Studi dan Perbandingan Algoritma RSA dan ECC dalam Enkripsi Data

Ananti Selaras Sunny (13507009)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if17009@students.if.itb.ac.id

Abstrak—Kriptografi asimestris sudah sering digunakan pada enkripsi data. Biasanya digunakan pada tanda tangan digital dan *password authetification*. Kriptografi asimetris yang sering digunakan adalah Algoritma RSA untuk tanda tangan digital. Kehandalan algoritma RSA sudah terbukti dengan digunakannya algoritma enkripsi ini untuk berbagai keperluan. RSA terkenal dengan kecepatan verifikasi data yang terenkripsi. Algoritma RSA mempunyai tingkat keamanan yang cukup tinggi, disebabkan faktorisasi bilangan yang digunakan cukup besar. Akan tetapi, beberapa tahun belakangan ini, banyak peneliti dan penulis yang berpendapat bahwa algoritma ECC merupakan salah satu algoritma yang akan berkembang kedepannya. Prinsip yang digunakan algoritma ECC mirip dengan algoritma RSA, karena keduanya menggunakan perhitungan matematika. Perbedaannya dengan algoritma RSA, algoritma ECC menggunakan kurva elips daripada menggunakan daerah perhitungan yang terbatas. Walaupun algoritma ECC tergolong masih lebih baru dibandingkan dengan RSA, tetapi sudah banyak yang menggunkan algoritma ECC ini. Karena algoritma RSA dan ECC ini memiliki prinsip yang mirip, studi dan perbandingan antara kedua algoritma ini menjadi menarik untuk dibahas. Pembahasan ini dilakukan untuk menemukan perbandingan performansi algoritma RSA dan ECC.

Kata Kunci—Algoritma RSA, Algoritma ECC, perbandingan.

I. PENDAHULUAN

Kriptografi dibedakan menjadi dua buah berdasarkan kunci yang ada, yaitu kriptografi simetris dan kriptografi asimetris. Kriptografi simetris menggunakan kunci untuk enkripsi dan dekripsinya sama. Kriptografi simetris ini bersifat konvensional karena menggunakan algoritma klasik. Kriptografi asimetris berbeda dengan kriptografi simetris. Pada kriptografi asimetris kunci yang digunakan untuk enkripsi berbeda dengan kunci yang digunakan, maka disebut asimetris. Untuk enkripsi data menggunakan kunci publik dan dekripsi menggunakan kunci privat. Kunci privat ini hanya diketahui oleh dua pihak yang saling berkomunikasi.

Kriptografi asimetris sudah banyak diterapkan pada aplikasi tertentu. Kriptografi asimetris yang sering digunakan adalah algoritma RSA. Algoritma RSA banyak yang digunakannya pada aplikasi. Algoritma RSA tergolong cukup aman karena bilangan faktor pembaginya cukup besar rangenya. Contoh aplikasi yang menggunakan algoritma RSA adalah Digital Signature. Algoritma RSA sudah lama ada hanya saja dan menjadi algoritma populer. Pengguna sampai saat ini masih mempercayai dan menggunakan algoritma RSA untuk keperluan pembuatan aplikasi online.

Meskipun algoritma RSA ini banyak yang sudah menggunakannya, bukan berarti algortima RSA tidak memiliki saingan. Para peneliti dan ahli kriptografi memprediksi algoritma ECC sebagai algoritma kriptografi masa depan. Kehadiran algoritma ECC dalam dunia kriptografi merupakan suatu yang baru.

Algoritma ECC dan RSA memiliki dasar perhitungan yang sama yaitu penggunaan rumus matematika. Perbedaannya algoritma ECC menggunakan curva elips dan tidak bergantung pada daerah perhitungan yang terbatas. Karena memiliki dasar algoritma yang mirip hanya berbeda cara perhitungannya, menjadikan algoritma ECC dan RSA menjadi menarik untuk dibahas. Terlebih lagi sekarang sudah banyak yang menggunakan algoritma ECC untuk enkripsi data.

