Pembangkitan Nilai MAC dengan Menggunakan Algoritma Blowfish, Fortuna, dan SHA-256 (MAC-BF256)

Sila Wiyanti Putri [1)]

1) Program Studi Teknik Informatika ITB, Bandung 40132, email: silawp@gmail.com

Abstract *– Integritas data merupakan hal yang penting dalam keamanan. Dalam praktek pengiriman data, komponen integritas dijaga dengan menggunakan* Message Authentication Code *(*MAC*).* MAC *merupakan sebuah algoritma yang dirancang untuk melakukan verifikasi terhadap keaslian dari suatu pesan. Dengan menggunakan* MAC *dapat diketahui apakah pesan tersebut mengalami perubahan (oleh pihak yang tidak berwenang).*
Makalah ini membahas mengenai sebuah alternatif algoritma untuk membangkitkan nilai MAC *dengan menggunakan algoritma* Blowfish*,* Fortuna*,* SHA-256*, dan permutasi. Algoritma* Blowfish *digunakan sebagai algoritma enkripsi simetris untuk mengenkripsi pesan sebelum membangkitkan nilai* MAC *pesan tersebut.* SHA-256 *digunakan sebagai fungsi* hash *untuk membangkitkan tanda tangan digital dari pesan yang sudah terenkripsi.* Fortuna *adalah salah satu algoritma* secure pseudo random number generator *yang digunakan untuk menghasilkan faktor acak dari nilai* hash *yang dihasilkan. Nilai acak pertama (x) akan digunakan sebagai penentu banyaknya permutasi yang akan dilakukan. Untuk setiap permutasi yang dilakukan, dibangkitkan dua buah nilai acak sebagai penanda posisi bit yang akan ditukar. Dengan demikian,* Fortuna *akan menmbangkitkan nilai acak sebanyak 2x+1 kali.*

Kata Kunci*: MAC, Blowfish, Fortuna, SHA-256, permutasi, MAC-BF256*

1. PENDAHULUAN

Manusia sebagai makhluk sosial memiliki kebutuhan untuk berkomunikasi. Salah satu bentuk dari komunikasi yang paling dasar adalah saling berkirim pesan. Dengan teknologi yang ada sekarang ini, proses pengiriman pesan dapat dilakukan dengan semakin mudah, di mana saja dan kapan saja dengan menggunakan berbagai media.

Dengan kemudahan ini, bentuk serangan yang mungkin terjadi juga menjadi semakin beraneka ragam. Hak ini menjadikan pesan atau dokumen yang dikirimkan menjadi semakin mudah pula dicuri, dilihat, atau diubah oleh pihak yang tidak bertanggung jawab. Hal tersebut menjadi masalah yang cukup serius karena dokumen yang dikirim dapat merupakan dokumen penting yang harus dijaga keintegritasannya.

Integritas data adalah salah satu komponen dalam keamanan yang menjadi tolak ukur keutuhan pesan atau dokumen. Komponen ini menjamin bahwasannya jika pesan atau dokumen yang dikirimkan diubah, ditambah, maupun dikurangi oleh pihak yang tidak berwenang, maka pihak yang berwenang akan mengetahuinya. Salah satu cara menerapkan komponen integritas data dalam pesan adalah dengan melalui sebuah algoritma yang dinamakan *MAC*.

Dalam makalah ini, penulis akan merancang algoritma untuk membangkitkan nilai *MAC* dengan menggunakan algoritma *Blowfish, Fortuna, SHA-256,* dan permutasi. Penulis juga mencoba melakukan implementasi rancangan algoritma ini dan kemudian melakukan analisa terhadap hasil implementasi tersebut. Dengan makalah ini, diharapkan pembaca dapat mendapatkan informasi yang berguna dan membantu dalam perkembangan ilmu kriptografi.

