Analisis *Overhead* Kinerja Komputasi Mode AES-GCM Terhadap AES-CBC untuk Implementasi *Authenticated Encryption*

Muhammad Fatihul Irhab - 13522143
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10 Bandung 40132, Indonesia
13522143@std.stei.itb.ac.id, fatihulirhab@gmail.com

Abstract*—Penerapan *Authenticated Encryption* (AE) sangat penting dalam mengamankan transmisi data modern untuk menjamin kerahasiaan dan integritas pesan secara bersamaan. Makalah ini bertujuan untuk menganalisis dan membandingkan *overhead* kinerja komputasi antara dua pendekatan *Authenticated Encryption* yang umum digunakan yaitu skema *Authenticated Encryption with Associated Data* (AEAD) menggunakan AES-GCM dan skema *Encrypt-then-MAC* menggunakan AES-CBC dengan HMAC-SHA256. Pengujian dilakukan menggunakan implementasi bahasa C++ dengan OpenSSL, mengukur metrik *Cycles Per Byte* (CPB) menggunakan instruksi CPU rdtsc pada variasi ukuran data mulai dari 64 *byte* hingga 1 MB untuk kunci 128-bit dan 256-bit. Hasil eksperimen menunjukkan bahwa AES-GCM secara konsisten mengungguli AES-CBC+HMAC pada seluruh pengujian. Pada ukuran data 1 MB dengan AES-128, mode GCM mencatat efisiensi sebesar 1.46 CPB, jauh lebih efisien dibandingkan mode CBC+HMAC yang mencatat 8.05 CPB, menghasilkan *speedup* hingga 5.52 kali. Sementara pada AES-256, GCM memiliki keunggulan dengan *speedup* mencapai 6.33 kali pada ukuran data 1 MB. Analisis menunjukkan bahwa keunggulan kinerja AES-GCM disebabkan oleh sifat *parallelizable* dari mode *Counter* (CTR) dan operasi GHASH yang dapat memanfaatkan akselerasi *hardware*, berbeda dengan sifat serial pada AES-CBC dan *overhead* tambahan dari pemrosesan HMAC. Makalah ini menyimpulkan bahwa AES-GCM adalah pilihan yang lebih optimal untuk implementasi sistem berkinerja tinggi dibandingkan skema tradisional *Encrypt-then-MAC*.

Keywords—Authenticated Encryption, AES-GCM, AES-CBC, HMAC-SHA256, Cycles Per Byte (CPB).

I. PENDAHULUAN

Dalam komunikasi digital modern, keamanan data tidak hanya bergantung pada kerahasiaan (*confidentiality*), tetapi juga pada integritas (*integrity*) dan autentikasi pesan, yang menuntut penerapan skema *Authenticated Encryption* (AE). Pendekatan tradisional untuk mencapai keamanan ini sering kali menggunakan komposisi generik *Encrypt-then-MAC*, seperti penggabungan mode AES-CBC (*Cipher Block Chaining*) dengan HMAC-SHA256, namun metode ini memiliki potensi kelemahan kinerja karena memerlukan dua kali pemrosesan data (satu *pass* untuk enkripsi dan satu *pass* untuk autentikasi) serta penanganan *padding* yang menambah *overhead* komputasi. Sebagai alternatif yang lebih modern, mode operasi *Authenticated Encryption with Associated Data* (AEAD) seperti AES-GCM (*Galois/Counter Mode*) dikembangkan untuk memberikan kerahasiaan dan integritas dalam satu proses, memanfaatkan mode *Counter* (CTR) yang memungkinkan pemrosesan paralel dan efisiensi tinggi pada arsitektur CPU modern.

Meskipun secara teoritis AES-GCM menawarkan keunggulan arsitektural dibandingkan skema serial AES-CBC+HMAC, analisis yang jelas mengenai besaran pengurangan *overhead* komputasi tetap diperlukan untuk memvalidasi kinerja implementasinya. Oleh karena itu, makalah ini bertujuan untuk mengukur dan membandingkan kinerja kedua skema tersebut menggunakan implementasi *library* kriptografi standar OpenSSL. Fokus utama makalah ini adalah menganalisis metrik *Cycles Per Byte* (CPB) dan total *CPU cycles* yang diukur secara akurat menggunakan instruksi rdtsc, untuk melihat seberapa besar dampak efisiensi yang dihasilkan oleh mekanisme AEAD dibandingkan pendekatan *Encrypt-then-MAC*.

