Autentikasi Multilayer

Rikysamuel
Teknik Informatika
Institut Teknologi Bandung
Bandung, Indonesia
rikysamueltan@gmail.com

***Abstract*— Paper ini berisi pemaparan dan analisis tentang pemanfaatan beberapa algoritma kriptografi dalam pengotentikasian dan verifikasi dalam sistem login suatu sistem yang lebih besar. Algoritma yang digunakan antara lain Diffie-Hellman, AES, RSA, MAC, dan fungsi hash MD5. Juga memanfaatkan protokol kriptografi dalam komunikasi antara server-client.**

Keywords — authentication, AES, MAC, Diffie-Hellman, TCP, socket

I. PENDAHULUAN

Privasi menjadi aspek yang sangat penting dimasa sekarang ini. Tanpa adanya privasi, orang bisa dirugikan dan kenyamanannya pun terganggu. Selain itu juga banyak data-data pribadi yang jika tidak dilindungi, dan bila tersebar dan jika sampai disalahgunakan oleh orang lain, bisa sangat buruk dampaknya. Oleh karena itu banyak usaha yang dilakukan untuk menjaga kerahasiaan/privasi pengguna.

Salah satu dari usaha yang disebutkan sebelumnya adalah *Authentication* (Autentikasi). Dengan menggunakan otentikasi, tidak semua orang secara bebas bisa masuk ke dalam suatu sistem. Hanya orang-orang tertentu saja yang memiliki kombinasi username-password (misalnya) yang diperkenankan untuk masuk ke dalam sistem yang bersangkutan. Teknik menggunakan autentikasi ini sangat populer bahkan hingga saat ini. Beberapa teknik autentikasi yang populer dari masa ke masa antara lain HTTP Basic Authentication, Form-Based Authentication, bahkan sekarang sudah menggunakan API seperti OAuth.

Walau demikian, tetap saja selalu ada celah yang dapat ditembus oleh pihak ketiga untuk dapat menembus sistem autentikasi ini. Pada makalah ini akan dibahas mengenai pengatasan permasalahan yang terjadi pada sistem autentikasi. Yang lebih disorot dalam pembahasan kali ini adalah keterhubungan/komunikasi anatara kedua belah pihak (Server-Client). Protokol yang digunakan untuk komunikasi adalah dengan menggunakan TCP melalui socket.

Sesuai dengan judul pada makalah ini, autentikasi multilayer membuat autentikasi pada suatu pesan (data login user) dirahasiakan sedimikian rupa dengan banyak algoritma kriptografi, sehingga *Intruder* akan menjadi sangat kesusahan unuk mendapatkan data yang penting dari user.

Secara umum gambaran dari sistem terkait adalah, pertama kali *TCP Connection* dimulai, digunakan algoritma pertukaran kunci Diffie-Hellman untuk mengoper kunci rahasia. Kemudian server mengirimkan string dummy yang harus di dekripsi oleh pengguna untuk mengecek apakah user tersebut asli. Baru kemudian setelah itu, data login user yang bersangkutan dikirim ke Server. Untuk simplifikasi, data yang di proses kedalam Autentikasi Multilayer hanyalah data password saja. Sedangkan username dibiarkan berbentuk plaintext.

II. DASAR TEORI

Algoritma yang digunakan untuk merancang Autentikasi Multilayer ini antara lain Diffie-Hellman, AES, RSA, MAC, dan MD5. Digunakan banyak algoritma dalam makalah ini bertujuan untuk mengkomplekskan lapisan-lapisan dalam autentikasi. Pada bagian dasar teori ini akan dibahas secara umum mengenai masing-masing dari algoritma yang disebutkan. Terlebih dahulu akan dibahas secara singkat mengenai Algoritma Diffie-Hellman.

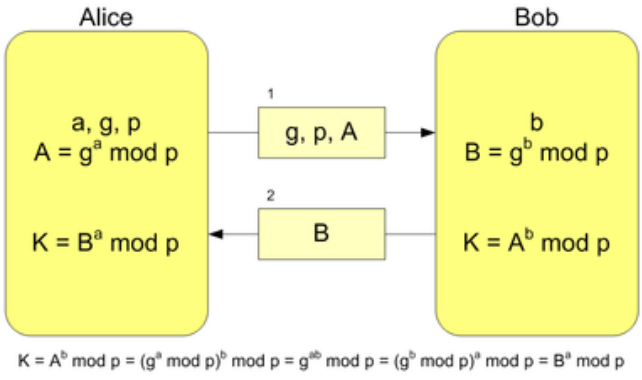
Pada algoritma Diffie-Hellman, dibahas mengenai aturan pertukaran data rahasia (kunci). Kedua belah pihak terlebih dahulu haruslah menyepakati beberapa nilai bersama, *n* dan *g*, dimana $g < n$. Kemudian kedua belah pihak saling bertukar informasi mengenai hasil perhitungan masing-masing dimana