2. LANDASAN TEORI

2.1. Message Authentication Code (MAC)

Message Authentication Code (MAC) adalah sebuah algoritma yang didesain secara khusus untuk memeriksa keaslian sebuah pesan atau dokumen sehingga apabila pesan tersebut mengalami perubahan yang dilakukan oleh pihak yang tidak berwenang, maka pihak yang berwenang dapat mengetahuinya.

Konsep dasar *MAC* adalah membuat sebuat tanda tangan digital dari sebuah pesan dengan menggunakan kunci simetris. Pada umumnya, nilai *MAC* sebuah pesan yang dikirimkan akan disisipkan ke dalam pesan tersebut. Apabila pesan mengalami perubahan, maka nilai *MAC* pesan tersebut juga akan berubah. Perubahan nilai *MAC* ini dapat digunakan untuk mengecek keaslian pesan. Selama pihak lain (di luar pihak yang bertukar pesan) tidak mengetahui kunci simetris yang digunakan untuk membangkitkan *MAC,* maka nilai *MAC* pesan yang telah diubah tidak dapat dibangkitkan ulang. Hal ini menjadi dasar bahwasannya *MAC* dapat digunakan untuk mengecek keaslian pesan.

Saat ini terdapat dua macam algoritma *MAC* yang

telah ditemukan, yaitu algoritma *MAC* berbasiskan algoritma kriptografi simetris (contoh: *MAC* menggunakan *DEA*), dan algoritma *MAC* berbasiskan fungsi *hash* (contoh: *HMAC-MD5, HMAC-SHA*).

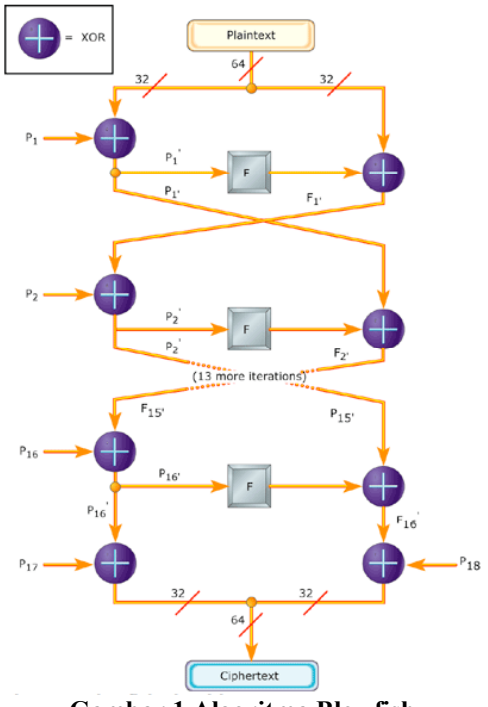
Dalam makalah ini, penulis akan merancang dan mengimplementasikan sebuah alternatif algoritma *MAC* yang disebut *MAC-BF256.* Algoritma ini mengkombinasikan empat buah algoritma untuk dapat menghasilkan nilai *MAC*. Keempat algoritma tersebut adalah algoritma *Blowfish, Fortuna, SHA-256,* dan permutasi. Masing-masing algoritma akan dibahas lebih dalam pada sub bab selanjutnya.

2.2. Algoritma Blowfish

Algoritma *Blowfish* dirancang oleh Bruce Scheiner dan dipublikasikan pertama kali pada tahun 1993. Algoritma *Blowfish* adalah sebuah algoritma enkripsi simetris, hal ini berarti algoritma ini menggunakan kunci yang sama baik untuk melakukan enkripsi maupun dekripsi.

Algoritma ini juga merupakan salah satu algoritma *cipher* blok, karena algoritma *Blowfish* membagi sebuah pesan manjadi blok-blok yang panjangnya sama pada saat melakukan proses enkripsi dan dekripsi pesan.

