STUDI PERBANDINGAN DAN IMPLEMENTASI ALGORITMA ELGAMAL DAN RSA UNTUK MELAKUKAN ENKRIPSI PESAN DAN PEMBUKTIAN OTENTIKASI PESAN SECARA BERSAMAAN

Yongke Yoswara and 13508034
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18034@students.if.itb.ac.id

***Abstract*—Saat ini algoritma kunci publik banyak diimplementasikan dalam kehidupan sehari-hari. Keuntungan dari algoritma kriptografi kunci publik adalah tidak diperlukan pengiriman kunci rahasia dan jumlah kunci dapat ditekan. Walaupun begitu, algoritma kriptografi kunci publik lebih lambat dalam melakukan enkripsi dan dekripsi pesan dibandingkan dengan algoritma kriptografi kunci privat. Walaupun masih berusia relatif muda, algoritma kunci publik mempunyai aplikasi yang sangat luas, antara lain untuk melakukan enkripsi/dekripsi pesan, tanda tangan digital, dan pertukaran kunci. Contoh algoritma untuk enkripsi/dekripsi pesan dan tanda tangan digital adalah algoritma ElGamal dan RSA.**

Kata kunci—Algoritma kunci publik, ElGamal, RSA, tanda tangan digital

I. PENDAHULUAN

Sekarang ini, teknologi berkembang sangat pesat. Hal serupa juga terjadi di dunia informatika. Perkembangan dunia informatika sangat cepat. Kemajuan dunia informatika juga mengakibatkan semakin berkembangnya teknologi informasi. Salah satu hal yang penting dalam teknologi informasi adalah nilai dari informasi itu sendiri. Nilai dari sebuah informasi sangat penting untuk sebagian orang yang tidak ingin isi dari informasi tersebut diketahui oleh orang yang tidak berhak melihatnya. Informasi itu disebarkan melalui media komunikasi yang tidak dapat kita ketahui dengan pasti keamanannya. Banyak media komunikasi yang tidak dapat menjamin keamanan data yang disalurkan melalui media tersebut. Contoh media komunikasi yang berkembang pesat sekarang ini adalah internet.

Oleh karena tidak adanya kepastian tentang keamanan suatu informasi yang disalurkan melalui media komunikasi, maka diperlukan metode yang dapat digunakan untuk menjaga isi dari informasi tersebut. Metode yang dimaksudkan adalah kriptografi. Kriptografi adalah sebuah seni dan bidang keilmuan dalam penyandian informasi atau pesan dengan tujuan menjaga keamanannya. Kriptografi sudah ada sejak jaman dahulu, namun seiring dengan berkembangnya teknologi, maka krptografi ini juga semakin berkembang. Perkembangan kriptografi ini dimakudkan supaya tidak ada yang bisa memecahkannya.

Awalnya, melakukan enkripsi dan dekripsi pesan pada kriptografi adalah dengan menggunakan sebauh kunci (disebut dengan kunci privat). Untuk melakukan enkripsi dan dekripsi pesan diperlukan kunci yang sama. Jasi untuk menjaga karahasiaan pesan, maka pengirim harus memberikan kunci rahasia tersebut kepada penerima melalui saluran yang benar-benar aman. Mengirimkan kunci privat pada saluran publik (seperti internet, pos, dll) sangat tidak aman. Oleh karena itu, kunci harus dikirim melalui saluran yang benar-benar aman. Jika kunci tidak dikirim melalui saluran yang benar-benar aman, maka penyadap dapat mengambil kunci tersebut dan melakukan dekripsi pada pesan tersebut.

Karena tidak ada jaminan bahwa kunci yang dikirim akan sampai di tangan penerima secara utuh dan tidak diketahui penyadap, maka sejak tahun 1970-an dikembangkan algoritma kriptografi kunci publik. Ide ini muncul pada tahun 1976. Pada kriptografi kunci publik, masing-masing pengirim dan penerima mempunyai sepasang kunci publik dan kunci privat. Kunci publik untuk mengenkripsi pesan sedangkan kunci privat digunakan untuk dekripsi pesan. Kunci publik dapat dikirim melalui saluran yang tidak aman. Dua keuntungan kriptografi kunci publik adalah tidak diperlukan pengiriman kunci secara rahasia dan jumlah kunci dapat ditekan.

Kriptografi kunci publik didasarkan pada fakta bahwa komputasi untuk enkripsi dan dekripsi pesan mudah dilakukan. Kemudian secara komputasi hamper tidak mungkin menurunkan kunci privat, d, bila diketahui kunci publik, e.

Walaupun begitu, ada beberapa kelemahan dalam algoritma kriptografi kunci publik tersebut. Kelemahannya antara lain adalah proses enkripsi dan dekripsi data umumnya lebih lambat dibandingkan dengan kriptografi simetri. Kemudian ukuran cipherteks yang dihasilkan dari proses enkripsi lebih besar daripada ukuran plainteksnya. Dan kelemahan lainnya adalah

ukuran kunci relatif lebih besar daripada ukuran kunci simetri.

Walaupun belum terlalu lama dikembangkan, algoritma kunci publik mempunyai aplikasi yang sangat luas. Contoh aplikasinya adalah untuk melakukan enkripsi/dekripsi pesan (algoritma ElGamal, RSA, Rabin). Kemudian aplikasi berikutnya adalah tanda tangan digital. Aplikasi ini bertujuan untuk membuktikan otentikasi data (contoh algoritma yang digunakan adalah RSA, ElGamal, DSA, GOST). Selain itu, algoritma kunci publik biasanya digunakan untuk mempertukarkan kunci simetri. Algoritma yang sering digunakan adalah Diffie-Hellman.

II. ELGAMAL

Algoritma ElGamal ditemukan oleh ilmuwan Mesir, yaitu Taher Elgamal pada tahun 1984, merupakan algoritma kriptografi kunci publik. Algoritma ElGamal terdiri atas 3 proses, yaitu pembentukan kunci, proses enkripsi, dan prosed dekripsi. Algoritma ini merupakan cipher blok, yaitu melakukan enkripsi pada blok-blok plainteks dan menghasilkan blok-blok cipherteks yang kemudian dilakukan proses dekripsi, dan hasilnya digabungkan kembali menjadi pesan yang utuh dan dapat dimengerti. Algoritma ElGamal mendasarkan kekuatannya pada fakta matematis kesulitan menghitung logaritma diskrit.