$$X = g^x mod(n) \text{ atau } Y = g^y mod(n)$$

Kemudian masing-masing menghitung kembali untuk mendapatkan nilai kunci yang bersangkutan, yaitu dengan:

$$K = Y^x mod(n) \text{ atau } K' = X^y mod(n)$$

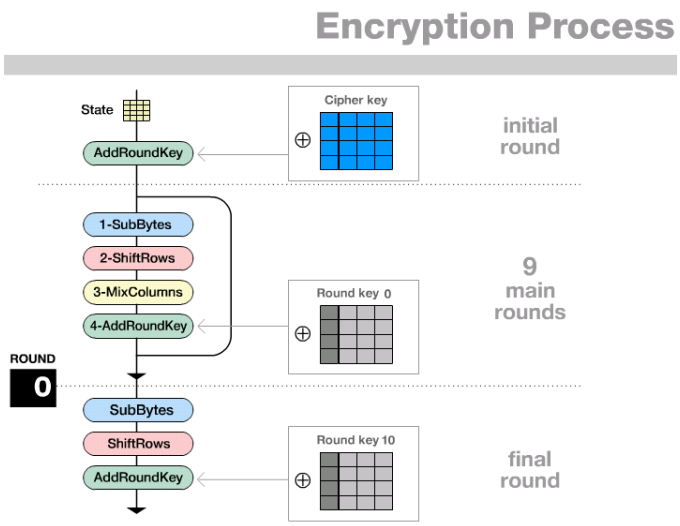
Kunci K inilah yang kemudian akan digunakan pada proses enkripsi-dekripsi selanjutnya.



Algoritma berikutnya adalah AES. AES pada aplikasi digunakan sebagai algoritma yang digunakan oleh Kunci K. AES menggantikan DES yang dianggap sudah tidak aman lagi. AES pertama kali dibuat pada jaman adanya lomba untuk

membuat algoritma yang baru. Pembuatnya adalah Rijndael (Vincent Rijmen dan Joan Daemen) yang merupakan finalis 5 besar dari lomba tersebut. AES merupakan block cipher algorithm yang mendukung panjang kunci hingga 128 sampai 256 bit dan dengan step hingga 32 bit. Proses pengenripsian mirip dengan DES (menggunakan ronde-ronde putaran).

Pada setiap putaran dalam ronde tersebut, menggunakan kunci internal yang berbeda-beda, sehingga algoritma ini menjadi sangat sulit untuk dipecahkan. Secara umum, gambaran proses dalam algoritma AES adalah sebagai berikut,



Kemudian algoritma selanjutnya yang digunakan adalah algoritma RSA. RSA dibuat oleh Ron Rivest, Adi Shamir, dan Len Addleman pada tahun 1976.

Pada Algoritma RSA, banyak menggunakan nilai-nilai yang prima. Seperti misalkan dalam proses pembangkitan kunci. Terlebih dahulu dicari nilai sembarang *p*, dan *q*. Kedua nilai random prima tersebut kemudian dikalikan dan menghasilkan nilai *n*. Dari hasil perhitunga tersebut kemudian bisa didapatkan nilai phi yang kemudian dibutuhkan untuk mendapatkan nilai privat/public dari pengguna terkait.

Kemudian untuk mendapatkan kunci enkripsi/dekripsi dicari lagi nilai prima tertentu. Untuk kunci dekripsi bisa didapatkan dari nilai *e* yang diinverskan yang telah dikali dengan dengan nilai modulo dari phi. Untuk melakukan enkripsi, hanya perlu melakukan sedikit rumus:

$$Ci = \mathrm{m}_i^e mod(n)$$

Dimana *m* merupakan pesan yang dibuat kedalam blok message, dan *e* merupakan kunci publik

Dan untuk melakukan dekripsi hanya tinggal melakukan perhitungan:

$$Mi = \mathrm{c}_i^d mod(n)$$

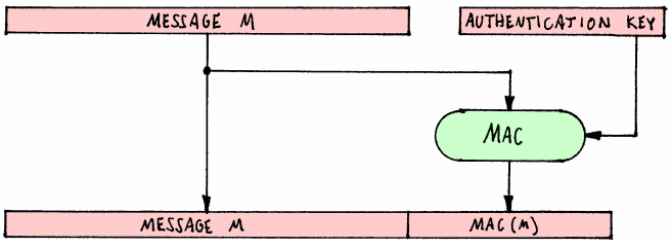
Dimana *c* merupakan block cipher dan *d* merupakan kunci privat.

