Studi dan Implementasi HMAC dengan Fungsi Hash Grøstl dan Perbandingannya dengan CMAC dengan Algoritma Cipher Blok AES

M. Albadr Lutan Nasution and 13508011[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*albadr.ln@itb.ac.id*

***Abstrak*— Algoritma Fungsi Hash Grøstl adalah salah satu finalist kompetisi algoritma fungsi SHA-3 yang diadakan oleh NIST USA. Fungsi hash ini banyak memakai berbagai komponen yang berasal dari algoritma kriptografi Rijndael. Makalah ini akan membahas secara singkat spesifikasi yang diajukan oleh tim Grøstl. Pada makalah ini juga diadakan implementasi algoritma fungsi hash Grøstl yang diaplikasikan sebagai *Message Authentication Code*. Akan dilakukan pula implementasi MAC dengan algoritma cipher blok AES pada mode CMAC. Dari kedua implementasi ini, akan dilakukan studi dan perbandingan kedua metode pembuatan MAC yang berasal dari algoritma Rijndael ini. Aspek perbandingan yang dilakukan adalah dari sisi fitur seperti perbedaan komponen dan teknik pengamanan yang digunakan serta hasil hash dan implementasi seperti kemudahan implementasi dan waktu pemrosesan.**

***Kata Kunci*—AES, CMAC, Grøstl Hash, HMAC, Rijndael**

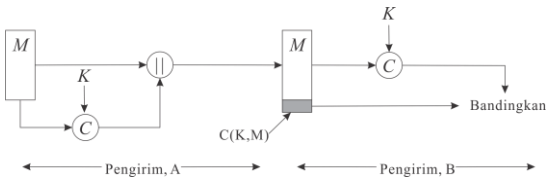
I. MESSAGE AUTHENTICATION CODE

Pada prinsipnya, Message Authentication Code (MAC) adalah informasi kecil berukuran tetap yang dapat digunakan untuk mengecek keutuhan dan kebenaran pesan dan mengecek keaslian pengirim pesan (*authentication* dan *integrity*). Sebuah algoritma pembangkit MAC, atau biasa juga disebut dengan algoritma fungsi hash berkunci menerima masukan berupa pesan sembarang ukuran M dan sebuah kunci rahasia K.

$$MAC = C(K, M) \quad [1]$$

Dua orang A dan B yang ingin berkomunikasi dengan melindungi pesannya melalui MAC harus memiliki kunci rahasia bersama K. Hal ini dapat dibedakan dari tanda tangan digital yang menerima kunci rahasia pembuat pesan saat pembangkitan dan menerima kunci publik pembuat pesan pada pengecekan.

Dalam skema pengecekan keaslian pesan, A, pengirim pesan, menghitung MAC dari pesan M tersebut dengan kunci K. Kemudian, pesan dan MAC dari pesan dikirim bersamaan kepada B, penerima pesan. B yang ingin memastikan keaslian pesan dapat mengecek keaslian pesan dengan menghitung MAC dari pesan yang diterima kemudian membandingkan dengan MAC yang ia terima. Skema ini dapat dilihat pada Gambar 1.



Gambar 1 Skema otentikasi pesan dengan MAC

Dalam implementasinya, terdapat beberapa metode pembangkitan MAC. Misalnya, MAC dapat dibangkitkan dengan menggunakan fungsi hash atau dengan algoritma enkripsi blok cipher.

Pembangkitan MAC dengan menggunakan fungsi hash cukup sederhana. Hal ini dapat dilakukan dengan menyambung kunci rahasia K dengan pesan asli M. Kemudian, dibangkitkan hash atau checksum CH dari hasil penyambungan pesan M dan kunci K sebelumnya. Pesan asli M dan checksum CH lah yang kemudian dikirim kepada penerima. [2]

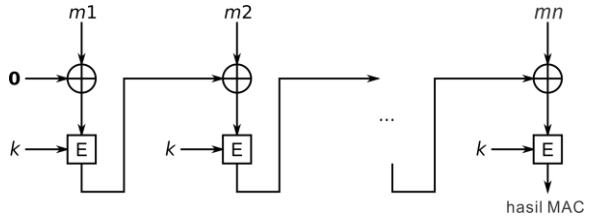
Pembangkitan MAC dengan menggunakan fungsi hash memiliki standar yang dinamakan HMAC. Pada standar ini, digunakan penambahan beberapa *padding* dan operasi XOR dalam pembangkitan MAC. Standar HMAC menggunakan operasi berikut :

$$MAC(M)_t = HMAC(K, M)_t$$
$$= H((K_0 \oplus opad) || H((K_0 \oplus ipad) || M))_t \quad [3]$$

Pada operasi di atas, dilakukan operasi XOR terhadap kunci K_0 dengan sebuah *padding opad* dan *ipad*. Kunci K_0 adalah hasil pemrosesan awal seperlunya dari kunci K. *Opad* atau *outer padding* adalah sebuah konstanta yang bernilai 0x5c diulang sebanyak ukuran blok. *Ipad* atau *innder apdding* adalah sebuah konstanta yang bernilai 0x36 diulang sebanyak ukuran blok. Ukuran blok yang digunakan adalah ukuran blok dari fungsi hash yang digunakan. Dengan demikian, MAC hasil algoritma

HMAC akan memiliki panjang yang sama dengan fungsi hash yang digunakan. Operasi || adalah operasi penyambungan dua array.

