Implementasi *Digital Signature* dengan Algoritma ElGamal dan MD5 pada Aplikasi Login Sederhana serta Perbandingannya dengan *Digital Signature* yang Menggunakan Algoritma RSA dan SHA-1

Muhammad Ghufron Mahfudhi / 13508020
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18020@students.if.itb.ac.id

***Abstrak*—*Digital signature* memiliki fungsi yang sama dengan tanda tangan pada dokumen secara umum. *Digital signature* dapat memberi keaslian terhadap suatu dokumen digital. Untuk membuat suatu *digital signature* dapat menggunakan algoritma yang menggunakan fungsi hash dan algoritma kunci publik. Contohnya yaitu algoritma GamalD5 yang merupakan gabungan algoritma ElGamal dan fungsi hash MD5 atau SHARSA yang merupakan gabungan algoritma RSA dan fungsi hash SHA-1. Kedua algoritma tersebut pasti mempunyai kelebihan dan kelemahan masing-masing. Kelebihan dari GamalD5 yaitu memiliki waktu yang cukup singkat dan kompleksitas isi hasil enkripsinya, sedangkan kelebihan dari SHARSA yaitu mempunyai kompleksitas algoritma yang tinggi dan ukuran hasil enkripsi yang lebih kecil. Dengan kelebihan tersebut, kedua algoritma tersebut dapat digunakan sesuai apa yang diprioritaskan. Akan tetapi, dengan menggabungkan kedua segi positif dari algoritma tersebut akan mendapatkan algoritma yang lebih baik. Sama halnya dengan *digital signature*. Dengan menggabungkan beberapa algoritma yang berbeda dengan kelebihannya masing-masing, akan mendapatkan algoritma *digital signature* baru yang labih baik dan aman.**

***Kata Kunci*—Digital Signature, ElGamal, MD5, RSA, SHA-1.**

I. PENDAHULUAN

Digital signature memiliki fungsi sebagai media otentikasi untuk mengecek keaslian pesan dan media anti penyangkalan. Jika kita melihat arti *signature* yang berarti tanda tangan, tanda tangan adalah bukti yang otentik yang tidak dapat dilupakan dan tidak dapat dipindah untuk digunakan ulang. Dokumen yang telah ditandatangani tidak dapat diubah dan tanda tangan pada dokumen tersebut tidak dapat disangkal (*repudiation*). *Digital signature* merupakan bentuk penerapan tanda tangan tersebut pada data digital.

Tanda tangan digital adalah nilai kriptografis yang bergantung pada isi pesan dan kunci. Tanda tangan pada dokumen cetak selalu sama, apa pun isi dokumennya. Akan tetapi, tanda tangan digital selalu berbeda-beda antara satu isi dokumen dengan dokumen lain.

Digital signature yang banyak digunakan adalah *Digital Signature Standard* (DSS) yang merupakan *digital signature* yang telah disepakati bersama untuk menjadi patokan (*standard*). *Digital signature* tersebut menggunakan fungsi hash SHA-1 dan algoritma *digital signature* yang merupakan pengembangan dari algoritma ElGamal. Akan tetapi, kita juga dapat membuat *digital signature* dengan algoritma lain seperti fungsi hash MD5 atau algoritma RSA.

Di antara algoritma *digital signature* pasti mempunyai kelebihan dan kekurangan masing-masing. Karena DSS sudah dijadikan standar, pasti kemangkusannya juga sudah terjamin. Oleh karena itu, dalam makalah ini akan dibahas mengenai modifikasi *digital signature* yang menggunakan algoritma ElGamal dan fungsi hash MD5. Sebagai perbandingan, akan dibahas juga mengenai modifikasi *digital signature* dengan menggunakan algoritma RSA dan fungsi hash SHA-1. Dengan begitu, dapat diketahui apakah *digital signature* dengan fungsi MD5 dan algoritma ElGamal itu lebih baik dari pada *digital signature* dengan fungsi SHA-1 dan algoritma RSA atau tidak.

II. ALGORITMA PADA *DIGITAL SIGNATURE*

A. Algoritma ElGamal

Keamanan algoritma ini terletak pada sulitnya menghitung logaritma diskrit. Masalah logaritma diskrit yaitu jika p adalah bilangan prima serta g dan y adalah sembarang bilangan bulat, harus mencari x sedemikian sehingga $g^x \equiv y \pmod p$.