Tidak cukup sebatas menggunakan AES dan RSA saja, MAC pun diimplementasikan dalam sistem autentikasi multilayer ini. Tujuannya untuk memastikan apakah pesan yang dibuat tersebut valid atau tidak.

MAC (Message Authentication Code) merupakan fungsi hash satu arah yang menggunakan suatu kunci rahasia. Untuk mendapatkan nilai MAC pada suatu pesan, pesan yang bersangkutan terlebih dahulu di-*concat* dengan kunci yang akan digunakan. Kemudain pesan+kunci tersebut dihash menggunakan fungsi hash tertentu. Hasil hash inilah yang merupakan nilai MAC. Selanjutnya nilai MAC dikirim bersama dengan pesan menuju pihak kedua. Pihak kedua memiliki kunci sehingga dapat mengautentikasi apakah pesan yang bersangkutan valid atau tidak.

Secara matematis, MAC dapat dituliskan sebagai:

$$MAC = C_k(m)$$



Untuk lebih menjaga keamanan data, pada sisi Server, data yang bersangkutan bisa disimpan dengan mengenkripsinya dalam database. Sehingga bila ada *intruder* menembus sistem dan masuk kedaam database, orang tersebut masih harus berusaha memecahkan data tersebut.

Dalam kasus ini yang digunakan adalah MD5. MD5 juga merupakan fungsi hash 1 arah. MD5 sendiri merupakan perbaikan dari MD4 yang ternyata ditemukan kolisinya (terdapat nilai hash yang sama).

Secara umum terdapat beberapa langkah garis besar dalam proses pengerjaan MD5, antara lain:

1. Penambahan bits-bits pengganjal
2. Penambahan nilai panjang pesan semula
3. Inisiasi penyangga (buffer) MD
4. Pengolahan pesan dalam blok berukuran 512 bit

Tapi walau begitu, MD5 sebenarnya sudah ditemukan kolisinya. Sehingga untuk penyimpanan dalam database bisa menggunakan algoritma lainnya.

III. Rancangan Sistem Autentikasi

Pada sistem autentikasi ini menggunakan TCP Connection antar Server-Client. Dalam TCP Connection, jaringan tidaklah aman dalam pengimplementasian makalah ini karena tidak menggunakan protokol SSL atau semacamnya. Oleh karena itu pertukaran kode rahasia dapat dilakukan menggunakan algoritma Diffie-Hellman. Karena algoritma dibutuhkan baik pada sisi Client dan Server, sehingga algoritma ini diterapkan baik pada aplikasi server maupun aplikasi client.

Pada Diffie-Hellman, terlebih dahulu harus disepakati antara kedua belah pihak mengenai nilai *g* dan *n*. Agar mempersulit, kedua nilai tersebut bisa dibuat dengan melakukan random berdasarkan *seed* tertentu, misalkan waktu. Dengan memanfaatkan seeder berupa time, nilai random tidak akan sama terus menerus. Setiap selang beberapa saat akan ada

perubahan nilai random, bergantung dari seeder yang digunakan.

Agar formatnya lebih mudah, *seed* waktu dapat diubah dari GMT menjadi *epoch*. Kemudian berdasarkan seed ini nilai random yang bersangkutan dapat dibuat berubah berkali-kali. Untuk dapat mengganti nilai random nilai *g* dan *n* sekali dalam sehari, dapat diambil nilai waktu dari tanggal hari ini pada jam spesifik, misalkan jam 00:00:00. Kemudian ubah satuan kedalam epoch. Setelah didapatkan nilai yang sudah disepakati untuk *g* dan *n*, kemudian client terlebih dahulu mengirimkan hasil perhitungannya dari $X = g^x mod(n)$. Nilai X tersebut oleh server dicari nilai K nya. Kemudian Server mengirimkan juga nilai hasill perhitungan $X = g^x mod(n)$ versi server kepada client.

