Implementasi *Token* Berbasis Waktu dengan Fungsi *Hash* untuk Mengotentikasi Transaksi E-Banking

William Sentosa / 13513026
Program Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
williamsentosa@itb.ac.id

***Abstract*—Transaksi e-banking merupakan hal yang sangat membantu kehidupan. Kita tidak perlu datang ke ATM untuk melakukan transaksi. Akan tetapi, e-banking rentan terhadap berbagai serangan, salah satunya serangan yang berhubungan dengan otentikasi. Oleh karena itu, diperlukan suatu metode otentikasi tambahan berupa *token* berbasis waktu yang hanya dimiliki oleh nasabah. *Token* ini dapat diimplementasikan dengan menggunakan fungsi *hash* yang dikonkat dengan waktu sehingga *token* ini akan berubah secara rutin selang beberapa waktu tertentu.**

Keywords— token, security token, two factor authentication, e-banking, message digest, fungsi hash, SHA, MD, WHIRLPOOL, RIPEMD

I. PENDAHULUAN

Transaksi e-banking atau biassa disebut *internet banking* telah menjadi suatu trend tertentu dewasa ini. Hampir setiap bank terkemuka menawarkan opsi untuk melakukan transaksi secara online. Nasabah tidak perlu lagi datang kea tm atau bank terdekat untuk melakukan berbagai jenis transaksi, seperti transfer dana, pembayaran tagihan ataupun pembelian (seperti *token* listrik atau pulsa).

Akan tetapi, transaksi tersebut harus terjamin keamanannya. Beberapa aspek keamanan yang harus dipenuhi adalah *privacy* dan *confidentiality* untuk menjaga inforamsi dari pihak yang tidak mendapat hak akses, integrity untuk menjaga informasi agar tidak diubah oleh pihak yang tidak berwajib, authentication untuk menjaga agar pihak yang boleh mengakses hanyalah pihak yang berwenang, serta availability untuk menjaga ketersediaan layanan.

Makalah ini akan berfokus kepada aspek otentikasi khususnya untuk mengotentikasi pengguna yang menggunakan layanan e-banking. Salah satu metode untuk mengotentikasi pengguna adalah dengan menggunakan ***two factor authentication* (2FA)**. *Two factor authentication* mengharuskan pengguna untuk memiliki dua proses otentikasi sebelum melakukan transaksi. Biasanya proses otentikasi pertama adalah memasukkan username dan password untuk login. Proses otentikasi kedua bisa dilakukan dengan memasukkan ***security token* yang berubah dalam jangka waktu tertentu (time synchronized one time password)**.

Token berbasis waktu ini hanya akan dimiliki oleh nasabah, sehingga selain diperlukan *username* dan *password*, transaksi *online* juga memerlukan kode *token* sebagai perlindungan ganda.

Token ini akan memiliki nilai yang secara berubah dari waktu ke waktu sehingga akan menyulitkan *hacker* yang ingin membobol transaksi. Nilai *token* tersebut akan diimplementasikan dengan menggunakan berbagai fungsi *hash* karena fungsi *hash* ini biassanya cenderung tidak membutuhkan komputasi yang terlalu tinggi sehingga cocok untuk diimplementasikan ke *device* yang memiliki kemampuan komputasi yang rendah.

II. DASAR TEORI

1. Fungsi *Hash*

Fungsi *hash* merupakan fungsi yang menerima masukan string yang panjangnya sembarang, lalu mengubahnya menjadi string keluaran yang panjangnya tetap (fixed), umumnya string keluaran berukuran jauh lebih kecil daripada ukuran string semula.

Fungsi *hash* ini memiliki nama lain yaitu *compression function*, *fingerprint*, *cryptographic checksum*, *message integrity check* (MIC), dan *manipulation detection code* (MDC). Fungsi *hash* merupakan fungsi satu arah sehingga pesan yang sudah diubah tidak dapat dikembalikan menjadi pesan semula.

