ALGORITMA MAC BERBASIS FUNGSI HASH SATU ARAH

Irma Juniati – NIM : 13506088

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung

Jl. Ganesha 10, Bandung

e-mail : if16088@students.if.itb.ac.id, irma.juniati@gmail.com

Abstrak

Pada era digital ini, tersedianya suatu mekanisme untuk memeriksa integritas pesan yang ditransmisikan melalui media elektronik apapun mutlak dibutuhkan. Keaslian pengirim pesan menjadi sangat penting, begitu juga dengan keaslian pesan yang dikirimkan dan diterima. *Message Authentication Code* (*MAC*) merupakan fungsi *hash* satu arah yang menggunakan sebuah kunci rahasia, yang dimiliki oleh pengirim dan penerima pesan. *MAC* digunakan untuk otentikasi pesan dan menjamin integritas isi arsip.
Makalah ini akan membahas sedikit *overview* tentang *MAC*, beserta kelebihan dan kekurangannya. Makalah ini juga akan membahas tentang dua pendekatan dalam merancang algoritma *MAC*. Akan tetapi, pembahasan akan lebih ditekankan kepada algoritma *MAC* berbasis fungsi *hash* satu arah, perancangan yang diambil sebagai contoh adalah *HMAC*. Pembahasan mengenai algoritma *MAC* berbasis *cipher* blok (*block cipher*) tidak akan dilakukan secara mendalam. Selain itu, juga akan dibahas sedikit mengenai beberapa modifikasi yang dilakukan terhadap *MAC* untuk meningkatkan performansi.

Selain itu, penulis melakukan implementasi algoritma *MAC* melalui sebuah program kecil, yang dapat mensimulasikan bagaimana proses otentikasi pesan dengan menggunakan *MAC*.

Kata kunci : *message authentication code*, *MAC*, fungsi *hash*, *HMAC*

1. PENDAHULUAN

1.1 *MAC* dan Aplikasinya

MAC adalah fungsi *hash* satu arah yang menggunakan kunci rahasia (*secret key*) dalam pembangkitan nilai *hash*. Dengan kata lain, nilai *hash* adalah fungsi dari pesan dan kunci. *MAC* disebut juga *keyed hash function* atau *key-dependent one-way hash function*. *MAC* memiliki sifat dan properti yang sama seperti fungsi *hash* satu arah lainnya, hanya saja terdapat komponen kunci pada *MAC*. Kunci ini juga akan digunakan oleh penerima pesan untuk melakukan verifikasi nilai *hash*.

Secara matematis, *MAC* dinyatakan sebagai berikut:

$$MAC = C_K(M)$$

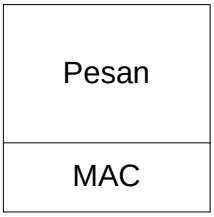
di mana *MAC* = nilai *hash*, *C* = fungsi *hash*, *M* = pesan yang akan dikirim, dan *K* = kunci rahasia.

Fungsi *C* memampatkan pesan *M* yang berukuran sembarang dengan menggunakan kunci *K*. Fungsi ini bersifat *many-to-one*, yang berarti beberapa pesan yang berbeda dapat memiliki *MAC* yang sama. Namun pada kenyataannya, menemukan pesan-pesan seperti itu sangatlah sulit (secara komputasi).

MAC digunakan untuk otentikasi pesan tanpa perlu merahasiakan (melakukan enkripsi) pesan. Mula-mula, pengirim pesan melakukan perhitungan *MAC* dari pesan yang akan ia kirim dengan menggunakan kunci rahasia *K*. Dalam hal ini diasumsikan bahwa pengirim dan oenerima pesan sudah berbagi kunci rahasia. Kemudian, *MAC* yang diperoleh ini dilekatkan pada pesan. Selanjutnya, pesan dikirim bersama-sama dengan *MAC* ke penerima.

Penerima kemudian akan menggunakan kunci *K* yang sama untuk menghitung *MAC* pesan dan membandingkannya dengan *MAC* yang ia terima. Jika kedua *MAC* ini sama, maka dapat disimpulkan bahwa pesan dikirim oleh orang yang seharusnya dan tidak terjadi pengubahan isi pesan selama transmisi. Kesimpulan tersebut dapat diambil berdasarkan dua kondisi berikut:

- Jika pesan tidak berasal dari orang yang seharusnya, *MAC* yang dihitung penerima tidak akan sama dengan *MAC* yang ia terima, karena pihak ketiga tidak mengetahui kunci rahasia *K*.
- Jika isi pesan diubah selam transmisi, maka *MAC* yang dihitung tidak akan sama dengan *MAC* yang diterima.