Algoritma ini menggunakan ukuran blok masukan 64 bit. Jika pesan yang masuk bukan kelipatan 8 byte, maka akan disisipkan bit-bit tambahan pada pesan tersebut (*padding*). Panjang kunci yang digunakan dapat bervariasi diantara 32 bit sampai dengan 448 bit. Algoritma *Blowfish* juga menggunakan sturuktur jaringan Feistel di dalamnya.

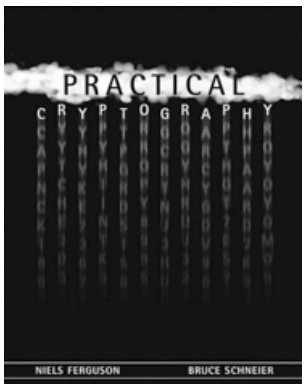


Gambar 1 Algoritma Blowfish

Dalam aplikasinya, algoritma ini dapat menggantikan algoritma yang sudah ada sebelumnya seperti DES atau IDEA. Karena sampai sekarang algoritma Blowfish tidak dipatenkan, maka algoritma ini bebas digunakan oleh masyarakat luas.

2.2. Algoritma Fortuna

Algoritma *Fortuna* merupakan salah satu algoritma pembangkit bilangan acak semu (*Pseudo-Random Number Generator/PRNG*). Algoritma ini pertama kali dipublikasikan oleh Niels Ferguson dan Bruce Schneier dalam bukunya *Practical Cryptography*. Nama Fortuna diambil dari nama dewi romawi, yaitu dewi keberuntungan dan kesempatan.



Gambar 2 Practical Cryptography oleh Ferguson dan Schneier

Algoritma ini dirancang secara spesifik sebagai algoritma pembangkit bilangan acak yang aman secara kriptografi (*secure PRNG*). Bagian-bagian dari algoritma ini adalah:

1. Generator
2. Akumulator entropi
3. Penyedia *state*

Generator adalah bagian yang menerima masukan *state*, dan berdasarkan itu akan membangkitkan data acak semu yang tidak terbatas jumlahnya. Generator dapat berupa algoritma blok *cipher* apa saja, misalnya *AES, Twofish,* atau *Serpent*. Ide dasar dari Generator dalam *Fortuna* adalah menjalankan *cipher* dengan menggunakan iterator, sehingga nilai yang dihasilkan dari proses awal, digunakan sebagai masukan dalam proses selanjutnya.

Akumulator entropi bertanggung jawab dalam mengumpulkan data acak yang dihasilkan generator untuk memperbaharui faktor acak yang tersimpan dalam penyedia *state*. Hal ini penting untuk memastikan data acak yang dibangkitkan selalu memiliki faktor acak yang baru. Dalam *Fortuna*, terdapat 32 *pool* entropi, masing-masingnya saling menyediakan faktor acak untuk satu sama lain.

Penyedia state adalah tempat menyimpan faktor acak awal untuk masukan Generator. Dengan *state* yang

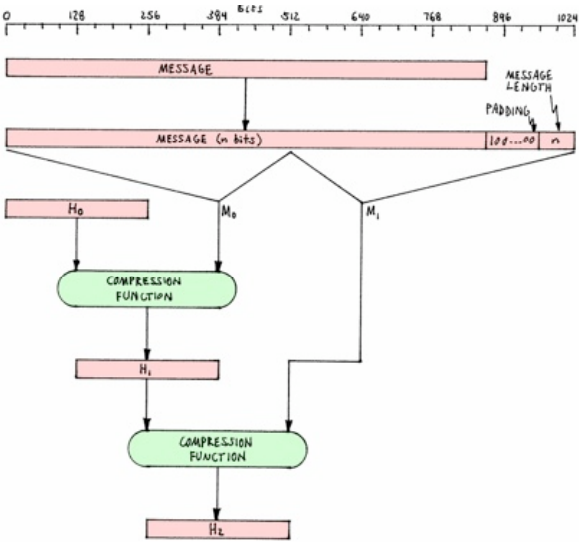
disediakan oleh penyedia state, Generator dapat langsung membangkitkan data acak begitu proses dimulai (tidak perlu menunggu).

