APLIKASI SIMULASI AUTENTIKASI DATA MENGGUNAKAN METODE *SCHNORR AUTHENTICATION* DAN *DIGITAL SIGNATURE SCHEME*

TUGAS AKHIR
Diajukan Sebagai Salah Satu Syarat
Untuk Memperoleh Gelar Sarjana

Oleh :
M. MUSTARI
10351022873

FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI SULTAN SYARIF KASIM RIAU
PEKANBARU
2010

APLIKASI SIMULASI AUTENTKASI DATA MENGGUNAKAN METODE SCHNORR AUTHENTICATION DAN DIGITAL SIGNATURE SCHEME

M.MUSTARI

10351022873

Tanggal Sidang : 30 Januari 2010

Periode Wisuda : Februari 2010

Jurusan Teknik Informatika

Fakultas Sains dan Teknologi

Universitas islam negeri sultan Syarif Kasim Riau

ABSTRAK

Pada situasi kurangnya memahami dan mempelajari kerja metode *Schnorr Authentication* dan *Digital Signature Scheme* maka diperlukan suatu alat bantu yang dapat mensimulasikan atau menggambarkan proses Autentikasi data. *Metode Schnorr Authentication* dan *Digital Signature Scheme* adalah suatu metode yang menerangkan tentang bagaimana mengidentifikasi dan menjamin integritas sumber dari sebuah pesan oleh masing – masing pihak yang saling berkomunikasi.

Metode *Schnorr Authentication* dan *Digital Signature Scheme* menggunakan bilangan prima dan perpangkatan modulo dalam proses pembentukan kunci, sedangkan proses pembentukan Kunci Privat dan Publik menggunakan Skema Otentikasi yang ditambahkan dengan sebuah fungsi Hash.

Dalam simulasi autentikasi data dilakukan oleh 2 orang pengguna yaitu pengirim dan penerima. Proses simulasi antara pengirim dan penerima dilakukan dengan 3 proses yaitu proses Pembentukan Kunci, Autentikasi, dan Tanda Tangan Digital.

Dari hasil pengujian aplikasi simulasi dapat membantu pemahaman terhadap pemahaman *Metode Schnorr Authentication* dan *Digital Signature Scheme*. Aplikasi simulasi dapat digunakan untuk mendukung kegiatan belajar mengajar, terutama dalam mata kuliah kriptografi.

Kata kunci : Authentication, Digital signature, Metode schnorr

SIMULATION APPLICATION DATA AUTHENTICATION USING AUTHENTICATION SCHNORR METHOD AND DIGITAL SIGNATURE SCHEME

M.MUSTARI

10351022873

Date of Final Exam: January 30th 2010

Graduation Ceremony Period : February 2010

Informatics Engineering Department

Faculty of Sciences and Technology

State Islamic University of Sultan Syarif Kasim Riau

ABSTRACT

Lack of understanding on the situation and learn how to work methods Schnorr Authentication and Digital Signature Scheme, it would require a tool that can simulate or describe the process of Authentication Data. Schnorr Method and Digital Signature Scheme is a method that explains about how to identify and ensure the integrity of the source of a message by each party to communicate with each other.

Methods Schnorr Authentication and Digital Signature Scheme using modulo primes and powers in the process of formation of the key, while the formation of Private and Public Key Authentication Scheme that is added using a Hash function.

In the simulation of authentication data is done by 2 users of the sender and receiver. Simulation process between sender and receiver is done by 3 main process Formation of Key, Authentication and Digital Signatures.

From the result of the simulation application testing can help the understanding of the methods Schnorr Authentication and Digital Signature Scheme. Simulation application can be used to support teaching and learning activities, especially in Cryptography courses.

Keywords : Authentication, Digital signature, Schnorr method

DAFTAR ISI

Halaman

BAB I
PENDAHULUAN

1.1. Latar Belakang

Otentikasi (*authentication*) merupakan identifikasi yang dilakukan oleh masing – masing pihak satu sama lainnya yang saling berkomunikasi. Informasi yang didapat oleh suatu pihak dari pihak lain harus diidentifikasi untuk memastikan keaslian dari informasi yang diterima. Identifikasi terhadap suatu informasi dapat berupa tanggal pembuatan informasi, isi informasi, waktu kirim dan hal-hal lainnya yang berhubungan dengan informasi tersebut. Otentikasi pesan memang berhasil melindungi kedua belah pihak yang saling bertukar pesan dari pihak ketiga. Tetapi, otentikasi pesan tidak bisa mencegah kemungkinan kedua belah pihak saling menyerang satu sama lain.

Pada situasi dimana tidak ada kepercayaan penuh antara pengirim dan penerima pesan, diperlukan suatu mekanisme yang lebih daripada sekedar otentikasi. Solusi yang paling menarik dari masalah ini adalah tanda tangan digital (*digital signature*). Tanda tangan digital adalah suatu mekanisme otentikasi yang memungkinkan pembuat pesan menambahkan sebuah kode yang bertindak sebagai tanda tangannya. Tanda tangan tersebut menjamin integritas dan sumber dari sebuah pesan.

Claus Schnorr's authentication dan *digital signature scheme* mengambil sekuritas dari permasalahan menghitung logaritma diskrit. Skema otentifikasi ini juga menggunakan bilangan prima dan perpangkatan modulo dalam proses pembentukan kuncinya. Tingkat kesulitan untuk memecahkan algoritma ini adalah sekitar 2^t, dimana nilai t ini dapat ditentukan sendiri. Skema ini dipatenkan di Amerika Serikat dan baru akan berakhir pada tanggal 19 Februari 2008. Skema otentifikasi dapat dimodifikasi menjadi skema tanda tanda digital (*digital signature scheme*). Proses pembentukan kunci privat dan publiknya sama seperti skema otentikasi, hanya saja pada skema tanda tangan digital ditambahkan sebuah fungsi *hash*.

Untuk memahami dan mempelajari Metode Schnoor Authentication menjadi lebih mudah, maka perlu adanya suatu alat bantu yang dapat mensimulasikan atau menggambarkan proses Autentikasi data menggunakan *Metode Schnorr Authentication dan Digital Signature Scheme*.

1.2. Perumusan Masalah

"Bagaimana mensimulasikan prosedur kerja dari *Metode Schnorr Authentication* dan *Digital Signature Schem Dalam Autentikasi Data"*.

1.3. Tujuan

Tujuan penyusunan tugas akhir ini adalah memahami *Schnorr Authentication* dan *Digital Signature Scheme*, serta membuat suatu aplikasi simulasi untuk membantu proses pemahaman terhadap *Schnorr Authentication* dan *Digital Signature Scheme*.

1.4. Batasan Masalah

Batasan masalah dalam membuat aplikasi simulasi *Schnorr Authentication* dan *Digital Signature Scheme* adalah sebagai berikut :

1. Aplikasi akan menampilkan tahap – tahap perhitungan dalam bentuk desimal.
2. Aplikasi menyediakan teori – teori dasar dari *Schnorr Authentication* dan *Digital Signature Scheme*.
3. Perpangkatan modulo bilangan besar menggunakan algoritma *Fast Exponentiation*.
4. Fungsi *hash* yang digunakan adalah fungsi SHA-1.

1.5 Sistematika Penulisan

Sistematika penulisan dalam penyusunan laporan tugas akhir ini adalah sebagai berikut:

BAB I PENDAHULUAN

Bab ini menjelaskan dasar-dasar dari penulisan laporan tugas akhir, yang terdiri dari latar belakang, rumusan masalah, batasan masalah, tujuan, serta sistematika penulisan laporan tugas akhir.

BAB II LANDASAN TEORI

Bab ini membahas teori-teori yang berhubungan dengan topik penelitian, yang terdiri dari Kriptografi, Authentication, Integrity, Nonrepudiation, Fungsi One-Way Hash SHA-1, simulasi, dan teori lainnya yang berhubungan dengan topik tugas akhir ini.

BAB III METODOLOGI PENELITIAN

Bab ini membahas tentang metodologi yang digunakan dalam penelitian dan pengembangan Aplikasi.

BAB IV ANALISIS DAN PERANCANGAN

Bab ini membahas tentang hasil analisis data masukan, analisis proses simulasi Metode , dan perancangan.

BAB V IMPLEMENTASI DAN PENGUJIAN

Bab ini membahas implementasi dan pengujian yang dilakukan terhadap simulasi yang telah dibuat yaitu Aplikasi Simualasi Schnorr Authentication dan Digital Signature Scheme.

BAB VI PENUTUP

Bab ini berisi kesimpulan dan saran yang dihasikan dari pembahasan tentang Aplikasi Simulasi Schnorr Authentication dan Digital Signature Scheme.

BAB II
LANDASAN TEORI

2.1 Keamanan Informasi

Keamanan informasi (*information security*) merupakan perlindungan terhadap informasi ketika informasi dikirim dari sebuah sistem ke sistem lainnya. Sistem kemanan informasi memiliki empat tujuan yang sangat mendasar, yaitu:

1. *Confidentiality*

   Menjamin apakah informasi yang dikirim tersebut tidak dapat dibuka atau tidak dapat diketahui oleh orang lain yang tidak berhak. Untuk data yang penting, dibutuhkan sekali tingkat kerahasiaan yang tinggi, yang hanya bisa diakses oleh pihak-pihak tertentu saja (pihak yang berhak).

2. ***Integrity***

   Menjamin keutuhan dan keaslian data, sehingga upaya pihak-pihak yang tidak bertanggung jawab untuk melakukan penduplikatan dan perusakan data dapat dihindari.

3. *Availability*

   Menjamin pengguna yang sah agar dapat mengakses informasi dan sumber miliknya sendiri. Tujuannya adalah untuk memastikan bahwa pihak-pihak yang memang berhak tidak ditolak untuk mengakses informasi yang memang menjadi haknya.

4. *Legitimate Use*

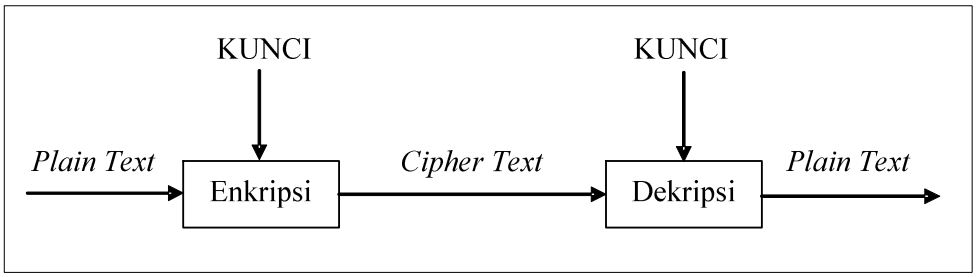
   Menjamin kepastian bahwa sumber tidak digunakan atau informasi tidak diakses oleh pihak-pihak yang tidak bertanggung jawab (pihak-pihak yang tidak berhak).

2.2 Kriptografi

Kriptografi adalah ilmu yang mempelajari teknik-teknik matematika yang berhubungan dengan aspek-aspek pada keamanan informasi misalnya kerahasiaan, integritas data, otentikasi pengirim / penerima data, dan otentikasi data. *Cryptography* (kriptografi) berasal dari bahasa Yunani yaitu dari kata '*cryptos*' dan '*graphia*' yang berarti penulisan rahasia. Kriptografi adalah suatu ilmu yang mempelajari penulisan secara rahasia. Kriptografi bertujuan menjaga kerahasiaan informasi yang terkandung dalam data sehingga informasi tersebut tidak dapat diketahui oleh pihak yang tidak sah.

Suatu pesan yang tidak disandikan disebut sebagai *plaintext* ataupun dapat disebut juga sebagai *cleartext*. Proses yang dilakukan untuk mengubah *plaintext* ke dalam *ciphertext* disebut *encryption* atau *encipherment*. Sedangkan proses untuk mengubah *ciphertext* kembali ke *plaintext* disebut *decryption* atau *decipherment*.

Secara sederhana, konsep umum kriptografi dapat dilihat pada gambar 2.1 berikut.



Gambar 2.1 Gambaran umum proses kriptografi

Kriptografi merupakan suatu ilmu atau seni mengamankan pesan yang dilakukan oleh seorang *cryptographer*. Sedangkan *cryptanalysis* adalah suatu ilmu dan seni membuka (*breaking*) *ciphertext* dan orang yang melakukannya disebut *cryptanalyst*.

Cryptographic system atau *cryptosystem* adalah suatu fasilitas untuk mengkonversikan *plaintext* ke *ciphertext* dan sebaliknya. Dalam sistem ini, seperangkat parameter yang menentukan transformasi pen-*cipher*-an tertentu disebut suatu set kunci. Proses enkripsi dan dekripsi diatur oleh satu atau beberapa kunci kriptografi. Secara umum, kunci-kunci yang digunakan untuk proses pengenkripsian dan pendekripsian tidak perlu identik, tergantung pada sistem yang digunakan.

Secara umum operasi enkripsi dan dekripsi dapat diterangkan secara matematis sebagai berikut :

$$EK (M) = C \quad \text{(Proses Enkripsi)}$$

$$DK (C) = M \quad \text{(Proses Dekripsi)}$$

Pada saat proses enkripsi, pesan M (*message*) disandikan dengan suatu kunci K (*key*), sehingga dihasilkan pesan C (*ciphertext*). Pesan C adalah pesan terenkripsi dan tidak dapat dibaca. Dalam hal ini, C berfungsi sebagai sandi rahasia yang hanya dapat dibaca oleh pihak-pihak yang berhak. Sedangkan pada proses dekripsi, pesan C tersebut disandikan kembali dengan menggunakan kunci K sehingga dihasilkan pesan M yang sama seperti pesan sebelumnya.

Dengan demikian, keamanan suatu pesan tergantung pada kunci yang digunakan dan tidak tergantung pada algoritma yang digunakan. Sehingga algoritma-algoritma yang digunakan tersebut dapat dipublikasikan dan dianalisis, serta produk-produk yang menggunakan algoritma tersebut dapat diproduksi massal. Tidak menjadi masalah apabila seseorang mengetahui algoritma yang kita gunakan. Selama ia tidak mengetahui kunci yang dipakai, ia tetap tidak dapat membaca pesan.

2.3 Aplikasi Kriptografi

2.3.1 Confidentiality dan Privacy

Confidentiality dan *privacy* terkait dengan kerahasiaan data atau informasi. Pada sistem *e-government* kerahasiaan data-data pribadi (*privacy*) sangat penting. Hal ini kurang mendapat perhatian di sistem *e-government* yang sudah ada. Bayangkan jika data pribadi anda, misalnya data KTP atau kartu keluarga, dapat diakses secara *online*. Maka setiap orang dapat melihat tempat dan tanggal lahir anda, alamat anda,

dan data lainnya. Data ini dapat digunakan untuk melakukan penipuan dan pemAndryolan dengan mengakui sebagai anda (atau keluarga anda).

Ancaman atau serangan terhadap kerahasiaan data ini dapat dilakukan dengan menggunakan penerobosan akses, penyadapan data (*sniffer, key logger*), *social engineering* (yaitu dengan menipu), dan melalui kebijakan yang tidak jelas (tidak ada).

Untuk itu kerahasiaan data ini perlu mendapat perhatian yang besar dalam implementasi sistem *e-government* di Indonesia. Proteksi terhadap data ini dapat dilakukan dengan menggunakan *firewall* (untuk membatasi akses), segmentasi jaringan (juga untuk membatasi akses), enkripsi (untuk menyandikan data sehingga tidak mudah disadap), serta kebijakan yang jelas mengenai kerahasiaan data tersebut. Pengujian terhadap kerahasian data ini biasanya dilakukan secara berkala dengan berbagai metode.

Ada beberapa jenis informasi yang tersedia didalam sebuah jaringan komputer. Setiap data yang berbeda pasti mempunyai grup pengguna yang berbeda pula dan data dapat dikelompokkan sehingga beberapa pembatasan kepada pengunaan data harus ditentukan. Pada umumnya data yang terdapat didalam suatu perusahaan bersifat rahasia dan tidak boleh diketahui oleh pihak ketiga yang bertujuan untuk menjaga rahasia perusahaan dan strategi perusahaan. *Backdoor*, sebagai contoh, melanggar kebijakan perusahaan dikarenakan menyediakan akses yang tidak diinginkan kedalam jaringan komputer perusahaan.

Kerahasiaan dapat ditingkatkan dan didalam beberapa kasus pengengkripsian data. Kontrol akses adalah cara yang lazim digunakan untuk membatasi akses kedalam sebuah jaringan komputer. Sebuah cara yang mudah tetapi mampu untuk membatasi akses adalah dengan menggunakan kombinasi dari *username*-dan-*password* untuk proses otentikasi pengguna dan memberikan akses kepada pengguna (*user*) yang telah dikenali. Didalam beberapa lingkungan kerja keamanan jaringan komputer, ini dibahas dan dipisahkan dalam konteks otentikasi.

Dalam menjaga kerahasiaan data, kriptografi mentransformasikan data jelas (*plaintext*) ke dalam bentuk data sandi (*ciphertext*) yang tidak dapat dikenali. *Ciphertext* inilah yang kemudian dikirimkan oleh pengirim (*sender*) kepada penerima (*receiver*). Setelah sampai di penerima, *ciphertext* tersebut ditranformasikan kembali ke dalam bentuk *plaintext* agar dapat dikenali.

2.3.2 Otentikasi (Authentication)

Otentikasi merupakan identifikasi yang dilakukan oleh masing – masing pihak yang saling berkomunikasi, maksudnya beberapa pihak yang berkomunikasi harus mengidentifikasi satu sama lainnya. Informasi yang didapat oleh suatu pihak dari pihak lain harus diidentifikasi untuk memastikan keaslian dari informasi yang diterima. Identifikasi terhadap suatu informasi dapat berupa tanggal pembuatan informasi, isi informasi, waktu kirim dan hal-hal lainnya yang berhubungan dengan informasi tersebut. Aspek ini berhubungan dengan metode untuk menyatakan bahwa

informasi betul-betul asli, orang yang mengakses atau memberikan informasi adalah betul-betul orang yang dimaksud, atau *server* yang kita hubungi adalah betul-betul *server* yang asli.

Masalah pertama, membuktikan keaslian dokumen, dapat dilakukan dengan teknologi *watermarking* dan *digital signature*. *Watermarking* juga dapat digunakan untuk menjaga "*intelectual property*", yaitu dengan menandai dokumen atau hasil karya dengan "tanda tangan" pembuat. Masalah kedua biasanya berhubungan dengan akses kontrol, yaitu berkaitan dengan pembatasan orang yang dapat mengakses informasi.