Tidak hanya itu saja, server mengirimkan sebuah string yang sudah dienkripsi kepada client yang tujuannya untuk didekripsi oleh client, untuk memastikan apakah client yang bersangkutan valid atau tidak (memiliki kunci atau tidak). Kemudian client juga mencari nilai K nya berdasarkan nilai X yang dikirimkan dari server. Setelah mendapatkan nilai K-nya, kemudian client barulah mengirimkan data loginnya, beserta dengan hasil dekripsi dari string yang dari Server.

Jika ada *intruder* yang meng-*intercept* komunikasi server menuju client, *intruder* tersebut tidak mengetahui kunci untuk mendekripsi string yang telah terenkripsi yang berasal dari server. Untuk mengetahui kunci yang digunakan, *intruder* tersebut harus melakukan operasi-operasi Matematika Diskrit. Walau pihak ketigas tersebut mengetahui nilai *g* dan *n* dari persamaan Diffie Hellman, tetap membutuhkan waktu yang lama untuk akhirnya mendapatkan kedua nilai konstanta *x* yang dimiliki Server dan Client.

Tapi walau begitu, pengamanan dengan menggunakan string test tersebut tidak cukup aman. Jika *intruder* yang bersangkutan memilih untuk melakukan *intercept* terhadap komunikasi dari client menuju Server, pihak ketiga tersebut "mungkin" dapat mencuri data login pengguna dan mengubahnya atau semacamnya.

Oleh karena itu, penambahan MAC perlu dilakukan untuk memverifikasi apakah inputan tersebut valid atau tidak. Jika pihak ketiga yang melakukan *intercept* tidak memiliki kunci rahasia kedua belah pihak sehingga perlu ada usaha lebih besar dari pihak ketiga tersebut untuk kemudian membuat MAC baru. Tapi usaha tersebut terbilang cukup besar dan sangatlah sulit.

Setelah request dari pengguna masuk, terlebih dahulu Server akan melakukan *checking* terhadap string test yang diberikan oleh Server kepada client untuk didekripsi. Jika hasil dekripsi tidak sesuai dengan yang dimiliki Server, maka login langsung dianggap gagal.

Tetapi jika dinyatakan hasil dekripsi sesuai dengan data yang dimiliki oleh Server, proses verifikasi dilanjutkan ke tahap berikutnya.

Proses enkripsi/dekripsi string tes tersebut menggunakan algoritma AES dengan kunci dari hasil pertukaran Diffie-Hellman. Untuk menerapkan AES, panjang pesan haruslah kelipatan 16. Sehingga apabila pesan yang dimiliki bukan kelipatan 16, bisa ditambahkan padding bit dengan karakter "\0" (null) sampai panjang pesan adalah kelipatan 16.

Konversi antara string ke byte dan sebaliknya sangat sering dilakukan pada proses dalam algoritma ini. Selain itu proses yang digunakan dalam AES ini adalah CBC.

Selanjutnya yang diperiksa terlebih dahulu adalah MAC. Isi dari pesan yang dihitung MAC-nya adalah password dari pengguna yang bersangkutan. Untuk memudahkan, fungsi hash yang digunakan dalam pembuatan MAC adalah dengan MD5. Fungsi hash selain MD5 dapat digunakan untuk lebih memperkuat sistem.

Kemudian server akan menghitung nilai MAC dari password pengguna dari basis data. Kemudian mencocokan kedua nilai tersebut. Apabila tidak didapatkan kecocokan, maka login dianggap gagal. Tetapi apabila ditemukan kecocokan, maka verifikasi diteruskan.

Pada tahap ini, yang dicek adalah kebenaran password tekait username pemilik. Password dari client terlebih dahulu di MD5 karena data password di database Server telah di MD5 semua. Selain itu, lebih aman untuk mentransmisikan data yang telah di hash daripada melakukan hashing di Server. Setelah di MD5, dari client nilai hash tersebut dipasangkan agoritma RSA.

Pada Server kunci public-private dapat dibuat untuk degenerate secara otomatis menggunakan rumus dan disimpan ke file agar selanjutnya hanya tinggal membuka file yang bersangkutan saja. Untuk melakukan enkripsi dari client menuju Server, pada client harus mengetahui nilai *e* dan *n* dari Server.

Dengan menggunkaan kunci publik dari Server, data password dienkripsi. Sampai saat ini, jika ada pihak ketiga yang berhasil lolos sampai ketahap ini, dia harus melakukan dekripsi terhadap data password yang telah di enkripsi dengan RSA dan di hash dengan MD5. Kenyataannya, untuk melakukan hal tersebut sangatlah sulit dan sangat membuang waktu.

