

Keamanan Sistem lanjut

EC 7010

## **Analisis Kinerja *Cryptography***

*Secure Hash Standard*

(SHA1, SHA224, SHA256, SHA384, dan SHA512)

pada *Digital Signature Standard*

(DSA, RSA, dan ECDSA)

**Laporan Tugas Akhir**

**Disusun oleh:**

Halga Tamici

23206013



Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2007

## Abstrak

---

Teknologi telah berkembang semakin pesat, tentunya tingkat keamanan yang tinggi juga semakin diperlukan bersamaan dengan majunya teknologi. Salah satu dari keamanan yang sangat diperlukan adalah *cryptography*. Terdapat berbagai macam jenis standarisasi *cryptography* pada FIPS (*Federal Information Processing Standards*) sesuai dengan fungsinya. Salah satu standar *cryptography* pada FIPS yang dapat membangkitkan tanda tangan digital adalah *Digital Signature Standard* (DSS). Pada DSS ini terdapat beberapa standarisasi, diantaranya adalah DSA (*Digital Signature Algorithm*), RSA (Ron Rives, Adi Shamir, dan Len Adleman), dan ECDSA (*Elliptic Curve Digital Signature Algorithm*). Namun pada implementasinya, suatu sistem pembuatan signature selalu memerlukan *hash function*. *Hash function* digunakan sebagai data integritas pada penggabungannya dengan skema *digital signature*. Salah satu jenis *hash function* yang saat ini paling banyak diimplementasikan adalah SHA-1 yang merupakan bagian dari *Secure Hash Standard* (SHS). Sehingga pada ketiga jenis DSS dapat digabungkan dengan SHA-1.

Dengan adanya kemajuan teknologi, tentunya kebutuhan ukuran message juga semakin besar begitu juga dengan tingkat ketahanan dari *attacker*. Sehingga NSA (*National Security Agency*) mengembangkan standarisasi SHS sehingga terdapat SHA-224, SHA-256, SHA-384, dan SHA-512 sesuai dengan *input* dan *output message* yang dihasilkan.

Pada penelitian ini dilakukan analisis penggabungan ketiga jenis DSS dan kelima jenis SHS beserta pengaruhnya terhadap *key generation* dan *signature* yang dihasilkan. Dengan masukan bit yang semakin besar, apakah dapat digunakan dengan inputan *image* sehingga prosesnya lebih efisien. Pada analisis ini dapat dilihat dari parameter efisiensi waktu proses, distribusi frekuensi, dan variansi distribusi. Sedangkan proses ketahanan terhadap *attacker* dilihat dari penyerangan terhadap *signature (key)* yang digunakan, baik secara *brute force attack* ataupun analisis *cracking*.

Dari penelitian ini, diharapkan terdapat pengembangan *digital signature cryptography* dengan penggabungan *hash function* sehingga dapat digunakan untuk bit lebih besar, efisien dalam waktu proses, tahan terhadap *attacker*, dan dapat digunakan pada berbagai aplikasi.

# Daftar Isi

---

Abstrak	2
Daftar Isi	3
<b>BAB I Pendahuluan</b>	<b>4</b>
1.1 Latar Belakang	4
1.2 Tujuan	6
1.3 Perumusan Masalah	6
1.4 Batasan Masalah	6
<b>BAB II Simulasi Cryptography SHS pada DSS</b>	<b>7</b>
2.1 Pemodelan sistem <i>hash function</i>	7
2.2 Pemodelan sistem digital signature	8
2.2.1 <i>Digital Signature Algorithm</i>	8
2.2.2 <i>RSA signature</i>	10
2.2.3 <i>Elliptic Curve Digital Signature Algorithm (ECDSA)</i>	11
<b>BAB III Hasil Analisis Simulasi</b>	<b>13</b>
3.1 <i>Secure Hash Standard (SHS)</i>	14
3.2 <i>Digital Signature Standard (DSS)</i>	16
3.2.1 <i>Analisis Key Generation</i>	16
3.2.2 <i>Analisis Signature Generation dan Signature Verification</i>	16
3.2.3 <i>Analisis Peluang Cracking</i>	18
3.2.4 <i>Kumpulan hasil simulasi</i>	19
<b>BAB IV Penutup</b>	<b>23</b>
4.1 Kesimpulan	23
4.2 Saran	24
Daftar Pustaka	25

BAB I

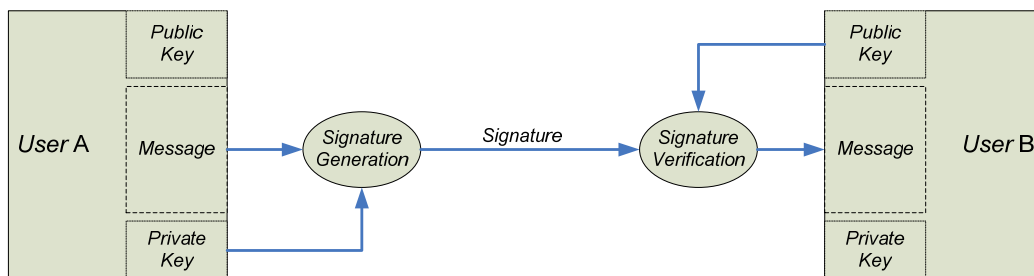
# Pendahuluan

## 1.1 Latar Belakang

Keamanan pada proses transmisi data telah berkembang cukup pesat. Salah satunya yaitu pada *cryptography*. *Cryptography* merupakan ilmu teknik matematika yang berhubungan dengan aspek keamanan informasi seperti kerahasiaan, integritas data, *authentication* dan keaslian data. Algoritma *cryptography* merupakan blok penting yang digunakan untuk memberikan keamanan pada jaringan komunikasi umum, seperti internet. Bersamaan dengan peningkatan pada konektivitas wireless dan data rate, keamanan protokol telah dikembangkan beberapa tahun lalu termasuk algoritma *cryptography* yang lebih *resource-friendly*.

Terdapat berbagai macam jenis standarisasi *cryptography* pada FIPS (*Federal Information Processing Standards*) sesuai dengan fungsinya. Salah satu standar *cryptography* pada FIPS yang dapat membangkitkan tanda tangan digital adalah *Digital Signature Standard* (DSS).

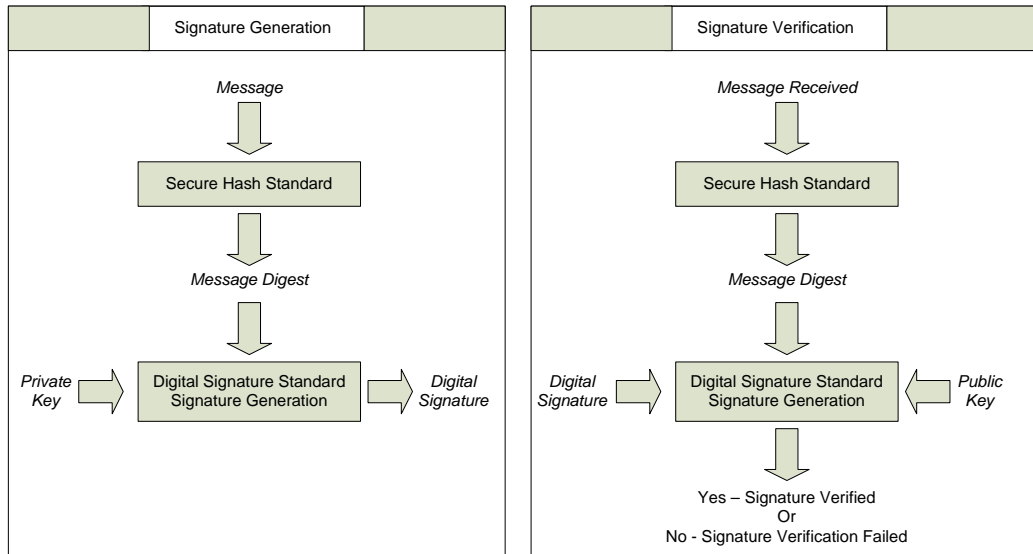
Pada kebanyakan *file-file* penting selalu diutamakan informasi identitas *user* pengirim untuk membuat suatu prifasi. Proses ini dilakukan untuk mengubah *signature* menjadi dokumen *digital*. Sehingga pada saat pengaksesan dokumen *digital* tersebut maka terdapat verifikasi pada *signature* yang diberikan.



**Gambar 1.1** Diagram blok *signature generation* dan *verification*

*Digital signature* memiliki *key* yang terdiri atas *public key* dan *private key*. *Cryptography public key* dapat mudah diserang dengan peniruan *public -key*, tetapi user bertanggung jawab untuk menjaga *private key* dengan aman. Dari *signature* yang diterima, harus memiliki *public key* yang dibangkitkan oleh *user* pengirim. *Public key* ini yang merupakan informasi *user* pengirim yang terdapat pada *user* penerima yang harus sesuai dengan *signature* yang dihasilkan oleh *user* pengirim.

Menurut FIPS 186 (*Federal Information Processing Standards*), pada bulan Mei 1994, standarisasi DSS yang pertama ditetapkan adalah DSA (*Digital Signature Standard*). Berikutnya pada bulan Mei 1997, NIST (*National Institute of Standards and Technology*) memberitahukan revisi bahwa memperbolehkan penggunaan RSA (Ron Rives, Adi Shamir, dan len Adleman) dan ECDSA (*Elliptic Curve Digital Signature Algorithm*) sebagai alternative dari DSA. Sehingga terdapat revisi pertama pada FIPS (FIPS 186-1) yang diutarakan bulan Desember 1998, dengan menambahkan RSA seperti ditetapkan pada ANSI X9.31. Dan pada bulan Januari 2000, revisi kedua (FIPS 186-2), dengan penambahan ECDSA (seperti ditetapkan pada ANSI X9.62) [3][5][6].