II.1 LOGARITSMA DISKRIT

Masalah logaritma diskrit adalah dengan memperhatikan hal berikut ini :

1. Jika diberikan suatu bilangan a, maka menghitung $b \equiv \alpha^a \pmod p$ adalah mudah
2. Tetapi jika diberukan suatu bilangan b, maka untuk menemukan a sehingga $b \equiv \alpha^a \pmod p$ adalah permasalahan yang sulit.

Selanjutnya bilangan a disebut sebagai logaritma diskrit terhadap b dengan basis α, dinyatakan dengan $a=\log_\alpha b$. Algoritma kriptografi pada umumnya menggunakan bilangan prima yang sangat besar, sehingga masalah logaritma diskrit pada penggunaannya dalam algoritma kriptografi meripakan suatu permasalahan yang sanagt sulit. Hal ini menjadi dasar kekuatan algoritma ElGamal.

II.2 PEMBENTUKAN KUNCI

Proses pertama dari algoritma ElGamal adalah pembentukan kunci yang terdiri atas kunci publik dan kunci rahasia. Pada proses ini dibutuhkan bilangan prima p yang digunakan untuk membentuk grup ℤP*. Kunci publik algoritma ElGamal berupa pasangan 3 bilangan, yaitu (p, α, dan b) dengan

$b = \alpha^a \pmod p$

Sedangkan kunci rahasianya adalah bilangan α tersebut.

Karena algoritma ElGamal menggunakan bilangan bulat dalam proses perhitungannya, maka pesan harus dikonversi ke dalam suatu bilangan bulat. Untuk mengubah pesan menjadi bilangan bulat, digunakan kode ASCII.

Berikut ini salah satu algoritma yang dapat digunakan untuk melakukan pembentuakn kunci.

- Masukan : bilangan prima aman p>255 dan elemen primitive α anggota ℤP*
- Keluaran : kunci publik dan kunci rahasia
- Langkah :

i. Pilih α anggota {0,1,…,p-2}
ii. Hitung $b \equiv \alpha^a \pmod p$
iii. Publikasikan nilai p, α, b. Serta rahasiakan nilai a.

II.3 PROSES ENKRIPSI

pada proses inim pesan dienkripsi menggunakan kunci publik dan sevarang bilangan acak rahasia k yang merupakan anggota himpunan {0,1,…,p-2}. Misalkan m adalah pesan yang akan dikirim. Selanjutnya m diubah ke dalam blok-blok karakter dan setiap karakter dikonversikan ke dalam kode ASCII, sehingga diperoleh plainteks $m_1,m_2,\ldots,m_n$. Untuk nilai ASCII bilangan 0 digunakan untuk menandai akhir dari suatu teks.

Proses enkripsi pada algoritma ElGamal dilakukan dengan menghitung

$$\gamma=\alpha^k mod\ p \quad \text{dan} \quad \delta=\beta^k.m\ mod\ p$$

dengan nilai k yang acak. Jadi diperoleh cipherteks (γ,δ)

Berikut ini adalah algoritma enkripsi:

- Masukan : pesan yang akan dienkripsi dan kunci publik (p, α, β).
- Keluaran : cipherteks (γ,δ).
- Langkah :

i. pesan dipotong-potong ke dalam bentuk blok-blok pesan dengan setiap blok adalah satu karakter pesan.
ii. konversikan masing-masing karakter ke dalam kode ASCII, maka diperoleh plainteks sebanyak n bilangan, yaitu $m_1,m_2,\ldots,m_n$
iii. Untuk I dari 1 sampai n kerjakan :
   a. pilih sebarang bilangan acak rahasia
   b. hitung $\gamma=\alpha^k mod\ p$
   c. hitung $\delta=\beta^k.m\ mod\ p$
iv. Diperoleh cipherteks yaitu (γ,δ), i = 1, 2, 3, .., n.

II.4 PROSES DEKRIPSI

Setelah menerima cipherteks, proses selanjutnya adalah melakukan dekripsi terhadap cipherteks tersebut menggunakan kunci publik p dan kunci rahasia a. Dapat ditunjukkan bahwa plainteks m dapat diperoleh dari dipherteks menggunakan kunci rahasia a.

Diberikan (p, α, β) sebagai kunci publik dan a sebagai kunci rahasia pada algoritma ElGamal. Jika diberikan cipherteks (γ,δ), maka

$$m=\delta.(\gamma^a)^{-1} mod\ p$$

dengan m adalah plainteks.

Berikut ini adalah algoritma dekripsi pada ElGamal.

- Masukan : cipherteks, kunci publik, kunci rahasia

- Keluaran : pesan asli (plainteks)
- Langkah :

i. untuk i dari 1 sampai n , lakukan langkah berikut :
   a. Hitung $\gamma_{p-1-a} mod\ p$
   b. Hitung $m_i = \delta . \gamma_{a-1} mod\ p$
ii. diperoleh plainteks m1,m2,...,mn
iii. Konverasikan masing-masing plainteks ke dalam karakter yang sesuai dengan kodenya. Kemudian hasilnya digabungkan.

III. RSA

Algoritma RSA diperkenalkan oleh tiga peneliti dari MIT, yaitu Ron Rivest, Adi Shamir, dan Len Addleman pada tahun 1976. RSA mendasarkan proses enkripsi dan dekripsi pada konsep bilangan prima dan aritmetika modulo. Baik kunci enkripsi dan dekripsi keduanyan merupakan bilangan bulat.

Kunci enkripsi tidak dirahasiakan dan diketahui umum (kunci publik), namun kunci untuk dekripsi bersifat rahasia. Kunci dekripsi dibangkitkan dari beberapa buah bilangan prima bersama-sama dengan kunci enkripsi. Untuk mendapatkan kunci dekripsi, orang harus memfaktorkan suatu bilangan non prima menjadi factor primanya. Kenyataannya, memfaktorkan bilangan non prima menjadi factor primanya bukanlah pekerjaan yang mudah. Semakin besar bilangan primanya tentu semakin sulit pula pemfaktorannya. Semakin sulit pemfaktoran, semakin kuat pula algoritma RSA.