Dalam hal ini pengguna harus menunjukkan bukti bahwa memang dia adalah pengguna yang sah, misalnya dengan menggunakan *password* Aspek / servis dari *security biometric* (ciri-ciri khas orang), dan sejenisnya. Ada tiga hal yang dapat ditanyakan kepada orang untuk menguji siapa dia:

• *What you have* (misalnya kartu ATM)

• *What you know* (misalnya PIN atau *password*)

• *What you are* (misalnya sidik jari, *biometric*)

Penggunaan teknologi *smart card*, saat ini kelihatannya dapat meningkatkan keamanan aspek ini. Secara umum, proteksi *authentication* dapat menggunakan *digital certificates. Authentication* biasanya diarahkan kepada orang (pengguna), namun tidak pernah ditujukan kepada *server* atau mesin. Pernahkan kita bertanya bahwa mesin ATM yang sedang kita gunakan memang benar-benar milik bank yang

bersangkutan? Bagaimana jika ada orang nakal yang membuat mesin seperti ATM sebuah bank dan meletakkannya di tempat umum? Dia dapat menyadap data-data (informasi yang ada di *magnetic strip*) dan PIN dari orang yang tertipu. Memang membuat mesin ATM palsu tidak mudah.

Tetapi, bisa anda bayangkan betapa mudahnya membuat *web site* palsu yang menyamar sebagai *web site* sebuah bank yang memberikan layanan *Internet Banking*. (Ini yang terjadi dengan kasus klikBCA.com)

2.3.3 Integritas (Integrity)

Aspek integritas (*integrity*) terkait dengan keutuhan data. Aspek ini menjamin bahwa data tidak boleh diubah tanpa ijin dari pihak yang memiliki hak. Acaman terhadap aspek integritas dilakukan dengan melalui penerobosan akses, pemalsuan (*spoofing*), virus yang mengubah atau menghapus data, dan *man in the middle attack* (yaitu penyerangan dengan memasukkan diri di tengah-tengah pengiriman data). Proteksi terhadap serangan ini dapat dilakukan dengan menggunakan *digital signature*, *digital certificate*, *message authentication code*, *hash function*, dan *checksum*. Pada prinsipnya mekanisme proteksi tersebut membuat kode sehingga perubahan satu bit pun akan mengubah kode.

Contoh permasalahan integritas dapat dilihat pada sistem perhitungan pemilihan umum tahun 2004 kemarin, dimana pada awal proses perhitungan masih

terdapat data uji coba. Hal ini menimbulkan keraguan atas integritas dari data yang berada di dalamnya.

Jaringan komputer yang dapat diandalkan juga berdasar pada fakta bahwa data yang tersedia apa yang sudah seharusnya. Jaringan komputer mau tidak mau harus terlindungi dari serangan (*attacks*) yang dapat merubah data selama dalam proses persinggahan (transmisi). *Man-in-the-Middle* merupakan jenis serangan yang dapat merubah integritas dari sebuah data yang mana penyerang (*attacker*) dapat membajak *session* atau memanipulasi data yang terkirim.

Di dalam jaringan komputer yang aman, partisipan dari sebuah transaksi data harus yakin bahwa orang yang terlibat dalam komunikasi data dapat diandalkan dan dapat dipercaya. Keamanan dari sebuah komunikasi data sangat diperlukan pada sebuah tingkatan yang dipastikan data tidak berubah selama proses pengiriman dan penerimaan pada saat komunikasi data. Ini tidak harus selalu berarti bahwa *traffic* perlu di enkripsi, tapi juga tidak tertutup kemungkinan serangan *Man-in-the-Middle* dapat terjadi. *Man in the middle attack* adalah serangan dimana seseorang menempatkan diri di tengah pembicaraan dan menyamar sebagai orang lain.

2.3.4 Nonrepudiation

Aspek ini menjaga agar seseorang tidak dapat menyangkal telah melakukan sebuah transaksi. Sebagai contoh, seseorang yang mengirimkan email untuk memesan barang tidak dapat menyangkal bahwa dia telah mengirimkan email

tersebut. Aspek ini sangat penting dalam hal *electronic commerce*. Penggunaan *digital signature, certificates,* dan teknologi kriptografi secara umum dapat menjaga aspek ini. Akan tetapi hal ini masih harus didukung oleh hukum sehingga status dari *digital signature* itu jelas legal.

Setiap tindakan yang dilakukan dalam sebuah sistem yang aman telah diawasi (*logged*), ini dapat berarti penggunaan alat (*tool*) untuk melakukan pengecekan sistem berfungsi sebagaimana seharusnya. *Log* juga tidak dapat dipisahkan dari bagian keamanan sistem yang dimana bila terjadi sebuah penyusupan atau serangan lain akan sangat membantu proses investigasi. *Log* dan catatan waktu, sebagai contoh, bagian penting dari bukti di pengadilan jika *cracker* tertangkap dan diadili. Untuk alasan ini maka *nonrepudiation* dianggap sebagai sebuah faktor penting didalam keamanan jaringan komputer yang berkompeten.

Definisi dari *nonrepudition* adalah sebagai berikut :

1. Kemampuan untuk mencegah seorang pengirim untuk menyangkal kemudian bahwa dia telah mengirim pesan atau melakukan sebuah tindakan.
2. Proteksi dari penyangkalan oleh satu satu dari entitas yang terlibat didalam sebuah komunikasi yang turut serta secara keseluruhan atau sebagian dari komunikasi yang terjadi.

Proses tranformasi dari *plaintext* menjadi *ciphertext* disebut proses *encipherment* atau enkripsi (*encryption*), sedangkan proses mentransformasikan kembali *ciphertext* menjadi *plaintext* disebut proses dekripsi (*decryption*).

Untuk mengenkripsi dan mendekripsi data, kriptografi menggunakan suatu algoritma (*cipher*) dan kunci (key). *Cipher* adalah fungsi matematika yang digunakan untuk mengenkripsi dan mendekripsi. Sedangkan kunci merupakan sederetan bit yang diperlukan untuk mengenkripsi dan mendekripsi data.

2.4

2.5 Fungsi One-Way Hash SHA-1

Sebuah fungsi *hash* (*hash function* atau *hash algorithm*) adalah suatu cara untuk menghasilkan sebuah *digital* "*fingerprint*" kecil dari sembarang data. Fungsi ini memecahkan dan mencampurkan data untuk menghasilkan *fingerprint* yang sering disebut sebagai nilai *hash* (*hash value*). Nilai *hash* ini sering direpresentasikan dengan sebuah *string* pendek dari huruf-huruf dan angka-angka yang kelihatan acak (berbentuk heksadesimal). Sebuah fungsi *hash* yang baik adalah suatu fungsi yang tidak (jarang) memiliki output nilai *hash* yang sama untuk input yang berbeda.

Secure Hash Algorithm, SHA-1 ini dikembangkan oleh NIST (*National Institute of Standard and Technology*). SHA-1 dapat diterapkan dalam penggunaan *Digital Signature Algorithm* (DSA) yang dispesifikasikan dalam *Digital Signature Standard* (DSS) dan SHA tersebut dapat diterapkan untuk aplikasi federal.

Untuk suatu pesan yang panjangnya < 2^{64}, SHA-1 akan menghasilkan keluaran sebanyak 160 bit dari pesan tersebut dan pesan keluaran itu disebut *message digest*. Panjang jarak *message digest* dapat berkisar antara 160 sampai 512 bit

tergantung algoritmanya. Berdasarkan cirinya SHA-1 dapat digunakan dengan algoritma kriptografi lainnya seperti *Digital Signature Algorithms* atau dalam generasi angka yang acak (*bits*).

SHA-1 dikatakan aman karena proses SHA-1 dihitung secara infisibel untuk mencari pesan yang sesuai untuk menghasilkan *message digest* atau dapat juga digunakan untuk mencari dua pesan yang berbeda yang akan menghasilkan *message digest* yang sama.

Menurut jenisnya SHA dapat dispesifikasikan menjadi 4 bagian yaitu: SHA-1, SHA-256, SHA-384, dan SHA-512. Berikut ini merupakan daftar-daftar properti dari keempat SHA.

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)	Security[2] (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

Gambar 2.2 Daftar-daftar properti dari keempat SHA

Untuk SHA-1 ukuran blok pesan -m bit- dapat ditentukan tergantung dari algoritmanya. Pada SHA-1 masing-masing blok pesan mempunyai 512 bit dimana dapat dilakukan dengan 16 urutan sebesar 32 bit.

SHA-1 digunakan untuk menghitung *message digest* pada pesan atau *file* data yang diberikan sebagai *input*. Tujuan pengisian pesan adalah untuk menghasilkan total dari pesan yang diisi menjadi perkalian dari 512 bits.

Beberapa hal yang dilakukan dalam pengisian pesan :

a. Panjang dari pesan,M adalah k bits dimana panjang $k < 2^{64}$. Tambahkan bit "1" pada akhir pesan. Misalkan pesan yang asli adalah "01010000" maka setelah diisi menjadi "010100001".

b. Tambahkan bit "0", angka bit "0" tergantung dari panjang pesan. Misalnya :Pesan asli yang merupakan bit string : abcde

   01100001 01100010 01100011 01100100 01100101.

   Setelah langkah (a) dilakukan

   01100001 01100010 01100011 01100100 0110010 1.

   Panjang k = 40 dan angka bit di atas adalah 41dan 407 ditambah bit "0" (448 – (40+1) = 407). Kemudian diubah dalam hex:

   ```
   61626364  65800000  00000000  00000000
   00000000  00000000  00000000  00000000
   00000000  00000000  00000000  00000000
   00000000  00000000
   ```

c. Untuk memperoleh 2 kata dari k,angka bit dalam pesan asli yaitu jika $k < 2^{32}$ maka kata pertama adalah semua bit "0". Maka gambaran dari 2 kata dari k = 40 dalam hex adalah 00000000 00000028.

   ```
   61626364  65800000  00000000  00000000
   00000000  00000000  00000000  00000000
   ```

```
00000000    00000000    00000000    00000000
00000000    00000000    00000000    00000028
```

SHA-1 menggunakan urutan fungsi logika yang dilambangkan dengan f_0, f_1,…, f_{79}. Untuk masing-masing f_t, dimana $0 \le t < 79$ akan menghasilkan output sebanyak 32 bit.

Fungsinya adalah sebagai berikut:

$$f_t(B, C, D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & 0 \le t \le 19 \\ B \oplus C \oplus D & 20 \le t \le 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & 40 \le t \le 59 \\ B \oplus C \oplus D & 60 \le t \le 79 \end{cases}$$

$\oplus$ = **fungsi XOR**

Konstanta kata yang digunakan pada SHA-1 yang disimbolkan secara berurutan dari K(0), K(1),…, K(79) dalam bentuk *hex* adalah sebagai berikut :

$$K = \begin{cases} 5A827999 & 0 \le t \le 19 \\ 6ED9EBA1 & 20 \le t \le 39 \\ 8F1BBCDC & 40 \le t \le 59 \\ CA62C1D6 & 60 \le t \le 79 \end{cases}$$

Algoritma SHA-1 dapat diringkas sebagai berikut:

a. Penghitungan menggunakan dua *buffer* dimana masing-masing *buffer* terdiri dari lima sebesar 32 bit kata dan urutan 80 juga sebesar 32 bit kata. Lima kata pertama pada *buffer* kata diberi nama A, B, C, D, E sedangkan lima kata

kedua diberi nama H_0, H_1, H_2, H_3, dan H_4. Kemudian pada 80 kata yang berurutan diberi nama W_0, W_1, …, W_{79} dan pada penghitungan ini juga memakai sebuah variabel sementara, TEMP.

b. Lakukan pengisian pesan, M dan kemudian parsingkan pesan tersebut ke dalam N 512 bit blok pesan, $M^{(1)}$, $M^{(2)}$, …, $M^{(n)}$. Caranya : 32 bit pertama dari blok pesan ditunjukkan ke $M_0^{(i)}$, lalu 32 bit berikutnya adalah $M_1^{(i)}$ dan selanjutnya berlaku hingga $M_{15}^{(i)}$.

c. Inisialisasi Nilai *Hash* (dalam bentuk hex) :

H_0 = 67452301 H_3 = 10325476

H_1 = EFCDAB89 H_4 = C3D2E1F0

H_2 = 98BADCFE

d. Lakukan proses M_1, M_2, …, M_n dengan cara membagi M_i ke dalam 16 kata W_0, W_1, …, W_{15} dimana W_0 merupakan *left most*.

e. Hitung : For t = 16 to 79

$$W_t = S^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$$

f. Inisialisasi 5 variabel A, B, C, D, dan E dengan nilai *Hash* :

$A = H_0$; $B = H_1$; $C = H_2$; $D = H_3$; $E = H_4$.

g. Hitung : For t = 0 to 79

$$TEMP = S^5(A) + f_t(B,C,D) + E + Wt + Kt$$

$E = D$; $D = C$; $C = S^{30}(B)$; $B = A$; $A = TEMP$.

h. Hitung Nilai *Hash* :

$H_0 = H_0 + A$; $H_1 = H_1 + B$;

$H_2 = H_2 + C$; $H_3 = H_3 + D$;

$H_4 = H_4 + E$.

Hasil dari *message digest* sebesar 160 bit dari pesan, M adalah : H_0 H_1 H_2 H_3 H_4**.**

2.6 Landasan Matematis Kriptografi

2.6.1 Bilangan Prima

Bilangan prima adalah bilangan *integer* yang lebih besar dari satu yang memiliki faktor bilangan satu dan bilangan itu sendiri. Dalam proses pembangkitan bilangan prima, sering dihadapkan dengan beberapa masalah berikut ini:

1. Permasalahan banyaknya bilangan prima yang tersedia. Ahli matematika telah menemukan bahwa jumlah bilangan prima yang tersedia pada bilangan 512 bit adalah sekitar 10^{151} (Berdasarkan buku '*Applied Cryptography*',Bruce Schneier).
2. Mungkinkah dua orang atau lebih mendapatkan dua bilangan prima yang sama? 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

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).

3. Permasalahan apakah seseorang mampu membuat *database* yang dapat menyimpan seluruh bilangan prima dari 2 hingga kurang dari 2^{1024}. Hal ini dapat dilakukan tetapi mustahil. Bila mempunyai *harddisk* yang berkapasitas 10^{35} word (untuk mempermudah perhitungan, dianggap setiap word sanggup menyimpan satu bilangan prima), maka akan diperlukan sekitar 10^{270} *harddisk* untuk menyimpan seluruh bilangan prima yang kurang dari 2^{1024}.

Selain itu, pemfaktoran bilangan prima besar juga tidak mudah. Bila mencari faktor prima besar sedemikian sulit, permasalahannya adalah bagaimana dapat membangkitkan bilangan prima dengan mudah. Triknya adalah dengan melakukan pengujian terhadap suatu bilangan prima apakah bilangan tersebut merupakan bilangan prima atau tidak.

Cara yang salah untuk mendapatkan bilangan prima adalah dengan membangkitkan suatu bilangan acak dan kemudian mencoba memfaktorkannya. Cara yang benar adalah membangkitkan bilangan acak dan kemudian mencoba apakah merupakan bilangan prima. Terdapat beberapa metode uji prima, pengujian menentukan apakah suatu bilangan termasuk bilangan prima atau bukan dengan tingkat keyakinan tertentu. Jadi kita tidak yakin seratus persen bahwa bilangan yang kita tes adalah betul-betul bilangan prima.

2.6.2 Algoritma Penguji Bilangan Prima Rabin-Miller

Salah satu metode yang paling banyak dipakai untuk menguji bilangan prima adalah metode Rabin-Miller. Algoritma Rabin-Miller menerima masukan berupa bilangan ganjil (n) yang lebih besar sama dengan 3 dan sebuah bilangan bulat (t) yang lebih kecil dari n. Bilangan t adalah parameter keyakinan, yaitu berapa kali pengujian dilakukan terhadap n untuk memastikan bahwa n adalah bilangan prima.

Adapun algoritma Rabin-Miller adalah sebagai berikut :

RABIN-MILLER (n,t)

1. $n-1 = 2^s r$; dengan s ganjil;
2. for $1 \leftarrow 1$ to t
3. do $a \leftarrow$ random; dengan $2 \leq a \leq n-2$
4. $y \leftarrow a^r \bmod n$
5. if $y \neq 1$ and $y \neq n-1$
6. do $j \leftarrow 1$
7. while $j \leq s-1$ and $y \neq n-1$
8. do $y \leftarrow y^2 \bmod n$
9. if y = 1 return false; bukan prima
10. $j \leftarrow j + 1$
11. if $y \neq n - 1$ return false; bukan prima

12. return true; prima

2.6.3 Greatest Common Divisor (GCD)

Dua buah bilangan dikatakan relatif prima jika mereka tidak memiliki faktor prima yang sama kecuali 1. Dengan perkataan lain, jika *Greatest Common Divisor* (GCD) dari a dan n adalah sama dengan satu, maka a dan n adalah relatif prima. Bentuk ini dapat ditulis seperti berikut :

gcd(a,n) = 1

Salah satu metode untuk menghitung GCD dari dua buah bilangan adalah dengan menggunakan algoritma *Euclidean* yang ditulis dalam bukunya '*Elements*' sekitar 300 tahun sebelum masehi. Algoritma ini bukan hasil rancangannya. Para ahli sejarah menyatakan bahwa algoritma mungkin 200 tahun lebih tua. Algoritma ini merupakan algoritma *nontrivial* tertua yang masih eksis di dunia. Knuth mendeskripsikannya dengan beberapa modifikasi modern dan dirancang dalam ilustrasi bahasa Pascal seperti berikut :

```
Begin
     P := A; Q := B;
     While Q ≠ 0 Do Begin
          R := P Mod Q;
          P := Q;
          Q := R;
     End
```

```
        GCD(A,B) := P
    End
```

2.6.4 Aritmatika Modular

Aritmatika modular merupakan operasi matematika yang banyak diimplementasikan pada metode kriptografi. Operasi modulo dua ini melibatkan bilangan 0 dan 1 saja sehingga identik dengan bit pada komputer.

Secara mendasar, a ≡ b (mod n) jika a = b + kn untuk beberapa bilangan *integer* k. Jika a merupakan non-negatif dan b di antara 0 sampai n, dimana b merupakan sisa hasil bagi dari a ketika dibagi dengan n. Dengan kata lain, b adalah hasil dari a modulo n. Kadang-kadang a juga disebut kongruen dengan b modulo n.

Operasi a mod n menghasilkan sisa bagi dari a, dimana sisa bagi merupakan beberapa bilangan integer dari 0 sampai n – 1. Operasi ini merupakan reduksi modular. Sebagai contoh, 5 mod 3 = 2.

Kriptografi banyak menggunakan perhitungan mod n, karena perhitungan logaritma diskrit dan akar kuadrat mod n merupakan persoalan sulit. Aritmatika modular lebih mudah dikerjakan pada komputer, karena operasi ini membatasi hasil yang didapatkan sehingga hasil operasi dari k-*bit* modulo n tidak akan lebih dari n.