Kemudian dipihak Server yang telah memiliki data login tersebut, mendekripsi dengan kunci privat-nya. Kemudian setelah didapatkan hasil dekripsinya, dicocokan dengan database. Apabila cocok, maka login akhirnya dinyatakan sukses. Demikian sebaliknya.

Untuk mengoper data request-response yang dilakukan antar Client-Server dilakukan menggunakan JSON sehingga mempermudah kirim-mengirim data antara kedua entitas. Perlu diperhatikan juga agar data yang dikirim dari Server ke Client dan sebaliknaya sebaiknya dikirim per byte dan bukan mengirim sebuah string dan mengakhirinya dengan karakter "\n" atau semacamnya.

IV. Eksperimen dan Pembahasan Hasil

Implementasi yang dilakukan untuk Sistem Autentikasi Multilayer ini berhasil dengan baik. Pengguna bisa melakukan login menuju Sistem dengan melakukan request login kepada Server dengan mengikuti protokol sistem yang telah dijelaskan sebelumnya.

Implementasi dilakukan dengan menggunakan Netbeans IDE dan program yang dibuat berupa Console Application. Data username adalah "*myusername*" dan data password adalah "*mypassword*" yang telah di hash dengan MD5 menjadi "*34819d7beeabb9260a5c854bc85b3e44*".

Untuk alur, pertama kali client merandom nilai *g* dan *n*. Pada pengujian yang dilakukan tanggal 10 Mei 2015, didapatkan kedua nilai berikut pada client:

g: 1901616422844974

n: 7547910616396980

Juga nilai random X client: 6925891895063144. Hasil random nilai *g* dan *n* dari Server didapati juga memiliki nilai yang sama karena seed nya besarnya sama. Sedangkan nilai random X server, yaitu 5384983200945584.

Kemudian Client mengirimkan JSON menuju Server berupa:

```
{
    "method":"key",
    "key":"6925891895063144"
}
```

Dimana artinya Client mengirimkan Request berupa "key" dan mengirimkan hasil perhitungan X nya menuju Server. Kemudian Server memberikan respon ke Client berupa:

```
{
    "method":"key",
    "value":[
            "5384983200945584",
            "some string"
            ]
}
```

Yang artinya method yang dikirim oleh Server adalah "key" dan value nya berupa array dimana elemen pertama adalah nilai X perhitungan dari Server, dan element kedua adalah string test yang di telah dienkripsi oleh Server menggunakan algoritma AES dengan menggunakan kunci simetri hasil rumus pencarian kunci dari nilai X yang didapatka, yang harus di dekripsi oleh Client.

Isi elemen kedua sebenarnya bukanlah "some string". Sebenarnya isinya adalah string panjang berupa representasi integer dari masing-masing blok byte pesan terenkripsi. Ditulis dengan "some string" dengan alasan kerapihan.

Kemudian, Client kembali membalas ke Server berupa:

```
{
    "password":"some string",
    "method":"login",
    "string":"hello world",
    "mac":"some string",
    "username":"myusername"
}
```

Client memberikan resquest berikutnya berupa "login" data. Yang dikirim ialah berupa password yang telah dienkripsi menggunakan algoritma RSA dengan kunci public server. Juga ada String test hasil dekripsi dengan algoritma AES dengan kunci simetri yang telah disepakati bersama. Juga ada nilai mac yang merupakan hasil perhitungan MAC dari pesan yang ingin dirahasiakan. Juga ada data username yang dikirim, yang dalam kasus ini tidak dipedulikan keamanannya.

Dalam proses pengimplementasian makalah ini, untuk simplifikasi, Client dianggap sudah mengetahui dan mencatat kunci public dari Server.

Kemudian setelah Server menerima request tersebut dari Client, server yang bersangkutan langsung melakukan parsing. Pertama kali yang dilakukan oleh Server adalah dengan mencari kecocokan antara string test verifikasi yang dimiliki client dengan string yang dikirimkan oleh Client.

Jika tidak didapati kecocokan antara kedua string tersebut, maka Server tidak mempedulikan data yang lainya, dan login dianggap gagal. Kemudian Server akan mengirimkan JSON failure ke Client berupa:

```
{
    "method":"login_resp",
    "value":"Login Failed"
}
```

Yang menyatakan bahwa login gagal. Tetapi jika ditemukan kecocokan antara kedua string tersebut, maka akan dicocokan berikutnya nilai MAC nya. Jika MAC nya tidak sesuai, maka login kembali dianggap gagal dan Server mengirim response Login failure. Tetapi jika ditemukan kecocokan, barulah yang dicocokan adalah data login user yang bersangkutan tersebut.