III.1 Pembangkitan Pasangan Kunci

Dalam melakukan proses membangkitkan pasangan kunci, berikut langkah yang diperlukan :

- Masukan : 2 buah bilangan prima sebarang, sebut saja a dan b
- Keluaran : kunci publik dan kunci privat
- Langkah :

i. Hitung n=a.b dan hasil n tidak perlu dirahasiakan (cukup merahasiakan nilai a dan b).
ii. Hitung m=(a-1)(b-1). Apabila nilai m telah dihitung, maka nilai a dan b dapat dihapus untuk mencegah diketahui oleh pihak lain.
iii. Pilih sebuah bilangan bulat untuk kunci publik, yaitu e yang relative prima terhadap m.
iv. Hitung kunci dekripsi,d, dengan kekongruenan ed=1 (mod m)

III.2 Proses Enkripsi

Dalam melakukan proses enkripsi, setiap plainteks dipecah menjadi blok-blok plainteks. Setelah dipecah-pecah, setiap blok plainteks tersebut dienkripsi dengan persamaan berikut :

ci=pi^e mod n

dengan e adalah kunci publik.

III.3 Proses Dekripsi

Proses dekripsi pada algoritma RSA cukup sederhana. Proses dekripsi dapat dilakukan dengan menggunakan persamaan

pi=ci^d mod n

IV. Tanda Tangan Digital

Tanda tangan digital dikembangkan untuk mendukung keamanan data dalam bentuk otentikasi pesan yang didasari pada tanda tangan di dokumen kertas. Tanda tangan digital merupakan sebuah tanda tangan elektronik yang dapat digunakan untuk mengotentikasi identitas pengirim pesan dan dapat digunakan untuk menjadmin keaslian dokumen atau pesan yang dikirimkan. Tanda tangan digital dapat dengan mudah dipindahkan, tidak dapat dipalsukan, dan memiliki time-stamp.

Skema tanda tangan digital harus mampu menangani keabsahan pengirim, keaslian pesan, dan anti penyanggahan. Pengecekan keabsahan pengirim dilakukan dengan melakukan proses verifikasi. Keaslian pesan berkaitan dengan keutuhan pesan. Jika terjadi modifikasi pesan atau modifikasi pada bagian tanda tangan digital akan menyebabkan kehahalan otentikasi. Anti penyanggahan menjamin pengirim tidak dapat menyangkal tentang isi pesan atau fakta bahwa ia yang mengirimkan pesan. Jika keabsahan pengirim dan keaslian pesan telah berhasil diverifikasi, maka pengirim tidak dapat menyanggah pesan yang dikirim.

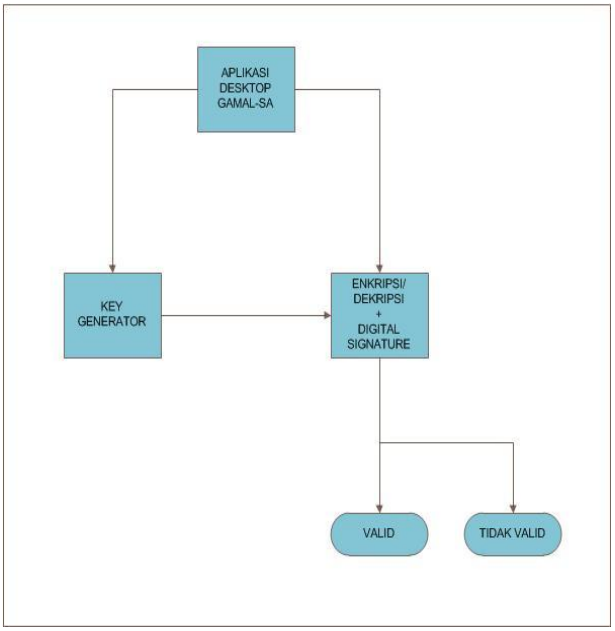
VI. Gamal-SA

Gamal-SA adalah sebuah perangkat lunak yang dikembangkan oleh penulis. Dalam perangkat lunak, ada dua pilihan algoritma yang dapat digunakan. Yang pertama adalah algoritma RSA dan yang kedua adalah algoritma ElGamal. Perangkat lunak ini dinamakan dengan Gamal-SA karena dapat menggunakan algoritma ElGamal atau juga RSA untuk melakukan enkripsi/dekripsi pesan dan otentikasi pesan.

Setelah mengetahui algoritma ElGamal dan RSA secara lebih spesifik, kemudian dilakukan implementasi pada sebuah aplikasi desktop. Aplikasi desktop tersebut dibagi menjadi dua. Yang pertama adalah aplikasi desktop untuk membangkitkan kunci (key generator). Dan yang kedua adalah aplikasi desktop yang akan melakukan enkripsi pesan dan tanda tangan digital secara bersamaan ketika menekan sebuah tombol.

Selain itu, aplikasi desktop ini nantinya dapat melakukan validasi pesan / file. Validasi pesan ini dilakukan dengan melihat keutuhan/keaslian tanda tangan digital yang sudah dimasukkan ke dalam pesan tersebut.

Berikut ini adalah arsitektur yang menggambarkan program tersebut.



Gambar 1 Gambar Arsitektur Program Gamal-Sa

Perangkat lunak yang digunakan untuk mengembangkan aplikasi Gamal-SA adalah Microsoft Visual Studio 2010 dengan bahasa pemrograman C#.