Metode yang dapat digunakan untuk menghitung a^m mod n adalah algoritma *fast exponentiation*. Operasi a^m mod n dapat dihitung dengan jumlah perkalian paling banyak $2.(^2\log (m + 1))$ dengan algoritma sebagai berikut :

1. Konversikan m ke biner : $b_k\ b_{k-1} \ldots b_1\ b_0$
2. {Initialisasi} I := k; Product := a

3. While I >= 0 Do

    a. If b_I = 0 Then Product := Product^2

       Else Product := Product * a;

    b. Product := Product mod n; I := I – 1;

2.6.5 Inverse Aritmatika Modular

Inverse merupakan operasi kebalikan. *Inverse* perkalian dari 4 adalah ¼ karena 4 * ¼ = 1. Dalam modulo, persoalan ini agak rumit. Misalkan

$$4 * x \equiv 1 \pmod 7$$

Atau bentuk penulisan ekivalen lainnya dapat dinyatakan dengan :

$$4x = 7k + 1$$

Dengan x dan k adalah bilangan *integer*.

Persoalannya adalah untuk mencari x sehingga,

$$1 = (a * x) \bmod n$$

Atau dapat ditulis seperti berikut :

$$a^{-1} \equiv x \pmod n$$

Persoalan *inverse* modular agak sulit untuk diselesaikan. Kadang-kadang dapat memiliki penyelesaian, dan kadang-kadang tidak. Sebagai contoh, *inverse* modular dari 5 modulo 14 adalah 3, sedangkan 2 mod 14 tidak memiliki *inverse* modular.

Secara umum, $a^{-1} \equiv x \pmod{n}$ memiliki sebuah solusi unik jika a dan n adalah relatif prima. Jika a dan n bukan relatif prima, maka $a^{-1} \equiv x \pmod{n}$ tidak memiliki solusi. Jika n adalah bilangan prima maka setiap bilangan dari 1 sampai (n – 1) adalah relatif prima dengan n dan memiliki tepat satu *inverse modulo* n dalam *range* tersebut.

Untuk menghitung *inverse modular* dari $a^{-1} \equiv x \pmod{n}$ dapat digunakan algoritma *extended Euclidean* yang dapat dijabarkan seperti berikut :

1. Bentuk Array A[3x2] dimana A[1,1] = n dan A[1,2] = a
2. Isikan A[2,1] .. A[3,2] dengan matriks Identitas.
3. Hitung bilangan bulat m dengan kondisi m * A[1,2] ≤ A[1,1]; usahakan m maksimum.
4. Hitung nX = A[1,1] - m * A[1,2].
5. Ubah nilai A[1,1] = A[1,2] dan A[1,2] = nX.
6. Lakukan langkah 4 dan 5 untuk baris kedua dan ketiga dari array A.
7. Ulang langkah 3 sampai 5 hingga elemen terakhir dari baris 1 = 0.
8. Jika A[3,1] ≥ 0 maka x = A[3,1] sebaliknya x = A[3,1] + n.

2.6.6 Logaritma Diskrit

Eksponensial modular merupakan salah satu fungsi yang sering dipakai dalam kriptografi. Fungsi ini dapat dirumuskan seperti berikut :

$$a^x \bmod n$$

Masalah invers dari eksponensial modular adalah mencari logaritma diskrit dari suatu nilai. Ini merupakan masalah yang sulit. Misalkan diberikan suatu persoalan untuk mencari nilai x dimana $a^x \equiv b \pmod n$.

Sebagai contoh, misalkan diketahui sebuah persamaan $3^x = 15 \pmod{17}$, maka nilai x adalah 6. Ini merupakan salah satu contoh logaritma diskrit yang memiliki penyelesaian. Tidak semua logaritma diskrit mempunyai penyelesaian. Contohnya adalah seperti berikut : $3^x \equiv 7 \pmod{13}$.

2.7 Authentication dan Digital Signature

2.7.1 Authentication

Otentikasi (*authentication*) merupakan sebuah istilah yang digunakan dalam pengertian yang luas. Secara tersirat kata tersebut mempunyai arti lebih dari sekedar menyampaikan ide yaitu bahwa alat tersebut telah menyediakan jaminan bahwa informasi tidak dimanipulasi oleh pihak yang tidak mempunyai wewenang. Otentikasi bersifat spesifik dalam topik keamanan yang berusaha dicapai. Contohnya meliputi pengendalian akses, otentikasi *entity*, otentikasi pesan, integritas data, *non-repudiation*, dan otentikasi kunci.

Otentikasi merupakan salah satu hal yang paling penting dalam keamanan informasi. Hingga pertengahan tahun 1970-an, dipercaya bahwa kerahasiaan dan otentikasi terhubung secara erat. Dengan penemuan dari fungsi-fungsi *hash* dan *digital signatures*, disadari bahwa kerahasiaan dan otentikasi sebenarnya adalah

masalah yang terpisah dan independen. Awalnya tidak kelihatan penting untuk memisahkan keduanya tetapi terdapat situasi dimana hal tersebut tidak hanya berguna tetapi juga penting. Contohnya, jika terdapat komunikasi dua pihak antara Anto dan Badu yang berlangsung, dimana Anto berada di suatu negara dan Badu di negara lainnya, Negara tuan rumah mungkin tidak memperbolehkan kerahasiaan dalam saluran komunikasi karena satu atau kedua negara mungkin ingin memonitor semua komunikasi. Namun, Anto dan Badu ingin meyakinkan identitas masing-masing, dan juga integritas serta keaslian dari informasi yang mereka kirim dan mereka terima.

Skenario diatas menggambarkan beberapa aspek otentikasi yang independen. Jika Anto dan Badu ingin meyakinkan identitas masing-masing, terdapat dua kemungkinan yang dapat dipertimbangkan.

1. Anto dan Badu dapat berkomunikasi tanpa penundaan waktu. Berarti mereka berkomunikasi secara *real time.*
2. Anto atau Badu dapat saling menukar pesan dengan penundaan waktu. Berarti pesan mungkin di-*routing* melalui jaringan yang berbeda, disimpan, dan disampaikan pada beberapa saat kemudian.

Dalam kemungkinan pertama, Anto dan Badu akan memverifikasi identitas masing-masing secara *real time*. Hal ini dapat dicapai oleh Anto dengan mengirimkan beberapa *challenge* kepada Badu dimana hanya Badu yang dapat meresponnya secara tepat. Badu dapat melakukan tindakan yang sama untuk mengidentifikasi Anto. Jenis

otentikasi ini secara umum dikenal sebagai otentikasi *entity* atau secara sederhana disebut identifikasi.

Dalam kemungkinan kedua, tidaklah tepat untuk mengirimkan *challenge* dan menunggu respon, dan selain itu jalur komunikasi mungkin hanya tersedia pada satu arah. Teknik yang berbeda sekarang diperlukan untuk meng-otentikasi keaslian pesan. Bentuk otentikasi ini disebut otentikasi keaslian data (*data origin authentication*).

2.7.2 Digital Signature

Tanda tangan digital adalah suatu mekanisme otentikasi yang memungkinkan pembuat pesan menambahkan sebuah kode yang bertindak sebagai tanda tangannya. Tanda tangan tersebut menjamin integritas dan sumber dari sebuah pesan.

Penandatanganan digital terhadap suatu dokumen adalah sidik jari dari dokumen tersebut beserta *timestamp*-nya dienkripsi dengan menggunakan kunci privat pihak yang menandatangani. Tanda tangan digital memanfaatkan fungsi *hash* satu arah untuk menjamin bahwa tanda tangan itu hanya berlaku untuk dokumen yang bersangkutan saja. Keabsahan tanda tangan digital itu dapat diperiksa oleh pihak yang menerima pesan.

Sifat yang diinginkan dari tanda tangan digital diantaranya adalah :

1 Tanda tangan itu asli (otentik), tidak mudah ditulis / ditiru oleh orang lain. Pesan dan tanda tangan pesan tersebut juga dapat menjadi barang bukti, sehingga penandatangan tak bisa menyangkal bahwa dulu ia pernah menandatanganinya.
2 Tanda tangan itu hanya sah untuk dokumen (pesan) itu saja. Tanda tangan itu tidak bisa dipindahkan dari suatu dokumen ke dokumen lainnya. Ini juga berarti bahwa jika dokumen itu diubah, maka tanda tangan digital dari pesan tersebut tidak lagi sah.
3 Tanda tangan itu dapat diperiksa dengan mudah.
4 Tanda tangan itu dapat diperiksa oleh pihak-pihak yang belum pernah bertemu dengan penandatangan.
5 Tanda tangan itu juga sah untuk *copy* dari dokumen yang sama persis.

2.8 Schnorr Authentication and Digital Signature Scheme

Claus Schnorr *Authentication* dan *Digital Signature scheme* mengambil sekuritas dari permasalahan menghitung logaritma diskrit. Skema ini juga menggunakan bilangan prima dan perpangkatan modulo dalam proses pembentukan kuncinya. Tingkat kesulitan untuk memecahkan algoritma ini adalah sekitar 2^t, dimana nilai t ini dapat ditentukan sendiri.

Skema ini dipatenkan di Amerika Serikat dan akan berakhir pada tanggal 19 Februari 2008. Skema otentikasi dapat dimodifikasi menjadi skema tanda tanda

digital (*digital signature scheme*). Proses pembentukan kunci privat dan publiknya sama seperti skema otentikasi, hanya saja pada skema tanda tangan digital ditambahkan sebuah fungsi *hash*.

2.8.1 Key Generation

Pembentukan kunci (*key generation*) berfungsi untuk menghasilkan kunci privat dan kunci publik yang akan digunakan dalam skema otentikasi dan skema tanda tangan digital. Proses pembentukan kunci adalah sebagai berikut:

1. Pilih 2 buah bilangan prima p dan q, dan sebuah nilai a yang memenuhi syarat berikut:
    a. Bilangan q adalah faktor prima dari (p-1). Ini berarti GCD(q, p-1) tidak boleh bernilai 1.
    b. Harus memenuhi operasi: a^q / 1 (mod p).
2. Pilih sebuah nilai s, dimana s < q. (s adalah kunci privat)
3. Hitung nilai v dengan rumus berikut:

    $v = a^{-s} \bmod p$

    (v adalah kunci publik).

Untuk lebih memahami proses pembentukan kunci, perhatikan contoh berikut:

1. Dipilih dua buah bilangan prima p = 816961, q = 23 dan a = 40433 yang memenuhi kedua syarat yang telah dijelaskan pada poin pertama proses pembentukan kunci.

2. Pilih sebuah nilai s, s = 15 (s lebih kecil dari q yang bernilai 23).
3. Hitung: $v = a^{-s} \bmod p$

   $v = (a^{-1})^{s} \bmod p$

   $v = ((a^{-1} \bmod p)^{s} \bmod p)$

   $v = ((40433^{-1} \bmod 816961)^{15} \bmod 816961)$

   $v = 578423$
4. Kunci privat: s = 15.
5. Kunci publik: v = 578423.

2.8.2 Protokol Otentikasi (Authentication)

Misalkan, Evi dan Andry sedang berkomunikasi di dua tempat berbeda melalui aplikasi *messenger* (*chatting*). Permasalahan yang muncul adalah bagaimana Evi dan Andry dapat mengidentifikasi identitas satu sama lain untuk memastikan bahwa orang yang sedang diajak komunikasi adalah benar-benar orang yang dimaksud. Dengan skema *Schnorr Authentication*, Evi dan Andry dapat memverifikasi identitas masing-masing secara *real time*. Hal ini dapat dilakukan dengan mengirimkan beberapa nilai matematis dimana hanya mereka berdua yang dapat meresponnya secara tepat.

Protokol dari Schnorr *Authentication* dapat dijabarkan sebagai berikut :

1. Evi mengambil sebuah bilangan acak r, yang lebih kecil daripada q dan menghitung $x = a^{r} \bmod p$.

2. Evi mengirimkan x kepada Andry.
3. Andry mengirimkan sebuah bilangan acak e yang berada di antara 0 dan $2^t - 1$.
4. Evi menghitung y = (r + se) mod q dan mengirimkan y kepada Andry.
5. Andry memverifikasi bahwa $x = a^y v^e \bmod p$.

Dengan menggunakan contoh proses pembentukan kunci pada subbab 2.7.1, perhatikan contoh skema otentikasi di bawah ini:

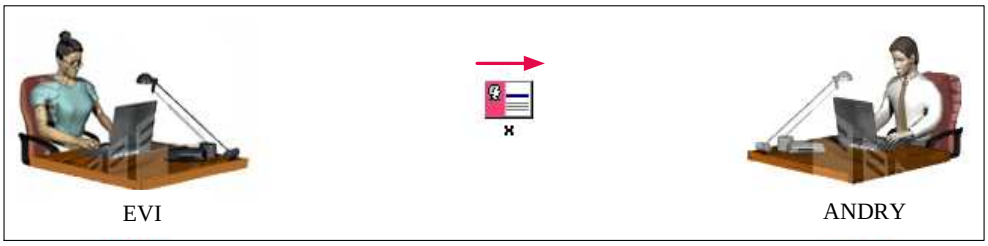
1. Evi memilih nilai r = 20 (r lebih kecil dari q yang bernilai 23) dan menghitung:

   $x = a^r \bmod p$

   $x = 40433^{20} \bmod 816961$

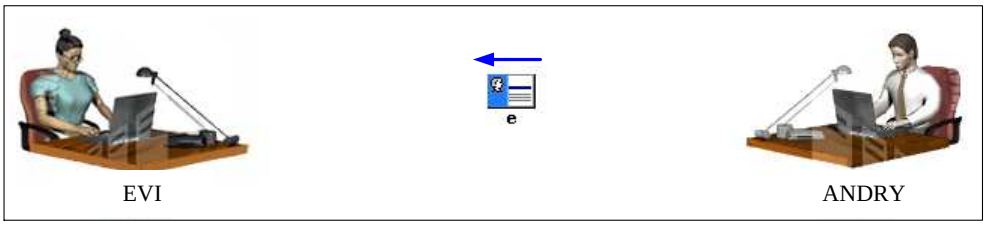
   x = 738635

2. Evi mengirimkan x = 738635 kepada Andry.



Gambar 2.3 Evi mengirimkan x = 738635 kepada Andry

3. Andry mengirimkan sebuah nilai e = 59195082.



Gambar 2.4 Andry mengirimkan e = 59195082 kepada Evi

4. Evi menghitung:

$$y = (r + se) \bmod q$$

$$y = (20 + 15 \,.\, 59195082) \bmod 23$$

$$y = 3$$

Evi mengirimkan y = 3 kepada Andry.

5. Andry memverifikasi bahwa:

$$x = ((a^y).(v^e)) \bmod p$$

$$x = ((a^y) \bmod p \,.\, (v^e) \bmod p) \bmod p$$

$$x = (40433^3 \bmod 816961) \,.\, (578423^{59195082} \bmod 816961) \bmod 816961$$

$$x = (346483 \,.\, 402127) \bmod 816961$$

738635 = 738635 (***True***), Proses otentikasi berhasil.

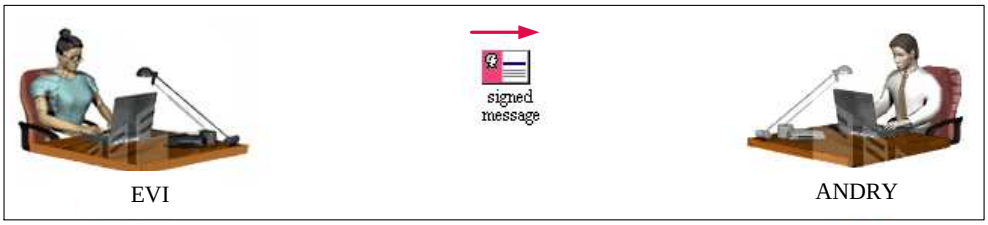
2.8.3 Protokol Tanda Tangan Digital (Digital Signature)

Protokol ini dapat dianalogikan seperti pengiriman surat yang menggunakan tanda tangan biasa. Andry memang dapat merasa yakin bahwa tanda tangan Evi adalah asli, tetapi bagaimana Andry dapat memastikan bahwa isi pesan adalah asli dan tidak diganti oleh pihak ketiga. Untuk memastikan keaslian dan keutuhan pesan, Evi membubuhkan tanda tangan digitalnya pada pesan tersebut. Tanda tangan digital itu tidak mudah ditiru orang lain, hanya sah untuk pesan itu saja dan dapat diperiksa oleh Andry.

Protokol dari Schnorr *Digital Signature Scheme* dapat dijabarkan sebagai berikut:

1. Evi mengambil sebuah bilangan acak, r, yang lebih kecil daripada q dan menghitung $x = a^r \bmod p$.
2. Evi menggabungkan M dan x, dan menghitung *hash value* dari hasil penggabungan.

   e = H(M, x).
3. Evi menghitung y = (r + se) mod q. Tandatangannya adalah e dan y. Evi mengirimkannya kepada Andry.



Gambar 2.5 Evi mengirimkan pesan dan tanda tangan kepada Andry

4. Andry menghitung $x' = a^y v^e \bmod p$ dan memverifikasi nilai e dengan nilai *hash* dari penggabungan dari M dan x'.

   e = H(M,x').

   Jika nilai e sesuai dengan nilai *hash*, maka tandatangan tersebut dianggap *valid.*

Dengan menggunakan contoh proses pembentukan kunci pada subbab 2.7.1, perhatikan contoh skema tanda tangan digital di bawah ini:

Misalkan nilai M (pesan) = 100

1. Evi memilih nilai r = 17 (r lebih kecil dari q yang bernilai 23) dan menghitung:

$x = a^r \bmod p$

$x = 40433^{17} \bmod 816961$

$x = 402127$

2. Evi menggabungkan M dan x, dan menghitung *hash value* dari hasil penggabungan.

   Hasil penggabungan M (100) dan x (402127) = 100402127

   Misalkan fungsi hash H(z) = (z * 2) + 1,

   maka e = H(M, x) = (100402127 * 2) + 1 = 200804255.

3. Evi menghitung:

   $y = (r + se) \bmod q$

   $y = (17 + 15 \cdot 200804255) \bmod 23$

   $y = 11$

   Tanda tangan digital adalah e (200804255) dan y (11). Evi mengirimkannya kepada Andry.