Pembangkitan MAC dengan menggunakan algoritma blok cipher dapat dilakukan misalnya dengan mengambil blok terakhir dari hasil enkripsi pesan M dengan cipher blok tersebut. Metode ini disebut dengan CBC-MAC (*Cipher Block Chaining Message Authentication Code*). Terdapat banyak metode lain untuk membangkitkan MAC dengan algoritma blok cipher dengan standar berbeda misalnya CMAC, OMAC, dan PMAC.



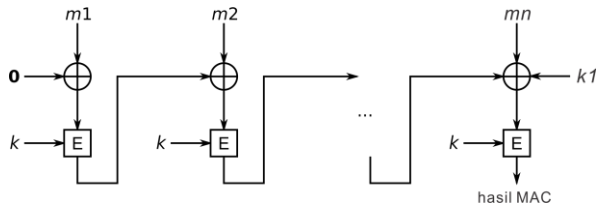
Gambar 2 Skema pembangkitaan MAC dengan CBC-MAC

Pada makalah ini diimplementasikan standar CMAC dengan algroritma blok cipher Rijndael atau biasa disebut AES. CMAC atau *Cipher-based Message Authentication Code* adalah hasil modifikasi dari CBC-MAC. CMAC membangkitkan dua upakunci K1 dan K2 untuk mengamankan enkripsi terakhir pada CBC-MAC.

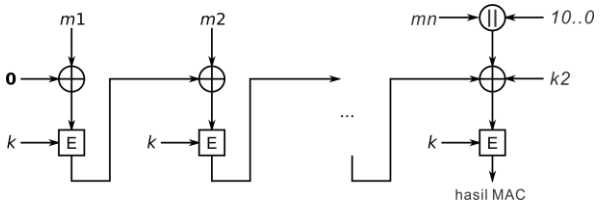
Dengan algrotima cipher blok C dengan panjang blok b bit, upakunci K1 dan K2 dibangkitkan dengan langkah-langkah berikut:

- Bangkitkan L dengan $L = C(K, 0^b)$. Nilai 0^b berarti bit string 0 sepanjang *b* bit.
- Jika MSB(L) = 0, $K1 = L << 1$,
- Jika tidak $K1 = (L << 1) \oplus R_b$, dengan R_b adalah konstanta bit string yang bergantung pada nilai b. Nilai R_b pada standar adalah $R_{128} = 0^{120}10000111$, dan $R_{64} = 0^{59}11011$.
- Kemudian, jika $MSB(K1) = 0, K2 = K1 << 1$
- Jika tidak $K2 = (K1<<1) \oplus R_b$.

Kunci K1 dan K2 digunakan dengan cara melakukan XOR dengan blok terakhir dari pesan M sehingga enkripsi pada blok terakhir adalah $XOR(C_{n-1},Mn,Kx)$ atau hasil XOR dari Mn (blok terakhir dari M), kunci K1 atau K2 dan hasil enkripsi blok sebelumnya. Pemakaian K1 dan K2 bergantung pada kasus kelengkapan Mn. Jika Mn adalah blok lengkap artinya sepanjang b bit, digunakan kunci K1. Sebaliknya, digunakan kunci K2. [4] Hal ini dapat terlihat dengan jelas pada skema berikut.



Gambar 3 Skema CMAC jika blok terakhir pesan adalah blok lengkap



Gambar 4 Skema CMAC jika blok terakhir pesan bukan blok lengkap

II. GRØSTL DAN RIJNDAEL

Grøstl adalah salah satu dari lima finalis kompetisi SHA-3 yang diadakan oleh NIST. Algoritma fungsi hash Grøstl seperti algoritma fungsi hash lainnya membagi pesan menjadi blok-blok dengan ukuran tetap tertentu. Setiap dari blok ini akan diproses dengan fungsi kompresi tertentu dan hasil dari setiap blok menjadi masukan bagi proses setelahnya. Akan tetapi tidak seperti MD5 atau SHA, Grøstl menyimpan *state* hash sementara (hasil proses kompresi sementara sampai blok tertentu) dengan ukuran minimal dua kali ukuran final hash yang diinginkan.

Fungsi kompresi *f* Grøstl memetakan blok pesan dan hash sementara sepanjang *l* ke keluaran sepanjang *l* juga. Panjang *l* untuk varian Grøstl yang menghasilkan panjang hash sampai 256 bit adalah 512 bit. Untuk varian yang lebih besar digunakan nilai *l* 512 bit. Hash sementara dan fungsi kompresi yang digunakan dalam Grøstl pada setiap blok didefinisikan sebagai :

$h_i = f(h_{i-1}, m_i)$ untuk i = 1, … , t

$f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h$ *[5]*

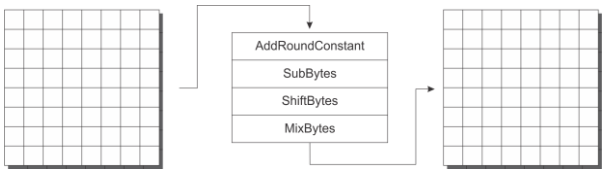
Fungsi P dan Q adalah fungsi permutasi. Kedua fungsi ini banyak didasari oleh fungsi-fungsi dasar pada algoritma kriptografi simetri Rijndael yang menjadi standar AES. Hanya saja Rijndael beroperasi pada 4x4 matriks byte sedangkan Grøstl beroperasi pada 8x8 atau 8x16 matriks byte.