Makalah ini membatasi ruang lingkup pada perbandingan algoritma AES-GCM dan AES-CBC+HMAC dengan konfigurasi kunci 128-bit dan 256-bit. Pengujian dilakukan terhadap variasi ukuran pesan mulai dari 64 *byte* hingga 1 MB (1.048.576 *byte*) dengan data acak, yang memungkinkan analisis mendalam mengenai pengaruh ukuran data terhadap biaya tetap (*fixed costs*) seperti inisialisasi dan finalisasi fungsi hash atau *cipher*. Metodologi pengukuran juga melibatkan mekanisme *warmup* iterasi untuk menstabilkan *cache* dan penghapusan *outlier* data untuk memastikan hasil yang konsisten.

Makalah ini diharapkan dapat menjawab rumusan masalah mengenai bagaimana perbandingan kuantitatif kinerja komputasi antara kedua skema tersebut, serta menentukan faktor percepatan (*speedup*) yang dihasilkan oleh AES-GCM pada berbagai skenario ukuran data. Secara teoritis, hasil makalah ini akan memberikan bukti mengenai efisiensi pemrosesan paralel pada mode GCM dibandingkan pemrosesan serial pada CBC. Sedangkan

secara praktis, makalah ini bermanfaat sebagai referensi dalam memilih skema enkripsi yang tepat, khususnya untuk implementasi yang membutuhkan *throughput* tinggi dan *latency* rendah, dengan menunjukkan secara jelas titik di mana *overhead* enkripsi terautentikasi menjadi efisien seiring bertambahnya ukuran data.

II. Dasar Teori

A. Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES) adalah algoritma kriptografi simetris berbasis *block cipher* yang mengenkripsi data dalam blok berukuran tetap 128-bit. AES mendukung variasi panjang kunci 128, 192, dan 256 bit. Dalam makalah ini, fokus diberikan pada varian AES-128 dan AES-256 untuk melihat dampak panjang kunci terhadap kinerja komputasi.

Secara internal, AES beroperasi melalui serangkaian transformasi *round* (substitusi, permutasi, dan *mixing*) yang jumlahnya bergantung pada panjang kunci. Karena AES hanya dapat memproses satu blok data (16 *byte*) pada satu waktu, diperlukan mode operasi (*mode of operation*) untuk mengenkripsi pesan yang ukurannya melebihi satu blok.

B. Cipher Block Chaining (CBC)

Mode CBC (*Cipher Block Chaining*) adalah mode operasi blok yang memperkenalkan ketergantungan antar blok *ciphertext*. Sebelum dienkripsi, setiap blok *plaintext* (P_i) dioperasikan XOR dengan blok *ciphertext* sebelumnya (C_{i-1}).

Persamaan matematis enkripsi CBC adalah sebagai berikut:

$$C_i = E_k(P_i \oplus C_{i-1})$$

Di mana C_0 adalah *Initialization Vector* (IV).

Karakteristik Kinerja CBC:

- Serialisasi, karena C_i bergantung pada C_{i-1}, proses enkripsi tidak dapat dilakukan secara paralel. Blok ke-2 harus menunggu blok ke-1 selesai.
- *Padding*, karena merupakan block cipher, jika data terakhir tidak memenuhi kelipatan 16 *byte*, diperlukan *padding* (pengisian data tambahan), yang menambah sedikit *overhead* ukuran data.

C. Message Authentication Code (HMAC)

Untuk menjamin integritas data (agar data tidak dimodifikasi di tengah jalan), enkripsi CBC saja tidak cukup. Diperlukan mekanisme otentikasi tambahan, yaitu HMAC (*Hash-based Message Authentication Code*).

Skema *Encrypt-then-MAC* yang diimplementasikan pada makalah ini:

1. Pesan (P) dienkripsi menjadi ciphertext (C).
2. HMAC dihitung berdasarkan *ciphertext* tersebut menggunakan fungsi hash SHA-256. *Tag* = HMAC(K_{mac}, C).
3. Penerima memverifikasi *tag* sebelum mendekripsi pesan.

Kombinasi ini memerlukan dua kali pemrosesan data (*two-pass*). Satu *pass* untuk enkripsi dan satu *pass* untuk *hashing*. Ini menciptakan *overhead* komputasi yang signifikan, terutama pada data berukuran besar.

D. AES-GCM

AES-GCM (*Galois/Counter Mode*) adalah mode operasi AEAD (*Authenticated Encryption with Associated Data*) yang menyediakan kerahasiaan dan integritas secara bersamaan tanpa memerlukan HMAC terpisah.