Algoritma RSA dan ECC merupakan kriptografi asimetris. Jika dilihat dari kinerja algoritma RSA yang sudah terbukti efektif ini, kinerja yang dihasilkan memiliki hasil yang bagus juga. Untuk algoritma ECC yang merupakan algoritma baru tetapi tidak kalah dalam hal kinerja algoritmanya terbukti bahwa ahli kriptografi memprediksi bahwa algoritma ECC merupakan algoritma kriptografi di masa yang akan datang.

Dengan melakukan perbandingan kinerja dua algoritma ini akan menghasilkan analisis kinerja kedua algoritma ini. Daftar kinerja ini akan membantu menentukan kehandalan dan kelemahan algoritma RSA dan ECC. Daftar kinerja ini juga bisa digunakan untuk pembanding dan pemilihan penggunaan algoritma dalam implementasinya.

II. ALGORITMA RSA

Algoritma RSA merupakan kriptografi simetris. RSA termasuk algoritma yang memiliki tingkat keamanan yang

cukup tinggi, karena pemfaktoran dari integer yang besar dengan menghitung berasal dari faktor prima. RSA adalah algoritma pertam yang cocok untuk tanda tangan digital dan enkripsi.

Algortima RSA terdiri dari tiga langkah, yaitu *key generation*, eknripsi, dan dekripsi.

- Key Generation
  Kunci RSA dibuat dengan mengenerate dari dua bilangan prima yaitu p dan q. Kemudian dihitunglah n dengan mengalikan p dan q.
  Hitunglah:

  $$\phi(n) = (p - 1)(q - 1)$$

  p dan q merupakan bilangan prima yang dirandom pada range tertentu dan kedua bilangan tidak sama nilainya. Setelah mendapatkan nilai p dan q, $\phi(n)$ dapat dicari selanjutnya dengan rumus diatas.
- Cari integer e yang bernilai relatif prima terhadap $\phi(n)$. Bilangan e ini akan digunakan dalam enkripsi data menggunakan kunci publik. Selain itu bilangan e ini akan digunakan untuk mencari bilangan d.
- Cari integer d dengan menggunakan rumus di bawah ini.

  $$d = e^{-1} \bmod \varphi(n)$$

  Bilangan integer d dapat ditemukan dengan menghitung inverse eksponen dari $e \bmod \varphi(n)$. Bilangan d ini akan menjadi private key untuk mendekripsi cipherteks.
- Enkripsi
  Rumus untuk menghitung enkripsi seperti di bawah ini,

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

  Sesuai dengan rumus di atas, hasil dari enkripsi berupa c, kemudian m adalah padding dari plainteks, bilangan n sediri didapatkan dari perkalian p dan q. Bilangan e sebelumnya sudah didapatkan.
- Dekripsi
  Rumus untuk mengembalikan cipherteks menjadi bentuk plainteks semula, seperti di bawah ini,

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

  Sesuai dengan rumus di atas, c adalah cipherteks hasil enkripsi menggunakan RSA kunci publik. Bilangan d adalah yang didapatkan dari perhitungan sebelumnya. Bilangan n didapatkan dengan perkalian p dan q. Hasil dari deskripsi itu sendiri adalah m, plainteks semula.

Kekuatan algoritma RSA terletak pada tingkat kesulitan dalam memfaktorkan bilangan menjadi faktor-faktor prima, yang dalam hal ini $n = a \times b$.

Jika n berhasil difaktorkan menjadi p dan q, maka $\phi(n) = (p - 1)(q - 1)$ dapat dihitung. Selanjutnya karena kunci enkripsi e tidak dirahasiakan (kunci publik), maka kunci dekripsi d dapat dihitung dari persamaan,
$ed \equiv 1 \pmod{n}$.