Pada algoritma ElGamal terdapat tiga tahapan utama, yaitu pembangkitan kunci, enkripsi, dan dekripsi. Proses pembangkitan kunci antara lain:

1. Memilih sembarang bilangan prima *p* (*p* dapat di-*share* di antara anggota kelompok).

2. Memilih dua buah bilangan acak, g dan x, dengan syarat $g < p$ dan $1 \le x \le p - 2$.
3. Hitung $y = g^x \bmod p$.

Hasil dari algoritma ini yaitu:

- Kunci publik: tripel (y, g, p)
- Kunci privat: pasangan (x, p)

Proses enkripsi pada algoritma ElGamal antara lain:

1. Menyusun plainteks menjadi blok-blok m_1, m_2, …, dengan nilai setiap blok di dalam selang $[0, p - 1]$.
2. Memilih bilangan acak k, dengan syarat $1 \le k \le p - 2$.
3. Setiap blok m dienkripsi dengan rumus
   $a = g^k \bmod p$
   $b = y^k m \bmod p$

Pasangan a dan b adalah cipherteks untuk blok pesan m. Jadi, ukuran cipherteks dua kali ukuran plainteksnya.

Proses dekripsi pada algoritma ElGamal antara lain:

1. Menggunakan kunci privat x untuk menghitung:
   $(a^x)^{-1} = a^{p-1-x} \bmod p$
2. Hitung plainteks m dengan persamaan:
   $m = b/a^x \bmod p = b(a^x)^{-1} \bmod p$

B. Fungsi Hash MD5

MD5 adalah fungsi hash satu-arah yang dibuat oleh Ron Rivest. MD5 merupakan perbaikan dari MD4 setelah MD4 berhasil diserang oleh kriptanalis. Algoritma *MD5* menerima masukan berupa pesan dengan ukuran sembarang dan menghasilkan *message digest* yang panjangnya 128 bit. Langkah-langkah pembuatan *message digest* tersebut antara lain:

1. Penambahan bit-bit pengganjal (*padding bits*).
   Pesan ditambah dengan sejumlah bit pengganjal sedemikian sehingga panjang pesan (dalam satuan bit) kongruen dengan 448 modulo 512. Jika panjang pesan 448 bit, maka pesan tersebut ditambah dengan 512 bit menjadi 960 bit. Jadi, panjang bit-bit pengganjal adalah antara 1 sampai 512. Bit-bit pengganjal tersebut terdiri dari sebuah bit 1 diikuti dengan sisanya bit 0.
2. Penambahan nilai panjang pesan semula.
   Pesan yang telah diberi bit-bit pengganjal selanjutnya ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula. Jika panjang pesan lebih dari 2^{64} maka yang diambil adalah panjangnya dalam modulo 2^{64}. Dengan kata lain, jika panjang pesan semula adalah K bit, maka 64 bit yang ditambahkan menyatakan K modulo 2^{64}. Setelah ditambah dengan 64 bit, panjang pesan sekarang menjadi kelipatan 512 bit.
3. Inisialisasi penyangga (*buffer*) *message digest*.
   MD5 membutuhkan 4 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit. Total panjang penyangga adalah $4 \times 32 = 128$ bit. Keempat penyangga ini menampung hasil antara dan hasil akhir. Keempat penyangga ini diberi nama *A*, *B*, *C*, dan D. Setiap penyangga diinisialisasi dengan nilai-nilai (dalam notasi HEX) sebagai berikut:
   A = 01234567
   B = 89ABCDEF
   C = FEDCBA98
   D = 76543210
4. Pengolahan pesan dalam blok berukuran 512 bit.
   Pesan dibagi menjadi L buah blok yang masing-masing panjangnya 512 bit (Y_0 sampai Y_{L-1}). Setiap blok 512-bit diproses bersama dengan penyangga menjadi keluaran 128-bit, dan ini disebut proses H_{MD5}. Proses H_{MD5} terdiri dari 4 buah putaran, dan masing-masing putaran melakukan operasi dasar MD5 sebanyak 16 kali dan setiap operasi dasar memakai sebuah elemen T. Jadi setiap putaran memakai 16 elemen Tabel T. Fungsi-fungsi f_F, f_G, f_H, dan f_I masing-masing berisi 16 kali operasi dasar terhadap masukan, setiap operasi dasar menggunakan elemen Tabel T.