6. Andry menghitung:

   $x' = ((a^y).(v^e)) \bmod p$

   $x' = ((a^y) \bmod p \cdot (v^e) \bmod p) \bmod p$

   $x' = (40433^{11} \bmod 816961) \cdot (578423^{200804255} \bmod 816961) \bmod 816961$

   $x' = (131633 \cdot 501222) \bmod 816961$

   $x' = 402127$

Hasil penggabungan M (100) dan x’ (402127) = 100402127

Andry memverifikasi: e = H(M, x’)

200804255 = (100402127 * 2) + 1

200804255 = 200804255 (***True***)

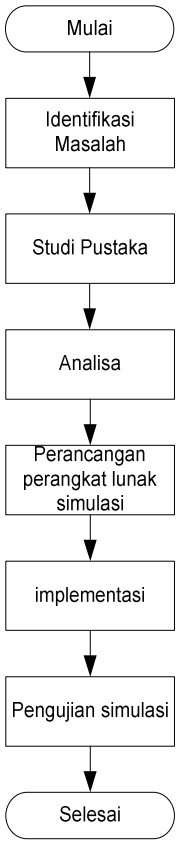
Verifikasi tanda tangan sesuai.

BAB III

METODOLOGI PENELITIAN

Pada bab ini akan dipaparkan tahapan-tahapan yang dilakukan dalam penelitian agar hasil yang dicapai tidak menyimpang dari tujuan yang telah dilakukan sebelumnya.

Metodologi penelitian yang digunakan dalam penyusunan tugas akhir ini akan melalui beberapa tahapan yang membentuk sebuah alur yang sistematis Tahap-tahap yang akan adalah sebagai berikut :



Gambar 3.1. Tahapan Pelaksanaan Tugas Akhir

3.1.1. Identifikasi Masalah

Berdasarkan latar belakang yang telah dipaparkan sebelumnya pada bab I, maka permasalahan yang diangkat pada tugas akhir ini adalah Bagaimana mensimulasikan prosedur kerja dari Metode Schnorr Authentication dan Digital Signature Schem Dalam Autentikasi Data.

3.2. Studi Pustaka *(Library Research)*

Studi kepustakaan atau kajian pustaka dilakukan untuk mencari dan mempelajari serta mendalami informasi tentang Metode Schnorr Authentication dan Digital Signature Schem Dalam Autentikasi Data. Sumber kepustakaan diambil karya ilmiah yang berasal dari buku-buku maupun internet. Karya ilmiah yang dimaksud adalah berupa tulisan ilmiah yang berbentuk artikel, prosiding, buku, *e-book* (buku elektronik), dan lainnya. Studi pustaka bertujuan untuk mensimulasikan prosedur kerja dari metode Schnorr Autentication dan Digital Signature Scheme dalam Autentikasi Data.

3.3. Analisa

Setelah melakukan kajian pustaka, langkah berikutnya adalah menganalisa hasil dari teori pendukung tersebut. Mempelajari prosedur Metode Schnorr Authentication dan Digital Signature Schem Dalam Autentikasi Data. Menganalisa kebutuhan-kebutuhan simulasi dan merancang suatu Aplikasi Simulasi. Hasil analisa kemudian dilanjutkan dengan perancangan meliputi perancangan antar muka, tabel dan perancangan lingkungan pengembangan.

3.4. Perancangan Aplikasi Simulasi

Dari hasil analisa, dibuatlah suatu perancangan dari Aplikasi simulasi ini menggunakan bahasa pemrograman *Microsoft Visual Basic 6.0* dengan rancangan sebagai berikut:

a. Perancangan *interface* simulasi.

b. Perancangan parameter inputan simulasi.

Setiap perancangan *interface* dari Metode Autentication Schnorr dan Digital Signature Scheme, diberi parameter *input*-an di antaranya:

1. Key Generation
    a. Variable p,q (bilangan prima)
    b. Variable nilai a (nilai acak)
2. Authentication.
    a. Variable r (nilai acak)
    b. Variable e (nilai acak)
3. Digital Signature.
    a. Pesan

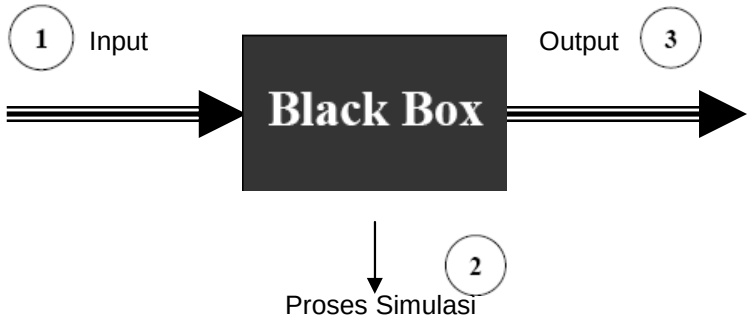
3.5. Implementasi

Hasil rancangan yang telah dilakukan pada tahap desain kemudian diubah menjadi bentuk yang dimengerti oleh mesin menggunakan bahasa pemrograman agar bisa diimplementasikan oleh pengguna. Tahapan ini mencakup penulisan kode program. Penulisan kode program dilakukan dengan menggunakan *Microsoft Virtual Basic 6.0* .

3.6. Pengujian Simulasi

Sebelum Aplikasi dapat digunakan, maka harus dilakukan pengujian terlebih dahulu. Pengujian dilakukan bertujuan untuk memberikan gambaran secara umum dan jelas prosedur yang dilakukan untuk mensimulasikan proses kerja Metode Authentication Schnorr dan Digital Signature Scheme.

Dalam pengujian untuk perangkat simulasi ini menggunakan metode uji *black box*. Pengujian *black-box* berfokus pada persyaratan fungsional Aplikasi. Pengujian ini memungkinkan analis sistem memperoleh kumpulan kondisi input yang akan mengerjakan seluruh keperluan fungsional program.



Gambar 3.2. Metode uji blackbox

Keterangan gambar:

1. *Input*-an parameter dari Aplikasi simulasi
2. Proses simulasi
3. *Output* dari simulasi

BAB IV

ANALISIS DAN PERANCANGAN

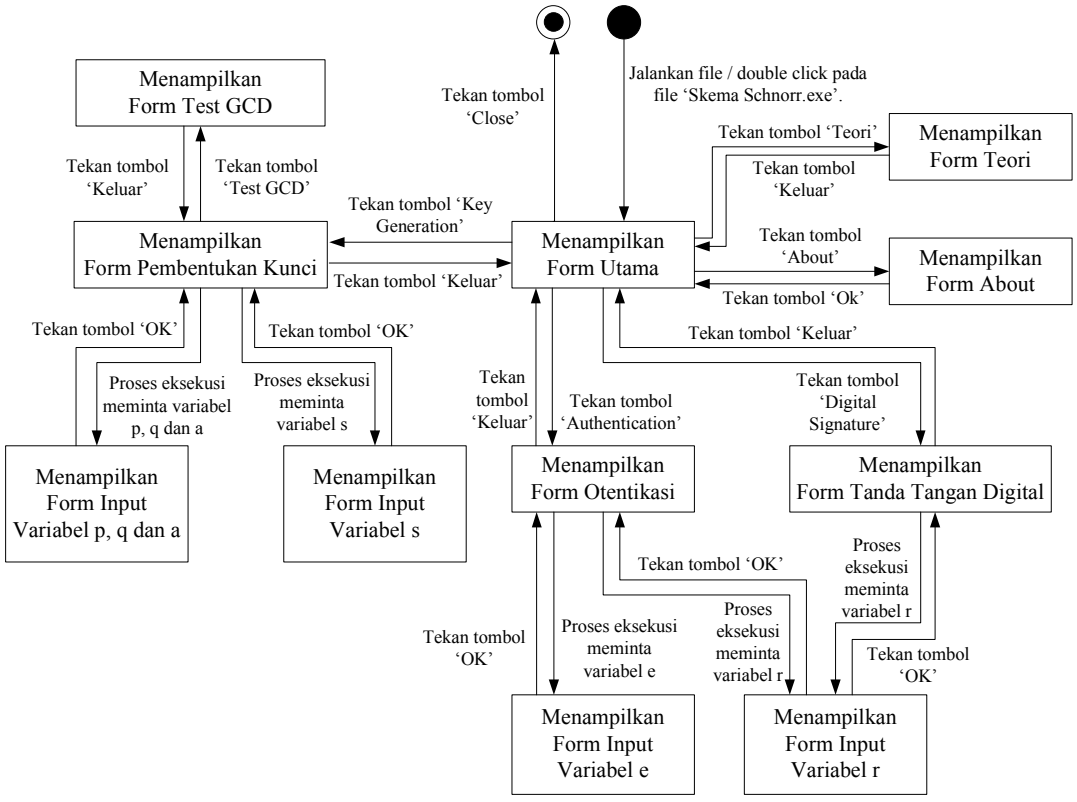
4.1 Analisis Perangkat Lunak

Pada bab ini akan mencakup alur kerja perangkat lunak, perancangan tampilan animasi, proses pembentukan kunci, proses kerja skema otentikasi (*authentication*), proses kerja skema tanda tangan digital (*digital signature scheme*) dan penjelasan terhadap *form-form* yang terdapat di dalam perangkat lunak.

4.1.1 Alur Kerja Perangkat Lunak

Perangkat lunak simulasi ini diawali dari *form* Utama yang berisi pilihan atau *link* ke *form* Pembentukan Kunci, *form* Skema Otentikasi, *form* Skema Tanda Tangan Digital, *form* Teori dan *form* About. *Form* Pembentukan Kunci mensimulasikan dan menjelaskan proses kerja pembentukan kunci. Dalam proses ini, dihasilkan dua buah kunci, yaitu s (kunci privat) dan v (kunci publik) yang akan digunakan pada proses otentikasi dan tanda tangan digital. *Form* Skema Otentikasi mensimulasikan dan menjelaskan prosedur kerja dari skema otentikasi. Dalam skema otentikasi, algoritma proses otentikasi dijelaskan secara bertahap. *Form* Skema Tanda Tangan Digital mensimulasikan dan menjelaskan prosedur kerja dari skema tanda tangan digital. Dalam skema ini, pesan yang akan dikirimkan, dibubuhi dengan tanda tangan digital terlebih dahulu. Tanda tangan digital akan dikirimkan bersamaan dengan pesan sebagai jaminan bahwa pesan tersebut adalah asli dan utuh.

Semua nilai variabel pada proses pembentukan kunci, otentikasi dan tanda tangan digital dapat di-*input* sendiri ataupun dihasilkan secara acak oleh komputer. Perangkat lunak juga memiliki *form* Teori, yang menyediakan teori-teori lebih lanjut mengenai skema Schnorr dan *form About* berfungsi untuk menampilkan informasi yang mengenai pembuat perangkat lunak. Alur kerja perangkat lunak digambarkan di dalam bentuk *State Transition Diagram* (STD) seperti terlihat pada gambar 3.1 berikut.



Gambar 3.1 *State Transition Diagram* (STD) Perangkat Lunak

4.1.2 Perancangan Tampilan Animasi

Proses animasi di dalam perangkat lunak menggunakan komponen *visual basic* yang dinamakan *timer*. *Timer* adalah objek yang akan mengerjakan prosedur yang dituliskan di dalamnya satu kali setiap interval waktu yang diberikan. Misalkan, interval waktu = 500 (dalam milidetik), maka setiap 0,5 detik *timer* akan mengeksekusi prosedur yang diberikan kepadanya.

Dalam perangkat lunak, prosedur yang diberikan pada *timer* adalah perintah untuk mengganti gambar dari satu keadaan ke keadaan berikutnya. Proses inilah yang membentuk tampilan animasi di dalam perangkat lunak. Berikut adalah animasi yang terdapat di dalam perangkat lunak:

1. Animasi Evi sedang mengetik

   Dalam perangkat lunak, Evi adalah pihak pertama yang diasumsikan melakukan proses pembentukan kunci, pihak yang di-otentikasi dan pihak yang diverifikasi tanda tangan digitalnya. Animasi ini akan dijalankan ketika Evi sedang memilih nilai variabel, menghitung nilai variabel atau sedang melakukan pengiriman. Gambar Evi sedang mengetik dapat dilihat pada gambar 3.2 berikut.

Gambar 3.2 Gambar Evi sedang mengetik

2. Animasi Andry sedang mengetik

Andry berfungsi sebagai pihak kedua yang diasumsikan menerima kunci publik, pihak yang melakukan proses otentikasi dan pihak yang memverifikasi tanda tangan digital dari Evi. Animasi ini akan dijalankan ketika Andry sedang memilih nilai variabel, menghitung nilai variabel atau sedang melakukan pengiriman. Gambar Andry sedang mengetik dapat dilihat pada gambar 3.3 berikut.

Gambar 3.3 Gambar Andry sedang mengetik

3. Proses pengiriman.

   Animasi proses pengiriman dijalankan ketika Evi mengirimkan variabel atau pesan kepada Andry dan sebaliknya. Animasi proses pengiriman menggunakan sebuah gambar surat dan menggerakkannya dari pengirim ke penerima. Gambar surat yang mewakili objek yang dikirimkan dapat dilihat pada gambar 3.4 berikut.

(a) (b)

Gambar 3.4 (a) Gambar surat mewakili objek yang akan dikirimkan oleh Evi, (b) Gambar surat mewakili objek yang akan dikirimkan oleh Andry

4.1.3 Proses Pembentukan Kunci

Proses pembentukan kunci adalah proses yang harus dijalankan sebelum menjalankan proses otentikasi ataupun proses tanda tangan digital. Proses pembentukan kunci menghasilkan dua buah kunci, yaitu s (kunci privat) dan v (kunci publik) yang akan digunakan pada proses otentikasi dan tanda tangan digital. Kunci privat adalah kunci yang dipegang oleh pihak pertama (Evi / pembuat skema), sedangkan kunci publik diberikan kepada pihak kedua (Andry).

Algoritma proses pembentukan kunci yang dijelaskan baris per baris di dalam perangkat lunak adalah sebagai berikut:

1. Pilih 2 buah bilangan prima p dan q, dan sebuah nilai a, dimana GCD(q, p-1) <> 1 dan (a^q) mod p = 1.
2. Pilih sebuah nilai s, dimana s < q. (s adalah kunci privat)
3. Hitung nilai v dengan rumus berikut:

   v = a^(-s) mod p

   (v adalah kunci publik).

4.1.4 Proses Kerja Skema Otentikasi

Skema otentikasi (authentication) merupakan protokol yang dijalankan apabila salah satu pihak dalam saluran komunikasi ingin memverifikasi keaslian (otentikasi) sumbernya. Algoritma proses kerja skema otentikasi yang dijelaskan baris per baris di dalam perangkat lunak adalah sebagai berikut:

1. Evi memilih sebuah nilai r (r < q).

2. Evi menghitung:

   x = a^r mod p

   Evi mengirim x kepada Andry.

3. Andry memilih sebuah nilai e (e diantara 0 sampai (2^t-1)) dan mengirim e kepada Evi.

4. Evi menghitung:

   y = (r + se) mod q

   dan mengirim y kepada Andry.

5. Andry melakukan verifikasi berikut:

   x = ((a^y).(v^e)) mod p

   Jika nilai x sesuai, maka verifikasi dan otentikasi berhasil.

4.1.5 Proses Kerja Skema Tanda Tangan Digital

Skema tanda tangan digital (*digital signature*) merupakan protokol yang dijalankan untuk dapat memverifikasi keaslian dan keutuhan dari pesan yang akan dikirimkan dalam suatu saluran komunikasi. Algoritma proses kerja skema tanda tangan digital yang dijelaskan baris per baris di dalam perangkat lunak adalah sebagai berikut:

1. Evi memilih sebuah nilai r (r < q) dan menghitung:

   x = a^r mod p

2. Evi menggabungkan (*concatenate*) M dan x dan menghitung nilai hash dari hasil penggabungan tersebut.

   e = H(M, x)

3. Evi menghitung:

   y = (r + se) mod q

   Tanda tangan adalah e dan y. Evi mengirimkan tanda tangan bersama pesan.

4. Andry menghitung:

   x’ = ((a^y).(v^e)) mod p

5. Andry menggabungkan (*concatenate*) M dan x’ dan melakukan proses verifikasi berikut:

   e = H(M, x’)

4.2 Perancangan

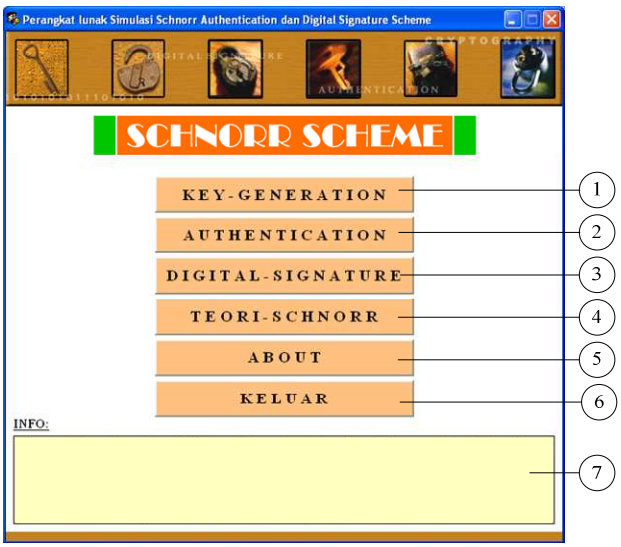
Perancangan perangkat lunak simulasi Schnorr *Authentication* dan *Digital Signature Scheme* menggunakan bahasa pemograman Microsoft Visual Basic 6.0. Perangkat lunak ini memiliki beberapa buah *form*, antara lain:

1. *Form* Utama.
2. *Form* Pembentukan Kunci.
3. *Form* Skema Otentikasi.
4. *Form* Skema Tanda Tangan Digital.
5. *Form Input* Variabel p, q dan a.

6. *Form Input* Variabel s.
7. *Form Input* Variabel r.
8. *Form Input* Variabel e.
9. *Form Test* GCD.
10. *Form* Teori.
11. *Form About.*

4.2.1 Form Utama

Form ini berfungsi sebagai *form* utama dari perangkat lunak. *User* dapat memilih untuk melihat proses pembentukan kunci, proses otentikasi, proses tanda tangan digital, teori yang berhubungan dengan skema Schnorr atau *form About*.



Gambar 3.5 Rancangan *Form* Utama

Keterangan:

1 : tombol ’Key Generation’, untuk menampilkan *form* Pembentukan Kunci.

2 : tombol ’Authentication’, untuk menampilkan *form* Skema Otentikasi.

3 : tombol ’Digital Signature’, untuk menampilkan *form* Skema Tanda Tangan Digital.

4 : tombol ’Teori Schnorr’, untuk menampilkan *form* Teori.