Server kemudian melakukan dekripsi terhadap data password yang dikirm dari Client dengan Algoritma RSA dengan menggunakan kunci privat dari Server itu sendiri. Setelah didapatkan hasil dekripsinya, kemudian didapatkan informasi password berupa hasil hash dari suatu kata aslinya. Server hanya tinggal mencocokan saja apakah hasil hash tersebut dengan hasil hash yang terdapat dalam basis data Server. Jika tidak didapati kecocokan, login lagnsung gagal dan Server mengirimkan login failure ke Client. Tetapi jika informasi password dari Client sesuai dengan basis data pada Server, Server mengirimkan Login Success berupa:

```
{
    "method":"login_resp",
    "value":"Login Success!"
}
```

Berikut adalah Screenshot ketika Server pertama kali dijalankan:

```
run:
Server is listening on localhost:1234
```

Kemudian Client meminta request dan sesaat kemudian langsung diberikan respon oleh Server. Tampilah Client menjadi:

```
run:
pass: 34819d7beeabb9260a5c854bc85b3e44
Seeder: 1431216000000
g: 1901616422844974
n: 7547910616396980
X: 583677601881676
response: {"method":"key","value":["2647076525836276","-40,11,41,-124,8
K is 6834767219142136
response: {"method":"login_resp","value":"Login Success!"}
Login Success!
```

Tampilan sedikit terpotong karena response dari Server yang terlalu panjang. Lalu tampilan Server menjadi:

```
run:
Server is listening on localhost:1234
client_message: {"method":"key","key":"583677601881676"}
Seeder: 1431216000000
g: 1901616422844974
n: 7547910616396980
X: 2647076525836276
K is 6834767219142136
{"method":"key","value":["2647076525836276","-40,11,41,-124,89,
client_message: {"password":"30,9,120,34,76,-124,-9,55,-12,104,
String match!
MAC match!
user: myusername
pass: 34819d7beeabb9260a5c854bc85b3e44
response: Login Success!
```

Dari proses implementasi, didapatkan hasilnya sesuai dengan ekspektasi. Proses enkripsi/dekripspi masing-masing algoritma pun berjalan dengan baik. Proses login dapat dilakukan dengan baik.

V. Analisis

Dengan menggunakan Autentikasi Multilayer, keamanan akan sistem autentikasi menjadi lebih baik. Akan tetapi untuk melakukan seluruh proses dari pertukaran kunci rahasia hingga verifikasi ke password, waktu yang dibuthkan juga menjadi lebih lama. Percobaan terakhir, dengan me-*restrart* ulang IDE dan menjalankan program, didapati waktu eksekusi dari sisi Client selama 1643 ms. Sedangkan setelah *system* melakukan cahcing, waktu eksekusi pada Client menjadi 609 ms. Pengetesan dilakukan dengan menggunakaan laptop Lenovo G405S dengan prosessor AMD A8 dan RAM sebesar 8 GB.

Waktu diatas 1 detik adalah waktu yang cukup lama untuk sebuah request. Hal tersebut dikarenakan algoritma yang digunakan banyak. Selain itu juga dikarenakan string yang diproses sangatlah panjang. Bahkan tipe data JAVA yang digunakan saat implementasi tidak cukup hanya sebatas LONG saja, tapi dengan BigInteger.

Tapi walau begitu proses login, dapat dijalankan dengan sempurna.

VI. Kesimpulan dan Saran

Sistem ini memberikan keamanan yang cukup namun tidak memiliki performansi yang baik. Memang antara performansi dan security sulit untuk didapatkan kondisi terbaik untuk keduanya. Jika pada sistem ini dirasa kurang aman, nilai MAC bisa ditambahkan ke respon server untuk mengecek apakah respon yang diterima benar dari Server atau bukan.

References

[1] Munir, Rinaldi. 2006. Kriptografi. Bandung:Teknik Informatika ITB

[2] https://blog.apigee.com/detail/api_authentication_and_how_it_got_that_way_from_http_basic_to_oauth_2.0

[3] http://www.kubieziel.de/blog/archives/937-Wie-AES-funktioniert.html

[4] http://www.cs.rit.edu/~ark/lectures/onewayhash/onewayhash.shtml

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, 11 Mei 2015

Rikysamuel - 13512089