GCM menggabungkan dua mekanisme utama:

1. *Counter Mode* (CTR) untuk enkripsi, dengan mengubah *block cipher* menjadi *stream cipher*. Ciphertext dihasilkan dengan meng-XOR *plaintext* dengan *keystream* yang dihasilkan dari enkripsi *counter* yang terus bertambah.

$$C_i = P_i \oplus E_k(Nonce||Counter_i)$$

2. GMAC (*Galois Message Authentication Code*) untuk integritas, dengan menggunakan operasi perkalian polinomial dalam *Galois Field GF*(2^{128}) yang disebut GHASH untuk menghasilkan *authentication tag*.

Keunggulan Kinerja GCM:

- Paralelisasi, karena nilai *counter* dapat diprediksi sebelumnya, blok-blok enkripsi pada CTR dapat diproses secara paralel tanpa menunggu blok sebelumnya selesai. Ini sangat optimal pada CPU modern dengan instruksi SIMD (*Single Instruction, Multiple Data*).
- *Single-pass* efisiensi**,** karena mekanisme autentikasi GHASH terintegrasi dengan alur data, menghilangkan kebutuhan untuk proses *hashing* terpisah seperti pada HMAC.
- Tanpa *padding*, karena beroperasi seperti *stream cipher*, GCM tidak memerlukan *padding* blok, sehingga tidak ada *overhead* ukuran pada pesan yang tidak genap 16 *byte*.

E. GHASH

Efisiensi AES-GCM sangat bergantung pada implementasi fungsi GHASH yang beroperasi di atas *field* biner GF(2^{128}). Berbeda dengan fungsi hash tradisional seperti SHA-256 yang menggunakan operasi *bitwise* dan penjumlahan modulo 2^{32} yang kompleks, GHASH didasarkan pada perkalian polinomial. Elemen-elemen dalam *field* ini direpresentasikan sebagai polinomial dengan koefisien biner. Secara matematis, tag autentikasi (T) dihitung dengan rumus:

$$T = GHASH_H(A, C) \oplus E_k(J_0)$$

Dimana $H = E_K(0^{128})$ adalah *Hash Subkey*, A adalah *Associated Data*, dan C adalah *Ciphertext*. Fakta bahwa operasi ini bersifat aljabar linear memungkinkan teknik optimasi tingkat lanjut seperti metode Horner untuk evaluasi polinomial, yang dapat diparalelkan lebih lanjut dibandingkan struktur Merkle-Damgård pada SHA-256 yang sangat serial.

F. Cycles Per Byte (CPB)

Cycles Per Byte menghitung berapa banyak siklus *clock* CPU yang dibutuhkan untuk memproses satu *byte* data.

Rumus dasar untuk menghitung CPB adalah:

$$CPB = \frac{Total\ Siklus\ CPU}{Ukuran\ Data\ (Byte)}$$

- CPB Rendah: Menandakan efisiensi tinggi (lebih sedikit kerja CPU per *byte*).
- CPB Tinggi: Menandakan efisiensi rendah atau *overhead* yang besar.

Makalah ini menggunakan *Cycles Per Byte* sebagai indikator utama untuk membandingkan *overhead* antara mode serial AES-CBC dan mode paralel AES-GCM.

G. Speedup

Untuk membandingkan seberapa jauh AES-GCM mengungguli AES-CBC+HMAC, digunakan rumus *speedup*:

$$Speedup = \frac{CPB_{CBC+HMAC}}{CPB_{GCM}}$$

Nilai *Speedup* > 1 menunjukkan bahwa AES-GCM lebih cepat. Sebagai contoh, *speedup* 5x berarti AES-GCM mampu memproses data lima kali lebih cepat dibandingkan skema CBC+HMAC pada ukuran data yang sama.

III. IMPLEMENTASI

Implementasi dirancang untuk membandingkan dua skema enkripsi terautentikasi yaitu skema *Encrypt-then-MAC* menggunakan AES-CBC dengan HMAC-SHA256, dan skema AEAD menggunakan AES-GCM.

A. Skema AES-CBC dengan HMAC-SHA256

Implementasi skema ini mengikuti paradigma *Encrypt-then-MAC*. Proses dibagi menjadi dua tahap sekuensial dalam setiap iterasi pengukuran:

1. Enkripsi, menggunakan EVP_EncryptInit_ex() dengan mode EVP_aes_128_cbc() atau EVP_aes_256_cbc().
2. Autentikasi, menggunakan EVP_MAC_init() dan EVP_MAC_update() dengan algoritma HMAC-SHA256 terhadap *ciphertext* yang dihasilkan.

```
EVP_CIPHER_CTX* ctx = EVP_CIPHER_CTX_new();
for (int w = 0; w < WARMUP_ITERATIONS; w++) {
    int len;
    EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, key128, iv);
    EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, size);
    EVP_EncryptFinal_ex(ctx, ciphertext + len, &len);
    size_t mac_len;
    EVP_MAC_init(hmac_ctx, hmac_key, 32, NULL);
    EVP_MAC_update(hmac_ctx, ciphertext, size);
    EVP_MAC_final(hmac_ctx, mac, &mac_len, EVP_MAX_MD_SIZE);
}
for (int i = 0; i < NUM_ITERATIONS; i++) {
    unsigned long long start = rdtsc();
    int len;
    EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, key128, iv);
    EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, size);
    EVP_EncryptFinal_ex(ctx, ciphertext + len, &len);
    size_t mac_len;
    EVP_MAC_init(hmac_ctx, hmac_key, 32, NULL);
    EVP_MAC_update(hmac_ctx, ciphertext, size);
    EVP_MAC_final(hmac_ctx, mac, &mac_len, EVP_MAX_MD_SIZE);
    unsigned long long end = rdtsc();
    cbc_cycles.push_back(end - start);
}
```

Gambar 3.1.1 *Code Skema Encrypt-then-MAC*
Sumber : Dokumen Pribadi

B. Skema AES-GCM

Implementasi AES-GCM menggunakan pendekatan AEAD (*Authenticated Encryption with Associated Data*). Fungsi enkripsi dan pembuatan *tag* autentikasi dilakukan dalam satu konteks *cipher*. Langkah implementasi yaitu:

1. Inisialisasi EVP_EncryptInit_ex() dengan mode EVP_aes_128_gcm() atau EVP_aes_256_gcm().
2. Proses enkripsi data menggunakan EVP_EncryptUpdate().
3. Pengambilan *authentication tag* menggunakan EVP_CIPHER_CTX_ctrl() dengan parameter EVP_CTRL_GCM_GET_TAG.

Skema ini tidak memerlukan pemanggilan fungsi hash terpisah, mengurangi jumlah instruksi yang dieksekusi CPU.

```
EVP_CIPHER_CTX* ctx = EVP_CIPHER_CTX_new();
for (int w = 0; w < WARMUP_ITERATIONS; w++) {
    int len;
    EVP_EncryptInit_ex(ctx, EVP_aes_128_gcm(), NULL, key128, iv);
    EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, size);
    EVP_EncryptFinal_ex(ctx, ciphertext + len, &len);
    EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, 16, tag);
}
for (int i = 0; i < NUM_ITERATIONS; i++) {
    unsigned long long start = rdtsc();
    int len;
    EVP_EncryptInit_ex(ctx, EVP_aes_128_gcm(), NULL, key128, iv);
    EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, size);
    EVP_EncryptFinal_ex(ctx, ciphertext + len, &len);
    EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, 16, tag);
    unsigned long long end = rdtsc();
    gcm_cycles.push_back(end - start);
}
```

Gambar 3.2.1 *Code Skema AEAD*
Sumber : Dokumen Pribadi

C. Skenario Pengujian

Untuk mendapatkan kinerja yang komprehensif, pengujian dirancang dengan memvariasikan parameter ukuran data dan panjang kunci. Variasi ukuran data bertujuan untuk mengamati perilaku *overhead* inisialisasi pada data kecil dan efisiensi *throughput* pada data besar.

Skenario pengujian dibagi menjadi beberapa kategori ukuran data:

1. Ukuran Kecil (64 *Byte* - 512 *Byte*): Merepresentasikan paket data jaringan standar atau pesan singkat, di mana *overhead* inisialisasi fungsi kriptografi biasanya mendominasi waktu eksekusi.
2. Ukuran Sedang (1 KB - 16 KB): Merepresentasikan ukuran *buffer* standar pada aplikasi transfer file atau *web server*.
3. Ukuran Besar (64 KB - 1 MB): Merepresentasikan enkripsi *bulk data* atau arsip, di mana kemampuan pemrosesan paralel dan *bandwidth* memori menjadi faktor penentu utama.

Setiap skenario dijalankan menggunakan dua variasi panjang kunci AES, yaitu 128-bit dan 256-bit, untuk menganalisis dampak jumlah *rounds* enkripsi terhadap total latensi.

D. Metodologi Pengukuran Kinerja

Untuk mendapatkan hasil pengukuran yang akurat dan minim *noise*, sistem mengimplementasikan mekanisme pengukuran waktu berbasis siklus CPU.

Pengukuran waktu eksekusi tidak menggunakan fungsi waktu sistem operasi (seperti clock()) karena resolusinya yang terbatas. Sistem menggunakan instruksi *assembly* rdtsc (*Read Time-Stamp Counter*) yang dibungkus dalam suatu fungsi. Fungsi ini digunakan tepat sebelum dan sesudah blok kode enkripsi dieksekusi. Selisih nilai kedua pembacaan tersebut merepresentasikan total siklus CPU yang dihabiskan.

Sebelum pengukuran data diambil, sistem melakukan iterasi *warmup* sebanyak WARMUP_ITERATIONS (1000 iterasi). Tujuannya adalah:

1. Memastikan instruksi kode dan tabel lookup AES masuk ke dalam *L1/L2 Cache* CPU.
2. Memicu mekanisme *branch prediction* pada prosesor.
3. Menghindari *cold start*

Setelah *warmup*, pengukuran dilakukan sebanyak NUM_ITERATIONS (10.000 iterasi). Data mentah hasil pengukuran kemudian diproses menggunakan fungsi calculate_mean_cycles(). Fungsi ini mengurutkan data dan membuang 5% data teratas dan 5% data terbawah sebelum menghitung rata-rata.

```
double calculate_mean_cycles(std::vector<unsigned long long>& data) {
    std::sort(data.begin(), data.end());
    size_t trim_count = static_cast<size_t>(data.size() * TRIM_PERCENT);
    std::vector<unsigned long long> trimmed_data(
        data.begin() + trim_count,
        data.end() - trim_count
    );
    unsigned long long sum = std::accumulate(trimmed_data.begin(), trimmed_data.end(), 0ULL);
    return static_cast<double>(sum) / trimmed_data.size();
}
```

Gambar 3.4.1 *Code calculate_mean_cycles*
Sumber : Dokumen Pribadi

Ambang batas 5% dipilih untuk menghilangkan gangguan eksternal (*noise*) yang signifikan, khususnya *context switching* sistem operasi dan interupsi *background processes* yang menyebabkan lonjakan siklus CPU ekstrem (data teratas), serta anomali pengukuran akibat inkonsistensi *cold/warm cache* (data terbawah). Dengan demikian, hasil rata-rata yang diperoleh lebih merepresentasikan kinerja murni algoritma kriptografi tanpa bias gangguan sistem.

IV. Pengujian dan Analisis

A. Hasil Pengujian AES-128

Data menunjukkan tren penurunan nilai CPB seiring bertambahnya ukuran data pada kedua skema, namun dengan laju yang berbeda signifikan.

Data (*Byte*)	AES-128-CBC + HMAC	AES-128-GCM	*Speedup*
64	155.73	99.99	1.56
128	87.07	48.16	1.81
256	49.54	25.41	1.95
512	27.83	14.28	1.95
1024	18.65	7.97	2.34
2048	12.55	4.75	2.75
4096	10.25	2.87	3.66
8192	9.16	2.47	3.70
16384	8.78	2.13	4.12
32768	8.28	1.65	5.02
65536	8.35	1.82	4.58
131072	8.42	1.82	4.63
262144	8.42	1.65	5.11
524288	8.11	1.47	5.50
1048576	8.05	1.46	5.52

Tabel 4.1.1 Perbandingan Kinerja Rata-rata (*Cycles Per Byte*) pada Kunci 128-bit.

Berdasarkan Tabel 4.1.1, pada ukuran data terkecil (64 *byte*), AES-CBC+HMAC mencatat 155.73 CPB, sedangkan AES-GCM mencatat 99.99 CPB, menghasilkan *speedup* awal sebesar 1.56x. Seiring bertambahnya ukuran data, kesenjangan kinerja semakin melebar. Pada data terbesar (1 MB), efisiensi AES-CBC+HMAC meningkat menjadi 8.05 CPB, namun AES-GCM mencapai efisiensi yang jauh lebih tinggi yaitu 1.46 CPB. Hal ini menghasilkan faktor *speedup* maksimum sebesar 5.52x untuk AES-128.

B. Hasil Pengujian AES-256

AES-256 memiliki jumlah *rounds* enkripsi yang lebih banyak (14 *rounds*) dibandingkan AES-128 (10 *rounds*), yang secara teoritis meningkatkan beban komputasi dasar.

Data (*Byte*)	AES-256-CBC + HMAC	AES-256-GCM	*Speedup*
64	107.97	65.99	1.64
128	65.78	41.73	1.58
256	38.86	18.02	2.16
512	24.00	12.11	1.98
1024	17.19	5.50	3.12
2048	12.86	3.14	4.09
4096	12.12	2.96	4.09
8192	10.50	1.81	5.80
16384	10.07	1.67	6.02
32768	9.93	1.80	5.51
65536	9.90	1.58	6.25
131072	9.76	1.59	6.13
262144	9.75	1.58	6.17
524288	10.17	1.56	6.50
1048576	9.75	1.54	6.33

Tabel 4.1.2 Perbandingan Kinerja Rata-rata (*Cycles Per Byte*) pada Kunci 256-bit.

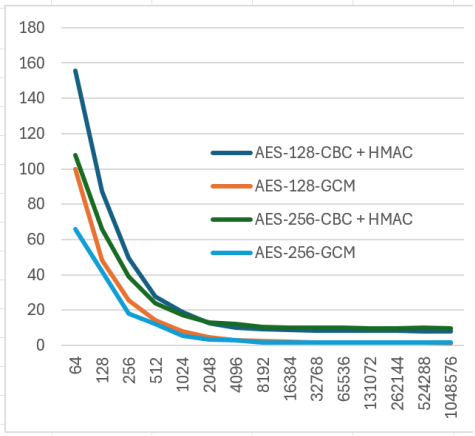
Berdasarkan Tabel 4.1.2, Tren yang teramati pada AES-256 konsisten dengan AES-128. Pada ukuran 1 MB, AES-CBC+HMAC mencapai 9.75 CPB, sedikit lebih tinggi dibandingkan versi 128-bitnya, yang mencerminkan biaya tambahan dari *rounds* ekstra AES-256. Namun, AES-GCM tetap mempertahankan efisiensi tinggi di angka 1.54 CPB. Menariknya, *speedup* yang dihasilkan pada ukuran data terbesar justru lebih tinggi, mencapai 6.33x. Hal ini mengindikasikan bahwa mode GCM mampu menangani peningkatan kompleksitas kunci dengan lebih efisien dibandingkan kombinasi CBC dan HMAC.

C. Analisis Dampak Ukuran Pesan dan Reduktsi Biaya Tetap (Fixed Cost)

Data pada kedua tabel menunjukkan nilai CPB yang sangat tinggi pada ukuran pesan kecil (misalnya 64 dan 128

byte). Hal ini terjadi karena setiap operasi kriptografi memiliki biaya inisialisasi dan finalisasi yang konstan, tidak peduli seberapa kecil datanya. Pada skema CBC+HMAC, biaya ini meliputi inisialisasi konteks AES, inisialisasi konteks HMAC, penanganan *padding* blok terakhir, dan finalisasi perhitungan hash.

Pada ukuran kecil, biaya tetap ini mendominasi total siklus CPU. Seiring bertambahnya ukuran pesan, biaya tetap ini tereduksi ke jumlah *byte* yang lebih banyak, sehingga nilai CPB menurun drastis dan mendekati kinerja asimtotik murni dari algoritma enkripsinya.



Gambar 4.3.1 ***Grafik Tren CPB terhadap Ukuran Data***
Sumber : Dokumen Pribadi

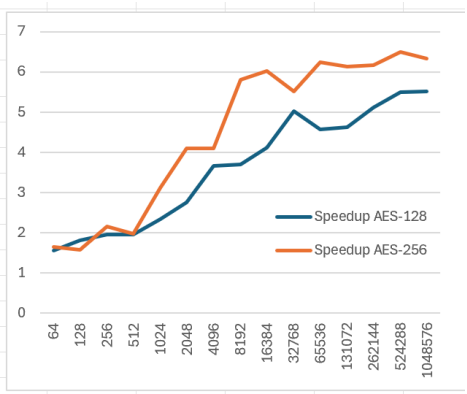
Gambar 4.3.1. Grafik memperlihatkan penurunan tajam nilai CPB pada rentang ukuran data kecil (< 1024 *byte*) untuk kedua skema, yang mengkonfirmasi dominasi biaya inisialisasi. Namun, terlihat divergensi yang jelas di mana kurva AES-GCM melandai jauh lebih cepat menuju angka asimtotik ~1.46 CPB, sedangkan kurva AES-CBC+HMAC tertahan di kisaran ~8.05 CPB. Jarak vertikal antara kedua garis ini merepresentasikan beban komputasi tambahan (*overhead*) yang harus ditanggung oleh skema *Encrypt-then-MAC*.

D. Analisis Efisiensi Serial vs Paralel

Faktor paling dominan yang menyebabkan keunggulan AES-GCM terletak pada perbedaan mendasar arsitektur mode operasinya.

Mode CBC bersifat serial secara inheren. Enkripsi blok saat ini (C_i) bergantung pada hasil enkripsi blok sebelumnya (C_{i-1}). Ketergantungan data ini menciptakan hambatan (*bottleneck*) dalam *pipeline* CPU, karena prosesor tidak dapat memulai enkripsi blok berikutnya sampai blok sebelumnya selesai diproses. Hal ini mencegah pemanfaatan instruksi SIMD (*Single Instruction, Multiple Data*) atau eksekusi paralel tingkat instruksi (*instruction-level parallelism*) secara maksimal untuk antar-blok.

GCM menggunakan mode *Counter* (CTR) untuk enkripsi intinya. Dalam mode CTR, *keystream* dihasilkan dengan mengenkripsi nilai *counter* yang dapat diprediksi. Karena nilai *counter* untuk setiap blok diketahui sejak awal dan tidak bergantung pada hasil blok data lain, proses enkripsi blok-blok data dapat dilakukan secara independen dan paralel. Arsitektur ini sangat optimal untuk CPU modern yang memiliki banyak unit eksekusi dan dukungan instruksi vektor (seperti AVX2 atau AVX-512), memungkinkan pemrosesan beberapa blok data sekaligus dalam satu siklus *clock*.



Gambar 4.4.1 ***Grafik Tren Speedup terhadap Ukuran Data***
Sumber : Dokumen Pribadi

Efektivitas arsitektur paralel tersebut dibuktikan oleh tren kenaikan *speedup* pada Gambar 4.4.1. Grafik tersebut menunjukkan korelasi positif antara ukuran dan faktor percepatan. Pada ukuran kecil, *speedup* relatif rendah (~1.5x) karena *pipeline* CPU belum terisi penuh. Namun, seiring dengan bertambahnya data yang mengisi instruksi SIMD secara kontinu, kurva pada grafik naik secara konsisten hingga mencapai puncaknya di angka 5.52x (AES-128) dan 6.33x (AES-256). Kenaikan ini memvalidasi bahwa AES-GCM mampu menggunakan *Instruction Level Parallelism* (ILP) secara maksimal pada beban kerja tinggi, berbeda dengan AES-CBC yang kinerjanya terbatasi oleh latensi serial.

E. Analisis Overhead Skema Two-Pass vs Single-Pass AEAD

Skema AES-CBC+HMAC menerapkan paradigma *Encrypt-then-MAC*, yang merupakan pendekatan *two-pass*. Data harus diproses sepenuhnya dua kali. Pertama untuk enkripsi CBC, dan kedua untuk perhitungan HMAC-SHA256 di atas *ciphertext* yang dihasilkan. Setiap proses membutuhkan akses memori dan siklus pemrosesan tersendiri.

Sebaliknya, AES-GCM adalah skema AEAD yang dirancang sebagai *single-pass*. Operasi enkripsi (CTR) dan operasi autentikasi (GHASH) terintegrasi dalam satu aliran pemrosesan data yang efisien. Meskipun GHASH sendiri adalah operasi matematika di *Galois Field*, implementasinya pada perangkat keras modern (menggunakan instruksi seperti PCLMULQDQ untuk perkalian polinomial) sangat cepat, sehingga total biaya gabungan GCM tetap jauh lebih rendah dibandingkan menjumlahkan biaya enkripsi CBC dan gabungan hashing SHA-256.

V. KESIMPULAN

Berdasarkan pengujian dan analisis mengenai kinerja komputasi antara mode AES-GCM (*Authenticated Encryption with Associated Data*) dan skema AES-CBC+HMAC (*Encrypt-then-MAC*), dapat ditarik beberapa kesimpulan utama sebagai berikut:

1. Hasil *benchmark* menunjukkan bahwa AES-GCM secara konsisten mengungguli AES-CBC+HMAC pada seluruh ukuran data yang diuji (64 *byte* hingga 1 MB), baik untuk panjang kunci 128-bit maupun 256-bit. Hal ini membuktikan bahwa desain AEAD modern yang terintegrasi jauh lebih efisien daripada komposisi generik *cipher* blok dan MAC.
2. Pada ukuran data terbesar (1 MB), di mana *overhead* tetap telah tereduksi, AES-GCM menunjukkan efisiensi yang luar biasa. Untuk AES-128, GCM mencapai ~1.46 CPB, sementara CBC+HMAC berada di ~8.05 CPB, menghasilkan faktor percepatan (*speedup*) sebesar 5.52 kali. Sedangkan AES-256, GCM mencapai ~1.54 CPB berbanding ~9.75 CPB pada CBC+HMAC, dengan *speedup* mencapai 6.33 kali. Angka-angka ini mengkonfirmasi bahwa GCM mampu menangani beban kerja kriptografi intensif dengan penggunaan siklus CPU yang jauh lebih sedikit.
3. Analisis menunjukkan bahwa keunggulan kinerja GCM terutama disebabkan oleh dua faktor arsitektural. Pertama pararelisasi, dengan mode CTR yang mendasari GCM memungkinkan pemrosesan blok data secara independen dan paralel, sangat cocok dengan arsitektur CPU modern yang mendukung instruksi SIMD dan akselerasi perangkat keras, berbeda dengan sifat serial dari mode CBC. Kedua *single-pass*, Mekanisme AEAD pada GCM melakukan enkripsi dan autentikasi dalam satu pemrosesan yang efisien, menghindari biaya tambahan dari perhitungan HMAC yang diperlukan pada skema AES-CBC+HMAC.
4. Pada data berukuran kecil, kinerja kedua skema sangat dipengaruhi oleh biaya tetap inisialisasi dan finalisasi, menghasilkan nilai CPB yang tinggi. Seiring bertambahnya ukuran data, biaya ini tereduksi, dan kesenjangan kinerja akibat perbedaan arsitektural (paralel vs serial) menjadi semakin dominan, yang tercermin dari peningkatan nilai *speedup*.

Hal ini menyimpulkan untuk pengembangan sistem yang membutuhkan *Authenticated Encryption*, sangat disarankan untuk menggunakan AES-GCM sebagai pilihan utama. Mode ini menawarkan kombinasi keamanan yang kuat dan kinerja komputasi yang baik, terutama pada infrastruktur perangkat keras modern. Penggunaan skema AES-CBC+HMAC sebaiknya dibatasi hanya untuk keperluan kompatibilitas dengan sistem *legacy* yang belum mendukung AEAD.

UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa, karena atas rahmat dan karunia-Nya, makalah ini dapat diselesaikan dengan baik. Makalah ini disusun untuk memenuhi salah satu tugas mata kuliah IF4020 Kriptografi.

Pada kesempatan ini, penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya kepada Bapak Dr. Ir. Rinaldi Munir, MT, selaku dosen pengampu mata kuliah IF4020 Kriptografi yang telah memberikan ilmu, arahan, dan bimbingan selama perkuliahan berlangsung.

Penulis menyadari bahwa makalah ini masih jauh dari kesempurnaan. Oleh karena itu, kritik dan saran yang membangun sangat diharapkan demi penyempurnaan karya tulis di masa mendatang.

REFERENSI

[1] R. Munir, Block Chiper (Bagian 1). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/14-Block-Cipher-Bagian1-2025.pdf, pada 13 Desember 2025.

[2] R. Munir, Block Chiper (Bagian 2). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/15-Block-Cipher-Bagian2-2025.pdf, pada 13 Desember 2025.

[3] R. Munir, Beberapa Block Cipher (Bagian 2). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/17-Beberapa-block-cipher-bagian2-2025.pdf, pada 13 Desember 2025.

[4] R. Munir, Beberapa Fungsi Hash. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/27-Beberapa-fungsi-hash-2025.pdf, pada 13 Desember 2025.

[5] R. Munir, Message Authentication Code (MAC). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/28-MAC-2025.pdf, pada 13 Desember 2025.

[6] A. Altigani, M. Abdelmagid, B. Barry, Analyzing the Performance of the Advanced Encryption Standard Block Cipher Modes of Operation: Highlighting the National Institute of Standards and Technology Recommendations. Diakses dari https://www.researchgate.net/publication/306006230_Analyzing_the_Performance_of_the_Advanced_Encryption_Standard_Block_Cipher_Modes_of_Operation_Highlighting_the_National_Institute_of_Standards_and_Technology_Recommendations, pada 13 Desember 2025.

[7] R. R. Sitraka, R. H. Malalatiana, Evaluations of Crypto-System AES Using Multiple Bloc Ciphering Mode. Diakses dari https://www.researchgate.net/publication/384775007_Evaluations_of_Crypto-System_AES_Using_Multiple_Bloc_Ciphering_Mode, pada 13 Desember 2025.

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, 24 Desember 2025

Muhammad Fatihul Irhab 13522143