Grøstl memiliki empat jenis permutasi yakni P_{512}, P_{1024}, Q_{512} dan Q_{1024}. Subscript 512 dan 1024 menggambarkan Proses permutasi P dan Q terdiri dari beberapa putaran. setiap putaran. Sama seperti AES, setiap putaran terdiri dari tahapan operasi :

1. AddRoundKey
   Pada tahap ini, dilakukan operasi XOR setiap byte pada state matriks dengan upakunci putaran yang telah dibangkitkan. Karena fungsi hash tidak memiliki kunci, pada Grøstl tahapan ini dinamakan AddRoundConstant dan digunakan konstanta kunci yang tetap. Akan tetapi, setiap jenis dari keempat permutasi memiliki konstanta kunci yang berbeda.
2. SubBytes
   Tahap SubBytes pada Grøstl sama seperti tahap SubBytes pada AES dan juga menggunakan Sbox yang sama. Setiap byte pada matriks *state*

disubstitusi dengan byte tertentu sesuai dengan Sbox yang didefinisikan pada Sbox Rijndael.

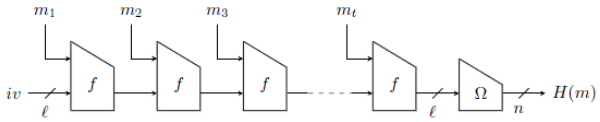
3. ShiftBytes
   Pada tahap ini, setiap baris pada matriks state per putaran digeser ke kiri sesuai dengan posisi baris. Pada AES, baris pertama dari matriks dibiarkan tetap. Baris kedua dari matriks digeser satu byte memutar ke kiri. Baris ketiga digeser dua byte dan baris keempat digeser tiga byte. Pada Grøstl, yang menggunakan matriks yang lebih besar proses ShiftBytes ini lebih dikembangkan lagi. Masing-masing dari empat permutasi yang didefinisikan memiliki proses ShiftBytes yang berbeda.
4. MixColoumns
   Pada tahap ini, masing-masing kolom dari matriks state ditransformasi dengan tranformasi linear tertentu. Tahap ini disebut MixBytes pada Grøstl.



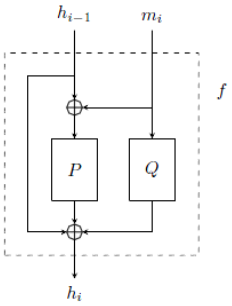
Gambar 5 Satu putaran Grøstl

Tidak seperti AES, setiap putaran pada Grøstl adalah identik dan Grøstl tidak memiliki tahapan AddRoundKey pada operasi terakhir. Jumlah putaran pada Grøstl merupakan parameter yang dapat dikonfigurasikan sendiri. Akan tetapi, jumlah putaran yang direkomendasikan pada setiap permutasi adalah 10 putaran untuk permutasi 512-bit dan 14 putaran untuk permutasi 1024-bit.

Hasil hash sementara yang terakhir memiliki ukuran dua kali lebih besar dari hasil hash yang diinginkan. Untuk memperoleh hash akhir, hasil ini kemudian ditranformasi dengan operasi $\Omega(h) = h \oplus P(h)$ yang kemudian dipotong sesuai dengan panjang hash yang diinginkan. [5]



Gambar 6 Skema fungsi hash Grøstl



Gambar 7 Fungsi kompresi dari Grøst

III. IMPLEMENTASI MAC

Dilakukan implementasi HMAC Grøstl sesuai dengan spesifikasi algoritma fungsi hash Grøstl yang ditulis pada [5] dan standar HMAC yang ditentukan pada [3]. Dilakukan pula implementasi CMAC dengan algoritma Advanced Encription Standar sesuai dengan spesifikasi algoritma AES pada standar yang ditentukan NIST pada [6]. Implementasi CMAC dilakukan sesuai dengan rekomendasi implementasi Cipher-based Message Authentication Code pada [4].

Kedua implementasi dilakukan pada lingkungan yang sama. Keduanya diimplementasikan pada bahasa Java. Pemilihan implementasi pada bahasa Java didasarkan pada kemudahan dalam menggunakan bahasa Java. Kedua implementasi (dan percobaan pada makalah ini) dilakukan pada mesin AMD Turion 64 x2 2,2 GHz, akan tetapi tidak dilakukan optimisasi pemograman sesuai mesin yang digunakan. Implementasi dilakukan sesuai dengan konsep abstrak dan algoritma semua yang tertulis pada masing-masing makalah yang membahasnya. Karena diimplementasikan pada lingkungan yang sama, diharapkan perbandingan yang dilakukan dapat mewakili perbandingan sebenarnya pada mesin lain dan jika dioptimasi sesuai dengan mesin.

Kedua algoritma pembangkitan MAC – HMAC Grøstl dan CMAC AES – memiliki banyak kesamaan dalam konsep dan rutin algoritmanya. Hal ini karena keduanya – Grøstl dan AES – didasarkan pada algoritma kunci simetri Rijndael. Keduanya pun memiliki struktur permutasi yang hampir sama dan *Substitution-Box* yang identik. Kesamaan konsep ini membuat perbandingan tingkat kesulitan implementasi keduanya tidak terlalu berbeda. Hanya saja Grøstl memiliki state yang lebih besar dibanding AES sehingga membutuhkan memori yang lebih besar. Grøstl juga memiliki dua tipe ukuran state sesuai dengan panjang *hash* yang ditentukan dalam parameter. Perbedaan ukuran state ini mungkin nantinya akan menyulitkan implementasi Grøstl pada *hardware*.