**Gambar 1.2** Diagram Blok Sistem Cryptography

Pada implementasinya, suatu sistem pembuatan signature selalu memerlukan *hash function*. *Hash function* biasanya diperlukan bila kita menginginkan pengambilan sidik jari suatu pesan. Sebagaimana sidik jari manusia yang menunjukkan identitas pemilik sidik jari. Fungsi ini diharapkan pula mempunyai kemampuan yang serupa dengan sidik jari manusia, di mana sidik jari pesan diharapkan menunjuk ke satu pesan dan tidak menunjuk kepada pesan lainnya. Fungsi ini juga dinamakan fungsi kompresi karena biasanya, masukan fungsi satu arah ini selalu lebih besar dari pada keluarannya, sehingga seolah-olah mengalami kompresi. Namun kompresi hasil fungsi ini tidak dapat dikembalikan ke asalnya sehingga disebut sebagai fungsi satu arah (*one way function*). Output *hash function* dinamakan *message digest*, karena seolah-olah merupakan inti sari pesan. Padahal tidak demikian. Sebab inti sari pesan mestinya merupakan ringkasan pesan yang masih dapat dipahami maknanya, sedangkan di *hash function* terjadi perlakuan sebaliknya, orang tidak tahu pesan aslinya. Standarisasi *hash function* yang ditetapkan di dalam FIPS adalah SHS (*Secure Hash Standard*) [2][4].

**Tabel 1.1** Spesifikasi SHA-1, SHA-256, SHA-384, dan SHA-512

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)	Security (bits)
SHA-1	$< 2^{64}$	512	32	160	80
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1024	64	384	192
SHA-512	$< 2^{128}$	1024	64	512	256

Versi pertama dari SHS yaitu SHA-1 (*Secure Hash Algorithm*) sesuai dari FIPS 180-1 pada bulan April 1997. SHA-1 merupakan jenis *hash function* yang saat ini paling banyak diimplementasikan. Dengan adanya kemajuan teknologi, tentunya kebutuhan ukuran message juga semakin besar begitu juga dengan tingkat ketahanan dari *attacker*. Sehingga NSA

(*National Security Agency*) mengembangkan standarisasi SHS sehingga pada bulan Agustus 2002, versi kedua dari SHS terdiri dari beberapa *hash function* yaitu SHA-224, SHA-256, SHA-384, dan SHA-512 [2].

**Tabel 1.2** Spesifikasi SHA-224

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)	Security (bits)
SHA-224	$< 2^{64}$	512	32	224	112

### 1.2 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Merancang penggabungan tiap-tiap SHS (SHA-1, SHA-224, SHA-256, SHA-384, dan SHA-512) dan DSS (DSA, RSA, dan ECDSA).
2. Merancang suatu sistem dengan masukan image yang akan dienkripsi dan pembalikan ke image setelah didekripsi.
3. Membandingkan kinerja tiap-tiap sistem penggabungan SHS dan DSS hasil perancangan.
4. Melakukan analisis performansi dengan masukan jumlah bit yang besar untuk tiap-tiap penggabungan SHS dan DSS.
5. Melakukan analisis statistik terhadap keluaran *cryptography*.

### 1.3 Perumusan Masalah

Beberapa permasalahan pada tugas akhir dapat didefinisikan sebagai berikut :

1. Proses pembuatan *message digest* dan parameter *message size*, *block size*, *word size*, *message digest size*, dan *security bits* pada *Secure Hash Standards*.
2. Perbandingan jenis-jenis SHS (SHA-1, SHA-224, SHA-256, SHA-384, dan SHA-512) dan memilih jenis SHS yang terbaik dilihat dari beberapa parameter dan kondisi.
3. Proses pembuatan *signature* (*public key* dan *private key*), *signature verification*, dan parameter yang diperlukan pada *Digital Signature Standards*.
4. Perbandingan jenis-jenis DSS (DSA, RSA, dan ECDSA) dan memilih jenis DSS yang terbaik dilihat dari beberapa parameter dan kondisi.
5. Teknik penggabungan ketiga jenis *Digital Signature* dengan kelima jenis *hash function* dan memilih penggabungan yang terbaik dilihat dari beberapa parameter dan kondisi.
6. Keuntungan dan kerugian pada proses penggabungan tersebut beserta parameter yang mempengaruhinya.
7. Teknik membuat simulasi untuk proses enkripsi dan dekripsi dengan inputan *image*.

### 1.4 Batasan Masalah

Batasan masalah yang digunakan dalam Tugas Akhir ini adalah:

1. Informasi yang akan dienkripsi dan dekripsi adalah *character* (huruf, angka, atau simbol) dan proses inputan *image*.
2. Proses *Digital Signature Standards* yang digunakan adalah DSA, RSA, dan ECDSA.
3. *Cryptography* RSA yang digunakan adalah *RSA signature*.
4. Proses *hash function* pada *Secure Hash Standard* yang digunakan adalah SHA-1, SHA-224, SHA-256, SHA-384, dan SHA-512.
5. Simulasi dilakukan dengan menggunakan software Matlab 7.0.1.

## BAB II

# Simulasi *Cryptography SHS* pada DSS

---

Perancangan model dan simulasi yang digunakan mengacu pada standar FIPS-180 (SHS), FIPS-186 (DSA, ECDSA), ANSI X9.31 (RSA), dan ANSI X9.62 (ECDSA). Tiap-tiap *digital signature* dan *hash function* saling berkaitan pada standar ini. SHA-1, SHA-224, SHA-256, SHA-384, dan SHA-512 dipilih sesuai standar algoritma yang digunakan pada FIPS-180-2. Algoritma ini digunakan untuk meng-*encrypt message* dan pembuat *pseudorandom* pada *key generation* DSS. Sedangkan DSA, RSA, dan ECDSA dipilih sesuai standar algoritma yang digunakan pada FIPS-186-2. Pada tugas ini akan merancang penggabungan SHS dan DSS sesuai besar *message* masukan, dan pemilihan optimalisasi yang terbaik untuk sistem *cryptography signature*.

Pada gambar diatas terlihat masukan berupa *character*, *image*, maupun *sound*. Masukan ini diproses dengan SHS sesuai dengan tipe *hash function* yang dipilih. Pada keluarannya diperkirakan data menjadi teracak dan jumlah bitnya mengecil sesuai tipe *hash function* yang digunakan. Keluaran SHS mulai dimasukkan ke DSS dan dilakukan pembuatan *signature* dengan menggunakan *public key* dan *private key*. *Signature* tersebut dikirim bersamaan dengan *message*. Pada penerima, untuk mengetahui *message* yang diterima benar-benar berasal dari pengirim, maka dilakukan verifikasi pada *signature* yang diterima. Apabila proses verifikasi menunjukkan bahwa *message* tersebut asli, maka penerima akan aman dengan *message* tersebut. Dan sebaliknya, apabila proses verifikasi menunjukkan bahwa *message* tidak asli, berarti *message* itu milik orang lain. Dengan adanya proses ini, memungkinkan keaslian data dapat dijaga dengan baik.

### 2.1 Pemodelan sistem *hash function*

Standar menetapkan *Secure Hash Algorithm* (SHA) yang diperlukan untuk menjamin keamanan *Digital Signature Standard* (DSS). Ketika pesan dengan sebarang panjang  $< 2^{64}$  bit dimasukkan, SHA-1 menghasilkan 160 bit keluaran yang disebut sebagai *message digest*. *Message digest* ini kemudian dimasukkan ke dalam DSA, yang menghitung tanda tangan digital untuk pesan tersebut. Penandatanganan *message digest* (dan bukannya penandatanganan pesan secara langsung) sering kali meningkatkan efisiensi proses, karena *message digest* biasanya jauh lebih kecil dibanding pesan aslinya. *message digest* pesan yang sama seharusnya dapat diperoleh oleh pemeriksa tanda tangan ketika menerima pesan dari pengirim dengan cara memasukkan pesan tersebut ke fungsi *hash* SHA. SHA dikatakan aman karena didesain supaya secara matematis tidak dimungkinkan untuk mendapatkan pesan aslinya bila diberikan hashnya atau tidak mungkin mendapatkan dua pesan yang berbeda yang menghasilkan *message digest* yang sama. SHA dibuat berdasarkan rancangan yang serupa dengan MD4 yang dibuat oleh Profesor Ronald L. Rivest dari MIT. SHA menghasilkan keluaran sidik jari 160 bit, lebih panjang dibanding MD5.

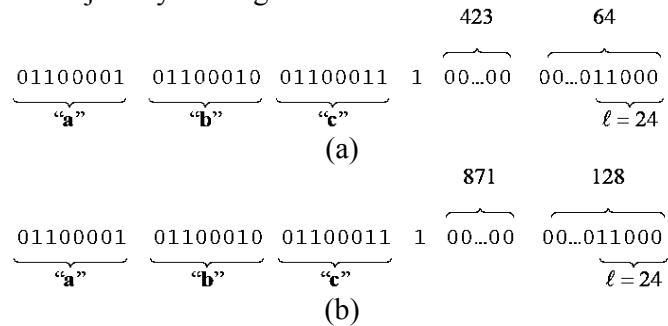
Pada simulasi ini dilakukan *message* yang sangat besar seperti *image* yang nantinya mempengaruhi efisiensi dari tiap-tiap blok yang digunakan. Sehingga dapat dianalisis performansi dari tiap-tiap jenis *hash function*.