Fungsi *hash* :

message digest = hash(message)

Keterangan :

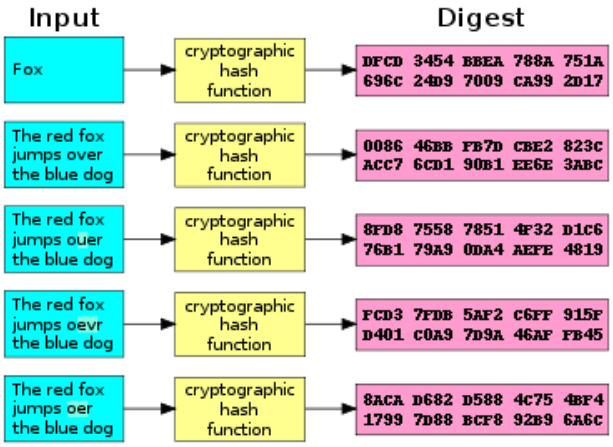
message = pesan berurkuran sembarang
hash = Fungsi *hash* yang digunakan
message digest = hasil *hash* yang berukuran *fixed*

Fungsi *hash* memiliki beberapa sifat yaitu

1. Fungsi *H* dapat diterapkan pada blok data berukuran berapa saja.
2. *H* menghasilkan nilai (*h*) dengan panjang tetap (*fixed-length output*).
3. *H*(*x*) mudah dihitung untuk setiap nilai *x* yang diberikan.
4. Untuk setiap *h* yang dihasilkan, tidak mungkin dikembalikan nilai *x* sedemikian sehingga *H*(*x*) = *h*. Itulah sebabnya fungsi *H* dikatakan fungsi *hash* satu-arah (*one-way hash function*).
5. Untuk setiap *x* yang diberikan, tidak mungkin

mencari $y \neq x$ sedemikian sehingga $H(y) = H(x)$.

6. Tidak mungkin mencari pasangan x dan y sedemikian sehingga $H(x) = H(y)$.



Gambar 1. Input dan output dari fungsi *hash*

Fungsi *hash* satu arah tidak tepat disebut sebagai sebuah proses enkripsi, meskipun nilai *hash* tidak memiliki makna, karena nilai *hash* tidak dapat ditransformasikan balik menjadi pesan semula, selain itu fungsi *hash* tidak memerlukan kunci untuk dapat berjalan.

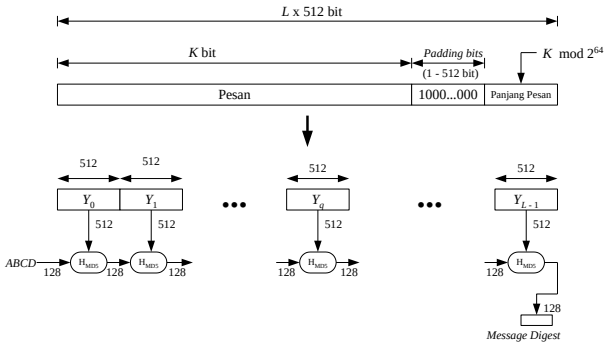
Terdapat banyak fungsi *hash* yang telah digunakan. Setiap fungsi *hash* memiliki algortima dan panjang *message digest* yang berbeda-beda. Beberapa fungsi *hash* juga rentan terkena kolisi, yaitu keadaan dua string sembarang memiliki nilai *hash* yang sama.

2. Algortima Fungsi *Hash* SHA

Berikut merupkan jenis-jenis fungsi *hash* yang cukup populer.