2.3. Algoritma SHA-256

SHA adalah fungsi *hash* yang dibuat oleh *NIST* dan digunakan bersama *DSS (Digital Signature Standard)*. Oleh *NSA, SHA* dinyatakan sebagai standar fungsi *hash* satu arah. *SHA* dapat dikatakan adalah penerus dari pendahulunya yang sudah lebih dulu populer dan digunakan secara luas di bidang kriptografi, yaitu *MD5*.

SHA dapat disebut aman karena aman karena algoritma ini dirancang sedemikian rupa sehingga secara komputasi tidak mungkin menemukan pesan yang berkoresponden denggan *message digest* yang diberikan.

Fungsi *hash* yang paling umum digunakan adalah *SHA-1*. Algoritma ini diimplementasikan antara lain dalam aplikasi dan protokol seperti *TLS, SSL, PGP, SSH, S/MIME*, dan *Ipsec.* Pada tahun 1995, dipublikasikan *SHA-2*. *SHA-256* adalah salah satu varian dari *SHA-2*.



Gambar 3 Diagram iterasi SHA-256

SHA-256 secara garis besar terdiri dari 2 tahap, yaitu:

1. Tahap pra proses
2. Tahap komputasi *hash*

Tahap pra proses terdiri dari penyisipan bit-bit pengganjal, pembagian pesan menjadi blok-blok berukuran *m* bit, dan inisiasi nilai-nilai yang akan dipergunakan pada proses komputasi *hash.* Tahap komputasi *hash* akan membangkitkan *message schedule* dari pesan yang sudah disisipi bit pengganjal. *Message schedule* yang dihasilkan kemudian akan digunakan sebagai masukan dalam proses iteratif pembangkitan nilai *hash*. Nilai *hash* yang terakhir kemudian akan menentukan *message digest*.

2.4. Algoritma Permutasi

Permutasi adalah penyusunan kembali suatu kumpulan objek dalam urutan yang berbeda dari urutan yang semula.

Di bawah ini adalah formula untuk menghitung banyaknya permutasi yang mungkin, dengan anggota set tidak boleh berulang:

$$P(n,r) = \frac{n!}{(n-r)!}$$

n: banyaknya anggota set
r: ukuran permutasi yang diinginkan

Untuk membangkitkan seluruh permutasi yang mungkin, dapat digunakan algoritma sebagai berikut:

```
function permutation(k, s) {
    var int factorial:= 1;
    for j:= 2 to length(s) {
       factorial:= factorial* (j-1);        // factorial= (j-1)!
       swap s[j- ((k / factorial) mod j)] with s[j];
    }
    return s;
}
```

Gambar 4 Algoritma permutasi

3. PEMBAHASAN

Saat ini terdapat dua macam algoritma *MAC* yang telah ditemukan, yaitu algoritma *MAC* berbasiskan algoritma kriptografi simetris (contoh: *MAC* menggunakan *DEA*), dan algoritma *MAC* berbasiskan fungsi *hash* (contoh: *HMAC-MD5, HMAC-SHA*).

Dalam makalah ini, penulis akan merancang dan mengimplementasikan sebuah alternatif algoritma *MAC* yang disebut *MAC-BF256.* Algoritma ini mengkombinasikan empat buah algoritma untuk dapat menghasilkan nilai *MAC*. Keempat algoritma tersebut adalah algoritma *Blowfish, Fortuna, SHA-256,* dan permutasi.

3.1. Alasan Pemilihan Algoritma

Algoritma *Blowfish* digunakan sebagai algoritma enkripsi simetris untuk mengenkripsi pesan sebelum membangkitkan nilai *MAC* pesan tersebut. Algoritma ini dipilih karena memiliki *s-box* besar yang tidak tergantung pada kunci sehingga memiliki tingkat keamanan yang cukup tinggi, ia juga menggunakan struktur *Feistel*, sehingga proses pembalikannya tidak rumit. Mode yang digunakan adalah *ECB*, sebagai mode paling sederhana.