Tabel 1. Fungsi-fungsi dasar MD5

Nama	Notasi	*g(b, c, d)*
f_F	*F(b, c, d)*	$(b \wedge c) \vee (\sim b \wedge d)$
f_G	*G(b, c, d)*	$(b \wedge d) \vee (c \wedge \sim d)$
f_H	*H(b, c, d)*	$b \oplus c \oplus d$
f_I	*I(b, c, d)*	$c \oplus (b \wedge \sim d)$

Catatan: operator logika AND, OR, NOT, XOR masing-masing dilambangkan dengan $\wedge$, $\vee$, $\sim$, $\oplus$.

Tabel 2. Nilai T[i]

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E5A51	T[35] = 69D96122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57C0FAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBCB	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

C. Algoritma RSA

Algoritma RSA merupakan algoritma kunci-publik yang paling terkenal dan paling banyak aplikasinya. Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan yang besar menjadi faktor-faktor prima. Prinsip yaitu dengan menggunakan:

Teorema Euler $a^{\phi(n)} \equiv 1 \pmod{n}$

dengan syarat:

1. a harus relatif prima terhadap n.
2. $\phi(n)$ = Toitent Euler = fungsi yang menentukan berapa banyak dari bilangan-bilangan 1, 2, 3, …, n yang relatif prima terhadap n.

Jika $n = pq$ adalah bilangan komposit dengan p dan q prima, maka $\phi(n) = \phi(p)\ \phi(q) = (p-1)(q-1)$.

Algoritma ini juga memiliki tiga tahapan utama, yaitu pembangkitan kunci, enkripsi, dan dekripsi. Proses pembangkitan kunci antara lain:

1. Memilih dua bilangan prima, p dan q (rahasia).
2. Menghitung $n = pq$.
3. Menghitung $\phi(n) = (p-1)(q-1)$.
4. Memilih sebuah bilangan bulat e untuk kunci public yang relatif prima terhadap $\phi(n)$.
5. Menghitung kunci dekripsi, d, dengan persamaaan: $ed \equiv 1 \pmod{\phi(n)}$ atau $d \equiv e^{-1} \bmod (\phi(n))$

Hasil dari algoritma di atas yaitu:

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

Proses enkripsi pada algoritma RSA antara lain:

1. Menyatakan pesan menjadi blok-blok plainteks: m_1, m_2, m_3, … (syarat: $0 < m_i < n-1$)
2. Menghitung blok cipherteks c_i untuk blok plainteks p_i dengan persamaan: $c_i = m_i^e \bmod n$, yang dalam hal ini, e adalah kunci publik.

Proses dekripsi pada algoritma RSA dilakukan dengan menggunakan persamaan: $m_i = c_i^d \bmod n$, yang dalam hal ini d adalah kunci privat.

D. Fungsi Hash SHA-1

SHA adalah fungsi hash satu-arah yang dinyatakan sebagai *standard* fungsi *hash* satu-arah. Algoritma *SHA* menerima masukan berupa pesan dengan ukuran maksimum 2^{64} bit (2.147.483.648 *gigabyte*) dan menghasilkan *message digest* yang panjangnya 160 bit, lebih panjang dari *message digest* yang dihasilkan oleh MD5. Terdapat enam varian fungsi SHA, yaitu SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512.

Langkah-langkah pembuatan fungsi SHA-1 antara lain:

1. Penambahan bit-bit pengganjal (padding bits).
2. Penambahan nilai panjang pesan semula.
3. Inisialisasi penyangga (buffer) MD.
4. Pengolahan pesan dalam blok berukuran 512 bit (H_{SHA}).

SHA membutuhkan 5 buah penyangga (*buffer*) yang masing-masing panjangnya 32 bit. Kelima penyangga ini diberi nama *A*, *B*, *C*, *D*, dan *E*. Setiap penyangga diinisialisasi dengan nilai-nilai (dalam notasi HEX) sebagai berikut:

A = 67452301
B = EFCDAB89
C = 98BADCFE
D = 10325476
E = C3D2E1F0

Proses H_{SHA} terdiri dari 80 buah putaran (MD5 hanya 4 putaran). Masing-masing putaran menggunakan bilangan penambah K_t, yaitu:

Putaran $0 \le t \le 19$ K_t = 5A827999
Putaran $20 \le t \le 39$ K_t = 6ED9EBA1
Putaran $40 \le t \le 59$ K_t = 8F1BBCDC
Putaran $60 \le t \le 79$ K_t = CA62C1D6

Tabel 3. Fungsi logika f_t pada setiap putaran

Putaran	$f_t(b, c, d)$
0 .. 19	$(b \wedge c) \vee (\sim b \wedge d)$
20 .. 39	$b \oplus c \oplus d$
40 .. 59	$(b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$
60 .. 79	$b \oplus c \oplus d$

III. IMPLEMENTASI APLIKASI LOGIN SEDERHANA

A. Komponen yang Digunakan

Implementasi aplikasi login sederhana pada makalah ini dilakukan pada perangkat lunak Microsoft Visual Studio 2010 dengan bahasa C# .NET dan basis data MySQL.