1. Algoritma MD5

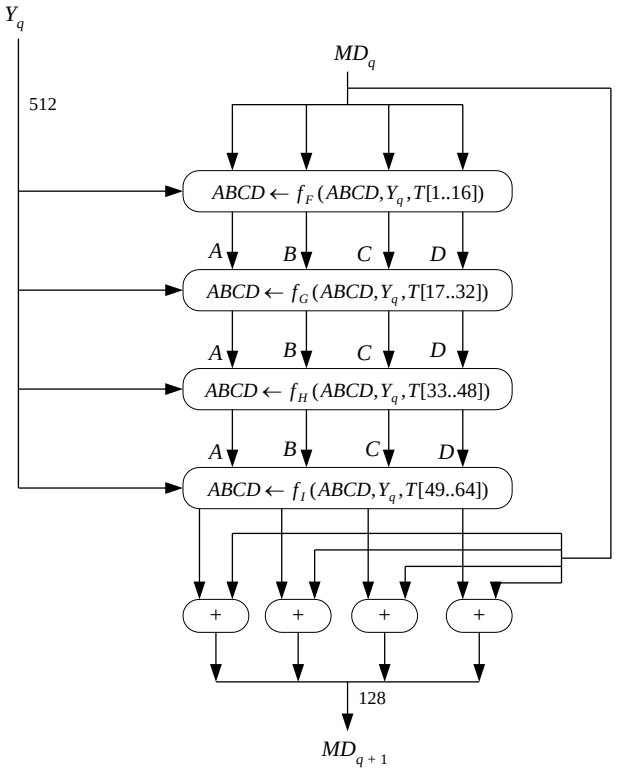
MD5 merupakan fungsi *hash* satu arah yang dibuat oleh Ron Rivest. MD5 ini merupakan perbaikan dari algoritma MD4 karena MD4 memiliki kolisi. Algortima ini akan mengahasilkan *message digest* yang panjangnya 128 bit.



Gambar 2. Algoritma MD5

Berikut merupakan langkah pembuatan *message digest* dengan menggunakan algoritma MD5.

- Menambahkan bit-bit pengganjal (*padding bits*), pesan ditambah degnan bit pengganjal sehingga panjang pesan kongruen dengan 445 (mod 512)
- Penambahan nilai panjang pesan semula, pesan tersebut ditambah lagi dengan 64 bit yang menyatakan panjang pesan semula
- Insialisasi buffer MD, fungsi ini membutuhkan empat buah buffer yang panjangnya 32 bit sehingga ukuran *message* yang dihasilkan adalah 128 bit.
- Pengolahan pesan dalam blok berukuran 512 bit, pesan dibagi menjadi sekian blok yang panjangnya 512 bit. Blok akan diproses dengan suatu mekanisme.



Gambar 3. Pemrosesan blok dengan MD5

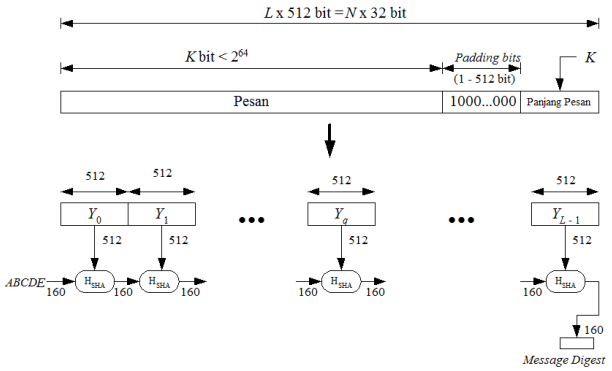
2. SHA (*Secure Hash Algorithm*)

SHA merupakan fungsi *hash* satu arah yang dibuat oleh NIST (National Insitute of Standards and Technology). SHA merupakan standard fungsi *hash* satu arah. Sebenarnya SHA didasarkan pada fungsi MD4. Algoritma ini menerima masukan pesan dengan ukuran maksimum 2^{64} bit dan akan menghasilkan *message digest* dengan panjang 160 bit.

Secara umum, terdapat enam varian fungsi SHA, yaitu

1. SHA-0
2. SHA-1
3. SHA-224
4. SHA256
5. SHA-384
6. SHA-512

Diantara keenam fungsi *hash* tersebut, fungsi *hash* yang paling sering dipakai adalah fungsi SHA-1.

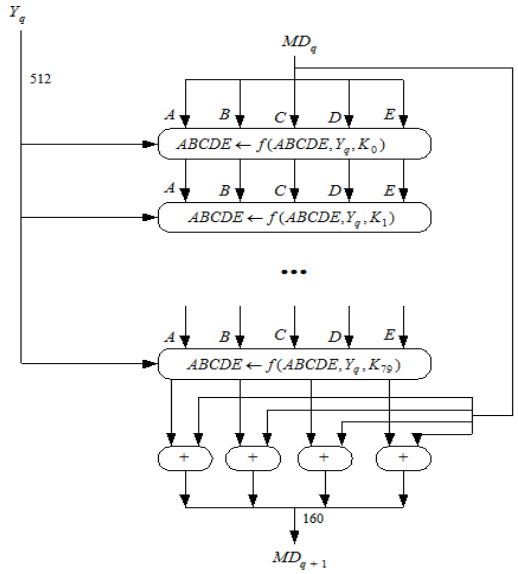


Gambar 4. Gambaran fungsi *hash* SHA-1

Berikut merupakan langkah pembuatan *message digest* dengan fungsi SHA-1.