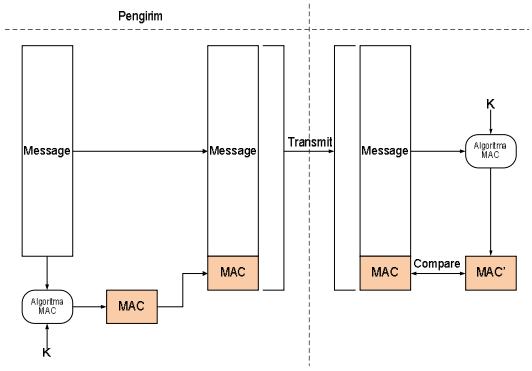
Gambar 1 *MAC* dilekatkan di akhir pesan untuk otentikasi

Aplikasi *MAC* lainnya adalah untuk menjaga otentikasi arsip yang digunakan oleh dua atau lebih pengguna, serta untuk menjamin integritas (keaslian) isi arsip terhadap perubahan, misalnya karena serangan virus. Caranya adalah dengan menghitung *MAC* dari isi arsip, kemudian menyimpannya ke dalam sebuah tabel dalam basis data. Jika menggunakan fungsi *hash* satu arah biasa, misalnya *MD5*, maka virus dapat menghitung nilai *hash* yang baru dari arsip yang sudah diubah, lalu mengganti nilai *hash* yang lama di dalam tabel. Tetapi, jika menggunakan *MAC*, virus tidak dapat melakukan hal ini karena tidak mengetahui kunci.

1.2 Algoritma *MAC*

Perancangan algoritma *MAC* dapat dilakukan dengan dua pendekatan. Pendekatan pertama adalah dengan menggunakan algoritma kriptografi kunci simetri berbasis blok (*block cipher*), dan pendekatan kedua adalah dengan menggunakan fungsi *hash* satu arah. Namun dalam pembahasan selanjutnya dalam makalah ini, pendekatan yang akan diuraikan secara lebih rinci adalah pendekatan kedua.

Gambar di bawah ini merupakan skema yang mengilustrasikan penggunaan *MAC* untuk otentikasi pesan.



Gambar 2 Skema *MAC* dan otentikasi pesan

2. ISI

2.1 *MAC* dan Tandatangan Digital

MAC berbeda dengan tanda tangan digital, meskipun prosesnya terlihat mirip. *MAC* didapatkan dengan kunci rahasia yang sama, sedangkan pada tanda tangan digital terdapat dua kunci, yaitu kunci publik dan kunci privat. Dengan demikian, pengirim dan penerima pesan sebelumnya harus melakukan persetujuan dulu tentang kunci rahasia yang akan digunakan. Ini berarti *MAC* mengimplementasikan kriptografi kunci simetri.

Oleh karena itu, *MAC* tidak memenuhi salah satu aspek kriptografi, yaitu *non-repudiation*, yang disediakan oleh tanda tangan digital. Semua pengguna yang dapat melakukan verifikasi *MAC* juga dapat membangkitkan *MAC* untuk pesan-pesan lainnya. Hal ini berbeda dengan tanda tangan digital yang menerapkan kriptografi kunci nonsimetri. Karena pemegang kunci privat hanyalah si pengirim pesan, tanda tangan digital membuktikan bahwa dokumen ditandatangani oleh pemegang kunci. Oleh karena itu, tanda tangan digital memenuhi aspek *non-repudiation*.

2.2 Perancangan Algoritma *MAC*

2.2.1 Algoritma *MAC* berbasis *block cipher*

Dua mode *block cipher* yang dapat digunakan untuk membangkitkan *MAC* adalah mode *CBC* dan *CFB*. Nilai *hash* yang menjadi *MAC* adalah hasil enkripsi blok terakhir. Misalkan *DES* digunakan sebagai *cipher block*, maka ukuran blok adalah 64 bit, dan kunci rahasia *MAC* adalah kunci *DES* yang panjanganya 56 bit.