B. Implementasi Gabungan Algoritma GamalD5

Algoritma GamalD5 merupakan penggabungan algoritma ElGamal dengan fungsi hash MD5. Pada fungsi hash MD5 masukan berupa *string*. Untuk mendapatkan *message digest* dari masukan tersebut, dilakukan fungsi hash sesuai yang telah dijelaskan pada bab sebelumnya.

Sedangkan pada algoritma ElGamal yang diimplementasikan, terdapat 3 fungsi utama, yaitu pembangkitan kunci berdasarkan *username*, fungsi enkripsi, dan fungsi dekripsi.

Pada proses pembangkitan kunci, dilakukan pencarian bilangan prima secara acak berdasarkan *username* masukan pengguna dengan menjumlahkan jumlah karakter *username* dengan representasi *integer* dari setiap karakter pada *username*. Bilangan prima inilah yang akan menjadi nilai *p*. Nilai *g* dan *x* didapatkan secara acak dari bilangan *p*, sedangkan nilai *y* didapatkan dari fungsi *modulo exponentiation* dari *g* dengan parameter *x* dan *p*. Dari nilai *p*, *g*, *x*, dan *y* didapatkan kunci publik dan kunci privat.

Pada proses enkripsi menggunakan ukuran blok 8 byte. Sebelum enkripsi dimulai, dicari nilai *k* dengan pencarian secara acak dari berdasarkan nilai *p*. Untuk setiap bloknya dilakukan fungsi enkripsi dengan kunci publiknya untuk mendapatkan nilai *a* dan *b*. Kemudian kedua nilai tersebut direpresentasikan ke dalam bentuk heksadesimal dengan format "*a*,*b*". Proses ini dilakukan secara terus-menerus untuk blok selanjutnya sampai selesai. Representasi hasilnya menjadi "a_0,b_0 a_1,b_1 a_2,b_2 ...".

Pada porses dekripsi, masukan berupa *string* seperti di atas dibagi untuk setiap bloknya. Pada proses dekripsi tiap blok, dilakukan pemecahan menjdi bilangan *a* dan *b*. Dari nilai *a*, *b*, *x*, dan *p* dilakukan proses dekripsi dengan kunci privatnya sehingga mendapatkan *array of byte* dari blok tersebut. Proses ini dilakukan secara terus-menerus sampai selesai. *Array of byte* tersebut kemudian dijadikan *string* sehingga menjadi pesan plainteksnya.

C. Implementasi Algoritma SHARSA

Algoritma SHARSA merupakan penggabungan algoritma RSA dengan fungsi hash SHA-1. Pada fungsi hash SHA-1 masukan berupa *string*. Untuk mendapatkan *message digest* dari masukan tersebut, dilakukan fungsi hash sesuai yang telah dijelaskan pada bab sebelumnya.

Sedangkan pada algoritma RSA yang diimplementasikan, terdapat 3 fungsi utama, yaitu pembangkitan kunci berdasarkan username, fungsi enkripsi, dan fungsi dekripsi.

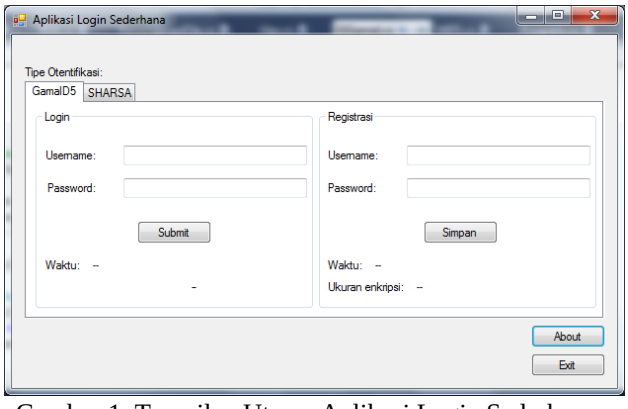
Pada proses pembangkitan kunci, dilakukan pencarian tiga bilangan prima secara acak berdasarkan username masukan pengguna dengan menjumlahkan jumlah karakter username dengan representasi integer dari setiap karakter pada username. Ketiga bilangan prima tersebut yang akan menjadi nilai *p*, *q*, dan *e*. Dari ketiga nilai tersebut dilakukan pembangkitan kunci sehingga mendapatkan kunci publik berupa nilai *e*, kunci privat berupa nilai *d*, dan modulo berupa nilai *n*.