VI. IMPLEMENTASI ALGORITMA ELGAMAL DAN RSA

Berikut ini adalah rancangan scenario yang terjadi dalam penggunaan perangkat lunak Gamal-SA

1. Skenario pembangkitan kunci
   1.1. Bob sebagai pengirim pesan melakukan pembangkitan kunci publik dan kunci privat menggunakan aplikasi Key Generator Gamal-SA
   1.2. Bob menyebarkan kunci publik kepada siapa saja (umum), termasuk juga Alice yang bertindak sebagai penerima pesan.
   1.3. Bob menyimpan kunci privat dan menjaganya dengan baik.
2. Skenario Enkripsi Dokumen
   2.1. Alice hendak mengirimkan sebuah dokumen / pesan kepada Bob tetapi Alice ingin pesan tersebut tidak diketahui oleh pihak lain. Dia ingin hanya Bob saja yang mengetahuinya. Jadi tidak ada orang lain yang mengetahuinya. Oleh sebab itu, Alice akan mengenkripsi pesan tersebut menjadi sebuah cipherteks.
   2.2. Alice melakukan enkripsi pesan dengan menggunakan perangkat lunak yang ada dan menggunakan kunci publik yang telah di-generate oleh Bob tadi.
   2.3. Alice mengirimkan dokumen cipherteks kepada Bob.
3. Skenario Dekripsi Dokumen
   3.1. Bob melakukan dekripsi dokumen dengan menggunakan perangkat lunak Gamal-SA.
   3.2. Dekripsi dokumen tersebut dengan menggunakan kunci privat yang telah dimilikinya.
   3.3. Bob dapat membaca dokumen/pesan yang diberikan Alice kepadanya.
4. Skenario Tanda Tangan Digital Dokumen.
   4.1. Bob ingin mengirimkan dokumen rahasia untuk dibaca oleh Alice. Tetapi Bob ingin dokumen tersebut terbukti keotentikannya. Jadi Bob akan melakukan proses enkripsi pesan kemudian diberikan tanda tangan digital. Hal ini bertujuan untuk menjaga keamanan dokumen/pesan. Supaya dokumen tersebut tidak terjadi perubahan sama sekali. Bob berkesimpulan untuk menandatangani secara digital terlebih dahulu sebelum dikirimkan kepada Alice.
   4.2. Bob melakukan enkripsi pesan bersamaan dengan menambah tanda tangan digital pada dokumen yang akan dikirim dengan menggunakan perangkat lunak Gamal-SA.
   4.3. Bob mengirimkan dokumen tersebut kepada Alice.
5. Skenario Verifikasi Dokumen
   5.1. Alice menerima dokumen dari Bob yang telah ditandatangani secara digital oleh Bob dan dilakukan proses enkripsi.
   5.2. Alice malakukan veridikasi dan dekripsi pesan menggunakan perangkat lunak Gamal-SA. Untuk melakukan verifikasi ini, Alice menggunakan kunci publik yang diberikan oleh Bob untuk mengetahui keotentikan dokumen cipherteks tersebut.

VI.1 PEMBUBUHAN TANDA TANGAN DIGITAL DENGAN ELGAMAL

Proses pembubuhan tanda tangan digital pada ElGamal berbeda dengan RSA (pembubuhan tanda tangan digital pada RSA akan dijelaskan di sub-bab berikutnya). Pda algoritma ElGamal, telah disediakan sebuah mekanisme khusus untuk pembubuhan tanda tangan digital, sehingga berkas tidak perlu dihitung nilai hash-nya, tetapi langsung mebgikuti skema ElGamal yang telah disediakan.

Misalkan sebuah bilangan x akan ditandatangan digital, dengan menggunakan kriptosistem ElGamal dengan $\kappa = (p,\alpha,a,\beta)$ dan sebuah bilangan random $k \in Zp-1^*$ yang bersifat rahasia, didefinisikan proses signature dokumen tersebut adalah sebagai berikut :

$$sig_{\kappa}(x,k) = (\gamma,\delta),$$

dimana

$$\gamma = \alpha^{k} \bmod p$$

dan

$$\delta = (x - a\gamma)k^{-1} \bmod (p-1).$$

Proses signature dengan kriptosistem ElGamal menghasilkan dua bilangan γ dan δ. Skema signature ElGamal bersifat non-deterministik, begitu pula dengan kriptosistem kunci publik ElGamal. Hal ini berarti terdapat banyak yanda yangan digital yang valid untuk sebuah pesan masukan. Algoritma verifikasi harus mampu

menerima semua masukan tanda tangan yang valid sebagai tanda tangan yang otentik.

VI.1.1 Verifikasi Tanda Tangan dengan ElGamal

Proses verifikasi tanga tangan digital dengan algoritma ElGamal menerima 3 masukan, yaitu pesan(x), γ, dan δ. Proses verifikasi dilakukan dengan mencocokan hasil dari dua proses penghitungan yang dijelaskan dalam skema berikut:

$$ver_\kappa(x,\gamma,\delta) = true \longleftrightarrow \beta^{\gamma}\gamma^{\delta} \equiv \alpha^{x} (mod\ p).$$

Skema verifikasi di atas dapat dibuktikan. Jika tanda tangan digital dibangun dengan benar, verifikasi akan sukse berjalan sebab

$$\beta^{\gamma}\gamma^{\delta} \equiv \alpha^{a\gamma}\alpha^{k\delta} (mod\ p)$$
$$\equiv \alpha^{x} (mod\ p),$$

dimana digunakan fakta bahwa

$$a\gamma + k\delta \equiv x\ (mod\ p-1)$$

VI.1.2 Implementasi Tanda Tangan Digital dengan ElGamal.

Parameter yang dibangkitkan program berukuran 256 bit dengan tujuan untuk meningkatkan keamanan tanda tangan digital. Karena bilangan yang digunakan termasuk bilangan besar dan tidak ada kelas primitive C# yang mampu menanganinya, digunakan sebuah kelas bantuan BigInteger yang mampu menangani penyimpanan bilangan-bilangan besar dan menyediakan fungsi-fungsi serta prosedur komputasi yang diperlukan.

Secara garis besar, program tanda tangan digital berjalan sebagai berikut :

a) Program membangkitkan parameter ElGamal, yaitu p, α, dan a.
b) Program membaca berkas yang akan ditandatangani sebagai masukan dari user.
c) Program menandatangi berkas dengan menggunakan parameter yang telah dimiliki dan menghasilkan sebuah berkas cipherteks yang berisi tanda tangan digital.
d) Program melakukan verifikasi tanda tangan digital dengan cara membandingkan dua buah proses komputasi dimana jika hasilnya sama, maka verifikasi berhasil dan jika tidak maka akan gagal.