Penemu algoritma ESA menyarankan agar nilai p dan q panjangnya lebih dari 100 digit. Dengan demikian hasil kali dari p dan q akan berukuran lebih dari 200 digit.

Usaha untuk mencari faktor bilangan 200 digit membutuhkan waktu komputasi selama 4 milyar tahun (dengan asumsi bahwa algoritma pemfaktoran yang digunakan adalah algoritma yang tercepat saat ini dan komputer yang dipakai dalam pencarian mempunyai kecepatan 1 milidetik).

Secara umum dapat disimpulkan bahwa algoritma RSA hanya aman jika n cukup besar. Jika panjang n hanya 256bit atau kurang, dapat difaktorkan dalam beberapa jam saja dengan sebuah komputer PC dan program yang tersedia di internet secara bebas. Jika panjang n adalah 512 bit atau kutang dapat difaktorkan dengan beberapa ratus komputer.

Jika dibandingkan dengan DES atau AES, RSA lebih lambat saat *sign* key-nya. Maka implementasi RSA yang sebenernya tidak digunakan untuk mengenkripsi sembarang plainteks. Plainteks yang dienkripsi oleh RSA sebelumnya sudah menjadi kunci simetri dengan kunci publik penerima pesan. Pesan plainteks sebelumnya dienkripsi menggunakan algoritma DES atau AES. Pesan dan kunci rahasia dikirimkan bersamaan. Penerima mendeskripsi kunci simetri dengan kunci privatenya, lalu mendekripsi dengan kunci simetri tersebut.

Serangan terhadap algoritma RSA adalah *Man-in-the-Middle Attack*, pengambilan informasi yang sedang dikirimkan, dan *Chosen-plaintext Attack*, mempelajari isi pesan yang akan dikirimkan.

III. ALGORITMA ECC

Algoritma ECC memiliki kepanjangan dari *Elliptic Curve Cryptpgraphy* ini ditemukan pada tahun 1985 oleh Victor Miller dan Neil Koblitz sebagai mekanisme alternatif untuk implementasi kriptografi asimetris. Algoritma ECC dibuat berdasarkan logaritma diskrit yang lebih menantang pengerjaannya pada kunci yang memiliki panjang yang sama.

Pada Februari 2005 National Security Agency (NSA) mempresentasikan strategi dan rekomendasi untuk mengamankan jalur komunikais pemerintahan Amerika dan jaringan komunikasi yang tidak diketahui.Protocol yang dipakai adalah Suite B, yang terdiri dari *Elliptic Curve Diffie-Hellman* (ECDH), *Elliptic Curve Menezes-Qu-Vanstone* (ECMQV) untuk pertukaran dan persetujuan kunci ; *Elliptic Curve Digital Signature Algorithm* (ECDSA) untuk *digital signatures*; *the Advanced Encryption Standard* (AES) untuk *symmetric encryption*; and the *Secure Hashing Algorithm* (SHA).

Secara umum, algoritma ECC memiliki kemanan yang kuat, efisien, skalabilitasnya di atas algoritma kriptografi

asimetris pada umumnya. Kelebihan dari ECC adalah sangat penting untuk NSA karena keamanan yang digunakan untuk mengamankan perangkat keras.

Sebuah kurva elips yang digunakan untuk perhitungan yang terdiri dari *points satisfying equations* bersama dengan sebuat point pembeda, yang menunjukkan bilangan tak hingga.

$$y^2 = x^3 + ax + b,$$

Koordinat disini dipilih dari daerah yang terbatas dan tetap yang tidak sama dengan 2 atau 3, karena dapat membuat perhitungan kurva menjadi lebih rumit. Keseluruhan keamanan algoritma ECC tergantung oleh kemampuan dalam menghitung perkalian titik dan ketidakmampuan untuk menghitung perkalian yang diberikan pada titik sebenarnya.