Ukuran state yang lebih besar pada algoritma fungsi hash Grøstl juga membuat sedikit-banyak perbedaan yang mencolok pada implementasi. Misalnya pada fungsi ShiftBytes yang memiliki beberapa variasi sesuai dengan tipe permutasi yang dijalankan. Hal ini berefek kepada pada jumlah prosedur yang harus diimplementasikan. Akan tetapi, tentu saja hal ini tidak terlalu berpengaruh pada implementasi *software*.

Ukuran state yang lebih besar pada Grøstl juga membuat fungsi MixBytes memiliki matriks pengali yang lebih besar dibanding pada matriks pengali pada fungsi MixColoums pada AES. Hal ini akan lebih memperlambat Grøstl dibanding AES karena jumlah perkalian matriks yang ada akan lebih banyak. Selain itu, karena perkalian dengan matriks tadi didasar pada *Rijndael Finite Galois Field*, implementasi kedua fungsi ini tidak mudah. Hal ini karena perkalian pada *Rijndael's Galois Field* adalah hal yang cukup kompleks. Implementasi perkalian pada

Rijndael's Galois Field pada AES dapat disederhanakan dengan cukup mudah karena bilangan yang digunakan relative kecil. Akan tetapi, karena bilangan yang digunakan pada Grøstl jauh lebih besar, penyederhanaan perkalian tidaklah semudah AES. Kekompleksan ini mungkin nantinya dapat diatasi dengan pembuatan tabel *lookup* yang dihitung sejak awal untuk perkalian sehingga implementasi lebih mudah dan performansi lebih cepat.

Permutasi satu blok milik Grøstl dan AES memiliki rutin yang cukup mirip. Akan tetapi, Grøstl menggunakan dua fungsi permutasi pada permutasi satu blok pesan. Akibatnya jumlah permutasi yang dipakai Grøstl dua kali lebih besar dari jumlah permutasi AES. Pada pembangkitan MAC, standar HMAC memanggil fungsi hash yang digunakan dua kali. Hal ini tentu akan membuat jumlah permutasi pada HMAC Grøstl lebih banyak lagi dengan kata lain kompleksitas yang lebih tinggi. Hal ini mungkin menyebabkan waktu proses pembangkitan MAC pada Grøstl minal empat kali lebih lama dibanding waktu proses pembangkitan MAC pada CMAC AES. Perbandingan waktu proses dapat diamati pada bagian V.

IV. Eksperimen Pembangkitan MAC

Pada bagian ini akan dipaparkan hasil MAC dari masing-masing implementasi. Akan dipaparkan pula hasil MAC dari perubahan satu bit pesan semula. Hal ini untuk melihat variasi perubahan hasil MAC terhadap manipulasi pesan, karena fungsi penghasil hash atau MAC yang bagus adalah yang tidak memiliki sifat kontinuitas. Artinya MAC atau hash yang dihasilkan antara dua pesan yang bersebelahan tidak berdekatan pula.

Pesan pertama yang dicoba dibangkitkan MAC-nya adalah pesan kosong. Seluruh pesan dalam percobaan berikut menggunakan kata sandi "Institut Teknologi Bandung". Berikut adalah hasil MAC pesan kosong dari masing-masing implementasi.

Tabel 1 MAC dari pesan kosong

Pesan	<kosong>
CMAC AES 128 bit	B4 3A 03 1D B4 4C BE 22 16 39 51 7F C5 AB 1A 87
HMAC Grøstl 224 bit	8D D2 AF D9 8E BA 4A 47 5B 96 9F AF 1F 66 5D 77 E8 99 83 FD B0 A5 C5 07 22 58 3D 5D
HMAC Grøstl 256 bit	02 AF 94 B5 70 83 25 DA 8E 6B 68 D8 93 D0 69 28 CF 61 36 4A 2E F8 18 CC 07 5C 01 CF A9 86 FF EB
HMAC Grøstl 384 bit	59 A0 5E E3 5E 8C 94 15 CA 60 5A 9F 39 F0 AD 74 39 11 27 CD 99 AD 82 6B 17 2E C2 4E 77 44 F3 18 28 CC E8 66 BA 43 DD 60 43 B0 60 D7 97 D3 69 33
HMAC Grøstl 512 bit	C9 44 3C 60 22 22 C1 49 D8 A6 5B D0 09 BA A2 7C 76 7F 28 C2 16 7C BC 85 34 0D 1E 77 04 08 E7 12 6F 5F 56 41 54 33 8E B4 46 C8 2C 46 1D 1B 0F 47 F2 A6 59 57 3B A1 C7 8F F8 96 2D 65 49 E9 FA 9C

Berikut adalah hasil MAC dari pesan kedua yang dicoba, yakni pesan sepanjang satu byte, berisi hanya karakter null atau karakter 0x00.