SHA-256 digunakan sebagai fungsi *hash* untuk membangkitkan tanda tangan digital dari pesan yang sudah terenkripsi. *SHA-256* dipilih karena merupakan algoritma *hash* standar dan sampai sekarang belum

ditemukan *collision*, sehingga baik untuk digunakan. *Fortuna* adalah salah satu algoritma *secure pseudo random number generator*. Dalam algoritma *MAC-BF256*, *Fortuna* digunakan untuk menghasilkan faktor acak dari nilai *hash* yang dihasilkan. Nilai acak pertama (x) akan digunakan sebagai penentu banyaknya permutasi yang akan dilakukan. Untuk setiap permutasi yang dilakukan, dibangkitkan dua buah nilai acak sebagai penanda posisi bit yang akan ditukar. Dengan demikian, *Fortuna* akan membangkitkan nilai acak sebanyak $2x+1$ kali.

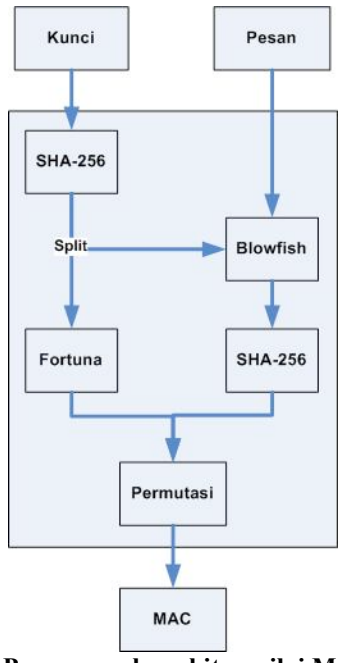
Selain alasan yang sudah dijelaskan di atas, algoritma-algoritma ini dipilih karena ketiganya merupakan algoritma yang bebas paten (dapat dipergunakan oleh publik secara terbuka), sehingga memudahkan dalam proses implementasi.

3.2. Algoritma MAC-BF256

3.2.1. Pembangkitan Nilai *MAC*

Untuk membangkitkan nilai *MAC* yang akan disisipkan ke dalam pesan, algoritma *MAC-BF256* membutuhkan dua buah data masukan yaitu:

1. Pesan yang ingin dibangkitkan nilai *MAC* nya. Apabila panjang pesan bukan kelipatan 8 byte, maka akan disisipkan bit-bit pengganjal sehingga penjang pesan merupakan kelipatan 8 byte.
2. Kunci simetris yang ingin digunakan untuk membangkitkan nilai *MAC*.



Gambar 5 Proses pembangkitan nilai MAC dengan MAC-BF256

Proses pembangkitan nilai *MAC* dimulai dengan melakukan proses *hashing* terhadap kunci simetris yang digunakan, dalam hal ini melalui algoritma *SHA-256*. Hasil dari proses ini akan dipisahkan menjadi dua bagian, bagian pertama akan menjadi kunci dalam enkripsi pesan dengan algoritma *Blowfish*, sedangkan yang kedua akan menjadi data masukan algoritma *Fortuna*.

Kemudian dilakukan enkripsi terhadap pesan dengan menggunakan algoritma *Blowfish*. Kunci yang digunakan untuk mengenkripsi pesan adalah bagian pertama dari hasil *hash* kunci simetris. Proses enkripsi ini dimaksudkan supaya pesan yang sama dengan kunci yang berbeda dapat menghasilkan nilai *MAC* yang berbeda.