VI.2 Pembubuhan Tanda Tangan Digital dengan RSA

Pada mulanya, pengirim pesan menghitung message digest dari pesan (M), yang diperoleh dengan mentransformasikan pesan(M) dengan menggunakan fungsi hash (H).

Selanjutnya message digest dienkripsi dengan algoritma kriptografi RSA kunci publik dan menggunakan kunci privat(SK) si pengirim. Hasil enkripsi inilah yang disebut dengan tanda tangan digital(S).

$$S = E_{SK}\ (MD)$$

VI.2.1 Verifikasi Tanda Tangan Digital Dengan RSA

Proses verifikasi tanda tangan digital dengan algoritma RSA adalah dengan cara berikut :

a) Tanda tangan digital(S) didekripsi dengan menggunakan kunci publik (PK) pengirim pesan, menghasilkan message digest(MD) semula, sebagai berikut:

$$MD = D_{PK}(S)$$

b) Penerima kemudian mengubah pesan(M) menjadi message digest(MD') menggunakan fungsi hash satu arah yang sama dengan fungsi hash yang dikirim oleh pengirim.
c) Jika MD = MD', berarti tanda tangan digital yang diterima otentik dan berasal dari pengirim yang benar.

VI.2.2 Implementasi Tanda Tangan Digital dengan RSA

Implementasi pada tanda tangan digital dengan menggunakan algoritma RSA tidak jauh berbeda prosedur kerjanya dengan implementasi pada alhoritma ElGamal. Pada algoritma RSA tetap dibutuhkan sebuah kelas BigInteger yang mampu menangani penyimpanan bilangan-bilangan besar dan menyediakan fungsi-fungsi serta prosedur komputasi yang diperlukan.

Secara garis besar, program tanda tangan digital dengan RSA berjalan sebagai berikut:

a) Program membangkitkan parameter
b) Program membaca pesan/berkas yang akan ditandatangani. Dalam hal ini, berkas /pesan yang ditandatangi sudah dilakukan proses enkripsi dengan RSA terlebih dahulu.
c) Program menandatangi berkas dengan parameter yang telah dimiliki.
d) Program melakukan berifikasi tanda tangan digital dengan membandingkan nilai hash dari dua buah berkas tersebut.

VI.3 Enkripsi / Dekripsi dengan ElGamal

Proses enkripsi dan dekripsi algoritma ElGamal sudah dijelaskan di bagian atas. Di bagian ini, akan dijelaskan lebih lanjut mengenai proses enkripsi dan dekripsi yang ada pada perangkat lunak yang telah dikembangkan.

VI.3.1 Enkripsi Berkas/Pesan ElGamal

Dalam perangkat lunak Gamal-SA, proses enkripsi pesan adalah sebagai berikut:

a) Pengirim pesan akan melakukan pembangkitan kunci terlebih dahulu (melalui key generator).
b) Proses enkripsi menggunakan kunci publik yang telah dibangkitkan dan sebuah bilangan integer acak k ($k \in \{0,1,..., p-1\}$) yang dijaga kerahasiaanya oleh penerima yang

mengenkripsi pesan.

c) Setiap karakter pada pesan akan ditransformasikan terlebih dahulu ke dalam kode ASCII sehingga diperoleh bilangan bulat M.
d) Hitung nilai r dan t dengan persamaan berikut :

$$r = \alpha^k \pmod p$$
$$t = \beta^k M \pmod p$$

e) Diperoleh cipherteks untuk karatker M tersebut dalam blok (r,t)
f) Proses di atas dilakukan untuk seluruh karakter dalam pesan termasuk karakter spasi.

VI.3.2 DEKRIPSI BERKAS/PESAN ELGAMAL

Dekripsi dari cipherteks ke plainteks menggunakan kunci rahasia a yang disimpan kerahasiaanya oleh penerima pesan.

Berikut ini adalah langkah-langkah dalam melakukan proses dekripsi:

a) Ambil sebuah blok cipherteks dari pesan yang telah dienkripsi
b) Dengan menggunakan a yang dirahasiakan oleh penerima, plainteks didapatkan dengan persamaan berikut ;

$$\gamma_{p-1-a} \, mod \, p$$
$$m_i = \delta . \gamma_{a-1} mod \, p$$

c) m_i adalah setiap blok plainteks yang dihasilkan
d) Lakukan langkah di atas sampai semua blok cipherteks didekripsi.

VI.4 ENKRIPSI/DEKRIPSI DENGAN RSA

Dalam algoritma RSA, ada 7 properti yang dimiliki oleh RSA. Property tersebut adalah sebagai berikut :

- p dan q bilangan prima (rahasia)
- $n = p \cdot q$ (tidak rahasia)
- $\phi(n) = (p-1)(q-1)$ (rahasia)
- e (kunci enkripsi) (tidak rahasia)
- Syarat: PBB(e, $\phi(n)$) = 1
- d (kunci dekripsi) (rahasia)
- d dihitung dari $d \equiv e^{-1} \bmod (\phi(n))$
- m (plainteks) (rahasia)
- c (cipherteks) (tidak rahasia)

VI.4.1 ENKRIPSI BERKAS/PESAN RSA

Untuk setiap enkripsi berkas/pesan dengan menggunakan RSA, langkah yang diperlukan adalah ;

a) Sebuah plainteks dibagi ke dalam blok-blok plainteks.
b) Untuk setiap blok, hitung cipherteks yang dihasilkan dengan menggunakan persamaan berikut:

$$c_i = m_i^e \textbf{ mod } n$$

dimana e adalah kunci publik

VI.4.2 DEKRIPSI BERKAS/PESAN RSA

Proses dekripsi pada RSA cukup sederhana. Untuk melakukan proses dekripsi, digunakan persamaan