### Blok-blok *hash function*

#### 1. Preprocessing

Pada blok ini dilakukan pembagian *message* yang sangat besar menjadi N-bits *message* blok sesuai dengan jenis *hash function*. Pada pembagian SHA-1, SHA-224 dan SHA-256 berdasarkan persamaan  $l+k \equiv 448 \pmod{512}$ .  $l$  merupakan panjang bit

masukan dan  $k$  merupakan jumlah bit 0 terkecil yang digunakan sehingga *padded message* berukuran 512 bit. Sedangkan pada pembagian SHA-384 dan SHA -512 berdasarkan persamaan  $l+1+k \equiv 896 \pmod{1024}$ , sehingga *padded message* berukuran 1024 bit. Untuk lebih jelasnya lihat gambar dibawah:



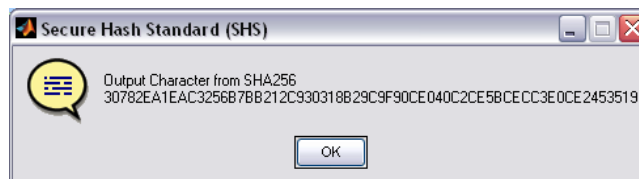
**Gambar 2.1** (a) *Padding message* pada SHA-1, SHA-224, dan SHA-256.  
 (b) *Padding message* pada SHA-384 dan SHA-512

Apabila masukan message sangat besar maka *message* tersebut dibagi menjadi N blok 512-bit ataupun N blok 1024-bit. Pada SHA-1, SHA-224, dan SHA-256 dilakukan pembagian tiap blok (512 bit) menjadi 16 buah 32-bit word. Tiap 32-bit ini akan diproses pada SHA secara terpisah dan didenotasikan menjadi  $M_j^{(i)}$ . Begitu juga pada SHA-384 dan SHA-512 dilakukan pembagian tiap blok (1024 bit) menjadi 16 buah 64-bit word.

Untuk menyamakan output *hash function*, dilakukan nilai inisial hash sesuai rekomendasi dari FIPS 180 [2]. Nilai-nilai tersebut terdapat pada lampiran A.

## 2. Main Process

Pada blok ini dilakukan proses komputasi *hash function* sesuai standarisasinya pada bagian 2.2. Pada blok ini dilakukan proses persiapan *message schedule*, inialisasi variabel yang bekerja, dan komputasi nilai *hash* di tengah proses. Sedangkan inisial awal hash dan konstanta hash yang digunakan terdapat pada lampiran.



**Gambar 2.2** Tampilan keluaran SHS

## 2.2 Pemodelan sistem digital signature

Simulasi dilakukan dai pembuatan *key* (*key generation*), pembuatan *signature* (*signature generation*) dan verifikasi *signature* (*signature verification*). Pemodelan sistem *digital signature* diilustrasikan dengan diagram blok sebagai berikut:

### 2.2.1 Digital Signature Algorithm

#### Key generation

Pada blok ini berfungsi untuk mencari *private key* dan *public key* dengan terlebih dahulu mencari prima  $p$  dan  $q$  untuk membangkitkannya. Nilai tersebut dapat diketahui dengan algoritma yang direkomendasikan dari NIST (Lihat 2.1.7). Pada simulasi ini menggunakan seed seperti dibawah ini (dalam heksa):

Seed = d5014e4b 60ef2ba8 b6211b40 62ba3224 e0427dd3

Setelah nilai p dan q didapat dilakukan pengecekan prima pada *probabilistic primality test*. Pada simulasi ini digunakan algoritma *Rabin-Miller test*. Pada umumnya tes prima dilakukan dengan algoritma Rabin-Miller dikarenakan algoritma ini dikenal sebagai *strong pseudoprime test* dan sangat cepat. Sehingga sangat sesuai dengan pengujian pembangkitan *pseudorandom DSA* (Lihat 2.1.5).

Setelah p dan q didapat, dilakukan komputasi untuk mendapatkan *public key* dan *private key*. Pada simulasi ini menggunakan rekomendasi dari NIST (Lihat 2.1.8). Dengan inputan bebas dari user dan settingan t awal seperti dibawah (dalam heksa):

t = 67452301 EFC DAB89 98BADCFE 10325476 C3D2E1F0

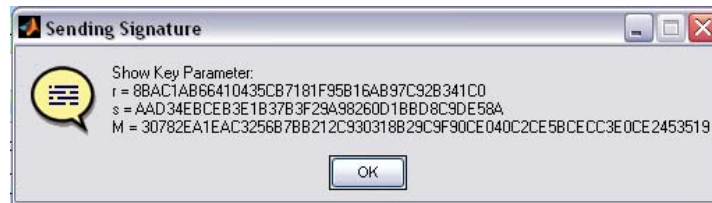
Sehingga didapat *private key* (x), *public key* (y), dan parameter *signature* (g). Pada proses ini dilakukan proses pseudorandom untuk mengacak nilai *private key* dan *public key* dengan menggunakan fungsi G dari SHA-1 dan DES. Pada simulasi ini digunakan fungsi G dari SHA-1, dikarenakan efisiensi waktu proses random. Selain itu, pencarian *multiplicative inverse* ( $n^{-1}$ ) menggunakan *greatest common divisor* (gcd). Pada simulasi ini, sistem gcd yang digunakan adalah *Euclidian algorithm* (Lihat 2.1.3). Algoritma ini berbentuk  $ax+by=d$ , dengan input a dan b mencari d, x, dan y.



Gambar 2.3 Tampilan *key parameter DSA*

### Signature generation dan Signature verification

Pada blok *signature generation* ini berfungsi untuk membangkitkan *signature* dengan kunci *private key* dan *modulus* q yang didapat dari *key generation*. Sedangkan pada blok *signature verification* ini berfungsi untuk memverifikasikan *signature* dengan kunci *public key* dan *modulus* n yang didapat dari *key generation* dan membandingkannya dengan *message* aslinya, apakah *message* yang diterima sama dengan hasil verifikasi atau tidak. Jika benar, *message* yang diterima benar-benar berasal dari pengirim. Jika tidak, terdapat *attacker* yang berusaha mencoba mengubah *signature* tersebut. Pada proses *signature generation* dan *signature verification* telah dibahas pada bagian 2.3.1. Pada blok ini terdapat proses pembuatan *signature* dan verifikasinya. Perbedaannya dengan algoritma yang lain, pada algoritma ini menggunakan teorema Elgamal yang menggunakan fungsi  $s = k^{-1}\{h(m)+xr\}$ .

Gambar 3.4 Tampilan *signature* DSA

## 2.2.2 RSA *signature*

### *Key Generation*

Pada blok ini berfungsi untuk mencari *private key* dan *public key* dengan terlebih dahulu mencari prima  $p$  dan  $q$  untuk membangkitkannya. Pembuatan *key* RSA (*private key* dan *public key*) dilakukan dengan pencarian prima  $p$  dan  $q$ . Nilai  $p$  dan  $q$  memiliki nilai yang sama besar dan pada simulasi ini digunakan *random search* menggunakan *Miller-Rabin test* (Lihat 2.1.6). Dan parameter lainnya dapat dicari dengan rumus yang terdapat pada bagian 2.3.2. Pada simulasi ini, pencarian nilai  $e$  dengan gcd menggunakan *Euclidian algorithm* (Lihat 2.1.1).

Perhatikan bahwa  $d$  dan  $n$  juga relatif prima. Bilangan  $d$  dan  $e$  merupakan kunci publik, sedangkan  $d$  kunci privat. Dua bilangan prima  $p$  dan  $q$  tidak diperlukan lagi. Namun  $p$  dan  $q$  kadang diperlukan untuk mempercepat perhitungan dekripsi.