Pada proses enkripsi menggunakan ukuran blok 8 byte. Untuk setiap bloknya dilakukan fungsi enkripsi dengan kunci publiknya untuk mendapatkan hasil cipherteks berupa representasi heksadesimal dari hasil penghitungan. Kemudian proses ini dilakukan secara terus menerus untuk blok selanjutnya sampai selesai. Representasi hasilnya menjadi "$Hex_0\ Hex_1\ Hex_2$ ...".

Pada porses dekripsi, masukan berupa string seperti di atas dibagi untuk setiap bloknya. Pada proses dekripsi tiap blok, dilakukan proses dekripsi dengan kunci privatnya sehingga mendapatkan *array of byte* dari blok tersebut. Proses ini dilakukan secara terus-menerus sampai selesai. *Array of byte* tersebut kemudian dijadikan string sehingga menjadi pesan plainteksnya.

D. Implementasi Keseluruhan

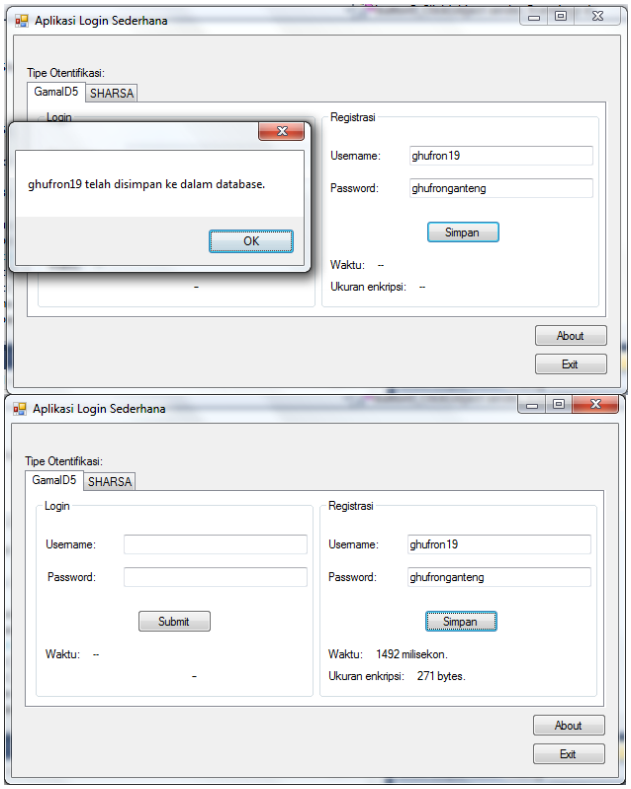
Secara keseluruhan, saya mengimplementasikan semua keempat algoritma tersebut ke dalam suatu aplikasi login sederhana. Tampilan utama aplikasi tersebut tampak pada gambar di bawah ini.