- Menambahkan bit-bit pengganjal (*padding bits*)
- Penambahan nilai panjang pesan semula
- Insialisasi buffer MD
- Pengolahan pesan dalam blok berukuran 512 bit

Fungsi SHA memerlukan lima buffer yang masing-masing panjangnya 32 bit sehingga total panjang buffer adalah 160 bit. Setiap penyangga diinisialisasi dengan sebuah nilai dan ditampung kedalam buffer A, B, C, D, dan E.



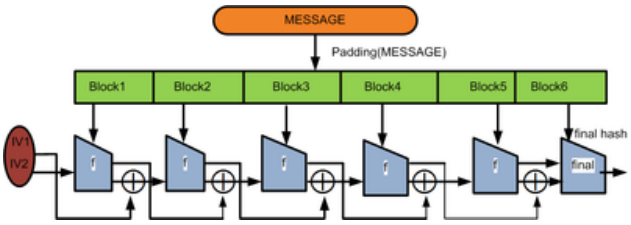
Gambar 5. Proses pengolahan blok

3. RIPEMD

Fungsi *hash* ini diajukan oleh konsorsium RIPE dan ditujukan untuk implementasi software pada mesin berarsitektur 32 bit. Sama seperti SHA, algoritma ini dikembangkan dari fungsi *hash* MD4. Fitur utama dari algoritma ini adalah adanya dua rnatai komputasi yang berbeda, independen dan parallel, dan hasil kedua komputasi ini digabungkan pada akhir prosesnya.

4. WHIRLPOOL

Fungsi *hash whirlpool* merupakan fungsi *hash* yang dirancang oleh untuk menerima *message* dengan panjang maksimum 2^{256} bit dan akan menghasilkan *message digest* dengan panjang 512 bit. Whirpool ini juga menggunakan blok-blok untuk pemrosesannya dan menggunakan konstruksi Merkle-Damgard untuk menghasilkan *message digest*.



Gambar 6. Konstruksi Merkle-Damgard

III. ALGORITMA PEMBANGKIT *TOKEN*

Untuk mengotentikasi pengguna dalam melakukan transaksi e-banking, pengguna perlu memasukkan suatu *token* yang hanya dimiliki oleh nasabah. Biasanya pihak bank akan memberikan suatu alat kecil yang dapat menghasilkan sekumpulan angka yang secara konstan berubah dalam jangka waktu tertentu.

Gambar 7. *Security token* berbasis waktu

Token berbasis waktu atau *time based password* dapat dibuat dengan menggunakan fungsi *hash*. ***Message* akan ditambahkan dengan waktu** sebelum *message* tersebut diproses dengan fungsi *hash*. Akibatnya *digest* akan memiliki nilai yang bervariasi.

Secara umum proses pembuatan *token* dibagi menjadi 5 tahap :

1. Pembangkitan *password* secara unik

   Password akan dibangkitkan untuk menjadi **input dari fungsi *hash***. **Pesan harus unik untuk setiap nomor rekening** untuk memastikan *token* tersebut hanya berlaku untuk satu rekening. **Ukuran password bebas dan password bertipe alpha numerik**. Pesan bisa saja dibangkitkan secara acak, asalkan keunikan pesan tetap terjaga. Dengan menggunakan pesan yang acak, hacker akan kesulitan untuk menebak password tersebut bila tidak membobol *database* sistem.