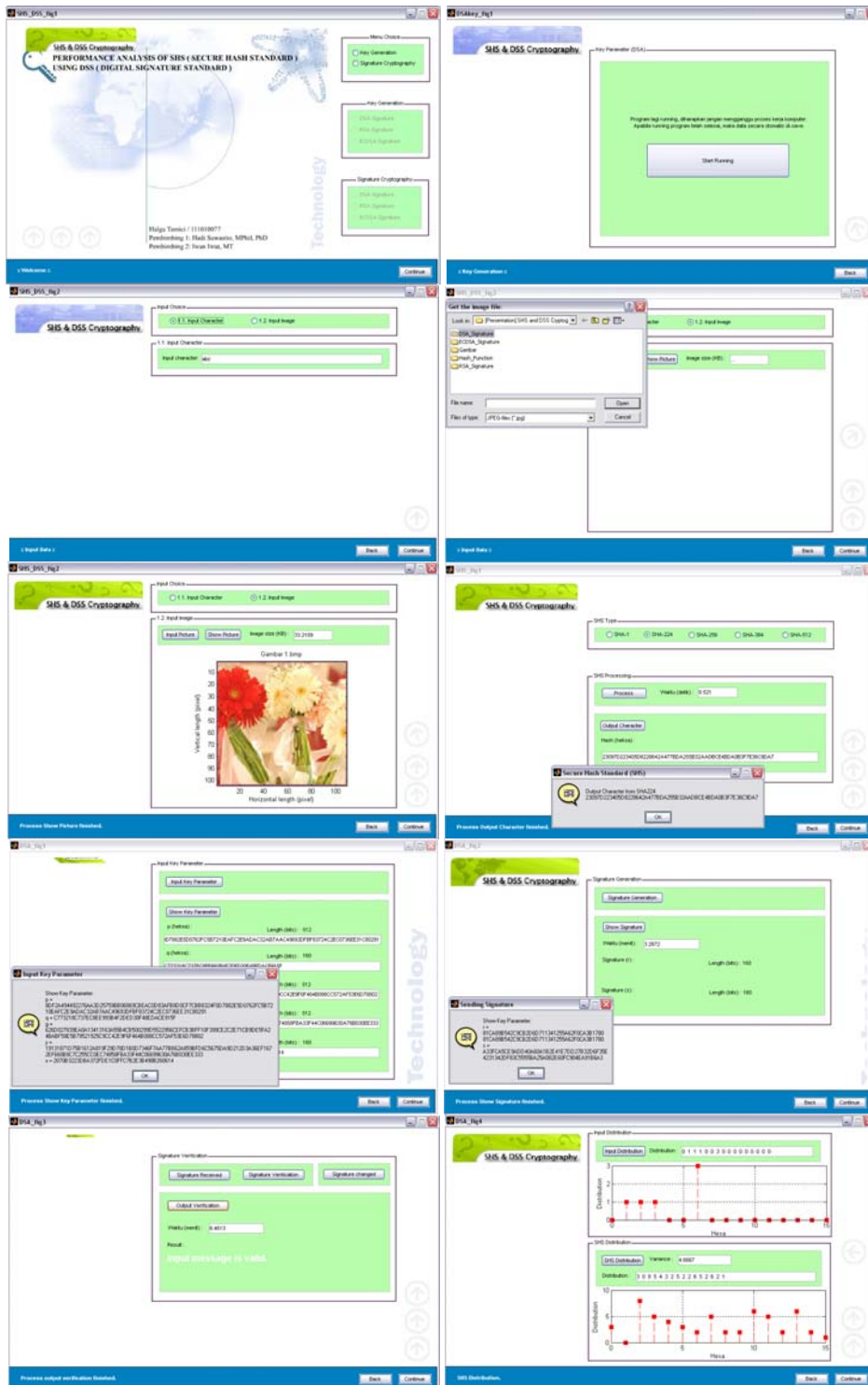
Untuk mengenkrip pesan  $m$ , pertama-tama bagi pesan kedalam blok-blok numerik yang lebih kecil dari  $n$  (dengan data biner, pilih pangkat terbesar dari 2 yang kurang dari  $n$ ). jadi bilangan  $p$  dan  $q$  bilangan prima 100 digit, maka  $n$  akan memiliki sekitar 200 buah digit setiap blok pesan  $m$ , seharusnya kurang dari 200 digit panjangnya. Jika kita perlu mengenkrip sejumlah blok pesan yang tetap, kita dapat menambahkan beberapa bit 0 disebelah kiri bilangan untuk menjamin bahwa pesan selalu kurang dari  $n$ . Pesan yang terenkrip  $c$ , akan tersusun dari blok-blok  $c_i$ , yang hampir sama panjangnya.

Gambar 2.5 Tampilan *key parameter* RSA

### *Signature generation dan Signature verification*

Pada blok *signature generation* ini berfungsi untuk membangkitkan *signature* dengan kunci *private key* dan *modulus*  $n$  yang didapat dari *key generation*. Sedangkan pada blok *signature verification* ini berfungsi untuk memverifikasikan *signature* dengan kunci *public key* dan *modulus*  $n$  yang didapat dari *key generation* dan membandingkannya dengan *message* aslinya, apakah *message* yang diterima sama dengan hasil verifikasi atau tidak. Jika benar, *message* yang diterima benar-benar berasal dari pengirim. Jika tidak, terdapat *attacker* yang berusaha mencoba mengubah *signature* tersebut. Pada proses *signature generation* dan *signature verification* telah dibahas pada bagian 2.3.2. Perbedaannya dengan algoritma yang lain, pada algoritma ini menggunakan pemangkatan dengan bilangan yang sangat besar, sehingga diharapkan sulit orang lain untuk mengetahui data asli.



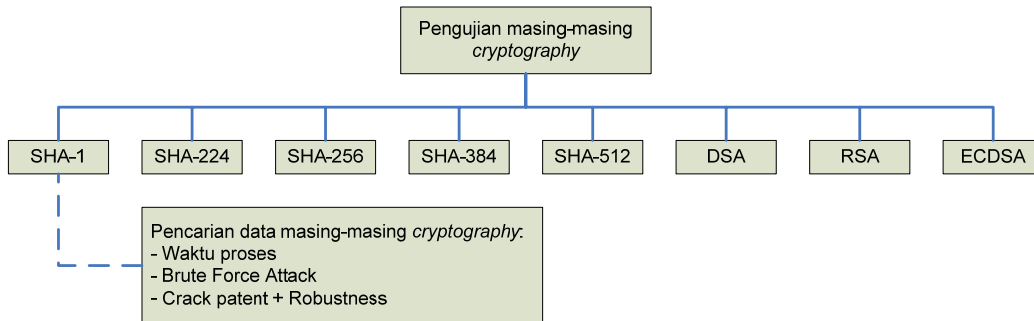


Gambar 2.7 Tampilan simulasi

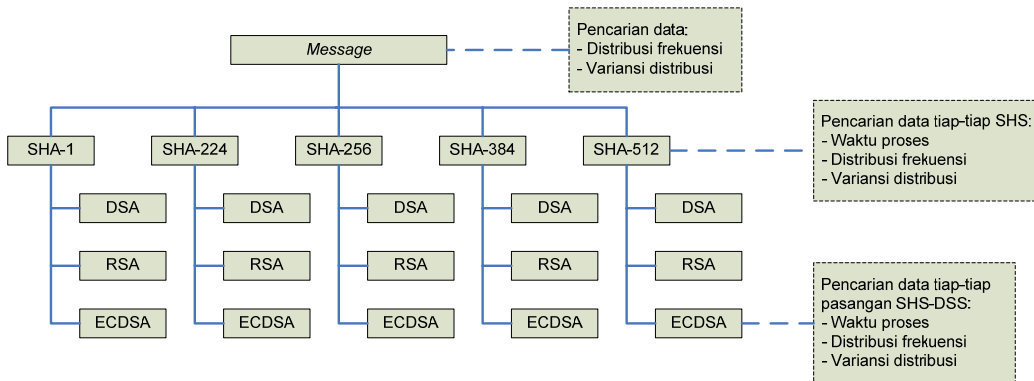
BAB III

# Hasil Analisis Simulasi

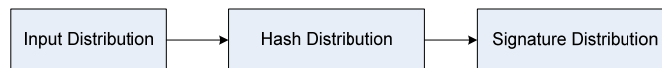
Sebelum dapat memecahkan kode rahasia, diperlukan identifikasi sistem enkripsi yang digunakan. Salah satu caranya adalah melihat frekuensi kemunculan hasil enkripsi pada transposisi yang sama dengan plaintext-nya. Pengujian ini dinamakan *identification system* dengan menggunakan analisis statistik (distribusi frekuensi dan variansinya). Pemodelan pengujian diilustrasikan dengan diagram blok sebagai berikut:



**Gambar 3.1** Blok diagram pengujian masing-masing SHS dan DSS



**Gambar 3.2** Blok diagram pengujian penggabungan SHS dan DSS

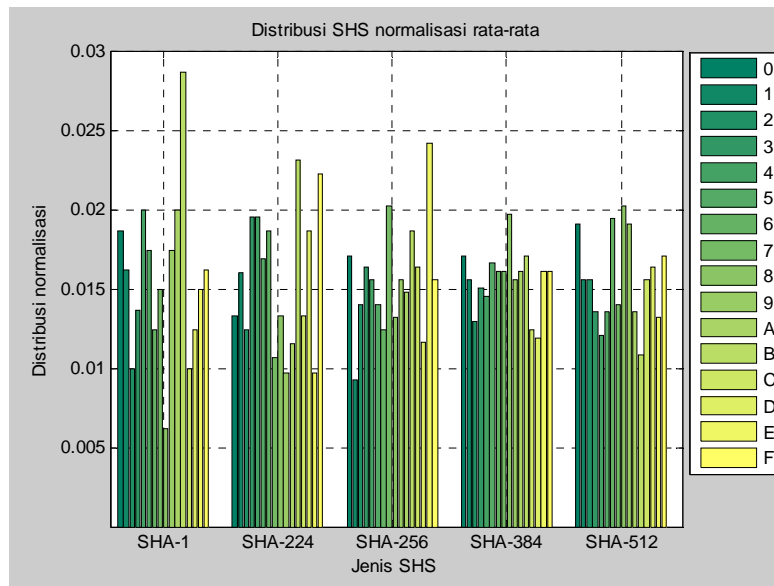


**Gambar 3.3** Blok diagram pengujian *identification system*

Seperti yang terlihat pada gambar, pengujian diawali dengan melihat kemampuan masing-masing *cryptography*. Pengujian ini melihat waktu proses, *Brute Force Attack*, dan *crack patent (robustness)*. Untuk pengujian gabungan SHS-DSS, sekali percobaan SHS akan dilakukan 3 kali pemasangan dengan DSS. Dan tiap pasangannya dilakukan pencarian waktu proses, distribusi frekuensi, dan variansi distribusi pada tiap-tiap *sample* dan membandingkannya.