Gambar 1. Tampilan Utama Aplikasi Login Sederhana

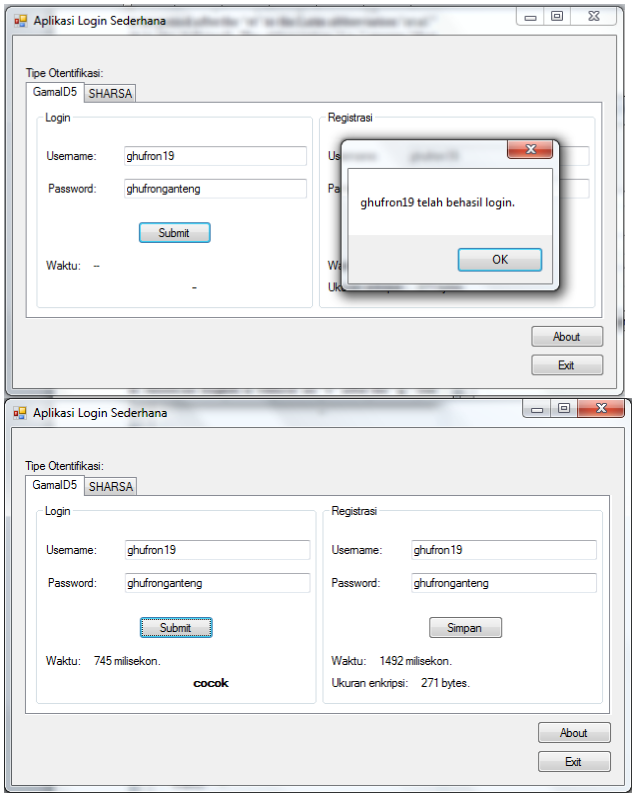
Pada gambar tersebut terdapat dua pilihan algoritma, yaitu GamalD5 dan SHARSA. Terdapat dua bagian pada setiap gabungan algoritma, yaitu login dan registrasi. Login merupakan proses pengecekan *username* dan *password* masukan pengguna dengan yang ada dalam basis data, sedangkan registrasi merupakan proses penyimpanan *username* dan *password* ke dalam basis data. Jika *username* atau *password* kosong saat tombol ditekan, maka akan muncul pesan kesalahan. *Username* masukan pengguna juga dibatasi hanya sampai 50 karakter. Proses registrasi hanya dapat dilakukan bila *username* belum ada dalam basis data, sedangkan proses login hanya dapat dilakukan apabila *username* terdapat pada basis data.

Pembangkitan kunci pada kedua gabungan algoritma tersebut berdasarkan *username*. Pada proses registrasi, *password* masukan pengguna diolah dengan fungsi hash terlebih dahulu. Kemudian hasil hash tersebut dienkripsi dengan algoritma ElGamal atau RSA dengan kunci publik yang telah dibangkitkan. Kemudian hasil enkripsi tersebut dimasukkan ke dalam basis data. Contoh proses registrasi tampak pada gambar di bawah ini.



Gambar 2. Tampilan saat Registrasi

Pada proses login, *password* masukan pengguna diolah terlebih dahulu dengan fungsi hash. Kemudian *password* hasil enkripsi diambil dari basis data dan didekripsi menggunakan algoritma ElGamal atau RSA dengan kunci privat yang telah dibangkitkan. Kemudian hasil dekripsi tersebut dibandingkan dengan hasil fungsi hash tadi. Apabila cocok, maka proses login berhasil. Tampilan proses login tampak pada gambar di bawah ini.



Gambar 3. Tampilan saat Login

Untuk menyimpan data *username* dan *password*, digunakan basis data MySQL. Basis data yang dibuat diberi nama "login" dengan sebuah tabel "userpass". Dalam tabel tersebut terdapat 4 buah kolom, yaitu index, *username*, *password*, dan algoritma. Index didapatkan secara otomatis dengan menambahkan angka 1 setiap ada masukan ke dalam basis data. *Username* merupakan *username* masukan dari pengguna. *Password* merupakan hasil enkripsi dari *password* masukan pengguna. Algoritma merupakan jenis gabungan algoritma yang digunakan untuk proses registrasi. Gambar di bawah ini menunjukkan struktur basis data yang dibuat.

index	username	password	algoritma
1	ghufron19	2798115E6657D3CB82224523B1EAE5798F851942E3162CF5B1...	GamalD5
3	Muhammad	37097D30B633C5B99FEF9AB55CAC15D9B4FE2E67471E486016...	GamalD5
4	kriptografi	1217033D1D32BE36014E941AD974CBBF1E2252D1D47DDD637E...	GamalD5

	Field	Type	Collation	Attributes	Null	Default	Extra
	index	int(11)			No	None	AUTO_INCREMENT
	username	varchar(50)	latin1_swedish_ci		No	None	
	password	text	latin1_swedish_ci		No	None	
	algoritma	enum('GamalD5','SHARSA')	latin1_swedish_ci		No	None	

Gambar 4. Struktur Basis Data Aplikasi Login Sederhana

Contoh aplikasi ini dapat diunduh pada pranala: https://cid-eaafaacd19fe3e76.skydrive.live.com/redir.aspx?resid=EAAFAACD19FE3E76!108&authkey=eSroAXCLKd4%24

IV. ANALISIS *DIGITAL SIGNATURE*

Untuk membandingkan algoritma GamalD5 dan SHARSA, saya melakukan proses registrasi dan login sebanyak 10 kali dengan *username* dan *password* yang sama untuk kedua algoritma tersebut. Hasilnya tampak pada tabel di bawah ini.