Sementara itu bagian kedua dari hasil *hash* kunci simetris akan digunakan untuk membangkitkan bilangan acak oleh algoritma *Fortuna*. Bilangan pertama yang dibangkitkan (x) merepresentasikan banyaknya permutasi yang dilakukan terhadap hasil enkripsi pesan dengan algoritma *Blowfish*. Untuk setiap permutasi yang dilakukan, dibangkitkan dua buah nilai acak sebagai penanda posisi bit yang akan ditukar. Dengan demikian, *Fortuna* akan membangkitkan nilai acak sebanyak $2x+1$ kali.

Proses ini akan menghasilkan nilai *MAC* sepanjang 256 bit yang kemudian akan disisipkan ke dalam pesan.

3.2.2. Verifikasi Nilai MAC

Untuk melakukan verifikasi suatu pesan terhadap nilai *MAC* yang dikandungnya, algoritma *MAC-BF256* membutuhkan dua buah data masukan yaitu:

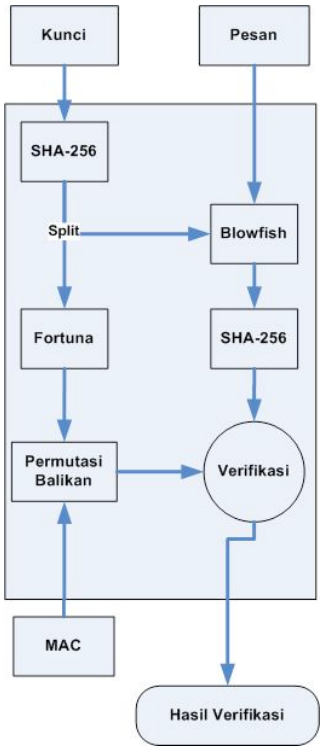
1. Pesan yang ingin diperiksa keasliannya.
2. Kunci simetris yang akan digunakan untuk melakukan proses verifikasi.

Dalam proses verifikasi nilai *MAC*, algoritma *MAC-BF256* menawarkan prinsip verifikasi ganda. Verifikasi ini bekerja dengan tahapan sebagai berikut:

1. Penerima pesan melakukan verifikasi nilai *MAC* yang disisipkan pada pesan yang dikirim (*MAC-S*) dengan nilai *MAC* yang dibangkitkan oleh penerima pesan (*MAC-R*).
2. Apabila berbeda, maka pesan telah diubah.
3. Apabila sama, maka dilakukan proses permutasi balikan (*R-Permutation*) untuk mendapatkan nilai *hash* (*Hc*) sebelum dipermutasi. Proses permutasi balikan ini dapat dilakukan karena proses permutasi yang dilakukan pada saat pembangkitan nilai *MAC* bersifat *reversible*.
4. Penerima pesan melakukan verifikasi nilai *Hc* yang didapat dari *MAC-S* dengan nilai *hash* pesan yang telah terenkripsi.
5. Apabila berbeda, maka pesan telah diubah. Sebaliknya jika ternyata didapati nilai tersebut sama, maka pesan yang diterima dapat dijamin

keasliannya.

Proses verifikasi yang dilakukan oleh algoritma *MAC-BF256* dapat dilihat pada diagram di bawah.



Gambar 6 Proses verifikasi dengan MAC-BF256

4. IMPLEMENTASI

Dalam melakukan implementasi, penulis menggunakan bahasa pemrograman Java, dengan lingkungan pembangunan sebagai berikut :

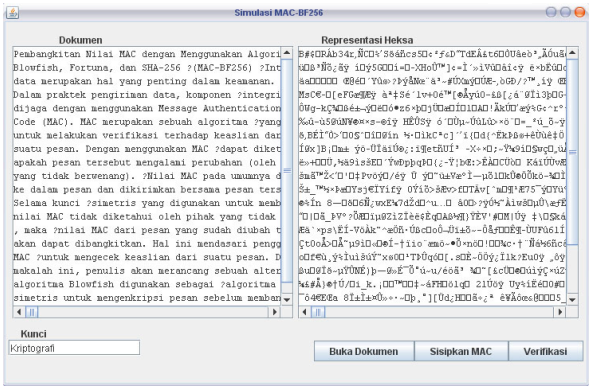
1. Java Development Kit (JDK) 6.
2. Perangkat pembangunan NetBeans IDE 5.5.