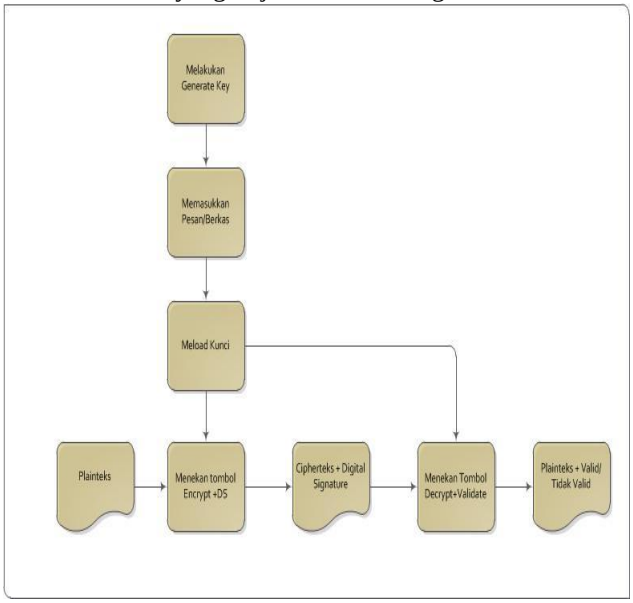
$$m_i = c_i^d \textbf{ mod } n,$$

dimana d adalah kunci privat.

VII PERANCANGAN GAMAL-SA

Dalam melakukan perancangan Gamal-SA, ketika kita menekan sebuah tombol Encrypt+DS maka pesan/berkas akan dienkripsi oleh Gamal-SA dan juga ditambahkan tanda tangan digital secara bersamaan.

Jadi scenario yang terjadi adalah sebagai berikut;



Gambar 2 Gambar Skenario yang dapat dijalankan pada Gamal-SA

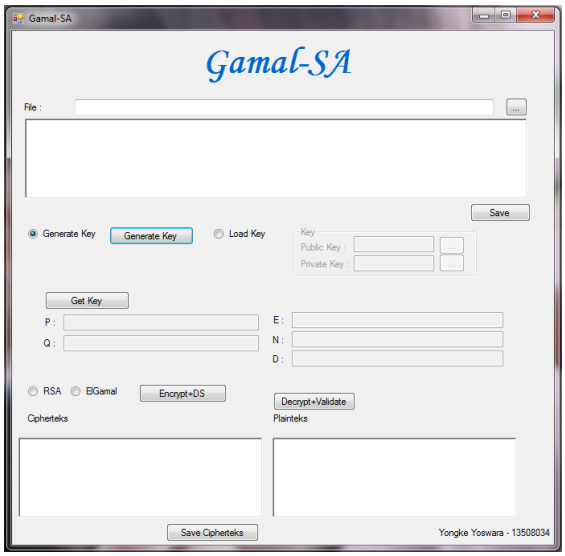
Berikut ini adalah deskripsi kelas yang ada dalam Gamal-SA

Tabel 1 Dekripsi Kelas Gamal-SA

Nama	Deskripsi
BigInteger	Kelas untuk biginteger. Dibutuhkan kelas ini karena angka yang digunakan sangat besar
RSA	Kelas yang berisi method untuk enkripsi/dekripsi RSA + tanda tangan digital
ElGamal	Kelas yang berisi method untuk enkripsi/dekripsi ElGamal + tanda tangan digital
SHA	Kelas untuk SHA
Form1	Form untuk melakukan enkripsi/dekripsi Gamal-SA
Form2	Form untuk melakukan generate key

Berikut ini adalah rancangan antarmuka yang ada.

Gambar 3 Gambar Antarmuka untuk Generate Kunci



Gambar 4 Gambar Antarmuka Program Utama

Berikut ini adalah kode dari algoritma RSA dan ElGamal yang digunakan dengan menggunakan bahasa C#

Enkripsi dengan RSA

```
publik string Encrypt(byte[] plain)
{
    string result = "";
    //GenerateKey();
    string plainASCII = ByteToASCIIString(plain);
    string[] AS = StringToArrOfString(plainASCII, n.ToString().Length - 1);
    string[] cipher = new string[AS.Length];

    if (AS[AS.Length - 1].Length < AS[0].Length)
    {
        for (int i = 0; i < (AS[0].Length - AS[AS.Length - 1].Length); i++)
        {
            AS[AS.Length - 1] = "0" + AS[AS.Length - 1];
        }
    }

    for (int i = 0; i < AS.Length; i++)
    {
        BigInteger a = new BigInteger(AS[i], 10);
        BigInteger b = new BigInteger(a.modPow(Enc, n));
        BigInteger cip = new BigInteger(b.ToString(), 10);
        cipher[i] = cip.ToHexString();
    }

    for (int i = 0; i < cipher.Length; i++)
    {
        if (i != cipher.Length - 1)
        {
            result += cipher[i] + " ";
        }
        else
        {
            result += cipher[i];
        }
    }
    return result;
}
```

Dekripsi dengan RSA

```
publik byte[] Decrypt(string ciphertext)
{
    string result = "";
    //GenerateKey();
    string[] AS = ciphertext.Split(' ');
    string[] plain = new string[AS.Length];
    for (int i = 0; i < AS.Length; i++)
    {
        BigInteger a = new BigInteger(AS[i], 16);
        BigInteger b = new BigInteger(a.modPow(Dec, n));
        if (b.ToString().Length < n.ToString().Length - 1)
        {
            string temp = b.ToString();
            for (int j = 0; j < (n.ToString().Length - 1 - b.ToString().Length); j++)
            {
                temp = "0" + temp;
            }
            plain[i] = temp;
        }
        else
        {
            plain[i] = b.ToString();
        }
    }
    for (int i = 0; i < plain.Length; i++)
    {
        result += plain[i];
    }
    string[] rs = StringToArrOfString(result, 3);
    byte[] res = new byte[rs.Length];
    for (int i = 0; i < rs.Length; i++)
    {
        res[i] = (byte)(int.Parse(rs[i]));
    }
    return res;
}
```

Tanda Tangan Digital RSA

```
mySHA1 sha = new mySHA1();
byte[] result = sha.ComputeHash(input);
//hashOutput = BitConverter.ToString(result).Replace("-", "");

RSA tempRSA = new RSA();
tempRSA.SetEnc(E);
tempRSA.Setn(N);

byte[] hasil = changeStringtoByte("<ds>" + tempRSA.Encrypt(result) + "</ds>");
byte[] ditulis = new byte[input.Length + hasil.Length];
for (int i = 0; i < input.Length; i++)
{
    ditulis[i] = input[i];
}
for (int i = input.Length; i < hasil.Length + input.Length; i++)
```

```
            {
                ditulis[i] = hasil[i - input.Length];
            }
```