Data Authentication Algortihm (*DAA*) adalah algoritma *MAC* yang berbasis *DES-CBC* yang digunakan secara luas. *DAA* menggunakan *IV* (*Initialization Vector*) = 0 dan bit-bit *padding* yang seluruhnya adalah bit 0. *DAA* mengenkripsi pesan dengan menggunakan *DES* dalam mode *CBC*. Hasil enkripsi blok terakhir menjadi *MAC*, atau hanya mengambil *n* bit paling kiri ($16 \leq n \leq 64$) dari hasil enkripsi blok terakhir sebagai *MAC*.

2.2.2 Algoritma *MAC* berbasis fungsi *hash* satu arah

Pada awalnya, algoritma *MAC* sebagian besar dikembangkan dengan basis *block cipher*. Namun, dewasa ini, pengembangan dengan basis fungsi *hash* satu arah seperti *MD5* dan *SHA* lebih disenangi. Alasan di balik pemilihan itu sangatlah sederhana. Komputasi fungsi-fungsi *hash* yang

telah tersedia jauh lebih cepat daripada *block cipher* dalam implementasinya.
Namun, fungsi *hash* bukan dirancang untuk otentikasi pesan. Pengguna perlu memodifikasi dalam penggunaannya sehingga fungsi-fungsi tersebut bisa digunakan untuk otentikasi pesan.
Contoh pemakaian secara sederhana adalah sebagai berikut:
Sambung pesan *M* dengan kunci *K*, lalu menghitung nilai *hash* dari hasil penyambungan itu.

$$H(M, K)$$

Nilai *hash* ini adalah *MAC* dari pesan yang ingin dikirimkan. Kemudian, pengirim mengirimkan *M* dan *MAC* kepada penerima, dan selanjutnya penerima akan melakukan otentikasi terhadap pesan karena ia mengetahui kunci rahasia *K*. Contoh penggunaan secara sederhana ini penulis implementasikan dan dapat dilihat pada bagian 2.4.

Salah satu contoh perancangan algoritma *MAC* berbasis fungsi *hash* adalah HMAC.

2.3 *Keyed-Hash Message Authentication Code* (HMAC)

2.3.1 *Overview*

Dalam kriptografi, *keyed-Hash Message Authentication Code* (HMAC) merupakan salah satu varian dari *MAC* yang nilainya dihitung dengan algoritma tertentu yang melibatkan fungsi *hash* yang sering dipakai dalam kriptografi, dikombinasikan dengan sebuah kunci rahasia. Seperti jenis *MAC* lainnya, *HMAC* juga digunakan untuk melakukan verifikasi terhadap integritas pesan dan otentikasi pesan. Perhitungan *HMAC* dapat menggunakan fungsi *hash* iteratif apapun, misalnya *MD5* atau *SHA*-1.

Kekuatan *HMAC* sangat bergantung pada kekuatan fungsi *hash* yang digunakan, yaitu pada panjang keluaran nilai *hash* yang dihasilkan (dalam bit), juga pada ukuran kualitas kunci rahasia yang digunakan.

Pembuatan dan analisis terhadap *HMAC* pertama kali dilakukan pada tahun 1996 oleh Mihir Bellare, Ran Canetti, dan Hugo Krawczyk. FIPS PUB 198 melakukan generalisasi dan standarisasi penggunaan *HMAC*. HMAC-SHA-1 dan HMAC-MD5 digunakan pada protokol Ipsec dan TLS.

2.3.2 Implementasi

Definisi formal *HMAC* secara matematis dapat dituliskan sebagai berikut:

$$\text{HMAC}(K, m) = h\{(K \oplus \text{opad}) \,.\, h\{(K \oplus \text{ipad}) \,.\, m\}\},$$

di mana:

$h\{\}$: fungsi *hash*
K : kunci rahasia, yang harus ditambahkan bit-bit *padding* 0 hingga sepanjang ukuran blok pada fungsi *hash* yang digunakan
m : pesan yang akan diotentikasi
. : melambangkan konkatenasi
$\oplus$: *exclusive or* (*XOR*)
opad : *outer padding* = 0x5c5c5c…5c5c
ipad : *inner padding* = 0x363636…3636