Tabel 4. Hasil Pengujian Proses Registrasi

Username	***Password***	**GamalD5**		**SHARSA**	
		Waktu (ms)	Ukuran (byte)	Waktu (ms)	Ukuran (byte)
ghufron19	ghufronganteng	1492	271	2973	206
Muhammad	GhufronMahfudhi	429	155	3329	186
kriptografi	elgamalmd5	244	157	862	134
pergikepasar	adabanyaksayuran	10357	381	2906	190
0123456789	9876543210	2084	215	2809	193
badaitubes	ada3tubes4tucil	705	268	3074	156
MaKaLahkE2	123haRusAmbisius	2937	317	2555	214
78SANGAT90	LA837PAR1SEKALI	5098	434	4452	243
4kYu64Alay	1n1p4sSw0RdR4hasia	898	237	3210	189
mauujiannichbesok	semogabisaberhasilsukses	267	187	834	157
Rata-rata		2451,1	262,2	2700,4	186,8

Tabel 5. Hasil Pengujian Proses Login

Username	***Password***	**GamalD5**	**SHARSA**
		Waktu (ms)	Waktu (ms)
ghufron19	ghufronganteng	931	2728
Muhammad	GhufronMahfudhi	146	3401
kriptografi	elgamalmd5	257	857
pergikepasar	adabanyaksayuran	9371	2859
0123456789	9876543210	1777	2951
badaitubes	ada3tubes4tucil	665	3365
MaKaLahkE2	123haRusAmbisius	2942	2497
78SANGAT90	LA837PAR1SEKALI	5285	4499
4kYu64Alay	1n1p4sSw0RdR4hasia	804	2880
mauujiannichbesok	semogabisaberhasilsukses	223	991
Rata-rata		2240,1	2702,8

Pada tabel pengujian proses registrasi, didapatkan waktu rata-rata proses registrasi dengan algoritma GamalD5, yaitu 2451,1 milisekon, lebih kecil dari pada proses registrasi dengan algoritma SHARSA, yaitu 2700,4 milisekon. Hal ini dikarenakan untuk mendapatkan nilai *e* secara acak pada pembangkitan kunci algoritma RSA juga harus memperhatikan sifat relatif prima dengan bilangan *p* dan *q*. Sedangkan pada pembangkitan kunci algoritma ElGamal tidak pengecekan relatif prima. Pengecekan sifat relatif prima tersebut pasti membutuhkan waktu lebih. Selain itu, fungsi hash SHA-1 melakukan 80 putaran, sedangkan fungsi hash MD5 hanya melakukan 4 putaran. Dari fungsi hash tersebut, SHA-1 membutuhkan waktu yang lebih lama dari pada MD5. Oleh karena itu, waktu yang dibutuhkan untuk melakukan proses registrasi dengan algoritma SHARSA lebih banyak dari pada proses registrasi dengan algoritma GamalD5.

Sedangkan ukuran rata-rata hasil algoritma GamalD5, yaitu 262,2 byte, lebih besar dari pada hasil algoritma SHARSA, yaitu 186,8 byte. Hal ini dikarenakan pada enkripsi algoritma ElGamal dihasilkan dua nilai, yaitu *a* dan *b*, yang direpresentasikan secara heksadesimal dengan ditambah karakter koma untuk setiap bloknya sehingga dapat menghasilkan *string* hasil yang bisa mencapai dua kalinya blok masukan. Sedangkan pada algoritma RSA,

pada setiap bloknya dihasilkan *array of byte* yang jumlahnya hampir sama dengan *array of byte* masukan. Dengan adanya fungsi hash SHA-1 sebagai masukan fungsi enkripsi RSA yang menghasilkan 160 bit *message digest*, tidak terlalu mempengaruhi selisih tersebut, karena hasil dari SHA-1 memiliki selisih yang cukup kecil dengan hasil MD5, yaitu 128 bit. Oleh karena itu, ukuran hasil algoritma GamalD5 lebih besar dari pada ukuran hasil algoritma SHARSA.