Untuk menggunakan ECC semua pihak harus setuju dengan elemen yang mendefinisikan kurva elips, yaitu *domain parameter*. Daerahnya didefinisikan dengan p pada kasus prima dan pasangan m dan f pada kasus biner. Kurva elips didefinisikan oleh konstan a dan b digunakan dalam pendefinisian persamaan. Subgrup siklisnya didefinisikan dengan generator (base point) G. Untuk aplikasi kriptografi dengan order G yang merupakan nilai yang tidak negatif terkecil dan $nG = \infty$, biasanya bilangan prima. Sejak n adalah ukuran subgroup $E(\mathbb{F}_p)$ yang diikuti dari teori lagrange bahwa nilai $h = \frac{|E|}{n}$ adalah sebuah integer.

Dalam aplikasi kriptografi nilai dari h, disebut kofaktor, yang nilainya harus $h \leq 4$ dan diharapkan h = 1. Jadi intinya pada kasus bilangan biner domain parameternya adalah (p,a,b,G,n,h) dan pada kasus bilangan biner adalah (m,f,a,b,g,n,h).

Algoritma tercepat yang diketahui dapat memecahkan ECDLP, membutuhkan $O(\sqrt{n})$ langkah. Contohnya untuk kemanan 128bit memerlukan satu kuva di atas $\mathbb{F}_q$ dimana $q \approx 2^{256}$. Ini bisa menjadi sangat kontras dengan daerah yang terdefinisi seperti pada algoritma DSA yang membutuhkan 3072 bit kunci publik dan 256 bit kunci privat, dan faktorisasi kriptografi seperti RSA yang membutuhkan 3072 bit kunci publik dan privat.

IV. PERBANDINGAN RSA&ECC

Untuk melakukan perbandingan algoritma RSA dan ECC maka dibuatlah suatu aplikasi untuk melihat kinerja kedua aplikasi ini.

Dengan menggunakan jsbn library, dibuatlah algoritma RSA dan ECC. Jsbn adalah library Big Integer untuk bahasa pemrograman javascript, yang bisa digunakan oleh aplikasi lain seperti desktop dan aplikasi *mobile* . Untuk pengujian kali ini menggunakan aplikasi berbasis web.

Sebenernya jika kita akan membandingkan kinerja dari algoritma kriptografi tidak etis membandingkan secara kode dan metode yang digunakan, biasanya kinerja algoritma ini dibandingkan dengan kecepatan enkripsi data dan dekripsi data.

Sesuai dengan dasar teori algoritma RSA, dibuatlah implementasinya dalam javascript. Method converternya tidak ditampilkan pada source di bawah ini.

```
// PKCS#1 (type 2, random) pad input string s to
n bytes, and return a bigint
function pkcs1pad2(s,n) {
  if(n < s.length + 11) { // TODO: fix for utf-8
    alert("Message too long for RSA");
    return null;
  }
  var ba = new Array();
  var i = s.length - 1;
  while(i >= 0 && n > 0) {
    var c = s.charCodeAt(i--);
    if(c < 128) { // encode using utf-8
      ba[--n] = c;
    }
    else if((c > 127) && (c < 2048)) {
      ba[--n] = (c & 63) | 128;
      ba[--n] = (c >> 6) | 192;
    }
    else {
      ba[--n] = (c & 63) | 128;
      ba[--n] = ((c >> 6) & 63) | 128;
      ba[--n] = (c >> 12) | 224;
    }
  }
  ba[--n] = 0;
  var rng = new SecureRandom();
  var x = new Array();
  while(n > 2) { // random non-zero pad
    x[0] = 0;
    while(x[0] == 0) rng.nextBytes(x);
    ba[--n] = x[0];
  }
  ba[--n] = 2;
  ba[--n] = 0;
  return new BigInteger(ba);
}

// "empty" RSA key constructor
function RSAKey() {
  this.n = null;
  this.e = 0;
  this.d = null;
  this.p = null;
  this.q = null;
  this.dmp1 = null;
  this.dmq1 = null;
  this.coeff = null;
}

// Set the public key fields N and e from hex
strings
function RSASetPublic(N,E) {
  if(N != null && E != null && N.length > 0 &&
E.length > 0) {
    this.n = parseBigInt(N,16);
    this.e = parseInt(E,16);
  }
  else
    alert("Invalid RSA public key");
}

// Perform raw public operation on "x": return
x^e (mod n)
function RSADoPublic(x) {
  return x.modPowInt(this.e, this.n);
}

// Return the PKCS#1 RSA encryption of "text" as
an even-length hex string
function RSAEncrypt(text) {
  var m =
pkcs1pad2(text,(this.n.bitLength()+7)>>3);
```

```
  if(m == null) return null;
  var c = this.doPublic(m);
  if(c == null) return null;
  var h = c.toString(16);
  if((h.length & 1) == 0) return h; else return
"0" + h;
}

// protected
RSAKey.prototype.doPublic = RSADoPublic;

// public
RSAKey.prototype.setPublic = RSASetPublic;
RSAKey.prototype.encrypt = RSAEncrypt;
```