Prosedur berikut menggambarkan bagaimana *HMAC* diimplementasikan.

```
function hmac(key, message)
    opad = [0x5c * blocksize]
    ipad = [0x36 * blocksize]

//kunci yang lebih panjang dari ukuran
blok diperpendek
    if (length(key) > blocksize) then
        key = hash(key)
    end if

    for i from 0 to length(key) – 1 step 1
        ipad[i] = ipad[i] ⊕ key[i]
        opad[i] = opad[i] ⊕ key[i]
    end for

    return hash(opad . hash(ipad .
message))

end function
```

2.3.3 Prinsip Perancangan

Spesifikasi perancangan *HMAC* didasari oleh adanya serangan-serangan yang dilakukan dengan mencoba mengombinasukan suatu kunci tertentu dengan fungsi-fungsi *hash*. Sebagai contoh, seseorang bisa saja mengasumsikan bahwa nilai *HMAC* diperoleh dengan menghitung nilai *hash* dari pesan yang sudah dikonkatenasi dengan kunci rahasia tertentu.

Metode ini memiliki kelemahan yang sangat mendasar. Seseorang bisa saja menambahkan isi pesan tanpa mengetahui kunci dan kemudian memperoleh *MAC* yang valid. Untuk mengatasi hal ini, seseorang dapat mengakalinya dengan mencantumkan panjanga pesan di bagian awal pesan yang akan dikirimkan, meskipun pada akhirnya cara ini telah dibuktikan mempunyai berbagai kelemahan.

Cara lain yang lebih baik yang dapat dilakukan adalah menghitung nilai *MAC* dengan menghitung nilai *hash* dari {kunci . pesan . kunci}. Dalam hal ini, kunci dikonkatenasi dengan isi pesan pada bagian awal dan akhir, baru kemudian dihitung nilai *hash*-nya. Namun, banyak analisis telah

membuktikan bahwa cara ini masih kurang efektif, bahkan jika kita menggunakan dua kunci yang berbeda, yaitu kunci1 dan kunci2.

Oleh karena itu, dicari cara sedemikian rupa sehingga keamanan *HMAC* dapat lebih terjamin. Nilai *HMAC* dilakukan dengan menghitung nilai *hash* dari *h*{kunci1 . *h*{kunci2 . pesan}}. Fungsi *hash* pada bagian luar memiliki parameter yang berupa nilai *hash* dari pesan yang dikonkatenasi dengan kunci2. Bit-bit *padding* internal dan eksternal yang digunakan dipilih sedemikian rupa sehingga memiliki tingkat kemiripan yang sangat jauh satu sama lain, hanya mempunyai sedikit bit yang sama.

2.3.4 Tingkat Keamanan

Secara kriptografi, keamanan *HMAC* sangat bergantung pada ukuran kunci rahasia yang digunakan. Serangan paling umum yang dilakukan terhadap *HMAC* adalah serangan *brute force* untuk menemukan kunci rahasia.

2.3.5 Contoh Penggunaan

Salah satu contoh penggunaan *HMAC* adalah pada penjualan barang secara *online*. Setiap pelanggan mempunyai kunci rahasia yang tersimpan dalam sebuah basis data. Pihak penjual, ketika menerima pesanan dari pelanggan, dapat melakukan otentikasi pesanan dengan menghitung *HMAC* dengan kunci rahasia yang sama. Dengan demikian, keaslian pesanan dan otentikasi pemesan dapat dilakukan dengan mudah.

Selain cara di atas, algoritma *MAC* banyak dikembangkan dengan berbasis *universal hashing*.

2.4 Universal Hashing

2.4.1 Overview

Universal hashing merupakan sebuah algoritma acak yang digunakan untuk memilih fungsi *hash F* dengan properti sebagai berikut:
Untuk setiap input *x* dan *y* yang berbeda, probabilitas $F(x) = F(y)$, meskipun itu untuk fungsi *hash* yang berbeda.

Terdapat beberapa metode *hashing* yang universal yang menghasilkan fungsi *F* yang dapat dievaluasi menggunakan komputer melalui beberapa instruksi.

Karena sebuah fungsi *hash* bersifat *many-to-one*, maka harus ada elemen-elemen yang bersinggungan dalam fungsi *hash* tersebut, sesuai dengan prinsip *pigeon-hole*. Pada umumnya, kita ingin merancang fungsi *hash* sedemikian rupa sehingga tidak ada elemen-elemen yang bertabrakan. Secara matematis, mencari pembuktian untuk hal ini dapat dikatakan hampir tidak mungkin dapat dilakukan.