Tabel 2 MAC dari pesan berukuran 1 byte, yakni 0x00

Pesan	00
CMAC AES 128 bit	54 94 0B 2B A2 0B 4D 48 A8 A7 33 F3 7E D6 34 FC
HMAC Grøstl 224 bit	E3 10 21 3E 5C BA 86 2D D4 C2 C6 8B 6A 50 CD 00 33 D9 CE A0 95 9C 2C 0C 13 1C 9B FD
HMAC Grøstl 256 bit	1C 1B FA E4 8C 2B 08 8B CA 93 9B F7 25 1E 55 55 62 61 1C E7 5D 4B A7 B5 34 76 DE 4F 7E 26 6B 76
HMAC Grøstl 384 bit	C2 0D 6B 28 45 8F 87 78 7D 48 66 1A 5B 85 90 77 1C 2B 61 20 2D 43 46 4A 52 B2 71 8D 8A 47 CD B2 A1 47 F0 C9 79 11 D4 9F 6A 1C 88 E7 BB 8F B2 80
HMAC Grøstl 512 bit	6A C0 B9 00 0C 3B E4 80 A6 DC AA EE F5 05 18 A4 04 1D F2 A8 2C F7 FD 58 2E 91 CD 73 C7 9D 2F 9A D1 BB 19 08 0B BC BA 20 37 D1 44 8E A7 9B 34 62 37 ED D3 E8 0D 06 7C 53 8F F9 B8 F7 A6 42 7F 75

Kemudian dicoba pembangkitan pesan yang hanya berbeda 1 bit dari pesan sebelumnya yakni pesan berukuran 1 byte yang berisi karakte 0x01.

Tabel 3 MAC dari pesan berukuran 1 byte, yakni 0x01

Pesan	01
CMAC AES 128 bit	E7 A3 02 DC F1 0C 18 94 10 43 F3 D8 DD 01 20 3F
HMAC Grøstl 224 bit	9A 5A 6A E9 69 84 80 6C E5 91 D4 84 B4 C9 C4 53 8F 44 B0 6F 47 CA 89 C4 34 EA 98 50
HMAC Grøstl 256 bit	B9 EF 24 A7 F8 07 FD 8A 5E E3 22 2C 74 F7 3A C2 2A E8 98 5B 9D 74 F7 85 6F AA CB 94 3F 7A 27 F9
HMAC Grøstl 384 bit	7D 7D 24 05 8D A3 24 4E 4A F5 65 4F 9C 05 92 5C 6D 63 7E 2B 70 C6 73 9F 45 4D 80 95 80 13 DC CB 24 6A F5 B7 E3 9B 57 A3 71 C9 54 1F A1 29 58 0D
HMAC Grøstl 512 bit	AB A4 D4 A0 01 2C F2 F1 1A F2 C3 4C 55 AF 7C 4F 49 5D BA 05 0B C4 43 4C 29 AD 34 21 0A 4C AA 10 93 34 30 7F A6 5B FA 52 71 EA 1C C4 EE B3 3B 44 D4 2F B0 D2 C5 23 0A 34 E6 4D 9D BC E2 65 DA C6

Dari ketiga tabel dapat terlihat bahwa perubahan MAC dari pesan kosong ke pesan berukuran 1 byte dan pesan 00 ke pesan 01 sangatlah jauh.

V. Perbandingan Performansi

Pada bagian ini akan dipaparkan hasil perbandingan performansi antara kedua implementasi. Perbandingan yang dilakukan adalah waktu proses pembangkitan MAC pada pembangkitan CMAC AES 128 bit, HMAC Grøstl 224 bit, HMAC Grøstl 256 bit, HMAC Grøstl 284 bit, dan HMAC Grøstl 512 bit.

A. Pengaruh Ukuran Pesan terhadap Waktu Proses Pembangkitan MAC

Pada subbagian ini akan dipaparkan hasil eksperimen pembangkitan MAC dengan ukuran pesan yang berbeda-

beda. Hal ini dilakukan untuk melihat perubahan waktu proses pembangkitan MAC pada masing-masing implementasi.

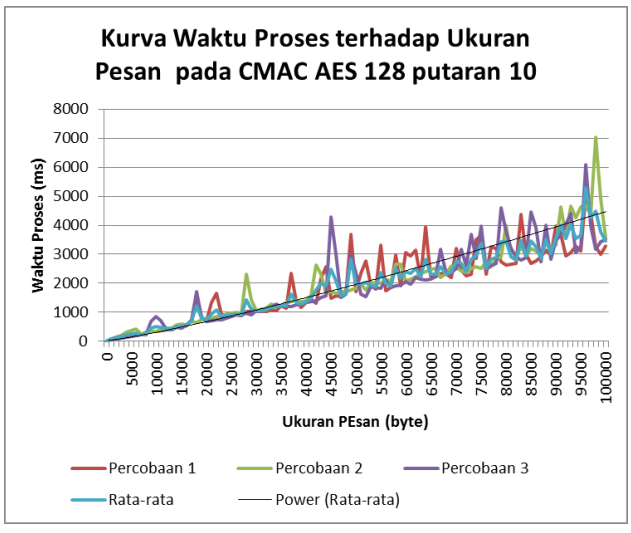
Perbandingan dilakukan dengan memakai parameter standar yang disarankan oleh pengembang, yakni menggunakan kunci 128 bit dan jumlah putaran 10 putaran pada CMAC AES 128 bit, menggunakan jumlah putaran 10 putaran pada HMAC Grøstl 224 bit dan 256 bit, serta menggunakan jumlah putaran 10 putaran pada HMAC Grøstl 224 bit dan 256 bit.