Enkripsi dengan ElGamal

```
    protected override byte[] ProcessDataBlock(byte[] p_block) {
        // the random number, K
        BigInteger K;
        // create K, which is the random number
        do {
            K = new BigInteger( );
            K.genRandomBits(o_key_struct.P.bitCount( ) -1,
o_random);
        } while (K.gcd(o_key_struct.P -1) != 1);

        // compute the values A and B
        BigInteger A = o_key_struct.G.modPow(K, o_key_struct.P);
        BigInteger B = (o_key_struct.Y.modPow(K, o_key_struct.P)
            * new BigInteger(p_block)) % (o_key_struct.P);

        // create an array to contain the ciphertext
        byte[] x_result = new byte[o_ciphertext_blocksize];
        // copy the bytes from A and B into the result array
        byte[] x_a_bytes = A.getBytes( );
        Array.Copy(x_a_bytes, 0, x_result, o_ciphertext_blocksize / 2
            - x_a_bytes.Length, x_a_bytes.Length);
        byte[] x_b_bytes = B.getBytes( );
        Array.Copy(x_b_bytes, 0, x_result, o_ciphertext_blocksize
            - x_b_bytes.Length, x_b_bytes.Length);
        // return the result array
        return x_result;
    }

    publik override byte[] EncryptData(byte[] p_data) {
        if (NeedToGenerateKey( )) {
            // we need to create a new key before we can export
            CreateKeyPair(KeySizeValue);
        }
        // encrypt the data
        ElGamalEncryptor x_enc = new
ElGamalEncryptor(o_key_struct);
        return x_enc.ProcessData(p_data);
    }
```

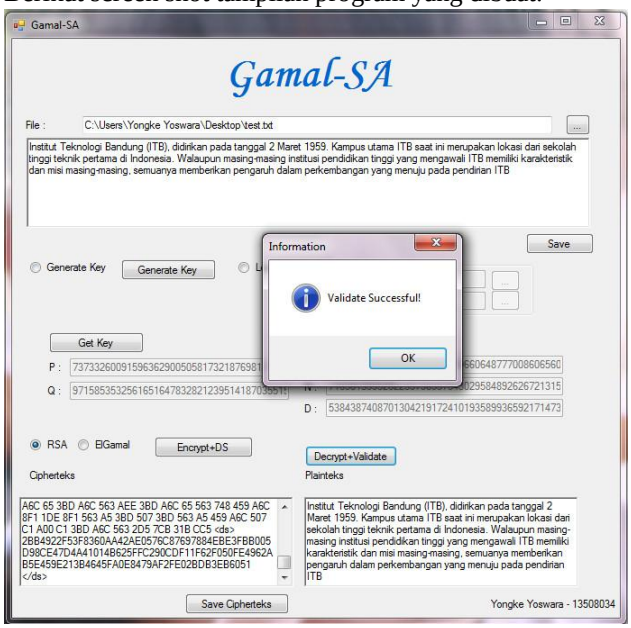
Dekripsi dengan ElGamal

```
    protected override byte[] ProcessDataBlock(byte[] p_block) {
        // extract the byte arrays that represent A and B
        byte[] x_a_bytes = new byte[o_ciphertext_blocksize / 2];
        Array.Copy(p_block, 0, x_a_bytes, 0, x_a_bytes.Length);
        byte[] x_b_bytes = new Byte[o_ciphertext_blocksize / 2];
        Array.Copy(p_block, x_a_bytes.Length, x_b_bytes, 0,
x_b_bytes.Length);
        // create big integers from the byte arrays
        BigInteger A = new BigInteger(x_a_bytes);
        BigInteger B = new BigInteger(x_b_bytes);
        // calculate the value M
        BigInteger M = (B *
            A.modPow(o_key_struct.X,
o_key_struct.P).modInverse(o_key_struct.P))
            % o_key_struct.P;
        // return the result - take care to ensure that we create
        // a result which is the correct length
        byte[] x_m_bytes = M.getBytes( );
        // we may end up with results which are short some leading
        // bytes - add these are required
        if (x_m_bytes.Length < o_plaintext_blocksize) {
            byte[] x_full_result = new byte[o_plaintext_blocksize];
            Array.Copy(x_m_bytes, 0, x_full_result,
                o_plaintext_blocksize - x_m_bytes.Length,
x_m_bytes.Length);
            x_m_bytes = x_full_result;
        }
        return x_m_bytes;
    }

    publik override byte[] DecryptData(byte[] p_data) {
        if (NeedToGenerateKey( )) {
            // we need to create a new key before we can export
            CreateKeyPair(KeySizeValue);
        }
        // encrypt the data
        ElGamalDecryptor x_enc = new
ElGamalDecryptor(o_key_struct);
        return x_enc.ProcessData(p_data);
    }
```

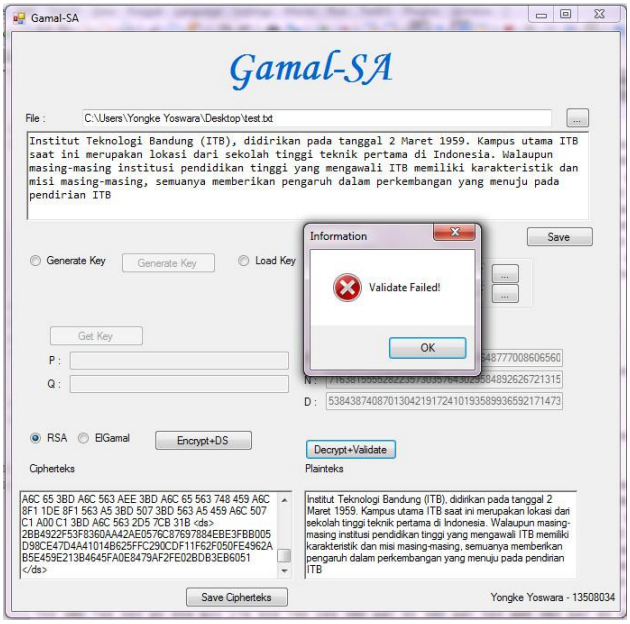
Tanda Tangan Digital dengan ElGamal

```
    DSACryptoServiceProvider MySigner = new
DSACryptoServiceProvider();

    FileStream file = new FileStream(args[0], FileMode.Open,
FileAccess.Read);
    BinaryReader reader = new BinaryReader(file);
    byte[] data = reader.ReadBytes((int)file.Length);

    byte[] signature = MySigner.SignData(data);

    string publikKey = MySigner.ToXmlString(false);
    Console.WriteLine("Signature: " +
Convert.ToBase64String(signature));
    reader.Close();
    file.Close();

    DSACryptoServiceProvider verifier = new
DSACryptoServiceProvider();

    verifier.FromXmlString(publikKey);

    FileStream file2 = new FileStream(args[0], FileMode.Open,
FileAccess.Read);
    BinaryReader reader2 = new BinaryReader(file2);
    byte[] data2 = reader2.ReadBytes((int)file2.Length);
```