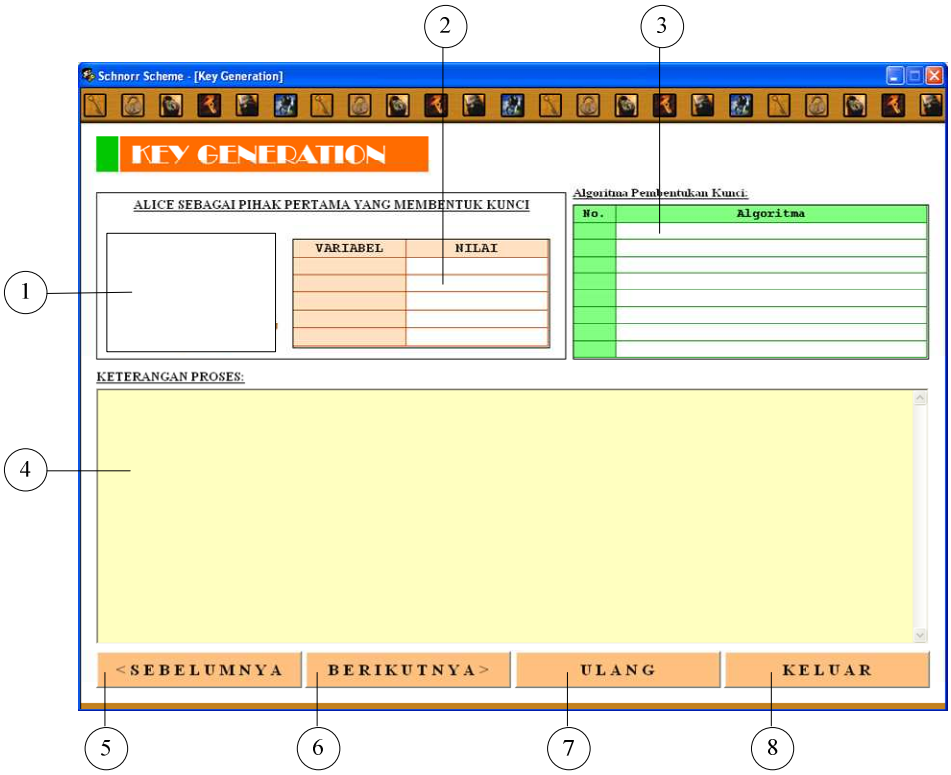
5 : tombol ’About’, untuk menampilkan *form About*.

6 : tombol ’Keluar’, untuk menutup *form*.

7 : label ’Info’, untuk menampilkan informasi dari tombol yang dipilih.

4.2.2 Form Pembentukan Kunci

Fungsi dari *form* ini adalah untuk menampilkan proses pembentukan kunci pada skema Schnorr.



Gambar 3.6 Rancangan *Form* Pembentukan Kunci

Keterangan:

1 : *picturebox*, sebagai daerah animasi.

2 : tabel variabel, untuk menampilkan isi dari variabel.

3 : tabel algoritma, untuk menampilkan algoritma dari proses.

4 : *textbox* ’Keterangan Proses’, sebagai tempat menampilkan keterangan proses.

5 : tombol ’Sebelumnya’, untuk kembali ke langkah atau algoritma sebelumnya.

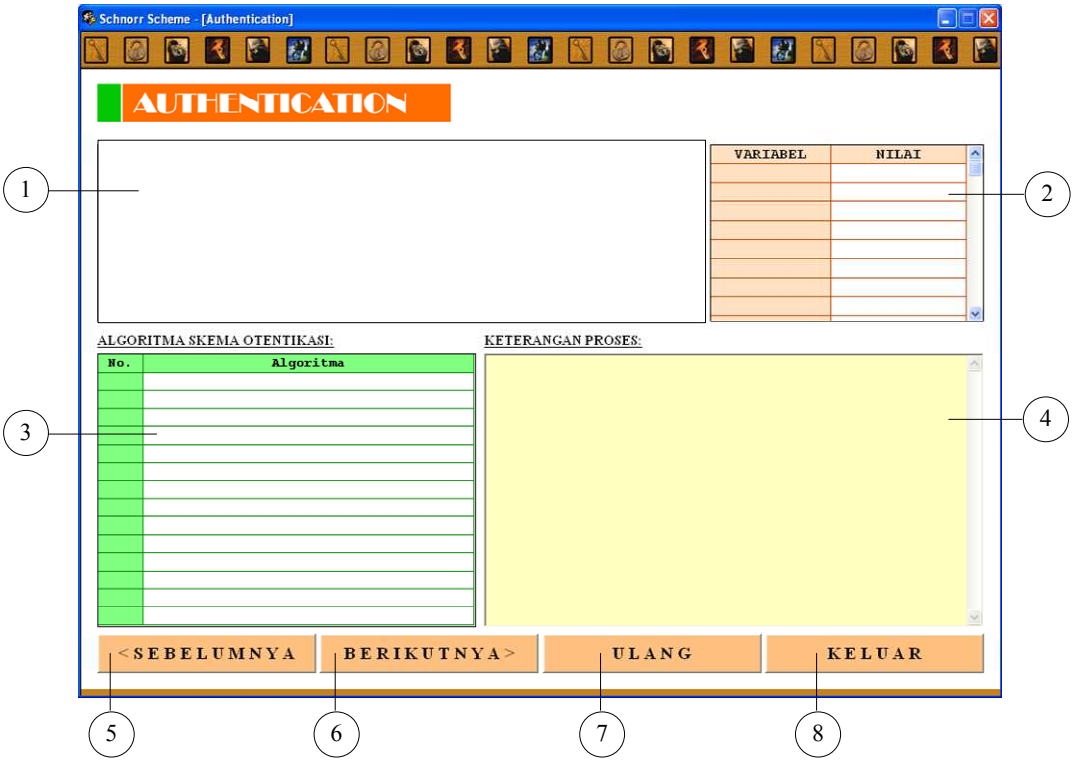
6 : tombol ’Berikutnya’, untuk menampilkan langkah atau algoritma berikutnya.

7 : tombol ’Ulang’, untuk mengulangi kembali algoritma dari awal.

8 : tombol ’Keluar’, untuk menutup *form*.

4.2.3 Form Skema Otentikasi

Fungsi dari *form* ini adalah untuk menampilkan proses otentikasi (*authentication*) pada skema Schnorr.



Gambar 3.7 Rancangan *Form* Skema Otentikasi

Keterangan:

1 : *picturebox*, sebagai daerah animasi.

2 : tabel variabel, untuk menampilkan isi dari variabel.

3 : tabel algoritma, untuk menampilkan algoritma dari proses.

4 : *textbox* ’Keterangan Proses’, sebagai tempat menampilkan keterangan proses.

5 : tombol ’Sebelumnya’, untuk kembali ke langkah atau algoritma sebelumnya.

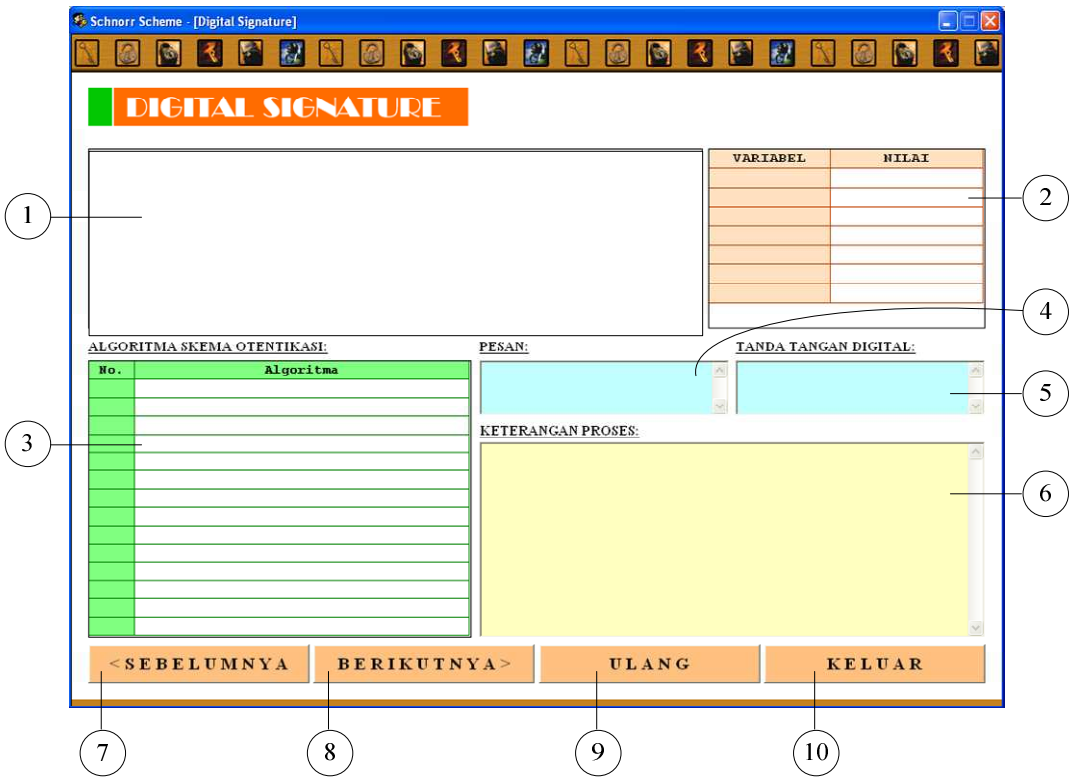
6 : tombol ’Berikutnya’, untuk menampilkan langkah atau algoritma berikutnya.

7 : tombol ’Ulang’, untuk mengulangi kembali algoritma dari awal.

8 : tombol ’Keluar’, untuk menutup *form*.

4.2.4 Form Skema Tanda Tangan Digital

Fungsi dari *form* ini adalah untuk menampilkan proses tanda tangan digital (*digital signature*) pada skema Schnorr.



Gambar 3.8 Rancangan *Form* Skema Tanda Tangan Digital

Keterangan:

1 : *picturebox*, sebagai daerah animasi.

2 : tabel variabel, untuk menampilkan isi dari variabel.

3 : tabel algoritma, untuk menampilkan algoritma dari proses.

4 : *textbox* ’Pesan’, sebagai tempat memasukkan pesan.

5 : *textbox* ’Tanda Tangan Digital’, untuk menampilkan tanda tangan digital.

6 : *textbox* ’Keterangan Proses’, sebagai tempat menampilkan keterangan proses.

7 : tombol ’Sebelumnya’, untuk kembali ke langkah atau algoritma sebelumnya.

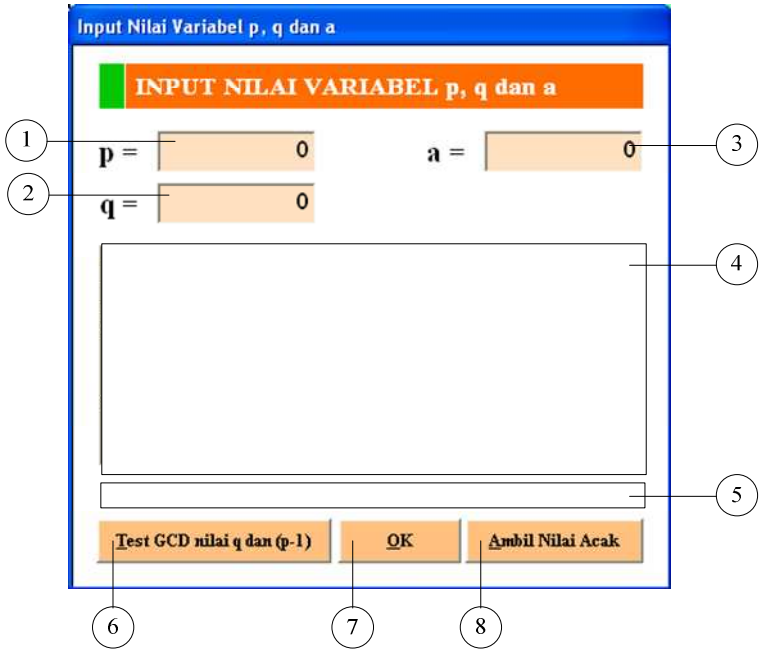
8 : tombol ’Berikutnya’, untuk menampilkan langkah atau algoritma berikutnya.

9 : tombol ’Ulang’, untuk mengulangi kembali algoritma dari awal.

10 : tombol ’Keluar’, untuk menutup *form*.

4.2.5 Form Input Variabel p, q dan a

Form ini berfungsi sebagai tempat memasukkan variabel p, q dan a pada proses pembentukan kunci. Selain dapat di-*input* sendiri, nilai variabel juga dapat dihasilkan secara acak oleh komputer.



Gambar 3.9 Rancangan *Form Input* Variabel p, q dan a

Keterangan:

1 : *textbox* ’p’, sebagai tempat memasukkan nilai variabel p.

2 : *textbox* ’q’, sebagai tempat memasukkan nilai variabel q.

3 : *textbox* ’a’, sebagai tempat memasukkan nilai variabel a.

4 : *textbox* ’Syarat’, sebagai tempat menampilkan syarat-syarat yang harus dipenuhi oleh variabel p, q dan a.

5 : *progressbar*, untuk menampilkan proses pembangkitan nilai variabel secara acak.

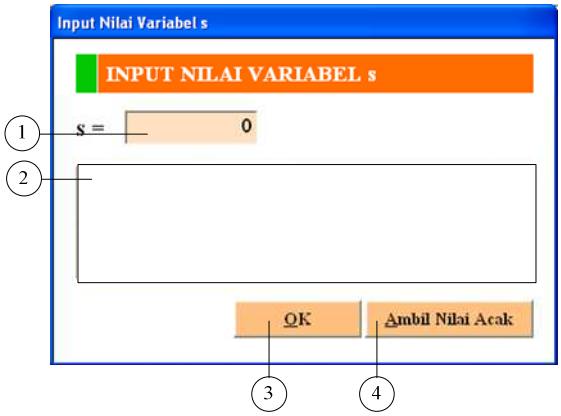
6 : tombol ’Test GCD’, untuk menampilkan *form Test* GCD.

7 : tombol ’OK’, untuk menyimpan nilai variabel dan menutup *form*.

8 : tombol ’Ambil Nilai Acak’, untuk menghasilkan nilai variabel p, q dan a secara acak.

4.2.6 Form Input Variabel s

Form ini berfungsi sebagai tempat memasukkan variabel s pada proses pembentukan kunci. Nilai variabel juga dapat dihasilkan secara acak oleh komputer.



Gambar 3.10 Rancangan *Form Input* Variabel s

Keterangan:

1 : *textbox* ’s’, sebagai tempat memasukkan nilai variabel s.

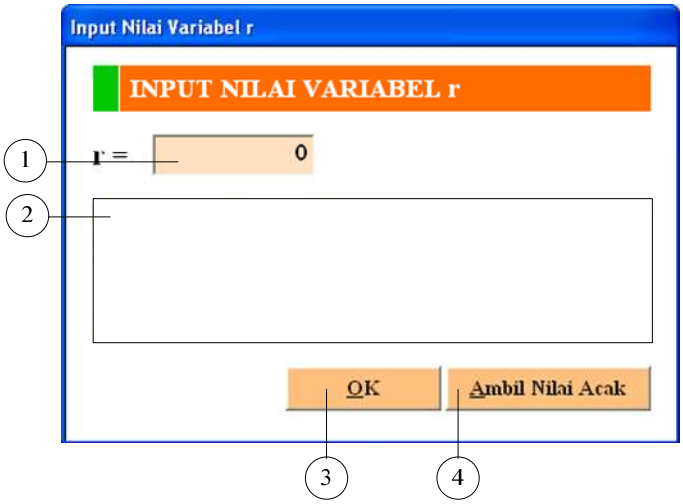
2 : *textbox* ’Syarat’, sebagai tempat menampilkan syarat-syarat yang harus dipenuhi oleh variabel s.

3 : tombol ’OK’, untuk menyimpan nilai variabel dan menutup *form*.

4 : tombol ’Ambil Nilai Acak’, untuk menghasilkan nilai variabel s secara acak.

4.2.7 Form Input Variabel r

Form ini berfungsi sebagai tempat memasukkan variabel r pada proses otentikasi dan tanda tangan digital. Nilai variabel juga dapat dihasilkan secara acak oleh komputer.



Gambar 3.11 Rancangan *Form Input* Variabel r

Keterangan:

1 : *textbox* ’r’, sebagai tempat memasukkan nilai variabel r.

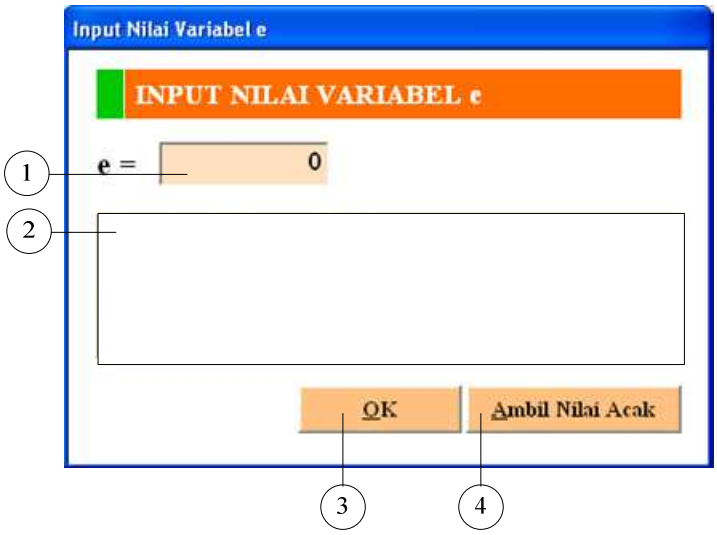
2 : *textbox* ’Syarat’, sebagai tempat menampilkan syarat-syarat yang harus dipenuhi oleh variabel r.

3 : tombol ’OK’, untuk menyimpan nilai variabel dan menutup *form*.

4 : tombol ’Ambil Nilai Acak’, untuk menghasilkan nilai variabel r secara acak.

4.2.8 Form Input Variabel e

Form ini berfungsi sebagai tempat memasukkan variabel e pada proses otentikasi. Nilai variabel juga dapat dihasilkan secara acak oleh komputer.



Gambar 3.12 Rancangan *Form Input* Variabel e

Keterangan:

1 : *textbox* ’e’, sebagai tempat memasukkan nilai variabel e.

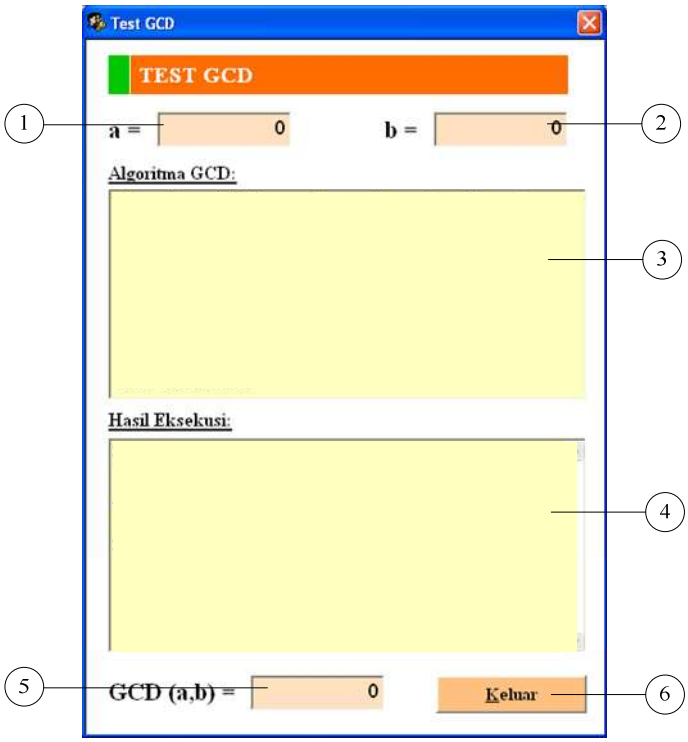
2 : *textbox* ’Syarat’, sebagai tempat menampilkan syarat-syarat yang harus dipenuhi oleh variabel e.