Tabel 4 Perbandingan Waktu Proses antar Implementasi

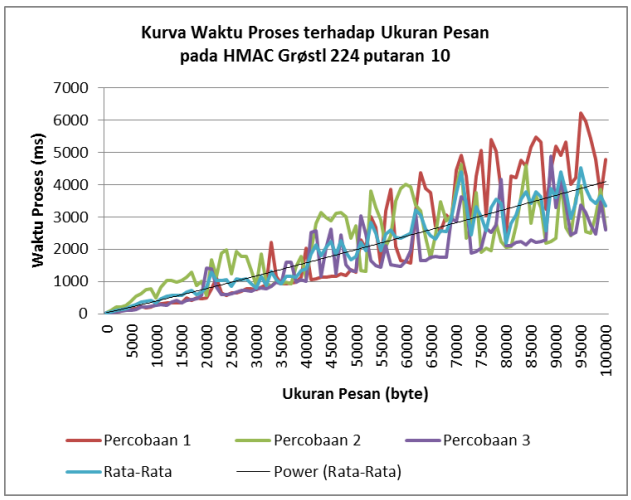
Ukuran Pesan (kB)	Waktu Pembangkitan MAC (ms)				
	CMAC AES 128bit	HMAC Grøstl 224 bit	HMAC Grøstl 256 bit	HMAC Grøstl 384 bit	HMAC Grøstl 512 bit
0	1	21	21	21	27
10	509	343	328	328	344
100	3.451	3.333	3.996	3.136	3.427
1000	12.817	61.245	34.164	40.903	57.517
10000	64.860	184.504	281.268	167.934	232.456

Pada tabel di atas tampak bahwa performansi HMAC Grøstl dan CMAC AES pada ukuran pesan kecil cukup seimbang. Akan tetapi, untuk ukuran pesan lebih besar di atas 1000 kByte, implementasi CMAC AES jauh lebih cepat dibanding HMAC Grøstl. Rasio perbandingan antara waktu proses HMAC Grøstl dan CMAC Grøstl dari tabel di atas berada pada 3 sampai 4. Hal ini sesuai dugaan yang berdasarkan pada jumlah permutasi yang diterapkan pada masing-masing algoritma seperti yang dijelaskan pada bagian III.

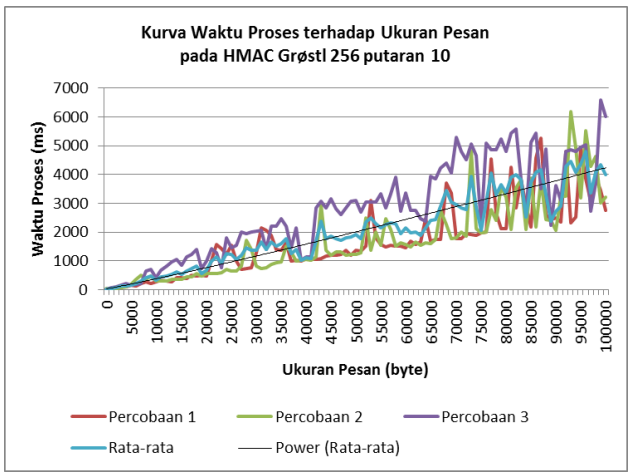
Lebih jelasnya lagi, dialukan tiga kali percobaan pada masing-masing pembangkitan MAC untuk melihat rata-rata laju peningkatan waktu proses berdasarkan ukuran pesan. Laju peningkatan waktu proses dari masing-masing algoritma dapat diamati pada grafik-grafik berikut.



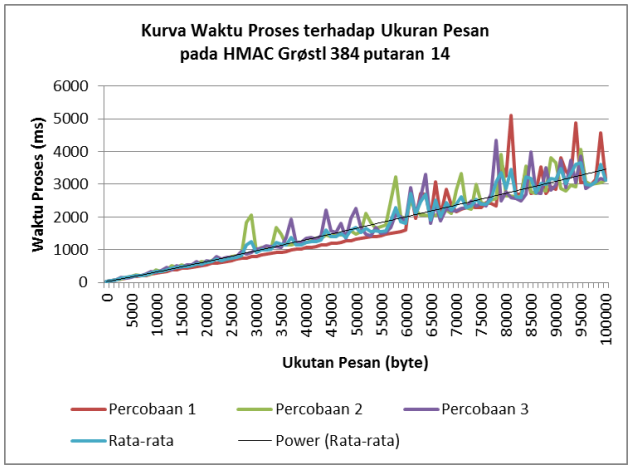
Gambar 8 Kurva Waktu Proses terhadap Ukuran Pesan pada CMAC AES 128-bit 10 putaran



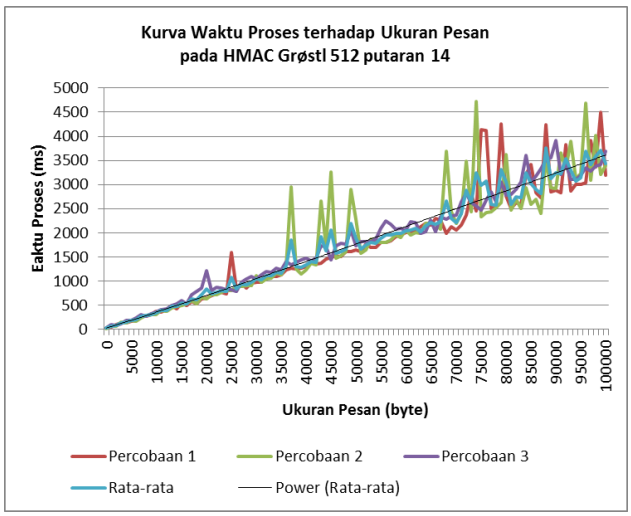
Gambar 9 Kurva Waktu Proses terhadap Ukuran Pesan pada HMAC Grøstl 224-bit, 10 putaran



Gambar 10 Kurva Waktu Proses terhadap Ukuran Pesan pada HMAC Grøstl 256, 10 putaran