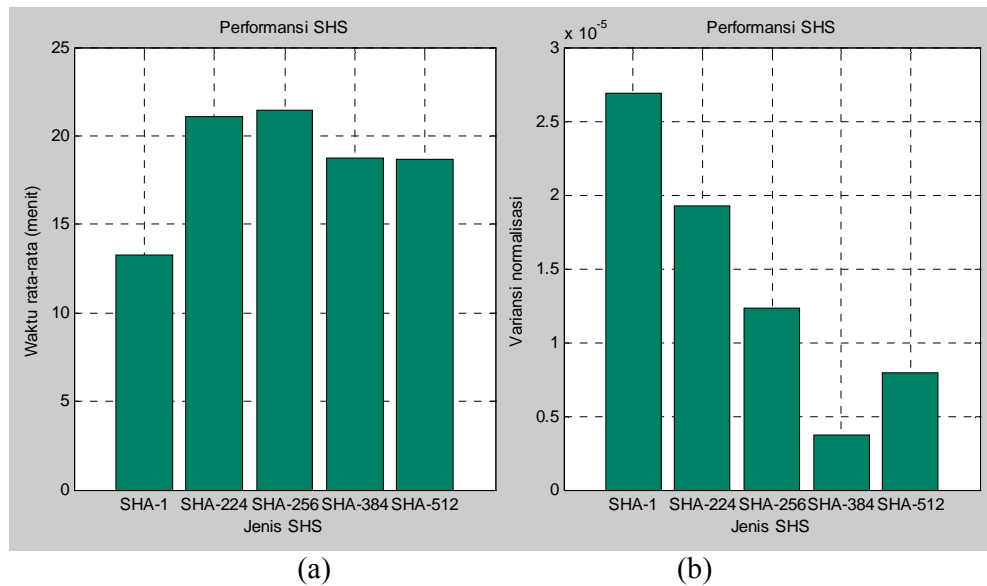
### 3.1 Secure Hash Standard (SHS)

Jika dianalisis dari distribusi frekuensi *heksa* yang dihasilkan, terdapat perbedaan dari inputan dan *message digest*, seperti yang terlihat pada gambar 4.4. Terdapat variasi yang semakin kecil pada *message digest* yang berarti bahwa terjadi pengacakan yang lebih baik pada *hash function*. Sehingga pada *digital signature*, menggunakan *hash function* sebagai *pseudorandom generator*.



**Gambar 3.4** Distribusi *message digest*

Waktu yang dibutuhkan untuk memproses *hash function* untuk tiap-tiap jenisnya tidak mengalami banyak perbedaan, seperti terlihat pada gambar 4.3(a). Proses *hash function* dilakukan untuk semua jenis SHS dengan inputan gambar BMP 1024 x 768 *pixel* 2,305 *Mbytes* (secara detail dapat dilihat pada lampiran D). Pada SHA-1 menunjukkan waktu proses yang paling cepat (56,12 % dibanding waktu terlama pada SHA-224). Tetapi terlihat dari SHA-224 memiliki waktu terlama, dan waktu proses semakin cepat sampai SHA-512 (83,07 % dibanding waktu terlama pada SHA-224). Pada SHA-224, SHA-256, SHA-384, dan SHA-512 memiliki perbedaan kecepatan proses dan waktu proses terbaik terdapat pada SHA-512, walaupun tidak mempengaruhi SHA-1. Hal ini terjadi disebabkan proses fungsi pada SHA-1 tidak memiliki banyak komputasi dibandingkan jenis-jenis SHS lainnya, sehingga waktu proses *hash function* terjadi paling cepat. Sedangkan proses fungsi pada SHA-1, SHA-224, dan SHA-256 memiliki komputasi yang menyerupai. Tetapi perbedaan waktu proses disebabkan oleh kapasitas masukan ke blok SHA berbeda. SHA-224 dan SHA-256 memiliki besar tiap blok masukan 512 bit, sedangkan SHA-384 dan SHA-512 memiliki besar tiap blok 1024 bit. Sehingga pada SHA-384 dan SHA-512 terjadi waktu proses paling cepat dibanding SHA-224 dan SHA-256. Antara SHA-224 dan SHA-256 tidak memiliki waktu proses yang jauh berbeda, begitu juga pada SHA-384 dan SHA-512.



**Gambar 4.5** (a) Waktu proses *hash function message* 2,3 MBytes  
 (b) Variansi proses *hash function*

Hasil dari *hash function* memiliki ukuran yang jauh lebih kecil (160-512bit). Apabila pada saat pengiriman bersamaan dengan *message*, tidak akan mempengaruhi ukuran total data. Sehingga pada *digital signature*, *hash function* digunakan sebagai kompresi data.

Apabila dianalisis dari kemungkinan *cracking* pada SHS, SHS memiliki *inisial hash value* yang dapat diubah tergantung user. Sehingga hasilnya sulit untuk diperkirakan oleh *attacker*. Jika input  $x$ ,  $x'$  dan output  $y$ ,  $y'$ , apabila *output hash* diketahui, *hash function* ini didesain sulit untuk mengkomputasi pencarian *input (preimage resistance)*. Misal untuk mencari nilai  $x'$  sehingga  $h(x')=y$ , ketika diketahui nilai  $y$  maka tidak akan menemukan nilai  $x'$ . Sehingga *hash function* ini hanya digunakan untuk satu arah saja (*one-way function*). SHS secara komputasional sulit untuk mencari input kedua jika ditemukan output yang sama (*2nd-preimage resistance*). Misal diketahui nilai  $x$ , untuk mencari input kedua  $x' \neq x$  sehingga  $h(x)=h(x')$ . Sehingga apabila dibandingkan kelima SHS dari analisis ini terdapat SHA-224 dan SHA-384 memiliki proteksi *cracking* paling tinggi. Karena pada komputasinya SHA-224 mengambil 7 nilai *hash* dari komputasi SHA-256 (dimana memiliki 8 nilai *hash*) dan pada komputasi SHA-384 mengambil 7 nilai *hash* dari komputasi SHA-512 (dimana memiliki 8 nilai *hash*) maka SHA-224 dan SHA-384 memiliki proteksi tertinggi. Proses ini akan membuat *attacker* semakin sulit untuk memperkirakan masukan. Sehingga terlihat pada parameter variansi distribusi SHA-384 memiliki nilai paling kecil. Sedangkan pada SHA-1 memiliki nilai variansi distribusi paling besar.

Hal ini dikarenakan komputasi pengacakan pada SHA-1 lebih jarang, karena komputasi dilakukan tiap masukan 32-bit dan jumlah *inisial hash* (5 buah) yang digunakan paling kecil dibanding *hash* yang lain (*hash* lainnya menggunakan 64-bit masukan dan jumlah *inisial hash* beragam 6, 7 dan 8). Pada SHA-224 memiliki variansi lebih kecil dari SHA-1, karena pada komputasinya SHA-224 mengambil 7 nilai *hash* dari komputasi SHA-256 (dimana memiliki 8 nilai *hash*). Hal ini membuat SHA-224 memiliki variansi lebih besar dibanding SHA-256. Namun dengan adanya proses ini membuat *attacker* menjadi lebih sulit memprediksi masukan, walaupun variansinya membesar tetapi masih termasuk dalam toleransi (karena secara umum variansi SHA-224 masih tergolong kecil). Begitu juga pada SHA-384, pada komputasi SHA-384 mengambil 7 nilai *hash* dari komputasi SHA-512

(dimana memiliki 8 nilai *hash*). Sedangkan SHA-224 dan SHA-256 memiliki variasi lebih besar dari SHA-384 dan SHA-512, karena looping masukan untuk satu blok *message* SHA-224 dan SHA-256 terjadi 64 kali. Pada SHA-384 dan SHA-512 terjadi *looping* 80 kali, sehingga variansinya lebih kecil dibanding SHA-224 dan SHA-256.

### **3.2 Digital Signature Standard (DSS)**

#### **3.2.1 Analisis Key Generation**

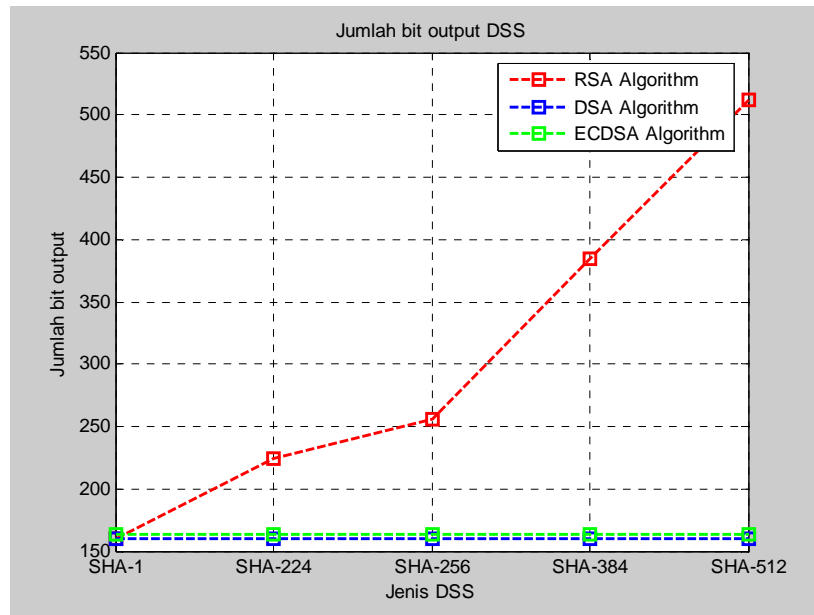
Waktu pengerjaan *key generation* untuk 512-bit prima pada DSA sama dengan RSA sekitar 10 jam di Matlab. Apabila RSA menggunakan 250-bit prima memiliki waktu pembuatan *key* sekitar 4 jam di Matlab. Proses ini terjadi dikarenakan proses pemangkatan bilangan yang begitu besar. Sekali pemangkatan kira-kira membutuhkan waktu 3 menit. Namun dalam sekali pencarian bilangan prima dilakukan beratus kali percobaan. Pada ECDSA memiliki waktu yang cepat karena menggunakan parameter yang sudah diuji cobakan di NIST.