Berdasarkan tabel hasil pengujian proses login, proses login dengan algoritma GamalD5 membutuhkan waktu rata-rata 2240,1 milisekon sedangkan algoritma SHARSA membutuhka waktu rata-rata 2702,8 milisekon. Hal ini dikarenakan algoritma RSA membutuhkan waktu lebih untuk pengecekan relatif prima sedangkan algoritma ElGamal tidak ada pengecekan relatif prima. Selain itu, fungsi hash SHA juga membutuhkan waktu lebih untuk melakukan 80 putaran, sedangkan fungsi hash MD5 hanya melakukan 4 putaran. Oleh karena itu, waktu yang dibutuhkan untuk proses login dengan algoritma SHARSA lebih besar dari pada dengan algoritma GamalD5.

Jika melihat dari sisi kompleksitas algoritma, algoritma SHARSA memiliki kompleksitas yang hampir sebanding dengan algoritma GamalD5. Jika melihat sesama fungsi hash, SHA-1 lebih kompleks dari pada MD5 karena melakukan 80 putaran sedangkan MD5 melakukan 4 putaran. Selain itu, pesan yang dihasilkan oleh SHA-1 juga lebih panjang dari pada pesan yang dihasilkan oleh MD5. Sedangkan dari sisi algoritma enkripsi dan dekripsi, ElGamal lebih kompleks dari pada RSA karena melakukan 2 fungsi pada proses enkripsi dan dekripsi. Hasil dari enkripsi ElGamal juga ada 2 sehingga lebih membingungkan orang yang ingin melakukan kriptanalisis. Akan tetapi, bila menggunakan bilangan prima yang berukuran besar, RSA lebih baik dari pada ElGamal karena membutuhkan waktu dan tenaga yang besar untuk dapat memfaktorkannya apabila melakukan kriptanalisis. Apabila dilihat dari persebaran waktu, SHARSA cukup stabil bila dibandingkan dengan GamalD5. Tampak pada tabel hasil pengujian, waktu rata-rata yang dibutuhkan untuk melakukan algoritma SHARSA

V. Simpulan dan Saran

Algoritma *digital signature* memiliki banyak kegunaan, terutama di bidang keamanan. Algoritma ini juga dapat dikembangkan dengan penggabungan beberapa algoritma yang telah ada, misalnya ElGamal dengan MD5 menjadi GamalD5 atau RSA dengan SHA menjadi SHARSA. Apabila dibandingkan, algoritma GamalD5 lebih baik dari pada algoritma SHARSA dalam waktu pemrosesan dan kompleksitas hasil enkripsi. Sedangkan algoritma SHARSA lebih baik dari pada algoritma GamalD5 dalam ukuran hasil enkripsi dan kompleksitas algoritma. Walaupun memiliki kelebihan masing-masing, apabila hal positif dari kedua algoritma tersebut digabungkan pasti akan mendapatkan suatu algoritma yang lebih baik. Fungsi hash SHA-1 memiliki kelebihan di bidang kompleksitas algoritma, sedangkan algoritma ElGamal memiliki kelebihan di bidang efisiensi waktu. Jika kedua hal ini digabungkan, maka akan menjadi algoritma yang lebih mangkus, seperti yang diterapkan pada *Digital Signature Standard* dengan fungsi SHA dan algoritma *digital signature* yang merupakan turunan dari algoritma ElGamal.

Untuk membuat *digital signature* dengan satu algoritma sudah bisa menjadi aman, tetapi dengan menggabungkan beberapa algoritma bisa menjadi lebih aman. Oleh karena itu, perlu dilakukan perkembangan terus terhadap algoritma ini agar selalu bisa menjadi lebih aman. Kriptanalisis dan pengujian juga perlu dilakukan sebagai parameter ketercapaian segi keamanan pada algoritma tersebut. Oleh karena itu, baik algoritma yang telah ada maupun algoritma baru yang akan dibuat harus selalu dikembangkan untuk menciptakan algoritma baru dengan tingkat keamanan yang lebih tinggi.

Daftar Pustaka

[1] Rinaldi Munir, Situs Perkuliahan Kriptografi, http://www.informatika.org/~rinaldi/Kriptografi/2010-2011/kripto10-11.htm.

[2] http://bodrato.it/presentazioni/Bodrato2008-TokyoUnivScience.pdf

[3] http://www.youtube.com/watch?v=rOJGP-W6v8Y

[4] http://en.wikipedia.org/wiki/Digital_signature

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

Muhammad Ghufron Mahfudhi
13508020