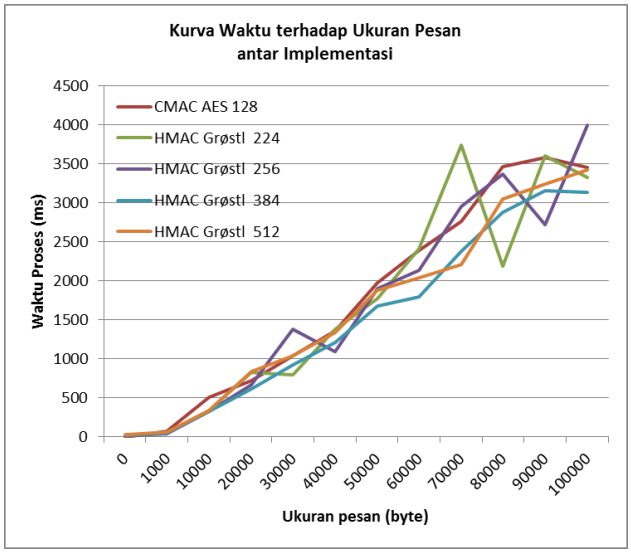


Gambar 11 Kurva Waktu Proses terhadap Ukuran Pesan pada HMAC Grøstl 284, 14 putaran



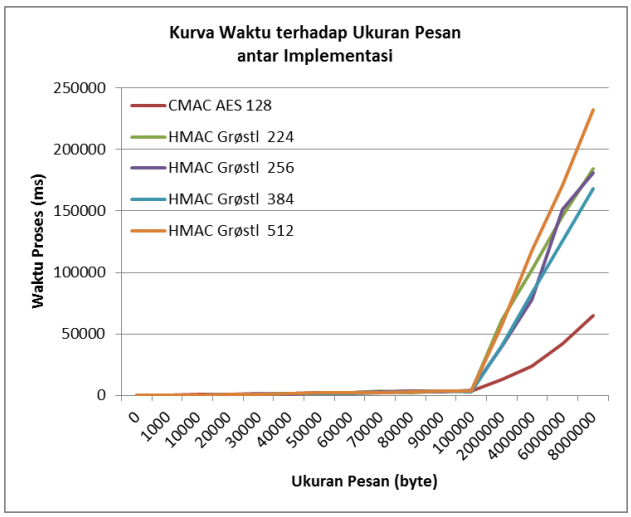
Gambar 12 Kurva Waktu Proses terhadap Ukuran Pesan pada HMAC Grøstl 512, 14 putaran

Dari grafik di atas terlihat bahwa laju peningkatan waktu proses terhadap peningkatan ukuran pesan hingga ukuran 100 kilobyte cukuplah linear. Kelima pembangkit MAC cukup seimbang dalam pesan berukuran kecil. Hal ini dapat dengan jelas diamati pada grafik berikut.



Gambar 13 Kurva perbandingan laju waktu proses antar implementasi sampai 100 kilobyte

Tampak pada gambar di atas bahwa kelima pembangkit pesan MAC dari dua implementasi yang dilakukan cukup seimbang satu sama lain. Akan tetapi, hal ini hanya berlaku hingga ukuran pesan sekitar 1000 kilobyte. Untuk pesan berukuran lebih besar, mulai tampak perbedaan yang mencolok dari masing-masing pembangkit MAC. Sesuai dengan Tabel 4, grafik berikut memperlihatkan perbandingan laju peningkatan waktu proses pembangkitan MAC pada masing-masing pembangkit pesan dari kedua implementasi yang dilakukan.



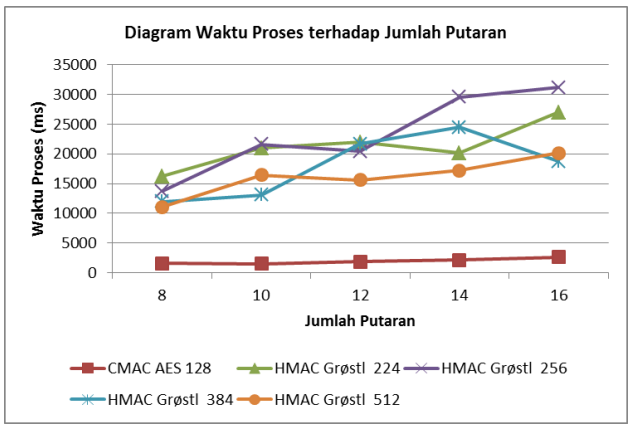
Gambar 14 Kurva perbandingan laju waktu proses antar implementasi sampai 8000 kilobyte

Dari grafik pada Gambar 14 tampak bahwa implementasi CMAC dengan AES jauh lebih unggul pada pesan berukuran besar. Waktu yang dibutuhkan untuk membangkitkan pesan sampai empat kali lebih cepat dari waktu pembangkitan pesan oleh HMAC Grøstl.

B. Pengaruh Jumlah Putaran terhadap Waktu Proses Pembangkitan MAC

Kedua implementasi MAC yang dilakukan memungkinkan pengguna untuk mengubah parameter putaran sesuai dengan kebutuhan keamanan. Pada bagian ini akan diperlihatkan perbandingan waktu yang dibutuhkan untuk membangkitkan MAC pada beberapa putaran.

Perbandingan dilakukan dengan ukuran pesan sebesar 500 kilobyte. Grafik berikut memperlihatkan hasil perbandingan yang dilakukan.