Algoritma acak ini menyediakan suatu cara untuk membuktikan hal tersebut. Kita dapat membangun sebuah kelas fungsi *hash* universal dengan ketentuan bahwa untuk setiap set masukan, nilai-nilai masukan tersebut akan tersebar dengan baik pada *range* yang telah ditentukan. Hal ini seperti memilih nilai-nilai *random* untuk semua masukan tersebut.

2.4.2 Definisi

Definisi formal mengenai *universal hashing* melibatkan satu set kunci *K*, nilai-nilai *V*, dan fungsi *hash H*, yang memetakan kunci ke nilai-nilai tersebut. Untuk setiap pasangan kunci *x* dan *y* pada *K*, *H* disebut **2-universal family** jika:

$$\Pr_{h \in H}[h(x) = h(y)] \leq \frac{1}{|V|}$$

Lebih ketat lagi, *H* disebut **strongly 2-universal family** jika:

$$\Pr_{h \in H}[h(x) = x' \, and \, h(y) = y'] = \frac{1}{|V|^2}$$

2.4.3 Contoh

Contoh kelas universal sederhana dari fungsi *hash* adalah semua fungsi *h* dengan bentuk:

$h(x) = f(g(x))$ dan
$g(x) = ((ax + b) \bmod p)$

dengan nilai *p* yang dijamin merupakan bilangan prima dengan nilai lebih besar dari semua nilai *input* dan setiap kombinasi *a* dan *b* membentuk fungsi yang berbeda di dalam kelas.

f menjadi fungsi yang memetakan elemen-elemen dengan rentang 0 hingga $p - 1$ ke rentang 0 hingga $n - 1$. Kemudian *f* akan menghitung nilai *g* mod *n*. Hanya ada satu fungsi *f* untuk semua fungsi dalam kelas ini. Untuk dapat melihat mengapa kelas ini disebut universal, perhatikan bahwa untuk setiap dua nilai masukan dan dua nilai keluaran, terdapat kira-kira p/n elemen yang dapat memetakan ke nilai keluaran apa pun, dan untuk setiap pasangan elemen sejumlah p/n tersebut, kita dapat menyelesaikan perhitungan dalam mod *p*. Sehingga untuk setiap pasangan nilai masukan, akan terdapat pasangan unik *a* dan *b* yang akan membawa ke elemen-elemen tersebut.

2.4.2 Penggunaan

Universal hashing banyak digunakan dalam bidang *computer science*, misalnya dalam kriptografi, juga dalam implementasi *hash tables*. Karena fungsi dipilih secara acak, maka sangat sulit untuk mengharapkan terjadinya banyak *hash collisions*.

Universal hashing telah digeneralisasi dalam banyak cara, sebagian besar disebut dengan ***k-wise independent hash functions***, yang mengharuskan fungsi tersebut berlaku seperti fungsi *random* terhadap setiap set masukan *k*.

2.5 Implementasi Sederhana Penggunaan MAC

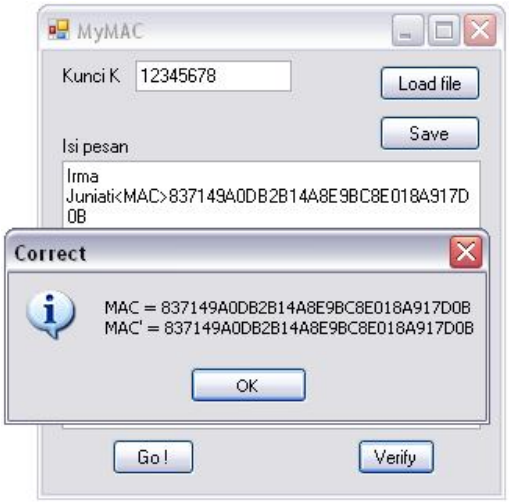
Berikut ini merupakan program kecil yang penulis implementasikan untuk mencoba melakukan simulasi proses pengiriman dan otentikasi oleh penerima pesan. Langkah-langkah yang dilakukan adalah sebagai berikut:

1. Lakukan penyambungan pesan *M* dengan *K*.
   Pada kasus yang diujikan, isi pesan adalah "Irma Juniati", dan nilai *K* = 12345678