Berikut screen shot tampilan program yang dibuat:



Gambar 5 Gambar Tampilan Ketika Validasi Berhasil



Gambar 6 Gambar Tampilan Ketika Salah Validasi (sebuah blok pada pesan terenkripsi dihapus)

VIII. KESIMPULAN

Ide dari algoritma kriptografi kunci publik muncul pada tahun 1976. Namun pada saat itu belum ditemukan algoritma kriptografi kunci publik yang sesungguhnya. Baru beberapa saat kemudian sampai sekarang ditemukan beragam algoritma kriptografi kunci publik. Kriptografi kunci publik / nirsimetri memiliki sepasang kunci, yaitu kunci publik dan kunci privat. Kunci publik adalah kunci yang digunakan untuk melakukan enkripsi pesan. Sedangkan kunci privat digunakan untuk mendekripsi pesan. Kunci publik dapat dikirim melalui saluran yang tidak perlu aman. Saluran yang tidak perlu aman ini mungkin sama dengan saluran yang digunakan untuk mengirim pesan.

Kelebihan kriptografi kunci publik antara lain adalah kita hanya perlu menjaga kunci privat, pasangan kunci publik / kunci privat tidak perlu diubah, dan tidak diperlukan pengiriman kunci rahasia. Kriptografi kunci publik saat ini banyak sekali digunakan di berbagai aplikasi, antara lain adalah dalam melakukan enkripsi/dekripsi pesan, tanda tangan digital, dan pertukaran kunci.

Ketiga aplikasi tersebut digunakan untuk menjaga kerahasiaan data, keotentikan data, keaslian pesan, dan anti penyangkalan. Untuk memperkuat kerahasiaan dan keaslian serta keotentikan pesan, maka dibuat sebuah perangkat lunak bernama Gamal-SA. Perangkat lunak ini menggunakan algoritma ElGamal/RSA untuk melakukan enkripsi pesan beserta pembubuhan tanda tangan digital secara otomatis. Hal ini bertujuan untuk meningkatkan keamanan dari pesan supaya tidak diubah oleh pihak yang tidak bertanggung jawab.

Jadi Gamal-SA diharapkan dapat meningkatkan keamanan pesan yang akan dikirim. Beberapa kesimpulan yang dapat diambil dari algoritma ElGamal dan RSA ketika diimplementasikan ke dalam Gamal-SA adalah

a) Algoritma ElGamal merupakan algoritma kriptografi yang lengkap, karena selain dapat digunakan untuk proses enkripsi.dekripsi pesan juga dapat dipakai untuk tanda tangan digital.
b) Penggunaan kunci pada algoritma ElGamal dengan panjang lebih besar membutuhkan waktu transformasi lebih lama dibandingkan.
c) Keamanan algoritma ElGamal ini terletak pada sulitnya menghitung logaritma diskrit
d) Properti yang dimiliki oleh ElGamal lebih sedikit dibandingkan dengan properti yang dimiliki RSA.
e) Algoritma ElGamal membutuhkan waktu yang lebih sedikit (efisien) dibandingkan dengan RSA untuk berkas/pesan yang dienkripsi dan dibubuhi tanda tangan digital.
f) Ukuran tandatangan pada ElGamal 2 (dua) kali ukuran tandatangan digital dengan RSA.
g) Algoritma RSA merupakan algoritma yang paling terkenal dan banyak pengaplikasiannya
h) Keamanan dari algoritma RSA adalah sulitnya memfaktorkan bilangan yang besar menjadi factor prima
i) Autentikasi dengan algoritma RSA lebih simple dibandingkan dengan ElGamal.

IX. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa karena atas rahmat-Nya, penulis dapat menyelesaikan makalah ini. Terima kasih juga penulis ucapkan kepada Bapak Rinaldi Munir, selaku dosen Kriptografi semester genap tahun ajaran 2010/2011 atas bimbingan yang diberikan selama ini. Makalah ini juga dapat terselesaikan berkat kuliah yang disampaikan oleh Bapak Rinaldi Munir.

REFERENCES

[1] http://en.wikipedia.org/wiki/ElGamal_encryption
Tanggal Akses: 29 April 2011, 17:00 WIB
[2] http://www.iusmentis.com/technology/encryption/ElGamal/
Tanggal Akses: 29 April 2011, 18:00 WIB
[3] http://www.math.uic.edu/~leon/mcs425-s08/handouts/el-gamal.pdf
Tanggal Akses: 29 April 2011, 21:30 WIB.
[4] http://www.informatika.org/~rinaldi/Kriptografi/kriptografi.htm
Tanggal Akses: 30 April 2011, 14:00 WIB
[5] http://en.wikipedia.org/wiki/RSA
Tanggal Akses : 30 April 2011, 18.00 WIB
[6] http://www.di-mgt.com.au/rsa_alg.html
Tanggal Akses: 2 Mei 2011, 18:30 WIB
[7] http://www.rsa.com/rsalabs/node.asp?id=2146
Tanggal Akses: 2 Mei 2011, 13:30 WIB.

PERNYATAAN

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

Bandung, 9 Mei 2011

Yongke Yoswara - 13508034