2. Penambahan password dengan waktu

   Password yang telah dibangkitkan akan dikonkat dengan waktu. Waktu tersebut bisa saja didapatkan dari hasil pengurangan antara waktu sekarang dengan *unix time* dalam bentuk *unix timestamp*. Password akan dikonkat dengan waktu baru **setiap 30 detik**.

```
password = concat(password, waktu)
```

3. Pengubahan password menjadi *message digest*

   Fungsi *hash* akan digunakan untuk mengubah password menjadi *message digest*. Fungsi *hash* apapun dapat digunakan untuk menghasilkan *message digest* ini.

```
digest = Hash(password)
```

4. Pengambilan sebagian *digest* untuk ditampilkan

   Karena *message digest* umumnya terlalu panjang untuk ditampilkan, *digest* yang diambil cukup **diambil 24 bit pertama**.

5. Pengubahan *digest* menjadi decimal

   Setelah mendapatkan *digest* yang sudah diperkecil, *digest* akan diubah menjadi decimal untuk ditampilkan kedalam alat. Karena ukuran *digest* hanya 24 bit. decimal yang didapatkan berkisar antara 0 sampai 16777215. Angka ini akan menjadi *token* dan berubah setiap 30 detik sekali, karena proses ini akan diulang setiap 30 detik.

Berikut merupakan *pseudocode* dari pembangkitan *token*.

```
password = random(); //harus unik
while (true) {
  time = timeNow() - unixTime();
  password = password + time; //concat
  token = hash(password);
  token = subString(password, 0, 5);
  number = convertToDecimal(token);
  display(number);
  wait(30);
}
```

Token ini akan digunakan untuk menjadi otentikasi tingkat kedua ketika pengguna ingin melakukan transaksi seperti transfer uang, pembayaran, ataupun pembelian. Password akan dibangkitkan secara random saat nasabah pertama kali mengaktifkan fitur e-banking di bank. **Server akan menyimpan *password* tersebut** dan waktu *device* akan disamakan dengan waktu server sehingga ketika pengguna memasukkan *token*, server dapat melakukan operasi pembangkitan *token* dan mencocokannya.

IV. Implementasi dan Analisis Algoritma

Token dapat dibangkitkan dengan menggunakan berbagai fungsi *hash*. Untuk pengujian, *token* akan dibangkitkan dengan beberapa algoritma kriptografi sebagai perbandingan.

Berikut merupakan hasil implementasi dari algoritma pembangkitan *token* dengan fungsi SHA-1, MD5, RIPEMD, dan WHIRPOOL.

Password = Kr1pt0gR4f1		
Fungsi *hash*	**Waktu (dalam s)**	***Token***
SHA1	1463588086	08939455
	1463588116	10721477
	1463588146	05475556
	1463588176	00482886
	1463588206	08074202
MD5	1463588086	16446649
	1463588116	14452479
	1463588146	16004693
	1463588176	16238958
	1463588206	16680918
RIPEMD-128	1463588086	01573817
	1463588116	06653476
	1463588146	03316699
	1463588176	07260847
	1463588206	09121935
WHIRLPOOL	1463588086	08545331
	1463588116	15407115
	1463588146	11931667
	1463588176	11463559
	1463588206	13790833

Berdasarkan tabel diatas, perubahan nilai *token* dari waktu ke waktu tidak dapat diprediksi dan ditemukan polanya. Hal ini menunjukan bahwa keempat algoritma diatas dapat digunakan sebagai fungsi *hash* pembangkit *token*. Akan tetapi, masing-masing fungsi *hash* memiliki kecepatan yang berbeda-beda dalam membuat *digest*.

Berikut merupakan perbandingan kecepatan dari masing fungsi *hash*.

Fungsi *hash*	**MiB/Second**	**Cycles per byte**
SHA-1	153	11.4
MD5	255	6.8
RIPEMD-128	153	11.4
WHIRLPOOL	57	30.5

Berdasarkan perbandingan diatas, algoritma MD5 merupakan algoritma yang paling cepat. Hal ini disebabkan karena algoritma ini hanya menghasilkan *message digest* yang paling pendek (128 bit). Akan tetapi, algoritma ini dapat terkena kolisi yaitu kondisi dimana *message* yang berbeda dapat menghasilkan *message digest* yang sama. Kolisi di pembuatan *token* tidak terlalu bermasalahh mengingat *token* akan diperbaharui setiap 30 detik. Oleh karena itu, **algoritma MD5 cocok digunakan** mengingat *device security token* memilliki kemampuan komputasi yang lemah.