3 : tombol ’OK’, untuk menyimpan nilai variabel dan menutup *form*.

4 : tombol ’Ambil Nilai Acak’, untuk menghasilkan nilai variabel e secara acak.

4.2.9 Form Test GCD

Form ini berfungsi untuk menampilkan analisis algoritma GCD terhadap nilai p dan nilai (q-1) pada *form Input* Variabel p, q dan a.



Gambar 3.13 Rancangan *Form Test* GCD

Keterangan:

1 : *textbox* ’a’, untuk menampilkan nilai variabel a.

2 : *textbox* ’b’, untuk menampilkan nilai variabel b.

3 : *textbox*, untuk menampilkan algoritma GCD.

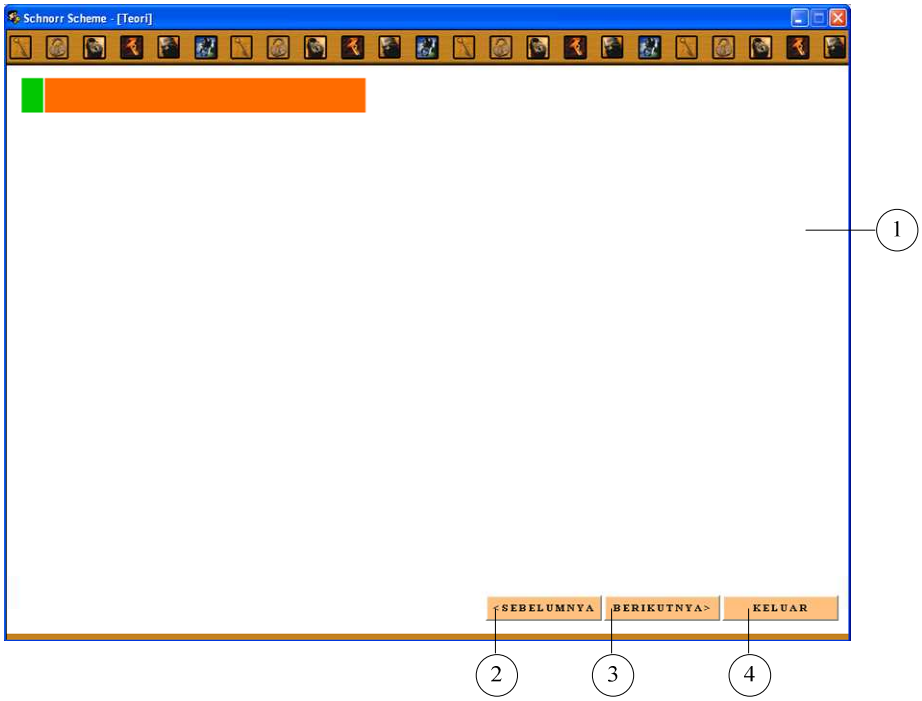
4 : *textbox*, untuk menampilkan hasil eksekusi algoritma GCD.

5 : *textbox* ’GCD’, untuk menampilkan nilai GCD.

6 : tombol ’Keluar’, untuk menutup *form*.

4.2.10 Form Teori

Form ini berfungsi sebagai tempat memasukkan variabel e pada proses otentikasi. Nilai variabel juga dapat dihasilkan secara acak oleh komputer.



Gambar 3.14 Rancangan *Form* Teori

Keterangan:

1 : *picturebox* ’Teori’, sebagai daerah tampilan teori.

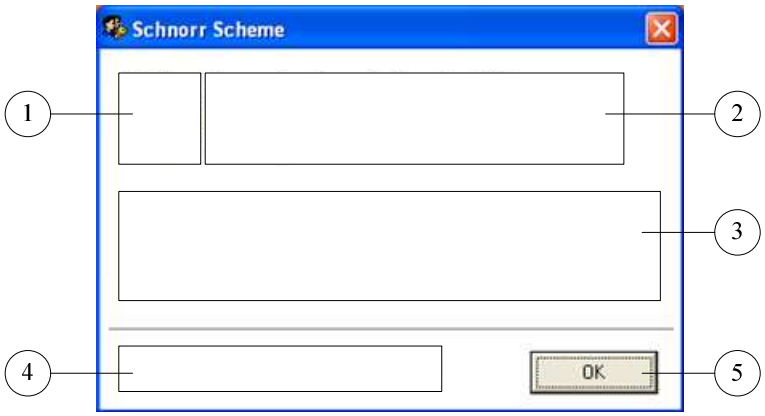
2 : tombol ’Sebelumnya’, untuk menampilkan teori sebelumnya.

3 : tombol ’Berikutnya’, untuk menampilkan teori berikutnya.

4 : tombol ’Keluar’, untuk menutup *form*.

4.2.11 Form About

Form ini berfungsi untuk menampilkan informasi mengenai pembuat perangkat lunak.



Gambar 3.15 Rancangan *Form About*

Keterangan:

1 : *picturebox* ’Logo’, untuk menampilkan logo perangkat lunak.

2 : nama perangkat lunak.

3 : nama, jurusan dan nomor induk mahasiswa (NIM) pembuat perangkat lunak.

4 : nama kampus, kota dan tahun pembuatan perangkat lunak.

5 : tombol ’OK’, untuk menutup *form*.

BAB IV
ALGORITMA DAN IMPLEMENTASI

4.1 Algoritma

Secara umum, algoritma yang digunakan untuk merancang perangkat lunak simulasi *Schnorr Authentication* dan *Digital Signature Scheme* adalah:

1. Algoritma pembentukan kunci.
2. Algoritma skema otentikasi.
3. Algoritma skema tanda tangan digital.
4. Algoritma tes prima Rabin Miller.
5. Algoritma *Fast Exponentiation*.
6. Algoritma *Greatest Common Divisor* (GCD).
7. Algoritma *Extended Euclidean*.

4.1.1 Algoritma Pembentukan Kunci

Proses pembentukan kunci merupakan proses awal sebelum melakukan proses otentikasi ataupun proses tanda tangan digital. Proses pembentukan kunci menghasilkan dua buah kunci, yaitu s (kunci privat) dan v (kunci publik) yang akan digunakan pada skema otentikasi dan tanda tangan digital. Kunci privat adalah kunci yang dipegang oleh pihak pertama (Alice atau pembuat skema), sedangkan kunci

publik diberikan kepada pihak kedua (Bob atau pihak lain yang berinteraksi dengan Alice).

Algoritma pembentukan kunci adalah sebagai berikut:

1. Pilih dua buah bilangan prima p dan q, dan sebuah nilai a, dimana GCD(q, p-1) <> 1 dan (a^q) mod p = 1.
    a. GCD(q, p-1) <> 1 artinya nilai variabel q dan (p-1) tidak relatif prima. Dengan kata lain, kedua variabel tersebut memiliki minimal satu faktor prima yang sama. Algoritma GCD dapat dilihat pada subbab 4.1.6.
    b. Operasi perpangkatan modulo (a^q) mod p = 1 menggunakan algoritma *Fast Exponentiation*. Algoritma *Fast Exponentiation* digunakan untuk menghindari *output* nilai yang melebihi batas maksimum (*overflow*) dari tipe variabel angka. Dengan perhitungan biasa, *overflow* sering terjadi apabila nilai q sangat besar. Algoritma *Fast Exponentiation* dapat dilihat pada subbab 4.1.5.
2. Pilih sebuah nilai s, dimana s harus lebih kecil dari q. Nilai s adalah kunci privat dan tidak boleh diketahui oleh pihak lain.
3. Hitung nilai v dengan rumus: v = a^(-s) mod p. Operasi ini diturunkan menjadi:

    v = (a^ (-1))^s mod p

    v = (a^ (-1))^s mod p

    v = (a^ (-1) mod p)^s mod p

a. Operasi (a^ (-1) mod p) diselesaikan dengan menggunakan algoritma *Extended Euclidean*. Algoritma *Extended Euclidean* dapat dilihat pada subbab 4.1.7.
b. Selanjutnya (operasi pada poin a) dipangkat dengan s modulo p diselesaikan dengan menggunakan algoritma *Fast Exponentiation.*
c. Hasil dari poin b adalah nilai v yang digunakan sebagai kunci publik. Kunci publik diberikan kepada pihak yang akan berinteraksi dengan pihak pertama (Alice) baik dalam skema otentikasi maupun skema tanda tangan digital.

4.1.2 Algoritma Skema Otentikasi

Skema otentikasi digunakan apabila salah satu pihak dalam saluran komunikasi ingin memverifikasi keaslian (otentikasi) dari pihak lainnya. Algoritma skema otentikasi adalah sebagai berikut:

1. Alice memilih sebuah nilai r, dimana nilai r lebih kecil dari q.
2. Alice menghitung: x = (a^r) mod p dan mengirimkan x kepada Bob.

   Operasi (a^r) mod p diselesaikan dengan menggunakan algoritma *Fast Exponentiation.*
3. Bob memilih sebuah nilai e yang berada di antara nilai 0 sampai (2^t – 1) dan mengirimkan e kepada Alice. Nilai t merupakan parameter kemanan yang dimiliki oleh Bob. Oleh karena itu disarankan agar t bernilai besar.

4. Alice menghitung: y = (r + (s * e)) mod q dan mengirimkan y kepada Bob. Operasi ini diselesaikan dengan operasi aritmatika biasa.
5. Bob melakukan verifikasi: x = ((a^y) . (v^e)) mod p. Oleh karena, operasi ((a^y) . (v^e)) dapat menghasilkan nilai yang *overflow*, maka operasi untuk menghasilkan nilai x dapat diturunkan menjadi:

   x = (((a^y) mod p) . ((v^e) mod p)) mod p

   Masing-masing operasi ((a^y) mod p) dan ((v^e) mod p) diselesaikan dengan menggunakan algoritma *Fast Exponentiation*. Jika nilai x sesuai atau sama dengan nilai hasil operasi, maka verifikasi berhasil. Jika tidak, maka verifikasi gagal.

4.1.3 Algoritma Skema Tanda Tangan Digital

Skema tanda tangan digital (*digital signature*) digunakan untuk menjaga keaslian dan keutuhan dari pesan yang akan dikirimkan. Pengubahan atau penghapusan terhadap satu bagian atau lebih dari pesan asli akan mengakibatkan verifikasi tanda tangan digital menjadi tidak benar. Pada skema ini, tanda tangan digital disertakan dalam pesan yang akan dikirimkan melalui suatu saluran komunikasi.

Algoritma skema tanda tangan digital adalah sebagai berikut:

1. Alice memilih sebuah nilai r, dimana nilai r lebih kecil dari q dan menghitung menghitung: x = (a^r) mod p. Operasi ini diselesaikan dengan menggunakan algoritma *Fast Exponentiation*.
2. Alice menggabungkan (*concatenate*) M dan x dan menghitung nilai hash dari hasil penggabungan tersebut: e = H(M, x).
    a. M adalah kode *ascii* dari setiap huruf di dalam pesan, sehingga banyak variabel e adalah sebesar jumlah huruf di dalam pesan.
    b. Hasil penggabungan M dengan x dimasukkan ke fungsi hash SHA-1. Kemudian, ambil 4 karakter dari *output* fungsi hash dan ubah setiap karakter menjadi nilai ascii. Penggabungan nilai *ascii* dari setiap karakter adalah hasil fungsi hash yang disimpan di variabel e.
3. Alice menghitung: y = (r + s.e) mod q.
    a. Karena variabel e berbentuk *array* dengan jumlah *array* sebesar jumlah huruf di dalam pesan, maka variabel y juga berbentuk *array* dengan jumlah *array* sebesar ukuran array dari variabel e.
    b. Tanda tangan digital dari pesan adalah e dan y. Format tanda tangan di dalam perangkat lunak disusun dalam format: ”$|e_1,y_1|e_2,y_2|e_3,y_3$...”.
4. Bob menghitung: x’ = ((a^y) . (v^e)) mod p. Oleh karena, operasi ((a^y) . (v^e)) dapat menghasilkan nilai yang *overflow*, maka operasi untuk menghasilkan nilai x’ dapat diturunkan menjadi:

x’ = (((a^y) mod p) . ((v^e) mod p)) mod p

Masing-masing operasi ((a^y) mod p) dan ((v^e) mod p) diselesaikan dengan menggunakan algoritma *Fast Exponentiation*.

5. Bob menggabungkan M dan x’ dan melakukan verifikasi: e = H(M, x’). Operasi ini sama dengan operasi yang telah dijelaskan pada poin 2, perbedaannya hanya terletak pada penggunaan variabel x’ dan x. Jika nilai e sesuai atau sama dengan nilai *hash*, maka verifikasi tanda tangan digital berhasil. Jika salah satu nilai e tidak sesuai, maka verifikasi tanda tangan digital akan gagal.

4.1.4 Algoritma Tes Prima Rabin Miller

Algoritma tes prima ini digunakan untuk menguji apakah suatu bilangan merupakan bilangan prima atau bukan. Algoritma ini dipakai dalam proses pembentukan kunci yang mengharuskan bilangan p dan q adalah bilangan prima. Dalam bentuk *pseudocode*, algoritma tes prima Rabin Miller adalah sebagai berikut:

```
'Tes Prima Rabin Miller terhadap bilangan P dengan parameter nilai A
Fungsi IsRabinMiller(pnP, pnA) As Boolean

    Set pnC ← pnP - 1

    [CARI nilai b]
    Set nTemp ← 0
    Selama (pnC Mod (2 ^ nTemp)) ← 0 DAN ((2 ^ nTemp) <
            pnP)
```

```
        Set nTemp ← nTemp + 1

    End Selama
    Set pnB ← nTemp - 1

    [CARI nilai m]
    Set pnM ← (pnC / (2 ^ pnB))

    Set pnJ ← 0
    Set pnZ ← FastExp(pnA, pnM, pnP)

    Jika (pnZ ← 1) ATAU (pnZ ← (pnP - 1)) Maka
        IsRabinMiller ← True (pnP adalah bilangan prima)
        KELUAR DARI FUNGSI
    End Jika

lAgain:

    Jika pnJ > 0 DAN pnZ ← 1 Maka
        IsRabinMiller ← False (pnP bukan bilangan prima)
        KELUAR DARI FUNGSI
    End Jika

    Set pnJ ← pnJ + 1
    Jika (pnJ < pnB) And (pnZ <> pnP - 1) Maka
        pnZ ← FastExp(pnZ, 2, pnP)
        GoTo lAgain
    End Jika

    Jika pnZ ← (pnP - 1) Maka
```

```
        IsRabinMiller ← True (pnP adalah bilangan prima)
        KELUAR DARI FUNGSI
    End Jika

    Jika (pnJ ← pnB) DAN pnZ <> (pnP - 1) Maka
        IsRabinMiller ← False (pnP bukan bilangan prima)
        KELUAR DARI FUNGSI
    End If

End Fungsi
```

4.1.5 Algoritma Fast Exponentiation

Algoritma *Fast Exponentiation* digunakan untuk mencari hasil dari operasi perpangkatan modulo bilangan besar (a^b mod c). Algoritma *Fast Exponentiation* digunakan karena perhitungan operasi perpangkatan modulo (a^b mod c) akan menghasilkan nilai *overflow* apabila variabel b bernilai sangat besar. Algoritma Fast Exponentiation ditampilkan dalam bentuk *pseudocode* adalah sebagai berikut:

```
[Fungsi mengembalikan nilai (A^B) mod C]
FUNGSI FastExp(A, B, C) As Double

    Set A1 ← A
    Set B1 ← B
    Set Product ← 1
    Selama (B1 <> 0)
        Selama B1 mod 2 ← 0
            Set B1 ← B1 div 2
            Set A1 ← (A1 * A1) mod C
```

```
        End Selama
        Set B1 ← B1 - 1
        Set Product ← (Product * A1) mod C
    End Selama

    [Kembalikan nilai fast exponentiation]
    Set FastExp ← Product

End FUNGSI
```

4.1.6 Algoritma Greatest Common Divisor (GCD)

Algoritma GCD digunakan untuk mencari nilai faktor persekutuan terbesar (FPB) dari dua buah bilangan. Algoritma ini digunakan dalam proses pembentukan kunci dimana GCD(q, p-1) tidak boleh bernilai 1. Algoritma GCD ditampilkan dalam bentuk *pseudocode* adalah sebagai berikut:

```
[Fungsi mengembalikan nilai GCD(A, B)]
FUNGSI GCD(A, B) As Long

    Set P ← A
    Set Q ← B
    Selama Q <> 0
        Set R ← P mod Q
        Set P ← Q
        Set Q ← R
    End Selama
```

```
    [Kembalikan nilai GCD]
    Set GCD ← P

End FUNGSI
```

4.1.7 Algoritma Extended Euclidean

Algoritma Extended Euclidean digunakan untuk mencari inversi modulo (untuk persamaan e^{-1} mod n). Algoritma Extended Euclidean ditampilkan dalam bentuk *pseudocode* adalah sebagai berikut:

```
[Fungsi mengembalikan nilai (NilaiX^-1) mod N]
FUNGSI ExtendedEuclidean(NilaiX, N) As Double

    Set Selesai ← False

    [Bentuk Array]
    Set A(1, 1) ← N
    Set A(1, 2) ← NilaiX

    [Matriks Identitas]
    Set A(2, 1) ← 1
    Set A(2, 2) ← 0
    Set A(3, 1) ← 0
    Set A(3, 2) ← 1

    [Lakukan looping]
    Selama bSelesai ← False
```

```
        [Hitung nilai m]
        Set nM ← A(1, 1) div A(1, 2)
        Untuk nI ← 1 Sampai 3
            [Hitung nilai x]
            Set nX ← A(nI, 1) - nM * A(nI, 2)
            [Ubah nilai]
            Set A(nI, 1) ← A(nI, 2)
            Set A(nI, 2) ← nX
            Jika (nI ← 1) DAN (nX ← 0) Maka
                Set Selesai ← True
            End Jika
        Next nI
    End Selama

    [Kembalikan hasil extended euclidean]
    Jika A(3, 1) >= 0 Maka,
        Set ExtendedEuclidean ← A(3, 1)
    Jika tidak,
        Set ExtendedEuclidean ← A(3, 1) + N
    End Jika

End FUNGSI
```

4.2 Implementasi Perangkat Lunak

4.2.1 Batasan Implementasi

1. Aplikasi akan menjelaskan prosedur kerja dengan menggunakan bantuan animasi gambar.
2. Jumlah orang dibatasi sebanyak 2 orang.