Implementasi yang dilakukan meliputi implementasi proses pembangkitan nilai *MAC,* penyisipan nilai *MAC* ke dalam pesan, dan proses verifikasi pesan.

Aplikasi yang dibuat oleh penulis terdiri dari dua buah kelas. Yang pertama kelas "Interface" yang merupakan kelas yang mengatur tampilan antarmuka, kelas ini merupakan kelas turunan dari JFRAME, yang kedua adalah kelas "MACBF256" yang berisi fungsi-fungsi untuk melakukan proses pembangkitan nilai *MAC,* penyisipan nilai *MAC* ke dalam pesan, dan proses verifikasi pesan.

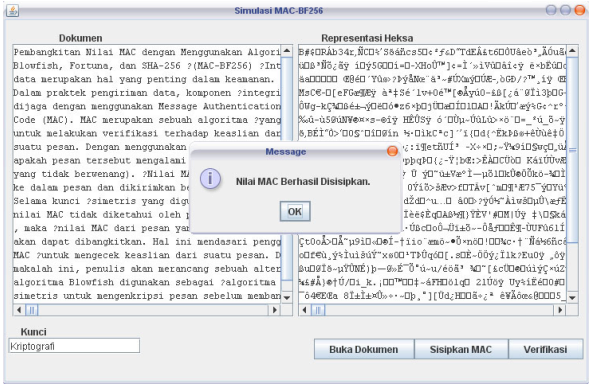
Aplikasi terdiri dari satu buah *form* untuk membuka pesan yang diinginkan (berupa dokumen) dan memasukan kunci simetris yang akan digunakan. Pengguna kemudian dapat memilih apakah ingin membangkitkan nilai *MAC* dan menyisipkannya, atau melakukan verifikasi terhadap pesan. Pilihan ini dilakukan dengan cara menekan tombol yang bersesuaian. Dokumen yang dibuka akan ditunjukkan isinya dan representasi heksanya pada dua buah *text area* di bagian tengah antarmuka.

Tampilan dari aplikasi yang telah dibangun dapat dilihat pada Gambar 7.



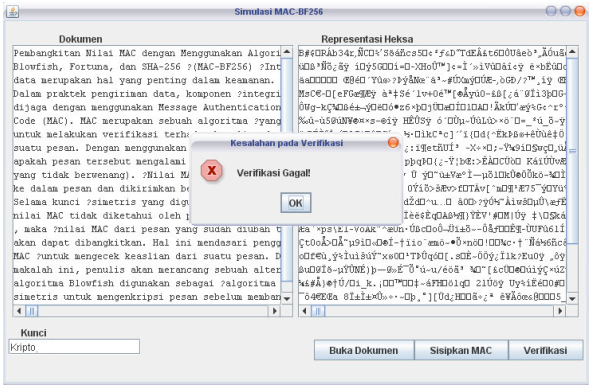
Gambar 7 Tampilan antarmuka aplikasi

Jika nilai *MAC* berhasil dibangkitkan dan disisipkan pada dokumen, maka akan muncul pesan berhasil, representasi heksa dari dokumen tersebut juga akan berubah karena sekarang sudah disisipkan nilai *MAC.*



Gambar 8 Penyisipan nilai MAC

Untuk proses verifikasi, pengguna cukup membuka dokumen, memasukkan kunci simetris, dan menekan tombol Verifikasi. Jika verifikasi berhasil, akan muncul pesan sukses. Namun jika dokumen telah diubah, atau pengguna tidak memasukkan kunci simertik yang sesuai, maka akan muncul pesan bahwa proses verifikasi gagal.