#### **3.2.2 Analisis Signature Generation dan Signature Verification**

Pada distribusi *signature* sebagian besar menunjukkan variasi yang lebih besar dari pada distribusi *hash function*. Sehingga hal ini menunjukkan peningkatan kinerja pengacakan pada sistem. Apabila dibandingkan RSA dengan DSA, DSA memiliki variasi hampir sama dengan RSA. Namun kelemahannya, waktu yang diperlukan dalam proses pembuatan *signature* berlangsung sangat lama dibanding RSA. Hal ini dikarenakan proses pemangkatan bilangan besar yang terjadi pada DSA berlangsung berkali-kali. Sedangkan pada RSA hanya berlangsung sekali.

Pada penggabungan SHS-DSA, terdapat SHA-384 memiliki kinerja yang lebih baik. Karena pada penggabungannya memiliki waktu proses *signature generation* paling kecil (1.331 menit) dan pada *signature verification* juga tergolong kecil (2.6475 menit), walaupun memiliki variasi *signature* yang paling besar dari jenis SHS lainnya, tetapi variansinya masih termasuk kecil, dan pada *brute force attack* memiliki waktu yang seimbang dengan jenis SHS lainnya, didukung lagi dengan adanya *hash* sebelumnya juga memiliki kinerja yang paling baik juga (seperti yang terlihat pada gambar 4.8-11).

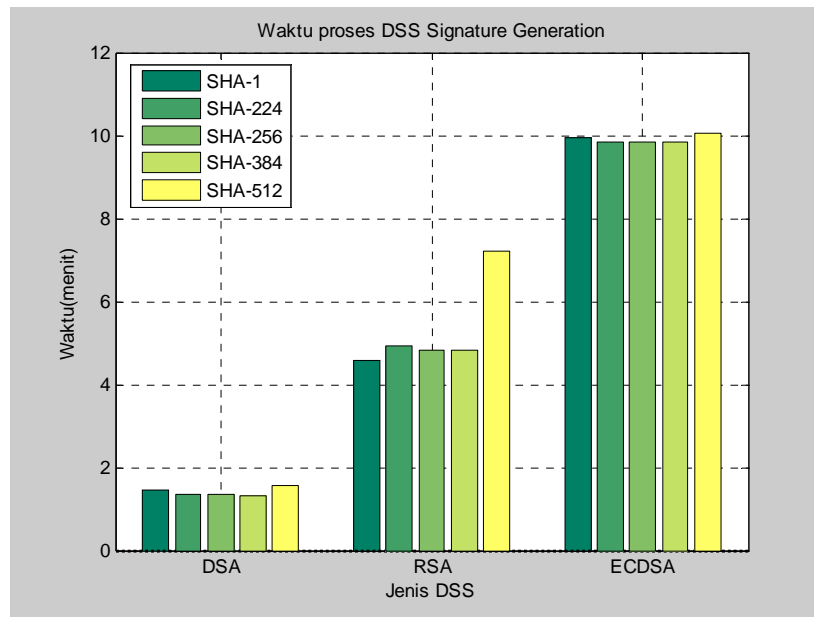
Pada penggabungan SHS-RSA, terdapat SHA-384 memiliki kinerja yang lebih baik. Karena pada penggabungannya memiliki waktu proses *signature generation* paling kecil (4.8135 menit) dan pada *signature verification* juga tergolong kecil (4.7616 menit), memiliki variasi *signature* yang paling kecil (6.7336) dari jenis SHS lainnya, dan pada *brute force attack* memiliki waktu yang seimbang dengan jenis SHS lainnya, didukung lagi dengan adanya *hash* sebelumnya juga memiliki kinerja yang paling baik juga (seperti yang terlihat pada gambar 4.12-14).



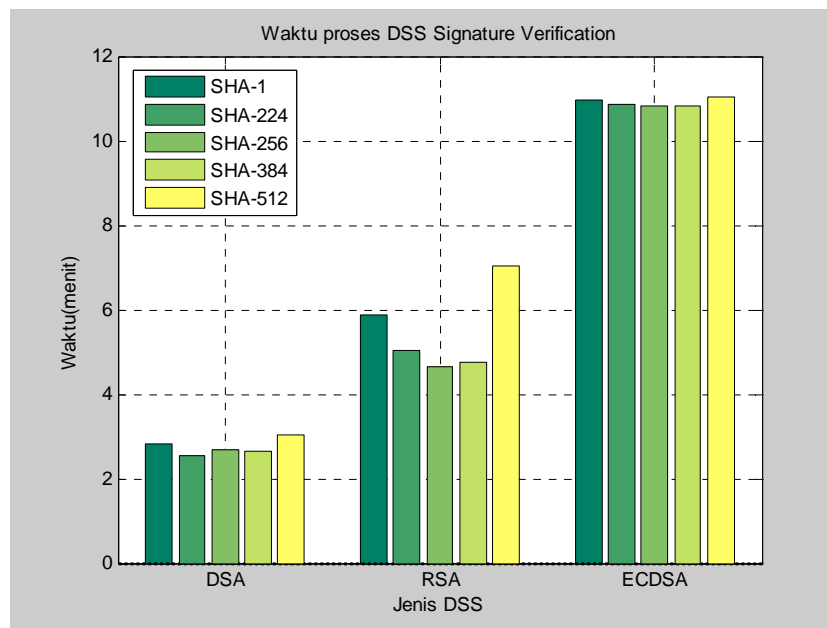
**Gambar 3.6** Perbandingan output DSA, RSA, dan ECDSA terhadap jenis SHS

Pada penggabungan SHS-ECDSA, terdapat SHA-384 memiliki kinerja yang lebih baik. Karena pada penggabungannya memiliki waktu proses *signature generation* paling kecil (4.8135 menit) dan pada *signature verification* juga tergolong kecil (4.7616 menit), memiliki variasi *signature* yang tergolong kecil (268.98), dan pada *brute force attack* memiliki waktu yang seimbang dengan jenis SHS lainnya, didukung lagi dengan adanya *hash* sebelumnya juga memiliki kinerja yang paling baik juga (seperti yang terlihat pada gambar 4.15-17).

Jika dilihat dari parameter waktu, DSA memiliki waktu proses paling baik dibanding jenis DSS lainnya. Jika dilihat dari parameter *brute force attack*, tidak memiliki banyak perbedaan antar jenis DSS, tetapi mengalami perbedaan pada antar jenis SHS inputannya. Jika dilihat dari parameter ketahanan matematis terhadap *attacker*, maka ECDSA mempunyai keunggulan dari kompleksitasnya di DLP dan *Elliptic Curve*.



**Gambar 3.7** Waktu Proses DSS Signature Generation



**Gambar 3.8** Waktu proses DSS Signature Verification

### 3.2.3 Analisis Peluang Cracking

*Key generation* memiliki pengaruh besar pada ketahanan dari *cryptography*. Untuk mendapatkan *key* harus mendapatkan prima terlebih dahulu. Pemilihan prima ini sangat banyak ditemukan dalam suatu *range*. Misal prima yang dibutuhkan sebesar 512 bit, maka terdapat sekitar  $10^{151}$  buah prima pilihan. Bila terdapat satu milyar ( $10^9$ ) orang yang masing-masing berusaha mendapatkan 1000 bilangan prima, maka diperlukan hanya  $10^{12}$  bilangan prima yang berbeda untuk memenuhinya. Bandingkan  $10^{12}$  bilangan prima yang diperlukan dengan  $10^{151}$  bilangan prima yang tersedia. Bayangkan pula bila disediakan bilangan prima 1024 bit, maka akan tersedia bilangan prima sekitar  $10^{305}$ . Angka ini diperoleh dari jumlah

bilangan prima yang kurang dari  $n$  adalah sekitar  $n/(\ln n)$ . Dan sulit sekali bagi *attacker* menyimpan *database* yang dapat menyimpan seluruh bilangan prima dari 2 hingga kurang dari  $2^{1024}$ . Hal ini dikarenakan apabila mempunyai *harddisk* berkapasitas  $10^{35}$  *word* (asumsi setiap *word* mampu menyimpan satu bilangan prima), maka akan diperlukan sekitar  $10^{270}$  *harddisk* untuk menyimpan seluruh blangan prima yang kurang dari  $2^{1024}$ .