3. Inputan Variable data berupa Pesan (*message*) dengan panjang maksimal 50 karakter, Sepasang bilangan prima p dan q, Bilangan acak a, r dan e, Bilangan eksponen (pemangkatan) t.
4. Panjang bilangan yang di-*input* dibatasi maksimal 50 digit bilangan bulat positif.

4.2.2 Spesifikasi Hardware dan Software

Spesifikasi perangkat keras yang direkomendasikan untuk menjalankan perangkat lunak simulasi *Schnorr Authentication* dan *Digital Signature Scheme* ini adalah sebagai berikut :

1. Prosesor *Pentium* IV 2.0 GHz.
2. *Harddisk* 80 *GigaByte*.
3. Memori (RAM) sebesar 256 MB.
4. Monitor SVGA yang mendukung resolusi minimum 1024 x768.
5. *VGA Card* 64 MB.
6. *Keyboard* dan *Mouse*.

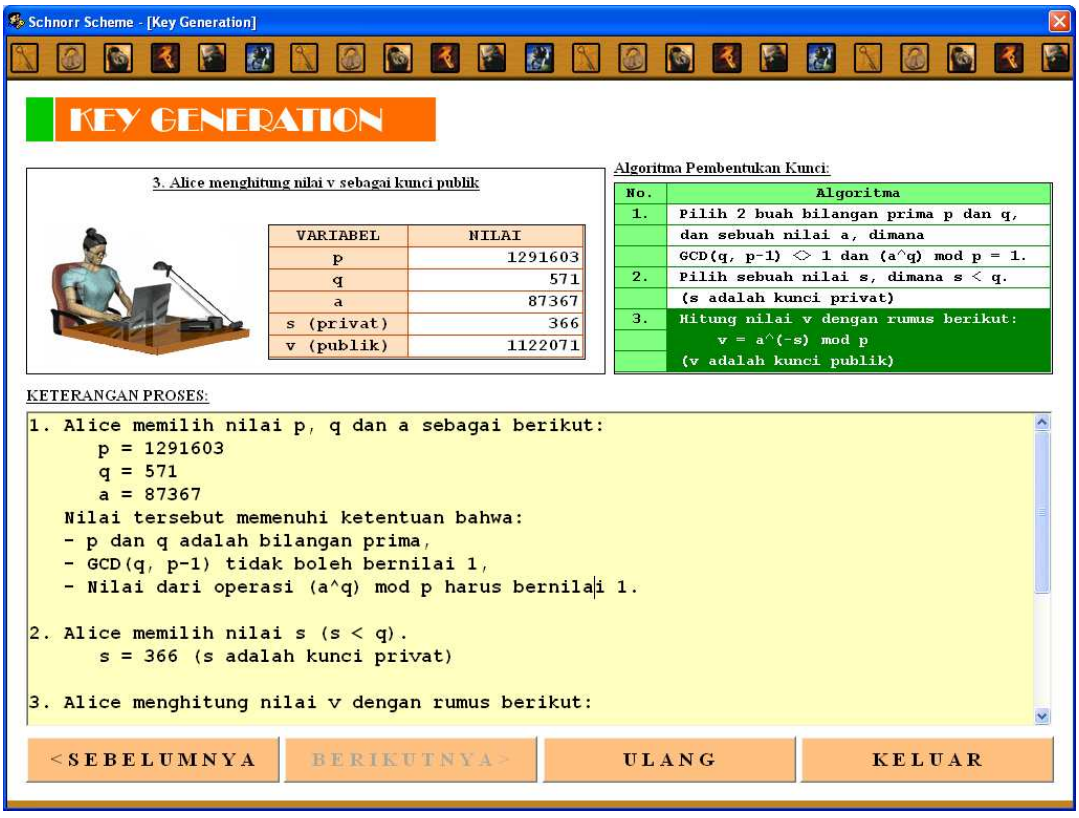
Adapun perangkat lunak (*software*) yang digunakan untuk menjalankan aplikasi ini adalah lingkungan sistem operasi Microsoft Windows 98 Second Edition atau Microsoft Windows NT / 2000 / XP.

4.2.3 Pengujian Program

Untuk menguji *output program*, pengujian akan dilakukan pada proses pembentukan kunci (*key generation*), proses otentikasi (*authentication*) dan proses tanda tangan digital (*digital signature*).

1. Proses Pembentukan Kunci

Form proses pembentukan kunci dapat dilihat pada gambar 4.1 berikut.



Gambar 4.1 Form Proses Pembentukan Kunci

Prosedur yang dilakukan pada proses pembentukan kunci adalah sebagai berikut:

```
1. Alice memilih nilai p, q dan a sebagai berikut:
       p = 1291603
```

q = 571

a = 87367

Nilai tersebut memenuhi ketentuan bahwa:

- p dan q adalah bilangan prima,
- GCD(q, p-1) tidak boleh bernilai 1,
- Nilai dari operasi (a^q) mod p harus bernilai 1.

2. Alice memilih nilai s (s < q).

   s = 366 (s adalah kunci privat)

3. Alice menghitung nilai v dengan rumus berikut:

   v = a^(-s) mod p

   v = 87367^(-366) mod 1291603

   v = ((87367^(-1) mod 1291603)^366) mod 1291603

   Selesaikan operasi (87367^(-1) mod 1291603) dengan algoritma extended euclidean

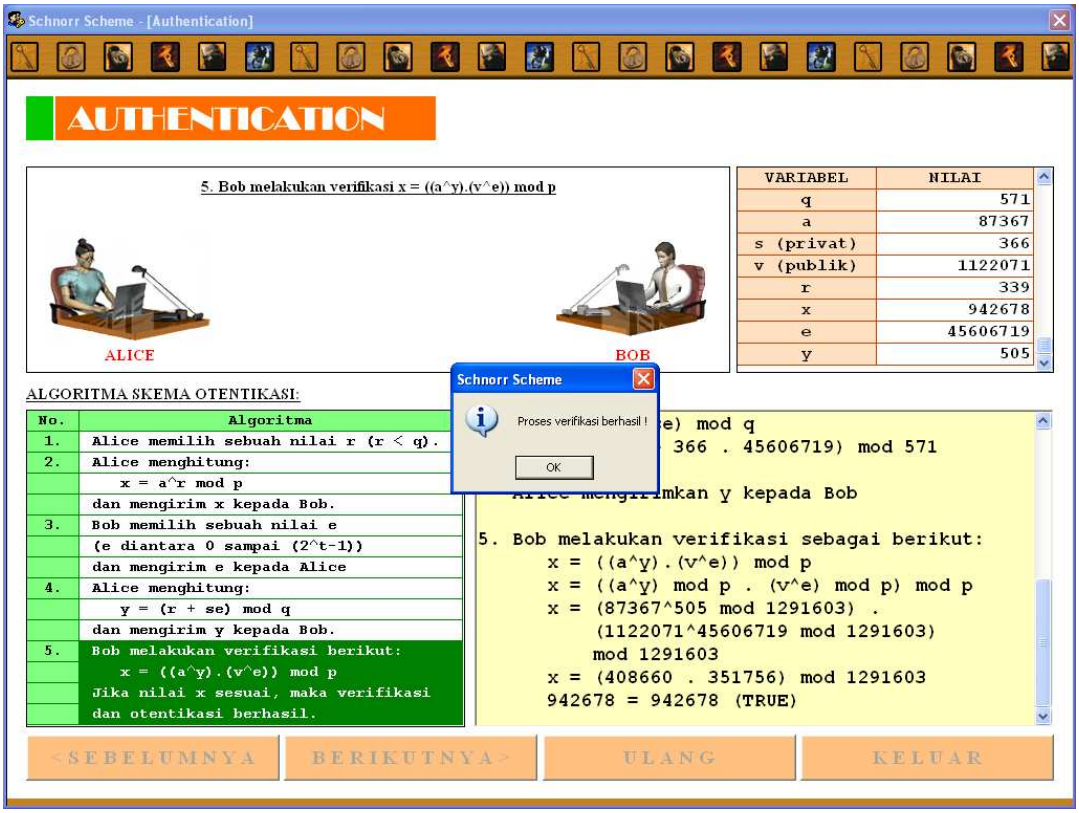
   (87367^(-1) mod 1291603) = 1253269

   v = (1253269^366) mod 1291603 (selesaikan dengan fast exponentiation)

   v = 1122071 (v adalah kunci publik)

2. Proses Otentikasi (*Authentication*)

Form proses otentikasi dapat dilihat pada gambar 4.2 berikut.



Gambar 4.2 Form Proses Otentikasi (*Authentication*)

Prosedur yang dilakukan pada proses otentikasi adalah sebagai berikut:

1. Alice memilih nilai r sebagai berikut:

   r = 339

2. Alice menghitung nilai x

   x = a^r mod p (selesaikan dengan fast exponentiation)

   x = 87367^339 mod 1291603

   x = 942678

   Alice mengirimkan x kepada Bob

3. Bob memilih nilai e sebagai berikut:

   e = 45606719

   Bob mengirimkan e kepada Alice

4. Alice menghitung nilai y sebagai berikut:

   y = (r + se) mod q

   y = (339 + 366 . 45606719) mod 571

   y = 505

   Alice mengirimkan y kepada Bob

5. Bob melakukan verifikasi sebagai berikut:

   x = ((a^y).(v^e)) mod p

   x = ((a^y) mod p . (v^e) mod p) mod p

   x = (87367^505 mod 1291603) .
   (1122071^45606719 mod 1291603)
   mod 1291603

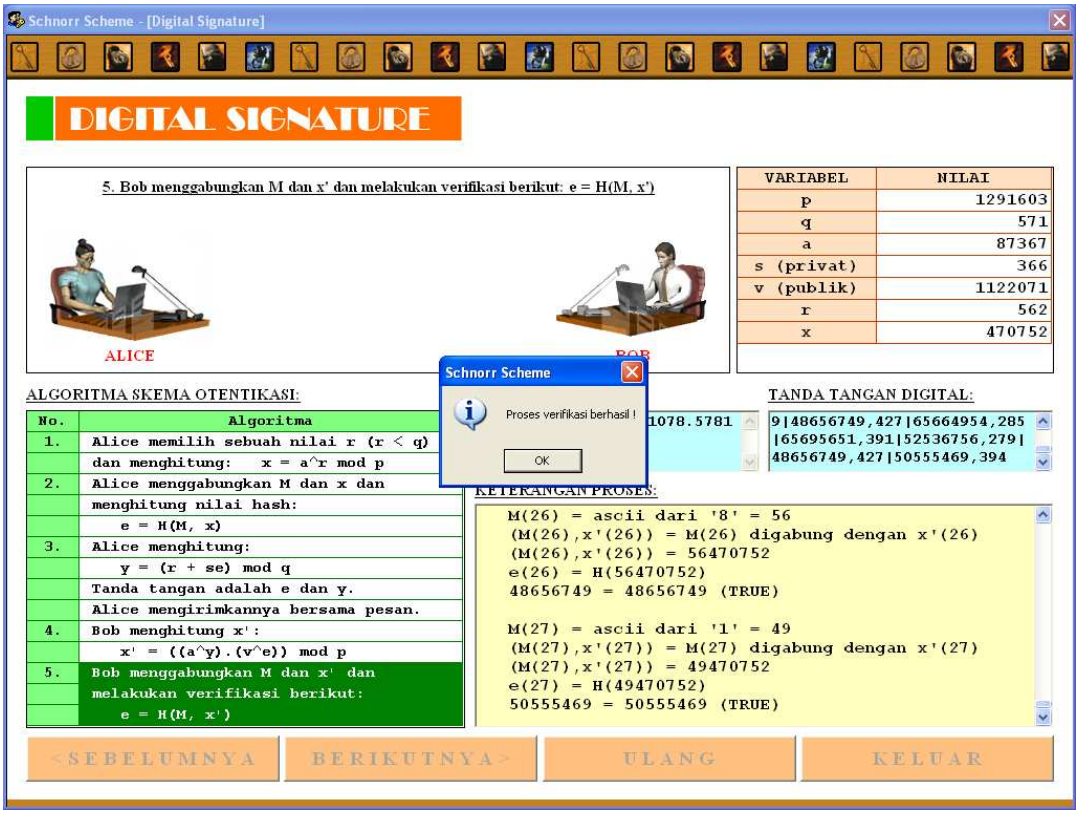
   x = (408660 . 351756) mod 1291603

   942678 = 942678 (**TRUE**)

Hasil perhitungan operasi (((a^y).(v^e)) mod p) sama dengan nilai x. Proses otentikasi berhasil.

3. Proses Tanda Tangan Digital (*Digital Signature*)

Form proses tanda tangan digital dapat dilihat pada gambar 4.3 berikut.



Gambar 4.3 Form Proses Tanda Tangan Digital (*Digital Signature*)

Misalkan pesan adalah ’No.rek bank = 125.1078.5781’.

Tanda tangan digital yang dihasilkan adalah:

|57556954,397|65496769,88|65664954,285|69534851,528|49506556,379|50666569,371|66505766,453|65655348,136|50685465,355|57666967,519|50666569,371|66505766,453|51674948,93|66505766,453|50555

```
469,394|66565254,260|65695651,391|65664954,285|50555469,394|56
486751,108|52536756,279|48656749,427|65664954,285|65695651,391
|52536756,279|48656749,427|50555469,394
```

Prosedur yang dilakukan pada proses tanda tangan digital adalah sebagai berikut:

```
1. Alice memilih nilai r sebagai berikut:
      r = 562

   Alice menghitung nilai x
      x = a^r mod p (selesaikan dengan fast
                     exponentiation)
      x = 87367^562 mod 1291603
      x = 470752

2. Alice menggabungkan M dan x dan
   menghitung nilai hash:  e = H(M, x)

   M(1) = ascii dari 'N' = 78
   (M(1),x) = M(1) digabung dengan x
   (M(1),x) = 78470752
   e(1) = H(78470752)
   e(1) = 57556954

   M(2) = ascii dari 'o' = 111
   (M(2),x) = M(2) digabung dengan x
   (M(2),x) = 111470752
   e(2) = H(111470752)
   e(2) = 65496769

   M(3) = ascii dari '.' = 46
   (M(3),x) = M(3) digabung dengan x
   (M(3),x) = 46470752
   e(3) = H(46470752)
   e(3) = 65664954

   M(4) = ascii dari 'r' = 114
   (M(4),x) = M(4) digabung dengan x
   (M(4),x) = 114470752
   e(4) = H(114470752)
   e(4) = 69534851

   M(5) = ascii dari 'e' = 101
   (M(5),x) = M(5) digabung dengan x
   (M(5),x) = 101470752
   e(5) = H(101470752)
   e(5) = 49506556
```

```
M(6) = ascii dari 'k' = 107
(M(6),x) = M(6) digabung dengan x
(M(6),x) = 107470752
e(6) = H(107470752)
e(6) = 50666569

M(7) = ascii dari ' ' = 32
(M(7),x) = M(7) digabung dengan x
(M(7),x) = 32470752
e(7) = H(32470752)
e(7) = 66505766

M(8) = ascii dari 'b' = 98
(M(8),x) = M(8) digabung dengan x
(M(8),x) = 98470752
e(8) = H(98470752)
e(8) = 65655348

M(9) = ascii dari 'a' = 97
(M(9),x) = M(9) digabung dengan x
(M(9),x) = 97470752
e(9) = H(97470752)
e(9) = 50685465

M(10) = ascii dari 'n' = 110
(M(10),x) = M(10) digabung dengan x
(M(10),x) = 110470752
e(10) = H(110470752)
e(10) = 57666967

M(11) = ascii dari 'k' = 107
(M(11),x) = M(11) digabung dengan x
(M(11),x) = 107470752
e(11) = H(107470752)
e(11) = 50666569

M(12) = ascii dari ' ' = 32
(M(12),x) = M(12) digabung dengan x
(M(12),x) = 32470752
e(12) = H(32470752)
e(12) = 66505766

M(13) = ascii dari '=' = 61
(M(13),x) = M(13) digabung dengan x
(M(13),x) = 61470752
e(13) = H(61470752)
e(13) = 51674948

M(14) = ascii dari ' ' = 32
(M(14),x) = M(14) digabung dengan x
(M(14),x) = 32470752
e(14) = H(32470752)
```

```
e(14) = 66505766

M(15) = ascii dari '1' = 49
(M(15),x) = M(15) digabung dengan x
(M(15),x) = 49470752
e(15) = H(49470752)
e(15) = 50555469

M(16) = ascii dari '2' = 50
(M(16),x) = M(16) digabung dengan x
(M(16),x) = 50470752
e(16) = H(50470752)
e(16) = 66565254

M(17) = ascii dari '5' = 53
(M(17),x) = M(17) digabung dengan x
(M(17),x) = 53470752
e(17) = H(53470752)
e(17) = 65695651

M(18) = ascii dari '.' = 46
(M(18),x) = M(18) digabung dengan x
(M(18),x) = 46470752
e(18) = H(46470752)
e(18) = 65664954

M(19) = ascii dari '1' = 49
(M(19),x) = M(19) digabung dengan x
(M(19),x) = 49470752
e(19) = H(49470752)
e(19) = 50555469

M(20) = ascii dari '0' = 48
(M(20),x) = M(20) digabung dengan x
(M(20),x) = 48470752
e(20) = H(48470752)
e(20) = 56486751

M(21) = ascii dari '7' = 55
(M(21),x) = M(21) digabung dengan x
(M(21),x) = 55470752
e(21) = H(55470752)
e(21) = 52536756

M(22) = ascii dari '8' = 56
(M(22),x) = M(22) digabung dengan x
(M(22),x) = 56470752
e(22) = H(56470752)
e(22) = 48656749

M(23) = ascii dari '.' = 46
(M(23),x) = M(23) digabung dengan x
```

```
(M(23),x) = 46470752
e(23) = H(46470752)
e(23) = 65664954

M(24) = ascii dari '5' = 53
(M(24),x) = M(24) digabung dengan x
(M(24),x) = 53470752
e(24) = H(53470752)
e(24) = 65695651

M(25) = ascii dari '7' = 55
(M(25),x) = M(25) digabung dengan x
(M(25),x) = 55470752
e(25) = H(55470752)
e(25) = 52536756

M(26) = ascii dari '8' = 56
(M(26),x) = M(26) digabung dengan x
(M(26),x) = 56470752
e(26) = H(56470752)
e(26) = 48656749

M(27) = ascii dari '1' = 49
(M(27),x) = M(27) digabung dengan x
(M(27),x) = 49470752
e(27) = H(49470752)
e(27) = 50555469
```

