa lokasi *file* kunci publik, lokasi *file* tanda tangan dan lokasi *file* pesan yang ditandatangani. Setelah mendekripsikan tanda tangan, *file* ini melakukan hash pesan dan membandingkannya dengan hash yang telah didekripsi untuk memvalidasi tanda tangan.

7. Implementasi Tanda Tangan Digital ECC

Implementasi tanda tangan digital ECC mengikuti garis langkah seperti yang dispesifikasikan untuk *Elliptic Curve Digital Signatures* (ECDSA) di ANSI X9.62.

Proses tanda tangan ECDSA dibagi menjadi tiga program : pembangkitan kunci, pembangkitan tanda tangan dan verifikasi tanda tangan. *Open source borZoi library* [8], versi 1.02 digunakan untuk mengimplementasi program ini.

7.1. ecdsaKeys.cpp(lihat Lampiran A4 untuk *source code*)

File ecdsaKeys bertanggung jawab untuk pembangkitan kunci ECSA untuk daerah biner yang telah disetujui NIST dengan khusus GF(2n) dari 163, 233, 283, 409 dan 571. *File* ini membangkitakan sebuah pasangan kunci publik dan kunci privat yang disimpan ke *disk* dalam sintaks ASN.1

DER (Distinguished Encoding Rules) sesuai dengan standar ANSI X9.62.

7.2. ecdsaSign.cpp (lihat Lampiran A5 untuk *source code*)

File ecdsaSign membaca sebuah pesan yang disimpan di *disk*, dan membangkitkan sebuah *file* tanda tangan sesuai dengan standar ANSI X9.62 berdasar pada *file* kunci privat pengguna dari ecdsaKeys. *File* tanda tangan disimpan pada disk dengan sintaks ASN.1 DER. Fungsi hash yang digunakan adalah SHA-1 (dijelaskan di FIPS 180-2 [7]).

7.3. ecdsaVerify.cpp (lihat Lampiran A6 untuk *source code*)

File ecdsaVerify melakukan proses verifikasi untuk menentukan apakah pesan dan tanda tangannya cocok dengan kunci publik dari pemilik pesan. *File* ini meyakinkan bahwa jika verifikasi sukses, maka pemilik pesan adalah yang menandatangani pesan, dan pesan tidak dimodifikasi selama pengiriman.

8. Kesimpulan

Hal yang saya dapatkan adalah bahwa pembangkitan kunci RSA jauh lebih lambat daripada pembangkitan kunci ECC untuk ukuran kunci RSA 1024 *bits* atau lebih. Mengingat ada piranti yang dapat menjebol kunci RSA yang lebih kecil dari 1024 *bits* dalam beberapa hari [10], biaya dari pembangkitan kunci dapat digolongkan sebagai faktor pemilihan sistem kunci publik untuk digunakan dalam tanda tangan digital, khususnya untuk piranti yang lebih kecil yang memiliki sumber perhitungan yang lebih sedikit daripada mesin uji coba saya.

Piranti yang tidak butuh pembangkitan kunci RSA setiap kali digunakan, tetapi menggunakan kunci RSA yang tetap, tidak akan mengalami kemunduran karena keterbatasan memori dan waktu dibandingkan dengan ECC, seperti hasil yang saya tunjukkan. RSA dapat dibandingkan dengan ECC untuk pembangkitan tanda tangan digital dalam hal waktu, dan lebih cepat daripada ECC untuk proses verifikasi tanda tangan. Jadi, untuk aplikasi yang membutuhkan verifikasi pesan lebih sering daripada pembangkitan tanda tangan, RSA adalah pilihan yang lebih baik.

ECC masih dalam masa pertumbuhan, oleh karena itu belum mengalami analisis ilmiah seperti skema RSA yang lebih tua. Semakin kecil ukuran kunci ECC semakin baik digunakan oleh piranti yang lebih rendah kekuatan perhitungannya seperti smart cards dan embedded system untuk transmisi data, verifikasi pesan dan guna yang lain.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. (2004). Bahan Kuliah IF5054 Kriptografi. Departemen Teknik Informatika, Institut Teknologi Bandung.

[2] Boneh D., Venkatesan. (2005). *Breaking RSA May be Easier Than Factoring*. http://theory.stanford.edu/~dabo/papers/no_rsa_red.pdf. Tanggal akses: 12 Desember 2006 pukul 13:00.

[3] *Elliptic Curve Digital Signature Algorithm.* http://www.comms.scitech.susx.ac.uk/fft/crypto/ecdsa.pdf. Tanggal akses: 12 Desember 2006 pukul 13:30.

[4] FIPS 186-2: *The Digital Signature Standard*. http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf. Tanggal akses: 12 Desember 2006 pukul 13.35.

[5] Crypto++ TM Library 5.1 – *a Free C++ Class Library of Cryptographic Schemes*. Ditulis oleh Wei Dai. http://www.eskimo.com/~weidai/cryptlib.html. Diakses tanggal 12 Desember 2006 pukul 14.00.

[6] PKCS#1 v2.1 : *RSA Cryptography Standard, RSA Laboratories*. 14 Juni 2002.

[7] FIPS 1980-2 : The Secure Hash Standard. http://csrc.nist.gov/publications/fips/fips180-2/fips180-2. Tanggal akses : 12 Desember 2006 pukul 13.40.

[8] borzoi 1.02 – *an open source Elliptic Curve Cryptography Library by Dragongate Technologies Ltd*. *http://www.dragongate-technologies.com/*. Tanggal akses : 12 Desember 2006 pukul 15.00.

[9] Abstract Syntax Notation One Standard. *http://asn1.elibel.tm.fr/en/*. Tanggal akses 12 Desember 2006 pukul 15.10.

[10] A. Shamir, and E. Tromer. Factoring Large Numbers with the TWIRL Device. *http://psifertex.com/download/twirl.pdf*. Tanggal akses : 13 Desember 2006 pukul 12.00.