2. Hitung nilai *hash* (*MAC*) dari hasil penyambungan.
   Fungsi *hash* yang digunakan adalah fungsi *hash* satu arah, yaitu *MD5*. *MAC* yang dihasilkan dituliskan di belakang *tag* <MAC>:

Gambar 3 Perhitungan *MAC*

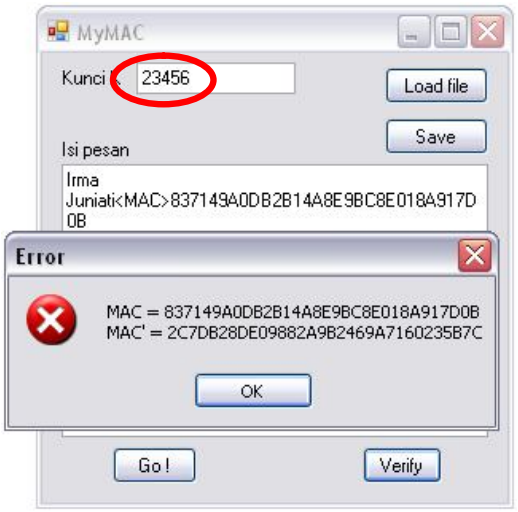
3. Otentikasi oleh penerima pesan
   Awalnya, penerima memisahkan isi pesan dengan *MAC* yang diterima, kemudian melakukan otentikasi dengan kunci *K* yang sama, yaitu 12345678. Jika *MAC* hasil perhitungan sama dengan *MAC* yang diterima, maka:



Gambar 4 Otentikasi Berhasil

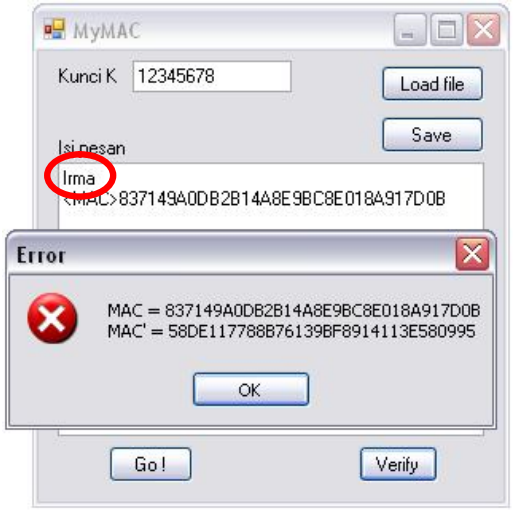
Berikut ini ditunjukkan dua contoh kasus yang dapat menyebabkan gagalnya otentikasi di sisi penerima pesan karena *MAC* hasil perhitungan berbeda dengan *MAC* yang diterima:

- Kunci yang digunakan tidak sesuai (misalnya: 23456).



Gambar 5 Otentikasi Gagal – Kunci Salah

- Terjadi perubahan terhadap isi pesan.



Gambar 6 Otentikasi Gagal - Isi Pesan Berubah

3. KESIMPULAN

Beberapa kesimpulan yang dapat diambil berdasarkan uraian dalam makalah ini adalah:

a. Meskipun cukup mirip dalam implementasinya, *MAC* bukanlah tanda tangan digital. *MAC* hanya menyediakan fungsionalitas untuk melakukan otentikasi pengirim dan integritas pesan.

b. Manajemen kunci pada *MAC* lebih mudah dibandingkan tanda tangan digital, karena *MAC* mengimplementasikan kriptografi kunci simetri.

c. *MAC* berbasis fungsi *hash* satu arah lebih digemari dibandingkan *MAC* berbasis *block cipher* karena komputasinya lebih cepat dan lebih mudah diimplementasikan.

d. Algoritma *MAC* yang memiliki kecepatan komputasi tinggi biasanya dibangkitkan dengan basis *universal hashing*.

DAFTAR PUSTAKA

[1] Munir, Rinaldi, (2006). Diktat Kuliah IF5054 Kriptografi, Departemen Teknik Informatika Institut Teknologi Bandung.

[2] http://www-cse.ucsd.edu/~mihir/papers/kmd5.pdf

[3] http://homes.esat.kuleuven.be/~preneel/preneel_mac_wcap06.pdf

[4] https://eprints.kfupm.edu.sa/3504/1/3504_1.pdf

[5] http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf