Analisis Algoritma dan Kinerja pada *Counter* dengan CBC-MAC *(CCM)* sebagai Fungsi Enkripsi Terotentikasi

Shelvie Nidya Neyman, Sugi Guritman, Ratna Purnama Sari

Departemen Ilmu Komputer,FMIPA IPB

Abstrak

Algoritma CCM merupakan suatu modus enkripsi terotentikasi yang mengkombinasikan modus enkripsi Counter dengan modus otentikasi CBC-MAC. CCM didasarkan pada algoritma blok kunci simetrik dengan panjang blok 128 bit, seperti algoritma AES. CCM melindungi tiga elemen data, yaitu: nonce, payload, dan associated data.

Berdasar analisis, CCM didesain untuk mendapatkan kerahasiaan dan otentikasi pesan secara simultan menggunakan sebuah kunci rahasia. Kebutuhan akan hanya fungsi forward cipher dari algoritma blok yang mendasari CCM menjadikan ukuran kode implementasi CCM yang lebih kecil. Meskipun demikian, beberapa timbal balik kinerja pada proses-proses dalam CCM harus dipikirkan terlebih dahulu sebelum mengaplikasikan algoritma ini. Semakin besar nonce, maka akan semakin kecil panjang maksimum dari payload yang bisa diproteksi CCM. Semakin kecil nonce, maka akan semakin kecil jumlah maksimum nonce yang berbeda. Semakin besar nilai MAC akan memberikan jaminan otentisitas yang semakin besar pula. Namun, dengan semakin besar nilai MAC maka semakin besar pula ruang penyimpanan yang harus disediakan bagi ciphertext. Dengan analisis algoritma, didapatkan CCM memiliki kompleksitas pada lingkup O(n) baik bagi proses generation-encryption maupun decryption-verification.

Melalui analisis uji implementasi menggunakan Matlab 6.5 dan analisis uji statistik (Independent-Samples T-Test), dapat disimpulkan bahwa running time proses CCM generation-encryption tidak berbeda nyata dengan running time proses CCM decryption-verification dengan selang kepercayaan 95%. Melalui analisis regresi, dapat disimpulkan bahwa untuk payload berukuran besar dan tanpa associated data, nilai running time proses generation-encryption dan decryption-verification pada AES-CCM adalah 1.8 sampai dengan 2.0 kali running time proses enkripsi pada AES-ECB.

Kata kunci: *CBC-MAC, cryptography, modus enkripsi terotentikasi, authenticated encryption mode, generation-encryption, decryption-verification.*

PENDAHULUAN

Seringkali ketika dua pihak berkomunikasi melalui suatu jaringan, ada dua tujuan utama yang ingin mereka capai dalam menjaga keamanan data atau pesan, yaitu kerahasiaan dan otentikasi pesan. Teknik kriptografi dapat digunakan untuk mencapai tujuan tersebut. Tujuan kerahasiaan dapat dicapai dengan menggunakan teknik kriptografi berupa penyandian pesan, sedangkan tujuan otentikasi dapat diraih dengan menggunakan teknik kriptografi berupa otentikasi pesan yang secara implisit memberikan integritas data.

Sehubungan dengan pencapaian tujuan keamanan data, para ahli kriptogafi berpendapat bahwa seseorang sebaiknya tidak melakukan penyandian pesan tanpa diikuti otentikasi pesan. Baru-baru ini ada sejumlah konstruksi baru yang meraih kerahasiaan dan otentikasi secara simultan dan seringkali lebih cepat daripada sembarang solusi yang menggunakan *generic composition (Black 2003)*. Konstruksi baru ini memanfaatkan modus-modus operasi sandi blok sehingga dikenal dengan istilah "*combined modes*" atau istilah yang lebih sering digunakan adalah "*authenticated-encryption modes*".

Penelitian ini akan menganalisis salah satu modus enkripsi terotentikasi yaitu algoritma *Counter with Cipher Block Chaining-Message Authentication Code*, disingkat *CCM*. *CCM* dapat dianggap sebagai suatu modus operasi algoritma blok. Seperti ditunjukkan oleh namanya, *CCM* mengkombinasikan modus enkripsi *Counter (CTR)* dengan modus otentikasi *Cipher Block Chaining-Message Authentication Code (CBC-MAC)*. *CCM* didasarkan pada algoritma sandi blok kunci simetrik dengan panjang blok 128 bit, seperti algoritma *AES*. Pada penelitian ini dilakukan analisis algoritma, serta implementasi algoritma *CCM* bagi analisis kinerja dengan uji *running time* proses *generation-encryption* dan *decryption-verification*.

Deskripsi Algoritma CCM

Algoritma *Counter with Cipher Block Chaining-Message Authentication Code (CCM)* diajukan oleh *Doug Whiting, Russ Housley,* dan *Niels Ferguson* pada tahun 2002. *CCM* didasarkan pada algoritma blok kunci simetrik yang telah diakui dengan ukuran blok 128 bit, seperti algoritma *AES* yang dipublikasikan dalam *FIPS* 197. Berikut ini adalah uraian lengkap mengenai algoritma *CCM*.

1. Algoritma Blok untuk CCM

Algoritma blok yang akan mendasari *CCM* pada penelitian ini adalah algoritma *AES*. *CCM* hanya memerlukan fungsi *forward cipher* dari *AES*. Kunci *CCM*, disimbolkan dengan *K*, adalah kunci untuk algoritma *AES*. Fungsi *forward cipher* dengan kunci ini

dinyatakan dengan $CIPH_K$, sedangkan panjang K dinyatakan dengan *Klen*.

2. *Elemen-elemen Data*

Data yang dilindungi *CCM* terdiri dari tiga elemen, yaitu:

1. Sebuah pesan atau data, yaitu suatu *bitstring* disebut *payload* (P), dengan panjang bit *Plen*. P akan diotentikasi dan dienkripsi sehingga *CCM* menyediakan jaminan keaslian dan kerahasiaan P.
2. Suatu bit *string* disebut *associated data*, (A). A bersifat opsional. A akan diotentikasi tetapi tidak dienkripsi sehingga *CCM* hanya menyediakan jaminan keaslian namun tidak memberikan jaminan kerahasiaan A.
3. Suatu bit *string* unik yang disebut *nonce* (N), diberikan pada pasangan data yang akan dilindungi, yaitu P dan A.

MAC yang dibangkitkan dalam *CCM* dinyatakan dengan T. Panjang bit T, dinyatakan dengan *Tlen*, merupakan suatu parameter yang akan ditetapkan untuk semua proses *CCM* dengan kunci yang diberikan *(Dworkin 2004)*.

3. *Pemformatan Input*

Elemen-elemen data *CCM* yaitu N, P, dan A akan diformat dengan sebuah fungsi pemformatan *(dalam Jonsson (2002) disebut dengan encoding function)* menjadi rangkaian tak-kosong dari blok data lengkap, dinyatakan dengan $B_0, B_1, \ldots, B_r$ untuk r bilangan bulat tak-negatif. Nilai r tergantung pada fungsi pemformatan dan elemen input. Tiga sifat berikut harus dimiliki oleh fungsi pemformatan *(Dworkin 2004)*:

1. Blok pertama, B_0, secara unik menentukan *nonce* N.
2. Data yang diformat secara unik menentukan P dan A; selain itu, jika (N, P, A) dan (N, P', A') adalah input triplet yang berbeda dengan format $B_0, B_1, \ldots, B_r$ dan $B_0', B_1', \ldots, B_{r'}'$, maka B_i berbeda dari B_i' untuk indeks i sedemikian sehingga $i \leq r$ dan $i \leq r'$.
3. Blok pertama, B_0, berbeda dari sembarang blok *counter* yang digunakan untuk semua proses *CCM* menggunakan kunci yang diberikan.

4. *Fungsi Pemformatan*

Syarat Panjang Peubah dalam CCM

Panjang bit untuk setiap *string* input, yaitu N, A, dan P, merupakan kelipatan dari 8 bit, yaitu setiap *string* input merupakan *string* oktet. Panjang oktet dari *string* ini dinyatakan dengan n, a, dan p. Demikian pula dengan parameter t yang menyatakan panjang oktet T.

Panjang oktet dari P *(yaitu bilangan bulat p)* direpresentasikan dalam blok pertama dari data terformat sebagai sebuah *string* oktet disimbolkan dengan Q. Panjang oktet Q yang dinyatakan dengan q, adalah sebuah parameter untuk fungsi pemformatan. Jadi, Q ekuivalen dengan $[p]_{8q}$, representasi biner p dalam oktet-oktet q.

Fungsi pemformatan yang digunakan dalam penelitian ini mensyaratkan panjang dari peubah berikut *(Dworkin 2004)*:

- t adalah elemen dari {4, 6, 8, 10, 12, 14, 16}
- q adalah elemen dari {2, 3, 4, 5, 6, 7, 8}
- n adalah elemen dari {7, 8, 9, 10, 11, 12, 13}
- $n + q = 15$
- $a < 2^{64}$.

Pemformatan Kontrol Informasi dan Nonce

Oktet awal blok pertama dari pemformatan, B_0, berisi empat *flag* (**Tabel 1**) untuk mengontrol informasi: yaitu dua bit tunggal (disebut *Reserved* dan *Adata*) dan dua *string* terdiri dari tiga bit yang digunakan untuk menyandikan nilai t dan q. Penyandian t adalah $[(t\text{-}2)/2]_3$, dan penyandian q adalah $[q\text{-}1]_3$. Bit *Reserved* dicadangkan untuk memungkinkan perluasan pemformatan di masa mendatang dan akan di-set dengan nilai '0'. Bit *Adata* bernilai '0' jika $a = 0$ dan '1' jika $a > 0$ *(Dworkin 2004)*. **Tabel 2** memperlihatkan pemformatan B_0.

Tabel 1 *Pemformatan oktet flag dalam* B_0

Bit	7	6	5	4	3	2	1	0
Isi	*Reserved*	*Adata*	$[(t\text{-}2)/2]_3$			$[q\text{-}1]_3$		

Tabel 2 *Pemformatan* B_0

Oktet	0	1 ... 15-q	16-q ... 15
Isi	*Flags*	N	Q

Pemformatan Associated Data

Jika $a = 0$ maka tidak ada blok yang disediakan untuk *associated data* dalam data yang diformat. Jika $a > 0$, maka a disandikan seperti dijelaskan di bawah ini, dan sandi dari a digabung dengan *associated data* A, diikuti oleh jumlah minimum bit-bit '0' *(mungkin juga tidak)* sedemikian sehingga *string* yang dihasilkan bisa dipartisi ke dalam 16-blok oktet. Blok-blok ini dinyatakan dalam data terformat sebagai $B_1, B_2, \ldots B_u$ untuk u bilangan bulat positif yang tergantung dari nilai a.

Nilai a disandikan berdasarkan tiga kasus berikut *(Dworkin 2004)*:

- Jika $0 < a < 2^{16}\text{-}2^8$, maka a disandikan sebagai $[a]_{16}$, yaitu dua oktet.
- Jika $2^{16}\text{-}2^8 \leq a < 2^{32}$, maka a disandikan sebagai 0xff || 0xfe || $[a]_{32}$, yaitu enam oktet.
- Jika $2^{32} \leq a < 2^{64}$, maka a disandikan sebagai 0xff || 0xff || $[a]_{64}$, yaitu sepuluh oktet.

Pemformatan Payload

Payload digabungkan dengan jumlah minimum bit-bit '0' *(mungkin juga tidak ada)* sedemikian sehingga hasilnya bisa dipartisi menjadi 16-blok oktet. Blok-blok ini dinyatakan dalam data yang telah diformat sebagai $B_{u+1}, B_{u+2}, \ldots B_r$, dengan $r = u + \lceil p/16 \rceil$ *(Dworkin 2004).*

Fungsi Pembangkitan Counter

Blok-blok *counter* Ctr_i dibentuk seperti terlihat pada **Tabel 3.** Dalam setiap blok Ctr_i, *flag* dibentuk seperti terlihat pada **Tabel 4**. Bit *Reserved* disediakan untuk perluasan di masa mendatang dan akan diset dengan nilai '0'. Bit 3, 4, dan 5 juga akan diset dengan nilai '0' untuk memastikan bahwa semua blok *counter* berbeda dari B_0. Bit 0, 1, dan 2 berisi penyandian q yang sama seperti dalam B_0 *(Dworkin 2004).*

Tabel 3 *Pemformatan* Ctr_i

Oktet	0	1 ... 15-q	16-q ... 15
Isi	*Flags*	N	$[i]_{8q}$

Tabel 4 *Pemformatan oktet flag dalam* Ctr_i

Bit	7	6	5	4	3	2	1	0
Isi	*Reserved*	*Reserved*	0	0	0	$[q\text{-}1]_3$		

Proses Generation-Encryption

Berikut ini adalah langkah-langkah proses *generation-encryption CCM:*

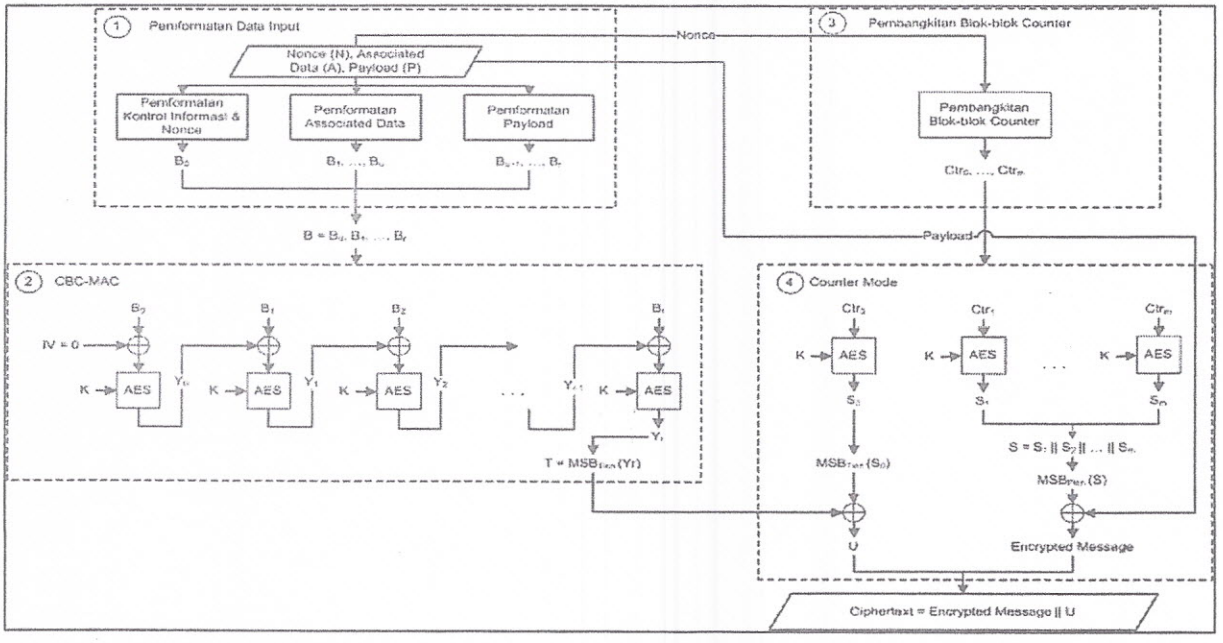
1. Jalankan fungsi pemformatan pada (N, A, P) untuk menghasilkan blok-blok $B_0, B_1, \ldots, B_r$.
2. Tetapkan $Y_0 = CIPH_K(B_0)$.
3. Untuk $i = 1$ sampai dengan r, lakukan $Y_i = CIPH_K(B_i \oplus Y_{i-1})$.
4. $T = MSB_{Tlen}(Y_r)$.
5. Jalankan fungsi pembangkitan *counter* untuk membangkitkan blok-blok *counter* $Ctr_0, Ctr_1, \ldots, Ctr_m$ dengan $m = \lceil Plen / 128 \rceil$
6. Untuk $j = 0$ sampai dengan m, lakukan $S_j = CIPH_K(Ctr_j)$.
7. $S = S_1 \| S_2 \| \ldots \| S_m$.
8. $C = (P \oplus MSB_{Plen}(S)) \| (T \oplus MSB_{Tlen}(S_0))$.

Ilustrasi proses *generation-encryption* dapat dilihat pada **Gambar 1.**

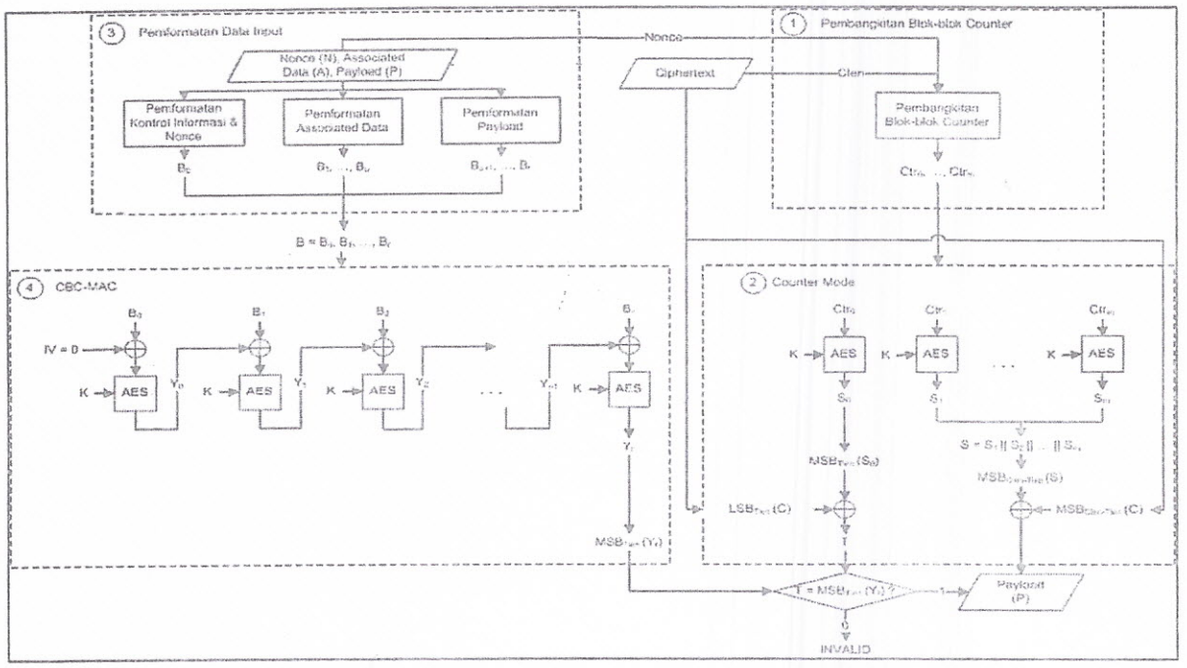
Proses Decryption-Verification

Berikut ini adalah langkah-langkah proses *decryption-verification CCM:*

1. Jika $Clen < Tlen$, maka kembalikan *INVALID.*
2. Jalankan fungsi pembangkitan *counter* untuk membangkitkan blok-blok *counter* $Ctr_0, Ctr_1, \ldots, Ctr_m$, dengan $m = \lceil (Clen - Tlen) / 128 \rceil$
3. Untuk $j = 0$ sampai dengan m, lakukan $S_j = CIPH_K(Ctr_j)$.
4. $S = S_1 \| S_2 \| \ldots \| S_m$.
5. $P = MSB_{Clen-Tlen}(C) \oplus MSB_{Clen-Tlen}(S)$.
6. $T = \text{LSB}_{Tlen}(C) \oplus MSB_{Tlen}(S_0)$.
7. Jika N, A, atau P tidak valid, maka kembalikan *INVALID,* selainnya lakukan fungsi pemformatan pada (N, A, P) untuk menghasilkan blok-blok $B_0, B_1, \ldots, B_r$.
8. Tetapkan $Y_0 = CIPH_K(B_0)$.
9. Untuk $i = 1$ sampai dengan r, lakukan $Y_j = CIPH_K(B_i \oplus Y_{i-1})$.
10. Jika $T \neq MSB_{Tlen}(Y_r)$, maka kembalikan *INVALID,* selainnya kembalikan P.



Gambar 1 Proses *generation-encryption C.*



Gambar 2 Proses *descryption-verification CCM*

Ilustrasi proses *decryption-verification* dapat dilihat pada **Gambar 2.**

HASIL DAN PEMBAHASAN

Analisis Algoritma CCM

Algoritma *CCM* bergantung pada pemilihan algoritma blok kunci simetrik yang mendasarinya. Dengan demikian, algoritma *CCM* merupakan sebuah modus operasi algoritma blok. Algoritma blok yang digunakan mendasari *CCM* merupakan algoritma yang sudah diakui dengan ukuran blok 128 bit. Sampai dengan sekarang satu-satunya algoritma blok 128 bit yang sudah diakui adalah algoritma *AES*.

AES didesain untuk melakukan proses penyandian secara rahasia dengan tingkat keamanan tak linear dengan kompleksitas waktu seefisien mungkin melalui penggunaan proses-proses transformasi yang ringan dalam implementasi seperti *XOR,* pergeseran, dan subtitusi. Akan tetapi, invers dari proses-proses tersebut terkadang memiliki efisiensi yang rendah, akibatnya proses dekripsi *AES* menjadi lambat dalam implementasi. Hal ini ditunjukkan dalam hasil penelitian *Giri (2004).*

Kekurangan ini dapat dihindari oleh *CCM. CCM* hanya memerlukan fungsi enkripsi *AES* baik dalam proses *generation-encryption* maupun *decryption-verification.* Hal ini dapat mengarah ke penghematan ukuran program *CCM* atau ukuran perangkat keras.

Data yang dilindungi *CCM* adalah *Nonce* (*N*), *Payload* (*P*), dan *Associated Data* (*A*). Ketiga elemen data ini dibentuk dengan cara yang unik menggunakan fungsi pemformatan menjadi rangkaian tak-kosong dari blok data input lengkap. Nilai *nonce* sebaiknya tak-berulang dengan pengertian bahwa pada sembarang dua pasangan data yang berbeda yang dilindungi oleh *CCM* selama umur hidup kunci akan diberikan *nonce* yang berbeda. Sebagai akibatnya, *nonce* menentukan suatu invokasi *(yaitu suatu operasi sandi blok)* dalam *CCM.* Dengan mengasumsikan bahwa tidak terdapat *associated data*, atau kalaupun ada, *associated data* tersebut bernilai kecil, maka dapat dikatakan bahwa jumlah total invokasi dalam *CCM* tidak akan melebihi 2^{61}.

Dari fungsi pemformatan dapat dilihat bahwa parameter *nonce n* menentukan parameter *q*, yaitu panjang oktet dari representasi biner panjang oktet *payload*, karena $q = 15 - n$. Parameter *q* menentukan panjang maksimum dari *payload* dengan $p < 2^{8q}$, maka *P* terdiri atas oktet-oktet kurang dari 2^{8q}, yaitu kurang dari 2^{8q-4} blok-blok 128-bit. Nilai *n* menentukan jumlah maksimum *nonce* yang berbeda, yaitu 2^{8n}. Dengan asumsi bahwa *associated data* bernilai kecil, maka semakin besar *nonce* akan mengakibatkan panjang maksimum dari *payload* yang bisa diproteksi *CCM* menjadi semakin kecil. Namun semakin kecil *nonce*, maka akan semakin kecil jumlah maksimum *nonce* yang berbeda.

CCM pada dasarnya menggabungkan dua algoritma kriptografik. Mekanisme pertama adalah *CBC-MAC* yang memberikan jaminan otentikasi, yang berarti musuh tidak dapat dengan mudah memalsukan valid *ciphertext* tanpa adanya akses terhadap kunci rahasia. *CBC-MAC* membangkitkan sebuah nilai *MAC* yang merupakan nilai otentikasi dari suatu pesan. Proses *CCM generation-encryption* akan mengekspansi

panjang *payload* dengan panjang *MAC*. *MAC* yang dibangkitkan dalam *CCM* dinyatakan dengan *T*. Panjang bit *T*, yaitu *Tlen*, merupakan suatu parameter yang ditetapkan untuk semua proses *CCM* dengan kunci yang diberikan. Nilai *Tlen* yang semakin besar memberikan jaminan otentisitas yang semakin besar pula. Namun, dengan semakin besar nilai *Tlen* maka semakin besar pula ruang penyimpanan untuk *ciphertext*.

Untuk kebanyakan aplikasi, Whiting *et al. (2002)* merekomendasikan penggunaan *Tlen* sekurang-kurangnya adalah 64. Dalam penelitian ini pun dipilih nilai minimum *Tlen* yang direkomendasikan yaitu 64 bit sehingga dapat memberikan jaminan otentisitas yang cukup serta ekspansi pesan pun tidak terlalu besar.

Mekanisme kedua dalam *CCM* adalah modus *CTR* yang memberikan jaminan kerahasiaan, yang berarti musuh tidak dapat dengan mudah mengambil sembarang informasi dari *ciphertext* tanpa adanya akses terhadap kunci rahasia. Tujuan utama seorang musuh dalam hal ini adalah untuk membedakan *ciphertext* dari "*random gibberish*". Karena nilai *nonce* selalu berbeda, maka blok pertama dari data terformat dan blok-blok *counter* akan selalu baru. Suatu *payload* yang sama dengan kunci rahasia yang sama pun akan menghasilkan *ciphertext* yang berbeda jika *nonce* yang digunakan juga berbeda. Dengan demikian, output *ciphertext* akan sangat menyerupai "*random gibberish*" dan pada modus *CTR* ini tidak terjadi *collision* sehingga *birthday attack* dapat dihindari.

Berbeda keadaannya jika *nonce* yang digunakan bernilai sama untuk beberapa aplikasi *CCM* dengan menggunakan kunci yang berbeda. *Meet-in-the-middle attack* atau *precomputation attack* dapat terjadi di sini. Langkah yang ditempuh adalah sebanyak 2^{64} dan bukan sebanyak ukuran ruang kunci yaitu 2^{128}. Dua senjata yang bisa digunakan untuk melawan serangan ini adalah dengan menggunakan ukuran kunci yang lebih besar serta dengan menggunakan nilai *nonce* yang selalu baru.

Modus *CTR* tidak hanya digunakan untuk mengenkripsi *payload* tetapi juga mengenkripsi *MAC*. Dengan mengenkripsi *MAC* maka *collision attack* pada proses *CBC-MAC* juga dapat dihindari dan musuh tidak akan memperoleh informasi apapun mengenai hasil *CBC-MAC*.

Pada proses *CCM decryption-verification*, seluruh atau sebagian *payload* tidak boleh dilepaskan sampai nilai *MAC* telah berhasil diverifikasi. Hal ini dapat mencegah terjadinya *chosen-ciphertext attack* yang akan mengambil informasi berharga dari kueri proses *decryption-verification* yang tak-valid.

Dari struktur *CCM* yang menggabungkan modus *CBC-MAC* dengan modus *CTR* dapat dilihat bahwa *CCM* bersifat *not fully parallelizable*. Pemparalelan tidak dapat dilakukan pada proses *CBC-MAC* namun mungkin untuk dilakukan pada proses *CTR*. Hal ini bisa dikatakan sebagai salah satu kekurangan *CCM*. Meskipun demikian, *CCM* mempunyai kelebihan dari sisi lain. Para perancang *CCM* tidak akan memberlakukan paten terhadap algoritma ini. Seluruh hak kekayaan intelektual terhadap algoritma *CCM* dilepaskan kepada publik.

Jika dipandang sebagai modus operasi algoritma blok, *CCM* memberikan jaminan keamanan yang lebih baik dari modus operasi *ECB* seperti yang diterapkan pada penelitian *Giri (2004)* terhadap algoritma blok *AES* karena selain jaminan kerahasiaan, *CCM* juga memberikan jaminan otentikasi pesan. Untuk dua blok *plaintext* yang sama dengan kunci yang diberikan, modus *ECB* akan menghasilkan blok *ciphertext* yang sama. Dengan alasan ini, *ECB* tidak baik digunakan untuk mengenkripsi *plaintext* berukuran panjang ataupun untuk suatu aplikasi yang kuncinya digunakan berulang-ulang untuk mengenkripsi lebih dari satu pesan. Jika hal ini dilakukan maka musuh dapat dengan mudah mematahkannya. Namun dari segi kecepatan, secara teoritis dapat dikatakan bahwa *ECB* lebih baik dari *CCM* karena dalam *ECB* hanya terdapat satu proses enkripsi algoritma blok dan tidak terdapat *chaining* di dalamnya, sedangkan dalam *CCM* terjadi dua proses yaitu otentikasi dengan *CBC-MAC* dan enkripsi dengan *CTR*, serta di dalam *CBC-MAC* terdapat proses *chaining*. Selain itu, struktur *ECB* yang memungkinkan pemrosesan secara paralel semakin dapat meningkatkan kecepatan algoritma *ECB*.

Berdasar analisis dapat dikatakan bahwa *CCM* didesain untuk mendapatkan kerahasiaan dan otentikasi pesan secara simultan menggunakan sebuah kunci rahasia. Kebutuhan akan hanya fungsi *forward cipher* dari algoritma blok yang mendasari *CCM* menjadikan ukuran kode implementasi *CCM* yang lebih kecil. Meskipun demikian, beberapa timbal balik kinerja pada proses-proses dalam *CCM* harus dipikirkan terlebih dahulu sebelum mengaplikasikan algoritma ini. Semakin besar *nonce*, maka akan semakin kecil panjang maksimum dari *payload* yang bisa diproteksi *CCM*. Semakin kecil *nonce*, maka akan semakin kecil jumlah maksimum *nonce* yang berbeda. Semakin besar nilai *Tlen* akan memberikan jaminan otentisitas yang semakin besar pula. Namun, dengan semakin besar nilai *Tlen* maka semakin besar pula ruang penyimpanan untuk *ciphertext*.

Analisis Algoritma Proses Generation-Encryption CCM

Langkah untuk melakukan proses *generation-encryption CCM* adalah:

1. Pemformatan input *nonce*, *associated data*, dan *payload*
2. Ekspansi kunci
3. Pembangkitan nilai otentikasi menggunakan *CBC-MAC*
4. Pembangkitan blok-blok *counter*
5. Enkripsi nilai otentikasi *(hasil pada langkah nomor 3)* dan *payload* menggunakan modus operasi *CTR* sehingga didapatkan *ciphertext*.

Waktu eksekusi pada *langkah 1* dan *2* adalah konstan *(misal α)* karena tidak dipengaruhi ukuran input. *Langkah 3 (misal waktu eksekusi = ε_1)* dipengaruhi jumlah blok data terformat, sehingga untuk input berukuran n_1 blok diperlukan waktu $\varepsilon_1 n_1$. *Langkah 4 (misal waktu eksekusi = ε_2)* dipengaruhi jumlah blok *payload*, sehingga untuk *payload* berukuran n_2 blok diperlukan waktu $\varepsilon_2(n_2+1)$. *Langkah 5 (misal waktu eksekusi = ε_3)* dipengaruhi jumlah blok *counter* hasil *langkah 4* sehingga diperlukan waktu sebesar $\varepsilon_3(n_2+1)$.

Secara keseluruhan, waktu eksekusi *generation-encryption CCM* adalah $\varepsilon_1 n_1 + \varepsilon_2(n_2+1) + \varepsilon_3(n_2+1) + \alpha$, dengan ε_1, ε_2, ε_3 dan α adalah suatu konstanta; n_1 adalah jumlah blok data terformat dan n_2 adalah jumlah blok *counter*. Notasi *O* untuk kasus terburuk proses *generation-encryption CCM* adalah:

$$GE_{(CCM)} = \varepsilon_1 n_1 + \varepsilon_2(n_2+1) + \varepsilon_3(n_2+1) + \alpha$$

Didapatkan kompleksitas $GE_{(CCM)}$ adalah dalam lingkup $O(n)$.

Analisis Algoritma Proses Decryption-Verification CCM

Langkah untuk melakukan proses *decryption-verification CCM* adalah:

1. Pembangkitan blok-blok *counter*
2. Ekspansi kunci
3. Dekripsi *ciphertext* menggunakan modus operasi *CTR* sehingga didapatkan *payload*
4. Pemformatan input *nonce*, *associated data*, dan *payload*
5. Pembangkitan nilai otentikasi menggunakan algoritma *CBC-MAC* untuk kemudian dilakukan verifikasi dengan nilai otentikasi yang terdapat pada *ciphertext* sehingga diperoleh *payload* terverifikasi.

Langkah 1 (misal waktu eksekusi = δ_1) dipengaruhi jumlah blok input *ciphertext*, sehingga untuk input berukuran n_1 blok *(setelah dikurangkan dengan panjang nilai otentikasi yang terdapat pada ciphertext)* diperlukan waktu $\delta_1(n_1+1)$. Waktu eksekusi pada *langkah 2* dan *4* adalah konstan *(misal β)* karena tidak dipengaruhi ukuran input. *Langkah 3 (misal waktu eksekusi = δ_2)* dipengaruhi jumlah blok *counter* hasil langkah 1 sehingga diperlukan waktu $\delta_2(n_1+1)$. *Langkah 5 (misal waktu eksekusi = δ_3)* dipengaruhi jumlah blok data terformat, sehingga untuk input berukuran n_3 blok diperlukan waktu $\delta_3 n_3$.

Secara keseluruhan, waktu eksekusi *decryption-verification CCM* adalah $\delta_1(n_1+1) + \delta_2(n_1+1) + \delta_3 n_3 + \beta$, dengan δ_1, δ_2, δ_3, dan β adalah suatu konstanta; n_1 adalah jumlah blok *counter* dan n_2 adalah jumlah blok data terformat. Notasi *O* untuk kasus terburuk proses *decryption-verification CCM* adalah:

$$DV_{(CCM)} = \delta_1(n_1+1) + \delta_2(n_1+1) + \delta_3 n_3 + \beta$$

Didapatkan kompleksitas $DV_{(CCM)}$ adalah dalam lingkup $O(n)$.

Analisis Uji Implementasi

Uji implementasi *CCM* dilakukan dengan lima macam percobaan. Setiap percobaan mengambil 32 ukuran *file payload* berupa teks sebagai objek kajian dalam selang 0 *byte* sampai dengan 1000 *byte*, dengan *n* ditetapkan dengan nilai 7 serta *t* ditetapkan dengan nilai 8. Untuk tiap percobaan diberikan nilai *a* yang berbeda, secara berturut-turut yaitu $a = 0, 8, 16, 24$, dan 32.

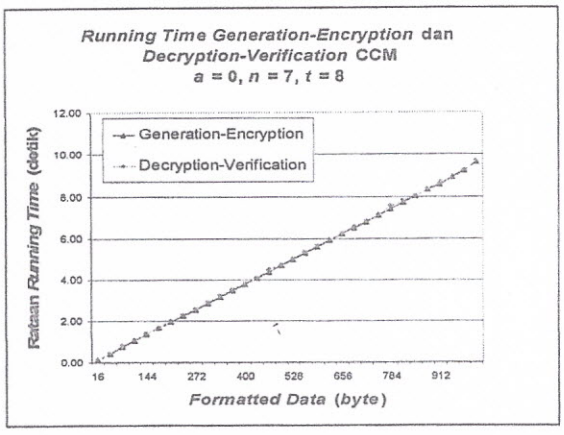
Berdasar uji implementasi, dengan meningkatnya ukuran *payload*, *running time* eksekusi proses uji baik *generation-encryption* maupun *decryption-verification* mengalami peningkatan. Hal ini terjadi karena semakin besar ukuran *payload* maka ukuran data terformat dan blok *counter* yang dibangkitkan menjadi semakin besar pula.

Gambar 3 adalah grafik hubungan antara proses *generation-encryption* dan *decryption-verification* terhadap ukuran data terformat untuk percobaan pertama, yaitu dengan $a = 0$. Dari grafik tersebut dapat dikatakan bahwa nilai *running time* proses *generation-encryption* dan *decryption-verification* adalah sama, dalam artian tidak terdapat beda atau selisih nilai *running time* yang signifikan di antara kedua proses tersebut. Hal ini dikarenakan *CCM* hanya memerlukan fungsi enkripsi dari *AES* baik untuk *CCM generation-encryption* maupun *CCM decryption-verification*. Dari keempat percobaan lainnya juga dapat dilihat keadaaan yang tidak berbeda, yaitu nilai *running time* proses *generation-encryption* dapat dikatakan sama dengan nilai *running time* proses *decryption-verification*.

Dengan adanya penambahan blok *associated data* maka ukuran data terformat semakin besar sehingga menambah jumlah invokasi yang terjadi. Penambahan jumlah invokasi ini mengakibatkan *running time* proses *generation-encryption* dan *decryption-verification* pada percobaan kedua sampai dengan kelima mengalami peningkatan dibandingkan percobaan pertama. **Gambar 4** memberikan gambaran hubungan antara proses *generation-encryption* dan *decryption-verification* terhadap ukuran *associated data* pada kelima percobaan yang dilakukan.

Pada percobaan ini dipilih nilai *t* minimum yang direkomendasikan yaitu sebesar 8. Seperti diuraikan sebelumnya bahwa nilai *MAC* akan mengekspansi panjang *ciphertext* sehingga panjang *ciphertext* menjadi sama dengan panjang *payload* ditambah dengan panjang *MAC*. Hal ini mengakibatkan *CCM* memerlukan penyimpanan *ciphertext* yang lebih besar dari modus operasi algoritma blok lainnya. Walaupun demikian, nilai perbandingan panjang *MAC* pada *ciphertext* terus mengalami penurunan seiring dengan pertambahan ukuran *payload*. Berdasar hal ini dapat dikatakan bahwa perluasan panjang *MAC* pada *ciphertext* ini menjadi semakin kecil dan tidak signifikan. Oleh karenanya, pemilihan nilai *t* yang lebih besar sebaiknya dilakukan agar mendapatkan jaminan otentisitas yang lebih besar

tanpa perlu mempermasalahkan secara berlebihan terhadap perluasan *ciphertext* yang akan terjadi.

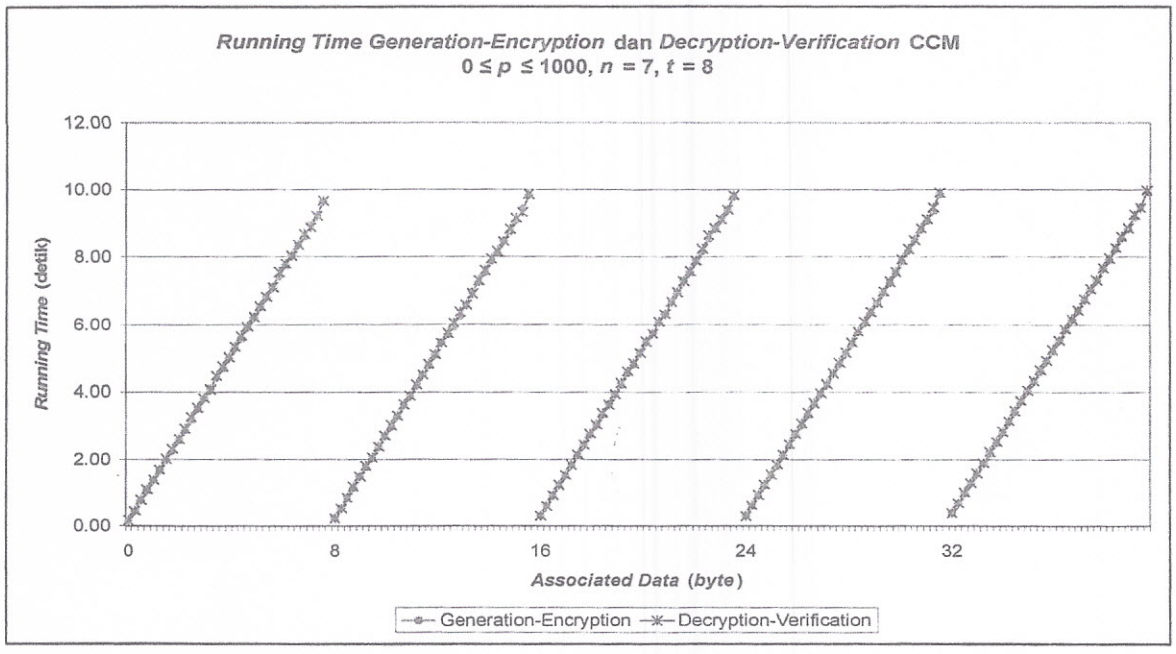


Gambar 3 Hubungan running time *generation-encryption* dan *descryption-verification* terhadap ukuran data terformat pada percobaan pertama.

Untuk menguji kesignifikanan dari beda *running time* antara proses *generation-encryption* dengan proses *decryption-verification* dilakukan analisis lanjut menggunakan uji statistik (*Independent-Samples T-Test*) dengan asumsi bahwa antara proses *generation-encryption* dan *decryption-verification* sebagai dua proses berbeda hanya ditentukan oleh dua peubah bebas, yaitu ukuran *file payload* dan *associated data* yang dieksekusi. Hasil uji statistik memperlihatkan nilai *Sig.* > 0.05 maka dapat disimpulkan dengan selang kepercayaan 95% proses *generation-encryption* tidak berbeda nyata dengan *decryption-verification*.

Melalui analisis regresi linier berganda pada masing-masing data kedua proses didapatkan koefisien grafik hubungan *running time* proses *generation-encryption* maupun proses *decryption-verification* terhadap ukuran *associated data* terformat *(ADF)* dan ukuran *payload* terformat *(PF)* yaitu *associated data* dan *payload* yang telah dibentuk oleh fungsi pemformatan. Secara umum persamaan garis dari grafik hubungan *running time* proses *generation-encryption* terhadap ukuran *ADF* (x_1) dan *PF* (x_2) dapat dituliskan sebagai $GE(x) = 0.004x_1 + 0.009x_2 + 0.136$. Sedangkan persamaan garis dari grafik hubungan *running time* proses *decryption-verification* terhadap ukuran *ADF* (x_1) dan PF (x_2) dapat dituliskan sebagai $DV(x) = 0.004x_1 + 0.010x_2 + 0.163$.

Kedua persamaan garis tersebut dapat digunakan untuk meramalkan *running time* jika diketahui ukuran *associated data* dan *payload* tertentu. Untuk *ADF* berukuran kecil dan *PF* berukuran besar dapat diasumsikan nilai *running time* proses *decryption-verification* adalah 1.111 kali nilai *running time* proses *generation-encryption*, sedangkan jika *ADF* berukuran besar dan *PF* berukuran kecil, nilai *running time* proses *decryption-verification* adalah 1.000 kali nilai *running time* proses *generation-encryption*, atau dengan kata lain



Gambar 4 Hubungan *running time generation-encryption* dan *decryption-verification* terhadap ukuran *associated data* pada kelima percobaan.

running time kedua proses tersebut akan bernilai sama.

Sebagai pembanding, pada penelitian ini juga dilakukan uji implementasi *AES-ECB* hasil penelitian *Giri (2004)* dengan menggunakan 32 ukuran *file* teks yang berbeda dalam selang 1 *byte* sampai dengan 1000 *byte* sebagai objek kajian.

Berdasar hasil pengujian didapatkan bahwa dengan meningkatnya ukuran pesan, *running time* eksekusi proses enkripsi dan dekripsi mengalami peningkatan. Hal ini terjadi karena semakin besar panjang blok pesan maka ukuran iterasi proses yang dilakukan pun akan semakin besar. Namun ukuran *running time* proses dekripsi selalu lebih besar dari *running time* proses enkripsi dengan *margin* yang semakin besar seiring membesarnya ukuran pesan.

Seperti telah dijelaskan sebelumnya bahwa algoritma kriptografi proses enkripsi *AES* didesain untuk melakukan proses penyandian secara rahasia melalui penggunaan proses-proses transformasi yang ringan dalam implementasi seperti *XOR*, pergeseran (*shift*), dan subtitusi. Akan tetapi, kebalikan dari proses-proses tersebut terkadang memiliki efisiensi yang rendah sehingga memungkinkan proses dekripsi *AES* menjadi lambat dalam implementasi.

Untuk mengetahui besar perbedaan antara proses enkripsi dan dekripsi dilakukan analisis lanjut dengan uji statistik (*Independent-Samples T-Test*) dengan asumsi bahwa proses enkripsi dan dekripsi adalah dua proses yang berbeda dan hanya ditentukan oleh satu peubah bebas, yaitu ukuran *file* pesan yang dieksekusi. Hasil uji statistik memperlihatkan nilai *Sig.* < 0.05 maka dapat disimpulkan dengan selang kepercayaan 95% proses enkripsi berbeda nyata dengan proses dekripsi.

Melalui analisis regresi linier pada masing-masing data enkripsi dan dekripsi didapatkan persamaan garis dari grafik hubungan *running time* proses enkripsi terhadap ukuran *file* pesan yang dapat dituliskan sebagai $E(x) = 0.005x + 0.010$. Sedangkan persamaan garis dari grafik hubungan *running time* proses dekripsi terhadap ukuran *file* pesan dapat dituliskan sebagai $D(x) = 0.007x + 0.016$, dengan x merupakan peubah bebas yang dalam hal ini adalah ukuran *file* pesan. Dari kedua persamaan tersebut maka untuk pesan berukuran berukuran besar dapat diasumsikan nilai *running time* proses dekripsi adalah 1.4 kali nilai *running time* proses enkripsi.

Persamaan garis dari grafik hubungan *running time* proses enkripsi terhadap ukuran *file* pesan pada *AES-ECB* dapat diperbandingkan dengan persamaan garis dari grafik hubungan *running time* proses *generation-encryption* dan *decryption-verification* terhadap ukuran *ADF* dan *PF*. Untuk *AES-CCM* tanpa *associated data*, dan untuk *file* pesan berukuran besar dapat diasumsikan nilai *running time* proses *generation-encryption* dan *decryption-verification* pada *AES-CCM* adalah 1.8 sampai dengan 2.0 kali *running time* proses enkripsi pada *AES-ECB*.

KESIMPULAN DAN SARAN

Kesimpulan

Berdasar analisis, *CCM* didesain untuk mendapatkan kerahasiaan dan otentikasi pesan secara simultan menggunakan sebuah kunci rahasia. Kebutuhan akan hanya fungsi *forward cipher* dari algoritma blok yang mendasari *CCM* menjadikan ukuran kode implementasi *CCM* yang lebih kecil. Meskipun demikian, beberapa timbal balik kinerja pada proses-proses dalam *CCM* harus dipikirkan terlebih dahulu sebelum mengaplikasikan algoritma ini. Semakin besar *nonce*, maka akan semakin kecil panjang maksimum dari *payload* yang bisa diproteksi *CCM*. Semakin kecil *nonce*, maka akan semakin kecil jumlah maksimum *nonce* yang berbeda. Semakin besar nilai *Tlen* akan memberikan jaminan otentisitas yang semakin besar pula. Namun, dengan semakin besar nilai *Tlen* maka semakin besar pula ruang penyimpanan yang harus disediakan bagi *ciphertext*.

Dengan analisis algoritma, didapatkan *CCM* memiliki kompleksitas pada lingkup $O(n)$. Hal ini berlaku bagi proses *generation-encryption* dan *decryption-verification*.

Bila dipandang sebagai modus operasi algoritma blok, *CCM* dapat memberikan jaminan keamanan yang lebih baik dari modus operasi *ECB*. Selain itu, *CCM* tidak hanya memberikan jaminan kerahasiaan, namun juga memberikan jaminan otentikasi pesan. Namun dari segi kecepatan, meskipun kedua modus memiliki notasi-O yang sama, modus ECB lebih baik dibandingkan *CCM* baik untuk proses enkripsi maupun dekripsi. Meskipun demikian, efisiensi yang rendah dari proses kebalikan dalam *AES* yang memungkinkan proses dekripsi *AES* menjadi lambat dalam implementasi, dapat dihindari oleh *CCM*.

Melalui analisis uji implementasi menggunakan Matlab 6.5 dan analisis uji statistik (*Independent-Samples T-Test*), dapat disimpulkan bahwa *running time* proses *CCM generation-encryption* tidak berbeda nyata dengan *running time* proses *CCM decryption-verification* dengan selang kepercayaan 95%. Melalui analisis regresi, dapat disimpulkan bahwa untuk *payload* berukuran besar dan tanpa *associated data*, nilai *running time* proses *generation-encryption* dan *decryption-verification* pada *AES-CCM* adalah 1.8 sampai dengan 2.0 kali *running time* proses enkripsi pada *AES-ECB*.

Saran

Ruang lingkup penelitian ini dibatasi pada algoritma *CCM* dengan *AES* sebagai algoritma blok yang mendasarinya. Penelitian dapat dikembangkan dengan memanfaatkan algoritma blok yang lain dengan panjang blok input 128 bit, seperti algoritma *Serpent, Twofish, MARS,* dan *RC6*.

Pada penelitian ini, *CCM* diaplikasikan pada *file* berbentuk teks. Untuk penelitian selanjutnya dapat dicoba untuk mengaplikasikan *CCM* pada *file* berbentuk lain, misalnya *file* gambar, atau *file* dokumen seperti *Portable Document Format (PDF).*

Algoritma *CCM* merupakan salah satu modus enkripsi terotentikasi yang secara simultan mendapatkan kerahasiaan dan otentikasi pesan dengan menggunakan sebuah kunci rahasia. Pada penelitian selanjutnya dapat dikaji modus enkripsi terotentikasi yang lain seperti algoritma *EAX,* Carter-Wegman *Counter (CWC),* Helix, dan *Galois/Counter Mode (GCM).*

DAFTAR PUSTAKA

Black J. 2003. Authenticated Encryption. www.cs.colorado.edu [7 Februari 2005].

Cormen TH, Leiserson CE, Riverst RL. 1990. *Introduction to Algorithms.* Massachusetts-London: The MIT Press.

Dworkin M. 2001. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. *NIST Special Publication* 800-38A. http://csrc.nist.gov/publications/ [24 Maret 2005].

Dworkin M. 2004. Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. *NIST Special Publication* 800-38C. http://csrc.nist.gov/publications/ [24 Maret 2005].

Ferguson N, Schneier B. 2003. *Practical Cryptography.* Indianapolis: Wiley Publishing, Inc.

Giri EP. 2004. Analisis Algoritme dan Waktu Enkripsi versus Dekripsi pada *Advanced Encryption*

Standard (AES) [skripsi]. Bogor: Fakultas Matematika dan Ilmu Pengetahuan Alam, Institut Pertanian Bogor.

Jonsson J. 2002. On the Security of CTR + CBC-MAC. http://csrc.nist.gov/encryption/modes/proposedmodes/ [7 Februari 2005].

Menezes A, Van Oorschot P, Vanstone S. 1996. *Handbook of Applied Cryptography.* CRC Press Inc.

Schneier B. 1996. *Applied Cryptography - Protocols, Algorithms and Source Code in C.* New York: John Wiley & Sons, Inc.

Stallings W. 2003. *Cryptography and Network Security Principles and Practice.* New Jersey: Pearson Education.

Whiting D, Housley R, Ferguson N. 2002. Counter with CBC-MAC (CCM). http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/ [24 Maret 2005].