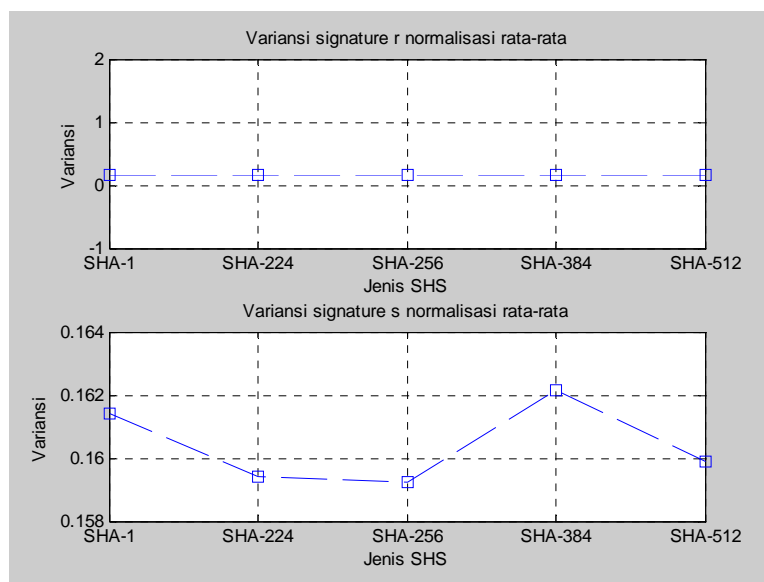
Secara matematis proses DSA, terdapat 2 kelebihan keamanan (kompleksitas) yang berhubungan dengan *Discrete Logarithm Problems* (DLP). Terdapat problem logaritmik pada  $Z_p$  sehingga menghasilkan metoda kalkulasi indeks yang sangat besar dan rumit untuk dipecahkan (*index-calculus attack*), dan juga terdapat problem logaritmik pada *subgroup cyclic* orde  $q$  dimana merupakan metoda paling baik untuk menjalankan “*square-root*” time.

Selain performansi yang didapat dengan adanya kunci dan DLP, apabila akan dibangkitkan beberapa *signature* yang berurutan, maka akan menghasilkan komputasi *signature* yang berbeda dengan *private key* yang sama. Sehingga *attacker* sulit memprediksi masukan *message* dari *signature* yang didapat.

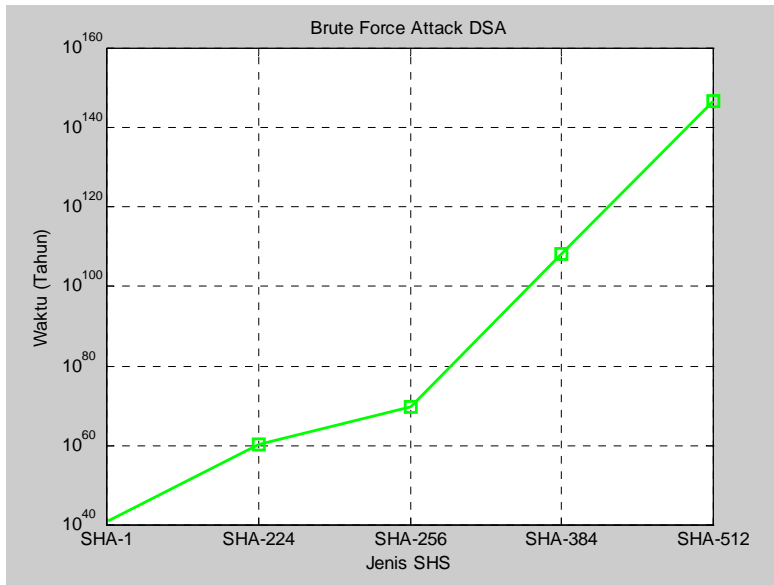
Secara matematis proses RSA, terdapat beberapa keunggulan (kompleksitas) pada proses *signature generation* dan *verification*. Apabila  $n = pq$  merupakan *2k-bit modulus* RSA dimana  $p$  dan  $q$  berukuran  $k$ -bit prima. Komputasi *signature*  $s = m^d \bmod n$  untuk *message*  $m$  membutuhkan bit operasi (tergantung *modular multiplication* dan *modular eksponensial*). Bilangan ini membutuhkan kalkulasi yang sangat besar dengan proses perkalian yang berulang-ulang tergantung panjang *private key* sehingga waktu proses RSA terbesar terpakai disini. Dan proses ini lebih efisien dibandingkan jika diketahui  $p$  dan  $q$  dan dilakukan komputasi terpisah  $s_1 = m^d \bmod p$ ,  $s_2 = m^d \bmod q$  (*chinese remainder theorem*). Proses verifikasi terjadi lebih cepat dibanding *signature generation* karena *public key* yang dipilih lebih kecil.

Secara matematis proses ECDSA memiliki kemiripan dengan DSA. Pada proses *signature generation* dan *signature verivication* terdapat kompleksitas pada *Discrete Logarithm Problems* (DLP). Namun ketahanannya diperkuat dengan pencarian bilangan prima, *private key* dan *public key* yang lebih sulit, yaitu dengan menggunakan kurva *elliptic*.