Untuk implementasi algoritma ECC pada javascript, terlalu panjang kodenya jika ditampilkan semua pada makalah ini, jadi sebagian method tidak ditampilkan pada makalah. Hanya bagian yang dirasa penting saja, yang ditampilkan method ECCurveFp, karena method ECPointFP terlalu panjang sehingga tidak dicantumkan.

```
// ECCurveFp

// constructor
function ECCurveFp(q,a,b) {
    this.q = q;
    this.a = this.fromBigInteger(a);
    this.b = this.fromBigInteger(b);
    this.infinity = new ECPointFp(this, null,
null);
}
// for now, work with hex strings because
they're easier in JS
function curveFpDecodePointHex(s) {
    switch(parseInt(s.substr(0,2), 16)) { //
first byte
    case 0:
          return this.infinity;
    case 2:
    case 3:
          // point compression not supported yet
          return null;
    case 4:
    case 6:
    case 7:
          var len = (s.length - 2) / 2;
          var xHex = s.substr(2, len);
          var yHex = s.substr(len+2, len);

          return new ECPointFp(this,

this.fromBigInteger(new BigInteger(xHex, 16)),

this.fromBigInteger(new BigInteger(yHex, 16)));

    default: // unsupported
          return null;
    }
}

ECCurveFp.prototype.getQ = curveFpGetQ;
ECCurveFp.prototype.getA = curveFpGetA;
ECCurveFp.prototype.getB = curveFpGetB;
ECCurveFp.prototype.equals = curveFpEquals;
ECCurveFp.prototype.getInfinity =
curveFpGetInfinity;
ECCurveFp.prototype.fromBigInteger =
curveFpFromBigInteger;
ECCurveFp.prototype.decodePointHex =
curveFpDecodePointHex;
```

Library Jsbn adalah library BigNum pada javascript yang tercepat sampai saat ini. Karena jsbn merupakan pure javascript, kinerjanya tergantung dengan perangkat keras yang digunakan.

Hasil tabel perbandingan aplikasi web algortima RSA dan ECC ini saat diujikan pada tiga macam browser, yaitu Chrome (ver 11.0.696.60) dan Mozilla (ver 4.0) didapatkan waktu yang dibutuhkan untuk komputasinya. Diujikan dengan spesifikasi komputer Sistem Operasinya Windows 7 64-bit, processor Intel Core 2 Duo T5500, RAM 3Giga.