V. Analisis Keamanan

Two factor authentication dengan menggunakan *token* telah memberikan jaminan keamanan yang lebih baik dibandingkan otentikasi yang hanya menggunakan username dan password. Namun, *token* yang digunakan sebagai otentikasi tingkat kedua ini harus tahan terhadap serangan *hacker*. *Token* ini akan berubah setiap 30 detik sekali dan hal ini akan membuat hacker kesulitan untuk menebak *token*. Selain itu, perubahan *token* relatif sulit untuk ditebak karena tidak ditemukan pola yang pasti sehingga hacker akan kesulitan untuk menebak berapa nilai *token* 30 detik kemudian.

Password yang menjadi inputan untuk fungsi *hash* juga dibangkitkan secara acak sehingga hacker tidak bisa menyerang dengan menggunakan *dictionary attack*. Satu cara yang bisa dilakukan oleh hacker adalah menerapkan teknik *brute force* dengan menelusuri seluruh kemungkinan *token*.

Token memiliki angka yang berkisar diantara 0 sampai 16777215. Artinya hacker harus menebak 16.777.216 kombinasi dalam waktu 30 detik atau sekitar 559.241

kombinasi per detik. Hal ini tidak mungkin dilakukan mengingat *response time* dari server hanya sekitar 20 ms, sehingga 1 detik hacker hanya dapat menebak 50 kemungkinan.

Bila aspek kemanannya masih dirasa kurang, *message digest* yang diubah menjadi *token* dapat diperbesar ukurannya sehingga kemungkinan *token* akan semakin banyak. Akan tetapi pengguna lebih sulit dalam memasukan *token*, karena *token* akan berubah setiap 30 detik sekali.

VI. KESIMPULAN

Untuk menyimpulkan, *token* berbasis waktu cocok untuk diterapkan sebagai metode otentikasi transaksi *e-banking*. Dari segi implementasi, hanya diperlukan alat dengan waktu yang sama atau sinkron dengan *server*. Dari segi keamanan, token ini cukup aman karena angka yang dihasilkan sulit ditebak dan akan berubah setiap tiga puluh detik sekali. Selain itu *password* untuk menjadi input message dari fungsi hash juga dibangkitkan secara acak sehingga mempersulit hacker untuk membobol transaksi.

Selain itu, berdasarkan hasil implementasi berbagai fungsi *hash*, algoritma MD5 paling cocok untuk diterapkan dalam pembangkitan *token*. Hal ini dikarenakan MD5 tidak membutuhkan komputasi yang terlalu tinggi sehingga cocok untuk dipakai oleh alat yang memiliki komputasi yang lemah seperti *security token device*.

VII. ACKNOWLEDGMENT

Pertama-tama saya ingin berterima kasih kepada Tuhan karena melalui anugrah-Nya saya bisa menyelesaikan makalah ini. Saya juga sangat berterima kasih kepada dosen saya Dr. Ir. Rinaldi Munir, karena telah memberikan inspirasi serta pengetajuan dalam bidang kriptografi bagi saya. Saya juga berterima kasih kepada Institut Teknologi Bandung, tempat dimana saya menuntut ilmu ketika saya menyelesaikan makalah ini.

REFERENCES

[1] Munir, Rinaldi, Diktat Kuliah Kriptografi, Bandung: Institut Teknologi Bandung.

[2] http://www.virtualdcs.co.uk/blog/should-two-factor-authentication-be-the-norm.html, diakses pada tanggal 18 Mei 2016, pukul 19.30.

[3] Schneier, Bruce. "Cryptanalysis of MD5 and SHA: Time for a New Standard". *Computerworld*. Retrieved 2016-04-20. Much more than encryption algorithms, one-way *hash* functions are the workhorses of modern cryptography."

[4] http://www.cryptopp.com/benchmarks.html, diakses pada tanggal 18 Mei 2016, pukul 23.45.

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, 19 Mei 2016

William Sentosa / 13513026