3. Alice menghitung nilai y sebagai berikut:
   $y = (r + se) \bmod q$

```
y(1) = (r + (s . e(1))) mod q
y(1) = (562 + (366 . 57556954)) mod 571
y(1) = 397

y(2) = (r + (s . e(2))) mod q
y(2) = (562 + (366 . 65496769)) mod 571
y(2) = 88

y(3) = (r + (s . e(3))) mod q
y(3) = (562 + (366 . 65664954)) mod 571
y(3) = 285

y(4) = (r + (s . e(4))) mod q
y(4) = (562 + (366 . 69534851)) mod 571
y(4) = 528

y(5) = (r + (s . e(5))) mod q
y(5) = (562 + (366 . 49506556)) mod 571
y(5) = 379

y(6) = (r + (s . e(6))) mod q
```

```
y(6) = (562 + (366 . 50666569)) mod 571
y(6) = 371

y(7) = (r + (s . e(7))) mod q
y(7) = (562 + (366 . 66505766)) mod 571
y(7) = 453

y(8) = (r + (s . e(8))) mod q
y(8) = (562 + (366 . 65655348)) mod 571
y(8) = 136

y(9) = (r + (s . e(9))) mod q
y(9) = (562 + (366 . 50685465)) mod 571
y(9) = 355

y(10) = (r + (s . e(10))) mod q
y(10) = (562 + (366 . 57666967)) mod 571
y(10) = 519

y(11) = (r + (s . e(11))) mod q
y(11) = (562 + (366 . 50666569)) mod 571
y(11) = 371

y(12) = (r + (s . e(12))) mod q
y(12) = (562 + (366 . 66505766)) mod 571
y(12) = 453

y(13) = (r + (s . e(13))) mod q
y(13) = (562 + (366 . 51674948)) mod 571
y(13) = 93

y(14) = (r + (s . e(14))) mod q
y(14) = (562 + (366 . 66505766)) mod 571
y(14) = 453

y(15) = (r + (s . e(15))) mod q
y(15) = (562 + (366 . 50555469)) mod 571
y(15) = 394

y(16) = (r + (s . e(16))) mod q
y(16) = (562 + (366 . 66565254)) mod 571
y(16) = 260

y(17) = (r + (s . e(17))) mod q
y(17) = (562 + (366 . 65695651)) mod 571
y(17) = 391

y(18) = (r + (s . e(18))) mod q
y(18) = (562 + (366 . 65664954)) mod 571
y(18) = 285

y(19) = (r + (s . e(19))) mod q
```

```
y(19) = (562 + (366 . 50555469)) mod 571
y(19) = 394

y(20) = (r + (s . e(20))) mod q
y(20) = (562 + (366 . 56486751)) mod 571
y(20) = 108

y(21) = (r + (s . e(21))) mod q
y(21) = (562 + (366 . 52536756)) mod 571
y(21) = 279

y(22) = (r + (s . e(22))) mod q
y(22) = (562 + (366 . 48656749)) mod 571
y(22) = 427

y(23) = (r + (s . e(23))) mod q
y(23) = (562 + (366 . 65664954)) mod 571
y(23) = 285

y(24) = (r + (s . e(24))) mod q
y(24) = (562 + (366 . 65695651)) mod 571
y(24) = 391

y(25) = (r + (s . e(25))) mod q
y(25) = (562 + (366 . 52536756)) mod 571
y(25) = 279

y(26) = (r + (s . e(26))) mod q
y(26) = (562 + (366 . 48656749)) mod 571
y(26) = 427

y(27) = (r + (s . e(27))) mod q
y(27) = (562 + (366 . 50555469)) mod 571
y(27) = 394

Tanda tangan digital adalah e dan y.
Alice mengirimkan pesan dan tanda tangan digital
kepada Bob.
```

4. Bob melakukan perhitungan sebagai berikut:

```
   x' = ((a^y).(v^e)) mod p

x'(1) = ((a^y(1)) mod p . (v^e(1)) mod p) mod p
x'(1) = (87367^397 mod 1291603) .
        (1122071^57556954 mod 1291603)
        mod 1291603
x'(1) = 470752

x'(2) = ((a^y(2)) mod p . (v^e(2)) mod p) mod p
x'(2) = (87367^88 mod 1291603) .
        (1122071^65496769 mod 1291603)
        mod 1291603
```

```
x'(2) = 470752

x'(3) = ((a^y(3)) mod p . (v^e(3)) mod p) mod p
x'(3) = (87367^285 mod 1291603) .
        (1122071^65664954 mod 1291603)
        mod 1291603
x'(3) = 470752

x'(4) = ((a^y(4)) mod p . (v^e(4)) mod p) mod p
x'(4) = (87367^528 mod 1291603) .
        (1122071^69534851 mod 1291603)
        mod 1291603
x'(4) = 470752

x'(5) = ((a^y(5)) mod p . (v^e(5)) mod p) mod p
x'(5) = (87367^379 mod 1291603) .
        (1122071^49506556 mod 1291603)
        mod 1291603
x'(5) = 470752

x'(6) = ((a^y(6)) mod p . (v^e(6)) mod p) mod p
x'(6) = (87367^371 mod 1291603) .
        (1122071^50666569 mod 1291603)
        mod 1291603
x'(6) = 470752

x'(7) = ((a^y(7)) mod p . (v^e(7)) mod p) mod p
x'(7) = (87367^453 mod 1291603) .
        (1122071^66505766 mod 1291603)
        mod 1291603
x'(7) = 470752

x'(8) = ((a^y(8)) mod p . (v^e(8)) mod p) mod p
x'(8) = (87367^136 mod 1291603) .
        (1122071^65655348 mod 1291603)
        mod 1291603
x'(8) = 470752

x'(9) = ((a^y(9)) mod p . (v^e(9)) mod p) mod p
x'(9) = (87367^355 mod 1291603) .
        (1122071^50685465 mod 1291603)
        mod 1291603
x'(9) = 470752

x'(10) = ((a^y(10)) mod p . (v^e(10)) mod p) mod p
x'(10) = (87367^519 mod 1291603) .
        (1122071^57666967 mod 1291603)
        mod 1291603
x'(10) = 470752

x'(11) = ((a^y(11)) mod p . (v^e(11)) mod p) mod p
x'(11) = (87367^371 mod 1291603) .
```

```
        (1122071^50666569 mod 1291603)
        mod 1291603
x'(11) = 470752

x'(12) = ((a^y(12)) mod p . (v^e(12)) mod p) mod p
x'(12) = (87367^453 mod 1291603) .
        (1122071^66505766 mod 1291603)
        mod 1291603
x'(12) = 470752

x'(13) = ((a^y(13)) mod p . (v^e(13)) mod p) mod p
x'(13) = (87367^93 mod 1291603) .
        (1122071^51674948 mod 1291603)
        mod 1291603
x'(13) = 470752

x'(14) = ((a^y(14)) mod p . (v^e(14)) mod p) mod p
x'(14) = (87367^453 mod 1291603) .
        (1122071^66505766 mod 1291603)
        mod 1291603
x'(14) = 470752

x'(15) = ((a^y(15)) mod p . (v^e(15)) mod p) mod p
x'(15) = (87367^394 mod 1291603) .
        (1122071^50555469 mod 1291603)
        mod 1291603
x'(15) = 470752

x'(16) = ((a^y(16)) mod p . (v^e(16)) mod p) mod p
x'(16) = (87367^260 mod 1291603) .
        (1122071^66565254 mod 1291603)
        mod 1291603
x'(16) = 470752

x'(17) = ((a^y(17)) mod p . (v^e(17)) mod p) mod p
x'(17) = (87367^391 mod 1291603) .
        (1122071^65695651 mod 1291603)
        mod 1291603
x'(17) = 470752

x'(18) = ((a^y(18)) mod p . (v^e(18)) mod p) mod p
x'(18) = (87367^285 mod 1291603) .
        (1122071^65664954 mod 1291603)
        mod 1291603
x'(18) = 470752

x'(19) = ((a^y(19)) mod p . (v^e(19)) mod p) mod p
x'(19) = (87367^394 mod 1291603) .
        (1122071^50555469 mod 1291603)
        mod 1291603
x'(19) = 470752
```

```
x'(20) = ((a^y(20)) mod p . (v^e(20)) mod p) mod p
x'(20) = (87367^108 mod 1291603) .
         (1122071^56486751 mod 1291603)
         mod 1291603
x'(20) = 470752

x'(21) = ((a^y(21)) mod p . (v^e(21)) mod p) mod p
x'(21) = (87367^279 mod 1291603) .
         (1122071^52536756 mod 1291603)
         mod 1291603
x'(21) = 470752

x'(22) = ((a^y(22)) mod p . (v^e(22)) mod p) mod p
x'(22) = (87367^427 mod 1291603) .
         (1122071^48656749 mod 1291603)
         mod 1291603
x'(22) = 470752

x'(23) = ((a^y(23)) mod p . (v^e(23)) mod p) mod p
x'(23) = (87367^285 mod 1291603) .
         (1122071^65664954 mod 1291603)
         mod 1291603
x'(23) = 470752

x'(24) = ((a^y(24)) mod p . (v^e(24)) mod p) mod p
x'(24) = (87367^391 mod 1291603) .
         (1122071^65695651 mod 1291603)
         mod 1291603
x'(24) = 470752

x'(25) = ((a^y(25)) mod p . (v^e(25)) mod p) mod p
x'(25) = (87367^279 mod 1291603) .
         (1122071^52536756 mod 1291603)
         mod 1291603
x'(25) = 470752

x'(26) = ((a^y(26)) mod p . (v^e(26)) mod p) mod p
x'(26) = (87367^427 mod 1291603) .
         (1122071^48656749 mod 1291603)
         mod 1291603
x'(26) = 470752

x'(27) = ((a^y(27)) mod p . (v^e(27)) mod p) mod p
x'(27) = (87367^394 mod 1291603) .
         (1122071^50555469 mod 1291603)
         mod 1291603
x'(27) = 470752
```

5. Bob menggabungkan M dan x' dan melakukan verifikasi: e = H(M, x')

```
M(1) = ascii dari 'N' = 78
```

```
(M(1),x'(1)) = M(1) digabung dengan x'(1)
(M(1),x'(1)) = 78470752
e(1) = H(78470752)
57556954 = 57556954 (TRUE)

M(2) = ascii dari 'o' = 111
(M(2),x'(2)) = M(2) digabung dengan x'(2)
(M(2),x'(2)) = 111470752
e(2) = H(111470752)
65496769 = 65496769 (TRUE)

M(3) = ascii dari '.' = 46
(M(3),x'(3)) = M(3) digabung dengan x'(3)
(M(3),x'(3)) = 46470752
e(3) = H(46470752)
65664954 = 65664954 (TRUE)

M(4) = ascii dari 'r' = 114
(M(4),x'(4)) = M(4) digabung dengan x'(4)
(M(4),x'(4)) = 114470752
e(4) = H(114470752)
69534851 = 69534851 (TRUE)

M(5) = ascii dari 'e' = 101
(M(5),x'(5)) = M(5) digabung dengan x'(5)
(M(5),x'(5)) = 101470752
e(5) = H(101470752)
49506556 = 49506556 (TRUE)

M(6) = ascii dari 'k' = 107
(M(6),x'(6)) = M(6) digabung dengan x'(6)
(M(6),x'(6)) = 107470752
e(6) = H(107470752)
50666569 = 50666569 (TRUE)

M(7) = ascii dari ' ' = 32
(M(7),x'(7)) = M(7) digabung dengan x'(7)
(M(7),x'(7)) = 32470752
e(7) = H(32470752)
66505766 = 66505766 (TRUE)

M(8) = ascii dari 'b' = 98
(M(8),x'(8)) = M(8) digabung dengan x'(8)
(M(8),x'(8)) = 98470752
e(8) = H(98470752)
65655348 = 65655348 (TRUE)

M(9) = ascii dari 'a' = 97
(M(9),x'(9)) = M(9) digabung dengan x'(9)
(M(9),x'(9)) = 97470752
e(9) = H(97470752)
50685465 = 50685465 (TRUE)
```

```
M(10) = ascii dari 'n' = 110
(M(10),x'(10)) = M(10) digabung dengan x'(10)
(M(10),x'(10)) = 110470752
e(10) = H(110470752)
57666967 = 57666967 (TRUE)

M(11) = ascii dari 'k' = 107
(M(11),x'(11)) = M(11) digabung dengan x'(11)
(M(11),x'(11)) = 107470752
e(11) = H(107470752)
50666569 = 50666569 (TRUE)

M(12) = ascii dari ' ' = 32
(M(12),x'(12)) = M(12) digabung dengan x'(12)
(M(12),x'(12)) = 32470752
e(12) = H(32470752)
66505766 = 66505766 (TRUE)

M(13) = ascii dari '=' = 61
(M(13),x'(13)) = M(13) digabung dengan x'(13)
(M(13),x'(13)) = 61470752
e(13) = H(61470752)
51674948 = 51674948 (TRUE)

M(14) = ascii dari ' ' = 32
(M(14),x'(14)) = M(14) digabung dengan x'(14)
(M(14),x'(14)) = 32470752
e(14) = H(32470752)
66505766 = 66505766 (TRUE)

M(15) = ascii dari '1' = 49
(M(15),x'(15)) = M(15) digabung dengan x'(15)
(M(15),x'(15)) = 49470752
e(15) = H(49470752)
50555469 = 50555469 (TRUE)

M(16) = ascii dari '2' = 50
(M(16),x'(16)) = M(16) digabung dengan x'(16)
(M(16),x'(16)) = 50470752
e(16) = H(50470752)
66565254 = 66565254 (TRUE)

M(17) = ascii dari '5' = 53
(M(17),x'(17)) = M(17) digabung dengan x'(17)
(M(17),x'(17)) = 53470752
e(17) = H(53470752)
65695651 = 65695651 (TRUE)

M(18) = ascii dari '.' = 46
(M(18),x'(18)) = M(18) digabung dengan x'(18)
(M(18),x'(18)) = 46470752
```

```
e(18) = H(46470752)
65664954 = 65664954 (TRUE)

M(19) = ascii dari '1' = 49
(M(19),x'(19)) = M(19) digabung dengan x'(19)
(M(19),x'(19)) = 49470752
e(19) = H(49470752)
50555469 = 50555469 (TRUE)

M(20) = ascii dari '0' = 48
(M(20),x'(20)) = M(20) digabung dengan x'(20)
(M(20),x'(20)) = 48470752
e(20) = H(48470752)
56486751 = 56486751 (TRUE)

M(21) = ascii dari '7' = 55
(M(21),x'(21)) = M(21) digabung dengan x'(21)
(M(21),x'(21)) = 55470752
e(21) = H(55470752)
52536756 = 52536756 (TRUE)

M(22) = ascii dari '8' = 56
(M(22),x'(22)) = M(22) digabung dengan x'(22)
(M(22),x'(22)) = 56470752
e(22) = H(56470752)
48656749 = 48656749 (TRUE)

M(23) = ascii dari '.' = 46
(M(23),x'(23)) = M(23) digabung dengan x'(23)
(M(23),x'(23)) = 46470752
e(23) = H(46470752)
65664954 = 65664954 (TRUE)

M(24) = ascii dari '5' = 53
(M(24),x'(24)) = M(24) digabung dengan x'(24)
(M(24),x'(24)) = 53470752
e(24) = H(53470752)
65695651 = 65695651 (TRUE)

M(25) = ascii dari '7' = 55
(M(25),x'(25)) = M(25) digabung dengan x'(25)
(M(25),x'(25)) = 55470752
e(25) = H(55470752)
52536756 = 52536756 (TRUE)

M(26) = ascii dari '8' = 56
(M(26),x'(26)) = M(26) digabung dengan x'(26)
(M(26),x'(26)) = 56470752
e(26) = H(56470752)
48656749 = 48656749 (TRUE)

M(27) = ascii dari '1' = 49
```

```
(M(27),x'(27)) = M(27) digabung dengan x'(27)
(M(27),x'(27)) = 49470752
e(27) = H(49470752)
50555469 = 50555469 (TRUE)
```

Hasil perhitungan operasi H(M, x') sama dengan nilai e. Proses verifikasi tanda tangan digital berhasil.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah menyelesaikan perangkat lunak simulasi *Schnorr Authentication* dan *Digital Signature Scheme*, penulis menarik kesimpulan sebagai berikut:

1. Hasil rancangan perangkat lunak dapat membantu pemahaman terhadap skema Schnorr, baik skema otentikasi maupun skema tanda tangan digital. Perangkat lunak dapat digunakan untuk mendukung kegiatan belajar mengajar, terutama dalam mata kuliah Kriptografi.
2. Perangkat lunak juga menyediakan layar teori yang berisi dasar-dasar teori dari algoritma-algoritma yang berada di dalam kedua skema Schnorr tersebut, sehingga dapat membantu pemahaman terhadap algoritma-algoritma tersebut.

5.2 Saran

Penulis ingin memberikan beberapa saran yang mungkin dapat membantu dalam pengembangan perangkat lunak ini yaitu :

1. Perangkat lunak dapat dikembangkan menjadi sebuah aplikasi *text editor* yang memberikan fasilitas tanda tangan digital (*digital signature*), sehingga diharapkan dapat digunakan oleh *user* dalam kehidupan sehari-hari.
2. Perangkat lunak dapat dikembangkan dengan menambahkan kriptanalisis terhadap skema Schnorr yang dibahas, sehingga dapat memberikan gambaran

mengenai kemanan yang diberikan oleh Schnorr. Sekuritas skema Schnorr berada pada permasalahan mencari logaritma diskrit. Pada proses pembentukan kunci, kunci publik (v) dihasilkan dengan rumus: $v = a^{-s} \bmod p$. Untuk mendapatkan kunci privat (s), maka penyerang (*attacker*) harus mampu menghitung $a^{-s} \equiv v \pmod p$. Ini merupakan permasalahan yang sulit dipecahkan.

DAFTAR PUSTAKA

Schneier, B., *Applied Crytography, Second Edition*, John Willey & Sons Inc., 1996.

Kurniawan, J., Kriptografi, Keamanan Internet dan Jaringan Komunikasi, Informatika, Bandung, 2004.

Munir, R., Kriptografi, Informatika, Bandung, 2006.

Andi, Panduan Praktis Pemograman Visual Basic 6.0 Tingkat Lanjut, Wahana Komputer, Yogyakarta, 2004.

Supardi, Y., Microsoft Visual Basic 6.0 Untuk Segala Tingkat, PT. Elex Media Komputindo, Jakarta, 2006.

Novian, A., Panduan MS. Visual Basic 6, Andi, Yogyakarta, 2004.

http://en.wikipedia.org/wiki/Schnorr_signature

http://en.wikipedia.org/wiki/Cryptographic_hash_function