Tabel 1 Waktu Enkripsi yang diperlukan

	Chrome	**Mozilla**
RSA public, 512 bit, e=3	2ms	3ms
RSA public, 512 bit, e=F4	2ms	8ms
RSA public, 1024 bit, e=3	2ms	3ms
RSA public, 1024 bit, e=F4	7ms	19ms
EC multiply, 128 bit	31ms	17ms
EC multiply, 160 bit	33ms	32ms
EC multiply, 192 bit	49ms	49ms
EC multiply, 224 bit	63ms	77ms
EC multiply, 256 bit	92ms	89ms

Dari Tabel 1, dapat dilihat kecepatan enkripsi data yang dilakukan oleh algoritma RSA dan ECC dalam bahasa pemrograman javascript. Kecepatan enkripsi algoritma RSA lebih cepat dibandingkan dengan algoritma ECC. Hal ini tidak sesuai dengan teori dan implementasi algoritma RSA dan ECC pada aplikasi non-web. Hal ini dimungkin karena perhitungan mengguna javascript lebih berat dan library jsbn memang lebih lambat daripada library BigNum lainnya yang digunakan pada aplikasi non-web.

Tabel 2 *Underlying Mathematical Problem*

Kriptografi Asimetris	**Persoalan Matematika**	**contoh**
Faktorisasi integer	Diberikan sejumlah n, dicari faktor bilangan prima	RSA
Elliptic Curve Discrete logarithm	Diberikan sebuah kurva elips dan titik P,Q pada E, dicari nilai x dari Q=xP	EC Diffle-Helman,ECDSA

Keamanan sistem berkaitan langsung dengan kesukaran dari persoalan matematikanya. Untuk setiap kasus, ada yang metode yang dapat memecahkan persoalan matematika pada Tabel 2. Karena yang terkenal untuk memecahkan ECDLP adalah fully exponential, dapat mengunakan substansial ukuran kunci yang lebih kecil utntuk mendapatkan kekuatan yang sama. Oleh karena itu, algoritma ECC menyediakan keamanan per bit dari setiap skema kunci publik.

Tabel 3 ***Run Times for Different Public-Key Schemes***

Kriptografi asimetris	**Cara memecahkan persoalan math**	***Running times***
Faktorisasi integer	jumlah field sleve : exp(1.923(log n)1/3(log log n) 2/3)	Sub-exponential
Elliptic Curve Discrete logarithm	Pollard-rho algoritm: square root of n	Fully exponential

Parameter yang lebih kecil dapat digunakan pada algoritma ECC, bukan algoritma RSA. Keuntungan yang bisa didapatkan dari parameter yang lebih kecil adalah kecepatan dan lebih kecil kuncinya. Keuntungan ini sangat penting dalam lingkungan yang serba terbatas. Dalam hal ini keterbatasan *processing power*, *Storage space*, *bandwidth*, dan *power consumption*.

Hasilnya adalah algoritma ECC cocok untuk lingkungan yang memiliki banyak konstrain seperti smart card, cellular phone, PDAs, dll.

Terlebih lagi penelitian pada ECDLP sudah menuju banyak hasil penting termasuk seleksi kurva, dengan paramater yang dikenal lemah dan jarang digunakan. Peneliti sudah mengeluarkan hasil pengembangan implementasi algoritma ECC yang efisien dan semakin kuat keamanannya.

Untuk menghindari kompromi keamanan sistem, NIST's FIPS 140-2 standar mengindikasikan bahwa kunci untuk kriptografi simetris seperti AES harus dicocokkan dengan kekuatan dari kriptografi asimetris seperti RSA dan ECC. Contohnya 128bit AES kebutuhan kunci pada algoritma RSAnya adalah 3072bit untuk mendapatkan keamanan yang sama, tetapi algoritma ECC hanya membutuhkan 256 bit kunci.

Tabel 4 NIST ***Guidelines for Public-Key Sizes***

RSA Min size (bits)	**ECC Min size (bits)**	**Security (bits)**
1024	160	80
2048	224	112
3072	256	128
7680	384	192
15360	512	256

Seperti yang diperlihatkan pada Tabel 4, ketika kunci ECC ukuran skalanya linear, RSA tidak begitu. Hasilnya bahwa rentang antar kedua algoritma tersebut meningkat. Hal ini sangat relevan untuk implementasi kemanan AES 256bit, yang membutuhkan 15360bit kunci RSA dibandingkan dengan ECC yang hanya membutuhkan 512bit kunci.