Gambar 15 Kurva perbandingan waktu proses terhadap jumlah putaran

Dari grafik di atas tampak bahwa jumlah putaran memengaruhi waktu proses pembangkitan MAC yang ada. Tampak terlihat dengan jelas pula bahwa peningkatan waktu proses pada HMAC Grøstl cukup linear dan CMAC AES memiliki perubahan waktu terhadap putaran

yang cukup stabil.

C. Perbandingan Waktu Proses Antara CMAC AES-128, HMAC Grøstl-512, dan MAC Grøst-512

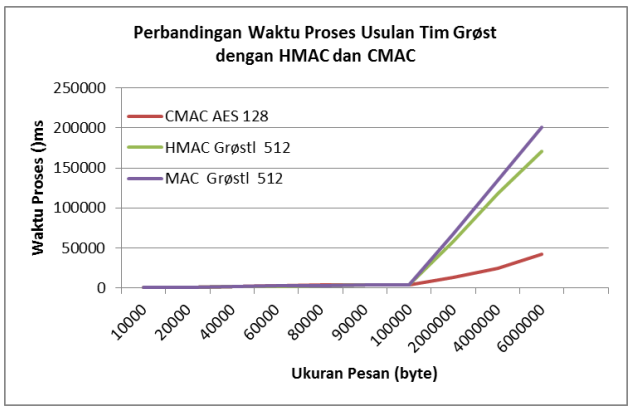
Dalam papernya, tim Grøst juga mengusulkan algoritma MAC lain sebagai pengganti HMAC. Hal ini diusulkan sebagai perbaikan dari HMAC karena penggunaan HMAC dengan Grøst membutuhkan pemanggilan algoritma Grøst dua kali.

Tim Grøst mengusulkan untuk menggunakan konstruksi amplop seperti berikut :

$MAC(K,M) = H(K_b||M_b||K)$,

dimana K_b adalah kunci yang telah di-*padding* sebanyak panjang blok yang digunakan dan M_b adalah pesan yang telah di-*padding* sehingga memiliki panjang kelipatan panjang blok yang digunakan. Tim Grøst mengklaim bahwa konstruksi ini memiliki keamanan yang sama dengan HMAC dan lebih efisien dibanding HMAC. [5]

Pada bagian ini akan dipaparkan hasil perbandingan implementasi konstruksi ini dengan HMAC Grøst-512 dan AES-128. Grafik berikut menggambarkan laju perubahan waktu proses pada ketiga implementasi yang dilakukan.



Gambar 16 Kurva perbandingan waktu proses terhadap ukuran byte dari HMAC, CMAC, dan MAC usulan tim Grøst.

Tampak pada gambar bahwa ternyata konstruksi amplop pembangkit MAC usulan tim Grøst membutuhkan waktu proses pembangkitan MAC yang lebih lama.

Kontradiksi dengan klaim bahwa lebih efisien. Menurut penulis, hal ini disebabkan karena implementasi konstruksi amplop yang dilakukan tidak efisien. Lebih tepatnya, pada konstruksi amplop terjadi dua kali penyambungan array. Kedua penyambungan array ini melibatkan pesan. Jika implementasi penyambungan array adalah dengan membentuk array baru dan menyalin dari kedua array yang akan disambungkan, dibutuhkan waktu yang lama. Penyebab utama lamanya proses adalah jumlah byte pada pesan yang sangat besar sehingga proses penyalinan array dibutuhkan kerja keras. Standar HMAC hanya melakukan satu kali penyambungan aray dengan pesan awal sehingga lebih cepat dari konstruksi amplop di atas.

Dengan demikian, untuk membuat konstruksi ini (dan juga HMAC dan konstruksi lain) dibutuhkan implementasi yang lebih efisien, terutama pada hal sederhana seperti penyambungan dua array.

VI. SIMPULAN

Dari hasil implementasi dan perbandingan terlihat bahwa pembangkitan MAC dengan CMAC AES jauh lebih efisien dibanding dengan pembangkitan MAC dengan HMAC Grøstl. Akan tetapi, karena CMAC AES hanya menghasilkan 128 bit MAC, CMAC AES ini hanya dapat menjadi alternatif penggunaan MAC jika lebih dibutuhkan kecepatan dibanding keamanan pada pesan.

Grøstl meskipun masih dalam pengembangan merupakan algoritma fungsi hash yang cukup baik. Apalagi banyak komponennya yang terinspirasi dari algoritma kriptografi simetri Rijndael yang telah teruji dan telah dipelajari dengan baik selama lebih dari sepuluh tahun.

REFERENSI

[1] Jueneman, R. R, Matyas, S. M., and Meyer, C. H. "*Message Authentication.*" IEEE Communication, Vol 23, No. 9, 1985, pp.22-40

[2] Munir, Ir. Rinaldi. "*Diktat kuliah Kriptografi*." Institut Teknologi Bandung. Bandung, 2006.

[3] FIPS-198a. "*The Keyed-Hash Message Authentication Code (HMAC)*". Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8900

[4] Dworkin, Morris. "*Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication."* National Institute of Standards and Technology, U.S. Department of Commerce, 2005.

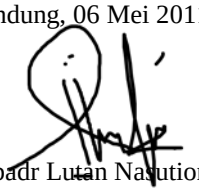
[5] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schlaffer, and S. S.Thomsen. "*Grøstl – SHA-3 candidate."* Submission to NIST, 2011.

[6] FIPS-197. "*Specification for the Advanced Encryption Standard (AES)."* Information Technology Laboratory, National Institute of Standards and Technology. November 26, 2001.

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, 06 Mei 2011



M Albadr Lutan Nasution
13508011