5. ANALISIS HASIL

Dalam menguji algoritma *MAC-BF256*, penulis meggunakan beberapa kasus uji. Dalam kasus-kasus uji yang diterapkan, istilah pesan selanjutnya mengacu pada dokumen tekstual dengan ukuran 49 KB.

5.1. Kasus Normal

Kasus uji pertama, pesan dibuka dengan menggunakan aplikasi, kemudian dilakukan penyisipan *MAC* dengan menggunakan kunci simetri "KRIPTOGRAFI". Proses penyisipan berhasil dilakukan dengan baik, hasil penyisipan dapat dilihat pada representasi heksa yang terdapat pada *text area* di sebelah kanan. Pesan kemudian diverifikasi dengan menggunakan kunci yang benar, yaitu "KRIPTOGRAFI". Muncul notifikasi bahwa verifikasi berhasil.

5.2. Kasus Kunci Tidak Bersesuaian

Kasus uji kedua, pesan dibuka dengan menggunakan aplikasi, kemudian dilakukan penyisipan *MAC* dengan menggunakan kunci simetri "KRIPTOGRAFI". Proses penyisipan berhasil dilakukan dengan baik, hasil penyisipan dapat dilihat pada representasi heksa yang terdapat pada *text area* di sebelah kanan. Pesan kemudian diverifikasi dengan menggunakan kunci yang tidak sesuai dengan kunci yang digunakan untuk membangkitkan nilai *MAC*, yaitu "KRIPTO". Muncul notifikasi bahwa verifikasi gagal dilakukan.

5.3. Kasus Dokumen Diubah

Kasus uji berikutnya, pesan dibuka dengan menggunakan aplikasi, kemudian dilakukan penyisipan *MAC* dengan menggunakan kunci simetri "KRIPTOGRAFI". Proses penyisipan berhasil dilakukan dengan baik, hasil penyisipan dapat dilihat pada representasi heksa yang terdapat pada *text area* di sebelah kanan. Pesan kemudian dibuka dengan editor teks untuk melakukan modifikasi isi pesan. Pesan kemudian diverifikasi dengan menggunakan kunci yang benar, yaitu "KRIPTOGRAFI". Muncul notifikasi bahwa verifikasi gagal dilakukan.

6. KESIMPULAN

Dari keseluruhan isi makalah ini, dapat diambil kesimpulan sebagai berikut:

1. Metode *Message Authentication Code (MAC)* merupakan metode yang efektif untuk melakukan verifikasi dokumen terkait dengan integritas data.
2. *MAC-BF256* merupakan algoritma alternatif dalam membangkitkan nilai *MAC* dan melakukan verifikasi pesan.
3. *MAC-BF256* mengadaptasi berbagai macam algoritma di dalamnya, yaitu: algoritma *Blowfish* untuk mengenkripsi pesan, algoritma *Fortuna* sebagai pembangkit bilangan acak, *SHA-256* sebagai fungsi *hash* satu arah, dan permutasi.
4. Algoritma-algoritma yang menyusun *MAC-BF256* seluruhnya adalah algoritma bebas paten sehingga dapat dipergunakan secara bebas untuk keperluan implementasi.
5. Dengan *MAC-BF256*, proses verifikasi yang dilakukan adalah verifikasi ganda, sehingga dokumen yang dinyatakan lulus verifikasi, lebih terjamin keasliannya.

DAFTAR PUSTAKA

Munir, Rinaldi, "Kriptografi", Institut Teknologi Bandung, 2006.

http://www.us.design-reuse.com/articles/5922/encrypting-data-with-the-blowfish-algorithm.html diakses pada Januari 2008

http://www.clipperz.com/learn_more/crypto_foundations/prng_fortuna_algorithm diakses pada Januari 2008

http://www.clipperz.com/learn_more/crypto_foundations/sha_2_secure_hash_algorithms diakses pada Januari 2008