### 3.2.4 Kumpulan hasil simulasi DSA

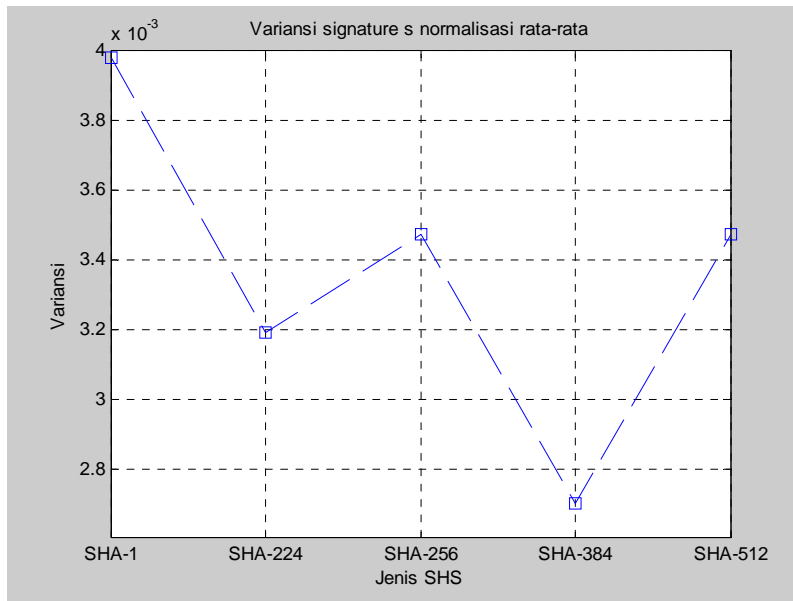


Gambar 3.9 Variansi *signature* normalisasi

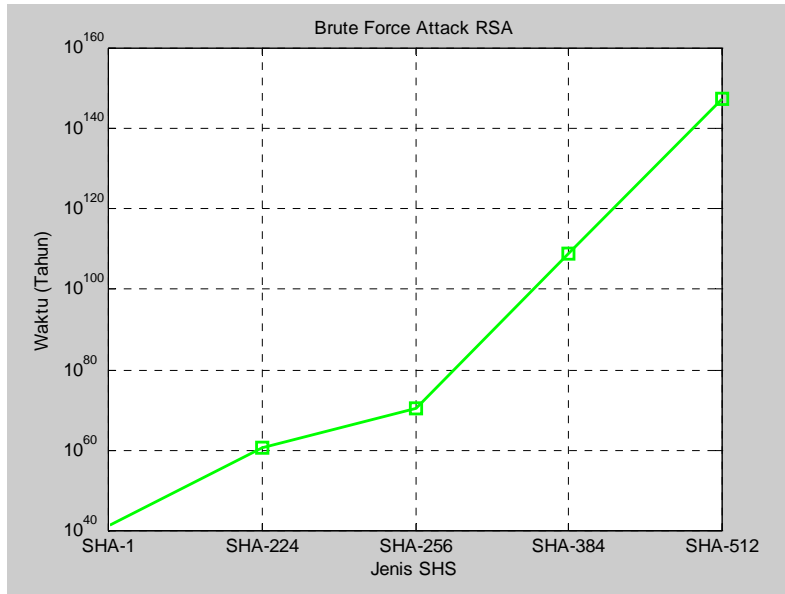


**Gambar 3.10** *Brute force attack* pada DSA

**RSA**

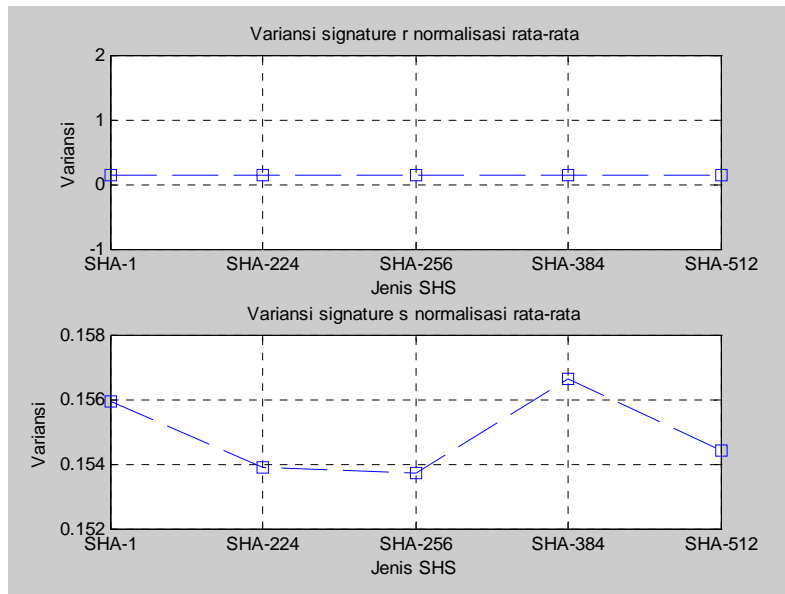


**Gambar 3.11** Variansi *signature* normalisasi

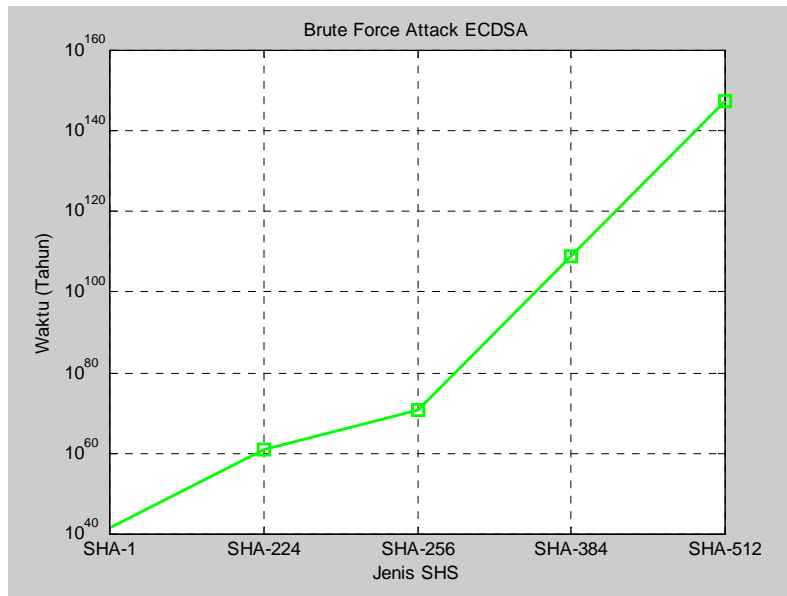


**Gambar 3.12** Brute force attack pada RSA

### ECDSA



**Gambar 3.13** Variansi *signature* normalisasi



**Gambar 3.14** Brute force attack pada ECDSA

## BAB IV

## Penutup

---

### 4.1. Kesimpulan

1. *Message digest* memiliki faktor pengacakan lebih besar dari input, dapat terlihat dari variansi distribusi *message digest* dan masukan. Sehingga SHS dapat dipergunakan sebagai *pseudorandom generator* baik pada *signature generation* maupun *key generation*.
2. *Message digest* memiliki ukuran sangat kecil (160, 224, 256, 384, dan 512 bit) sehingga menghemat bandwidth pada transmisi *message* beserta *signature*-nya.
3. Jika dilihat dari parameter waktu, SHA-1 memiliki waktu relatif paling cepat (56,12 % dibanding waktu terlama pada SHA-224). Dan diikuti waktu kedua paling cepat SHA-512 (83,07 % dibanding waktu terlama pada SHA-224). Jika dilihat dari parameter variansi distribusi normalisasi, SHA-1 memiliki nilai terbesar (0,0005) dan diikuti oleh SHA-512 (0,00069987). Tetapi jika dianalisis secara ketahanan terhadap *cracking*, SHA-224 dan SHA-384 memiliki ketahanan terhadap *cracking* paling besar.
4. Apabila ukuran *message* kecil ( $<2^{64}$  bit), maka sebaiknya menggunakan SHA-1 karena waktunya lebih cepat dibanding jenis-jenis SHS yang lain. Tetapi apabila *message* kecil sangat penting kerahasiaannya maka lebih baik menggunakan SHA-224 atau SHA-256. Apabila ukuran *message* besar ( $<2^{128}$  bit), maka sebaiknya menggunakan SHA-384. Tetapi apabila *message* besar sangat penting kerahasiaannya maka lebih baik menggunakan SHA-512.
5. Secara umum dari analisis kelima SHS, SHA-384 memiliki performansi paling baik. Hal ini dikarenakan SHS tersebut paling tahan terhadap analisis *cracking* (ketahanan *cracking* terbaik pada SHA-224 dan SHA-384), memiliki variansi dan waktu proses lebih kecil dibanding SHA-224, dan ketahanan *Brute Force Attack* paling tinggi dibanding SHA-1, SHA-224, dan SHA-256. Hasil analisis ini merupakan analisis secara umum, namun pada implementasinya tergantung pada keterbatasan *processor* dan *memory* (sehingga diambil waktu proses paling efisien), jumlah masukan *message*, dan lain-lain.
6. Suatu serangan dapat dihadang dengan proteksi dari DSS dengan berbagai macam jenis kehandalan dan lebih didukung lagi dengan adanya problem SHS yang menghasilkan masukan DSS menjadi *random* sehingga tingkat kompleksitas keluaran menjadi jauh lebih baik. Dan faktor pencarian masukan menjadi jauh lebih kompleks dan memakan waktu lebih banyak dari faktorisasi DSS dan SHS.
7. Pada penggabungan SHS-DSA, terdapat SHA-384 memiliki kinerja yang lebih baik. Karena pada penggabungannya memiliki waktu proses *signature generation* paling kecil (1.331 menit) dan pada *signature verification* juga tergolong kecil (2.6475 menit), walaupun memiliki variansi *signature* yang paling besar dari jenis SHS lainnya, tetapi variansinya masih termasuk kecil, dan pada *brute force attack* memiliki waktu yang seimbang dengan jenis SHS lainnya, didukung lagi dengan adanya *hash* sebelumnya juga memiliki kinerja yang paling baik juga.

8. Pada penggabungan SHS-RSA, terdapat SHA-384 memiliki kinerja yang lebih baik. Karena pada penggabungannya memiliki waktu proses *signature generation* paling kecil (9.831 menit) dan pada *signature verification* juga tergolong kecil (10.831 menit), memiliki variasi *signature* yang paling kecil (6.7336) dari jenis SHS lainnya, dan pada *brute force attack* memiliki waktu yang seimbang dengan jenis SHS lainnya, didukung lagi dengan adanya *hash* sebelumnya juga memiliki kinerja yang paling baik juga.
9. Pada penggabungan SHS-ECDSA, terdapat SHA-384 memiliki kinerja yang lebih baik. Karena pada penggabungannya memiliki waktu proses *signature generation* paling kecil (4.8135 menit) dan pada *signature verification* juga tergolong kecil (4.7616 menit), memiliki variasi *signature* yang tergolong kecil (268.98), dan pada *brute force attack* memiliki waktu yang seimbang dengan jenis SHS lainnya, didukung lagi dengan adanya *hash* sebelumnya juga memiliki kinerja yang paling baik juga.
10. Jika dilihat dari parameter waktu, DSA memiliki waktu proses paling baik dibanding jenis DSS lainnya. Jika dilihat dari parameter *brute force attack*, tidak memiliki banyak perbedaan antar jenis DSS, tetapi mengalami perbedaan pada antar jenis SHS inputannya. Jika dilihat dari parameter ketahanan matematis terhadap *attacker*, maka ECDSA mempunyai keunggulan dari kompleksitasnya di DLP dan *Elliptic Curve*.
11. Apabila dibandingkan antar jenis DSS, terdapat keunggulan pada ECDSA yang memiliki kinerja lebih baik. Karena pada penggabungannya memiliki kompleksitas problem paling tinggi, variasi distribusi yang lebih baik dibanding DSA, dan efisiensi dalam pemakaian jumlah masukan DSS.

#### 4.2 Saran

1. Pada umumnya semua implementasi SHA menggunakan SHA-1. Bersamaan dengan majunya teknologi membuat kebutuhan ukuran *message* semakin besar. Sehingga diharapkan untuk kedepannya semua implementasi diarahkan pada SHA yang lebih tinggi (direkomendasikan SHA-224, SHA-256, SHA-384, dan SHA-512).

## Daftar Pustaka

---

- [1] Coron, J.S. 2002. *Security Proof for Partial-Domain Hash Signature Schemes*. Gemplus Card International.
- [2] Federal Information Processing Standards Publication 180-2. 2002. *Secure Hash Standard*. National Institute of Standards and Technology.
- [3] Federal Information Processing Standards Publication 186-2. 2000. *Digital Signature Standard (DSS)*. National Institute of Standards and Technology.
- [4] Kurniawan, Y. 2004. *Kriptografi: Keamanan Internet dan Jaringan Komunikasi*. Bandung: Penerbit Informatika.
- [5] Menezes, A. 2002. *Evaluation of Security Level of Cryptography: RSA Signature Schemes*. University of Waterloo.
- [6] Menezes, A.J. and D.B. Johnson. *Elliptic Curve DSA (ECDSA): An Enhanced DSA*. University of Waterloo.
- [7] Menezes, A. et al. 2001. *Evaluation of Security Level of Cryptography: ECDSA Signature Scheme*. Certicom Research.
- [8] Menezes, A.J., P.C.V. Oorschot, and S.A. Vanstone. 2001. *Handbook of Applied Cryptography*. CRC Press.
- [9] Vaudenay, S. *Digital Signature Schemes with Domain Parameters*. Ecole Polytechnique Federale de Lausanne.