Hal in kan berpengaruh secara signifikan pada komunikasi sistem yang kinerja komputasionalnya relatif dengan kinerja algoritma ECC dan RSA. Perbedaannya menjadi lebih dramatis karena ukuran kunci RSA semakin membengkak yang berpengaruh pada kenaikan komputasionalnya. Dari 1024bit RSA ke 3072-bit RSA membutuhkan 27 kali (3^3) sedangkan untuk algoritma ECC hanya akan meningkatkan komputasional 4 kali (1.6^3).

Bahkan dalam sistem yang menggunakan 1024bit RSA memiliki pengaruh desain yang penting. Sebagai contoh, ECC-160 memiliki 6kali lebih kecil dibandingkan dengan 1024bit RSA dan dapat mengenerate tanda tangan 12 kali lebih cepat daripada StrongARM7. Dari segi kinerja yang dilakukan algoritma ECC lebih tinggi bahkan dengan level keamanan yang lebih tinggi.

Algoritma ECC adalah algoritma yang compact dan sangat efisien. ECC membuat kebutuhan pemrosesan berkurang pada device yang memiliki resource yang terbatas dibandingkan dengan algoritma RSA. Terlebih lagi, ECC memiliki standar, dan memastikan interoperabilitas antar device, dan diteliti dengan baik dan terbukti, hal ini menjawab pabrik manufaktur dalam hal reliability.

Keuntungan lainnya dari algoritma ECC adalah scalabilitas yang linear, software footprint kecil, implementasi hardware yang dibutuhkan murah, bandwidth yang dibutuhkan rendah, dan kinerja algoritma ECC tinggi. Untuk alasan tersebut ECC mendapatkan banyak dukungan perusahaan-perusahaan besar dan baru saja menerima validasi yang kuat dari NSA.

Hal ini jelas bahwa keamanan adalah sebuah komponen esensial dari komunikasi sekarang ini dan akan terus begitu sampai pada jangka waktu yang lama. ECC memiliki algoritma yang superior dibandingkan algoritma lainnya pada kriptografi asimetris ketika berhubungan dengan keamanan. Karena ECC menawarkan keamanan yang tinggi daripada algoritma lain, seharusnya sudah tidak diragukan lagi jika algoritma ECC merupakan generasi selanjutnya dalam kriptografi asimetris.

V. SIMPULAN

- Dibandingkan dengan algoritma RSA, algoritma ECC membutuhkan ukuran kunci yang lebih sedikit untuk mendapatkan level keamanan yang sama.
- Algoritma ECC sesuai untuk lingkungan yang memiliki resouce terbatas seperti perangkat mobile.
- Waktu yang dibutuhkan untuk enkripsi data lebih cepat algoritma ECC dibandingkan RSA.
- Algoritma ECC merupakan algoritma yang compact dan sangat efisien.
- Implementasi algoritma ECC menggunakan bahasa javascript tidak *powerful* sesuai dengan penelitian ahli kriptografi.

REFERENSI

[1] http://en.wikipedia.org/wiki/RSA (tanggal akses 5 Mei 2011)

[2] http://en.wikipedia.org/wiki/Elliptic_curve_cryptography (tanggal akses 5 Mei 2011)

[3] http://www.certicom.com/index.php/ecc (tanggal akses 5 Mei 2011)

[4] http://www-cs-students.stanford.edu/~tjw/jsbn/ (tanggal akses 5 Mei 2011)

[5] http://www.eetimes.com/electronics-news/4144307/ECC-Holds-Key-to-Next-Gen-Cryptography (tanggal akses 5 Mei 2011)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 06 Mei 2011

Ananti Selaras Sunny (13507009)