Implementasi dan Analisis Komparasi Kinerja Algoritma ChaCha20-Poly1305 Terhadap AES-GCM pada Sistem Penyimpanan Berkas Multimedia

Christian Justin Hendrawan
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13522135@std.stei.itb.ac.id, christian.justin23@gmail.com

***Abstract*—Pertumbuhan eksponensial data multimedia beresolusi tinggi menuntut adanya sistem penyimpanan yang tidak hanya aman, tetapi juga efisien dalam penggunaan sumber daya komputasi. *Advanced Encryption Standard* (AES) dengan panjang kunci 256-bit saat ini menjadi standar industri untuk keamanan data. Namun, kinerja AES sering kali mengalami penurunan efisiensi pada implementasi berbasis perangkat lunak murni atau pada lingkungan tanpa akselerasi perangkat keras khusus. Makalah ini melakukan studi komparatif antara kinerja algoritma ChaCha20-Poly1305 dan AES-256-GCM yang diterapkan pada sistem penyimpanan berkas multimedia menggunakan mekanisme *Streaming Authenticated Encryption with Associated Data* (AEAD). Penelitian ini menerapkan teknik segmentasi data atau *chunking* untuk menjaga kompleksitas memori tetap konstan terlepas dari ukuran berkas. Pengujian dilakukan dengan mengukur waktu eksekusi, *throughput*, dan penggunaan memori puncak pada variasi ukuran berkas hingga 1 GB. Hasil eksperimen menunjukkan bahwa ChaCha20-Poly1305 memiliki kinerja yang lebih unggul dibandingkan AES-256-GCM dalam lingkungan perangkat lunak, dengan capaian *throughput* enkripsi puncak sebesar 648,71 MB/s berbanding 416,58 MB/s pada AES-256-GCM. Selain itu, ChaCha20-Poly1305 menunjukkan efisiensi memori yang lebih baik dengan penggunaan rata-rata sekitar 3 MB dibandingkan 4 MB pada AES. Berdasarkan hasil tersebut, ChaCha20-Poly1305 direkomendasikan sebagai alternatif yang tangguh dan efisien untuk pengamanan aset multimedia, khususnya pada arsitektur sistem yang heterogen.**

***Index Terms*—Kriptografi, ChaCha20-Poly1305, AES-256-GCM, Keamanan Multimedia, Streaming AEAD.**

I. PENDAHULUAN

Pertumbuhan eksponensial data digital, khususnya format multimedia beresolusi tinggi seperti video 4K dan audio *lossless*, telah meningkatkan kebutuhan akan sistem penyimpanan yang andal dan aman. Dalam era di mana pertukaran informasi terjadi secara masif melalui jaringan global, aspek kerahasiaan (*confidentiality*) dan integritas (*integrity*) data menjadi prioritas utama. Penerapan mekanisme kriptografi yang tangguh menjadi krusial sebagai garis pertahanan terakhir dalam memitigasi risiko kebocoran data serta akses tidak sah terhadap aset digital.

Saat ini, *Advanced Encryption Standard* (AES) dengan panjang kunci 256-bit merupakan standar industri yang paling luas digunakan untuk mengamankan data statis [2]. AES-256 dikenal memiliki tingkat keamanan yang sangat tinggi dan telah teruji. Namun, tantangan muncul ketika algoritma ini diterapkan pada berkas multimedia berukuran besar. Proses enkripsi dan dekripsi pada data bervolume besar membutuhkan sumber daya komputasi yang signifikan. Meskipun AES memiliki performa optimal pada perangkat keras yang mendukung instruksi khusus seperti AES-NI, kinerjanya dapat mengalami penurunan efisiensi pada lingkungan perangkat lunak murni atau perangkat dengan daya komputasi terbatas, yang berpotensi memengaruhi pengalaman pengguna akibat latensi akses data [6]. Perbandingan latensi pada operasi dasar antara varian AES dan ChaCha20, sebagaimana ditunjukkan pada Gambar 1, mengilustrasikan bagaimana perbedaan frekuensi kerja dan jenis operasi memengaruhi waktu pemrosesan pada level primitif.

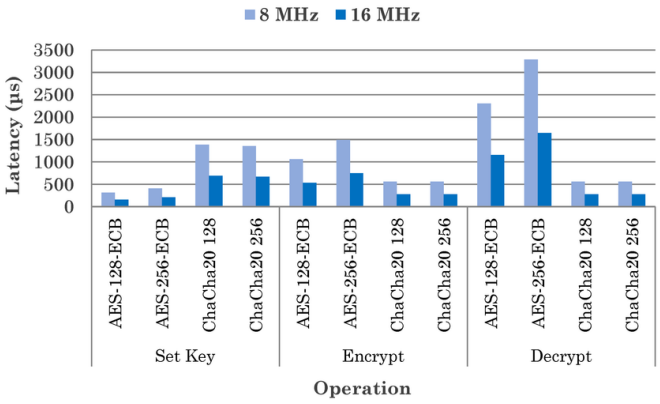

Fig. 1. Perbandingan antara latensi algoritma AES dan ChaCha [9]

Sebagai alternatif, algoritma ChaCha20-Poly1305 muncul sebagai kandidat yang menjanjikan dalam ranah kriptografi modern. Dikembangkan sebagai penyempurnaan dari Salsa20,

ChaCha20 adalah *stream cipher* yang didesain untuk memberikan keamanan tinggi dengan efisiensi performa yang unggul, terutama pada implementasi berbasis perangkat lunak tanpa akselerasi perangkat keras khusus [4]. Lebih jauh lagi, integrasi ChaCha20 dengan Poly1305 sebagai *Message Authentication Code* (MAC) menghasilkan sebuah skema *Authenticated Encryption with Associated Data* (AEAD) yang komprehensif. Konstruksi ini mampu menjamin kerahasiaan sekaligus integritas data dengan *overhead* komputasi yang minimal, sehingga kini menjadi standar yang diadopsi secara masif oleh protokol komunikasi modern. [1].

Makalah ini bertujuan untuk melakukan studi komparatif antara kinerja algoritma ChaCha20-Poly1305 dan AES-256, khususnya dalam konteks sistem penyimpanan berkas multimedia. Penelitian ini akan berfokus pada analisis metrik kinerja yang meliputi waktu eksekusi enkripsi dan dekripsi, serta penggunaan sumber daya sistem saat memproses berkas dengan variasi ukuran. Melalui analisis ini, diharapkan dapat ditarik kesimpulan mengenai efektivitas implementasi ChaCha20-Poly1305 sebagai alternatif standar enkripsi pada sistem penyimpanan modern.

II. TEORI DASAR

A. Kriptografi Kunci-Simetri

Kriptografi simetris, atau sering disebut sebagai *secret-key cryptography*, merupakan skema fundamental dalam pengamanan data yang menggunakan satu kunci rahasia yang sama digunakan untuk proses enkripsi dan dekripsi [7]. Dalam paradigma ini, keamanan data sepenuhnya bergantung pada kerahasiaan kunci tersebut. Jika pihak ketiga yang tidak berwenang berhasil mendapatkan kunci, maka kerahasiaan seluruh pesan yang diamankan dengan kunci tersebut akan terkompromi.

Secara matematis, proses dalam kriptografi simetris dapat didefinisikan sebagai transformasi data asli (*plaintext*) menjadi data tersandi (*ciphertext*) menggunakan fungsi enkripsi E dan kunci rahasia K. Sebaliknya, proses pengembalian pesan asli dilakukan menggunakan fungsi dekripsi D dengan kunci yang sama. Jika P merepresentasikan *plaintext* dan C merepresentasikan *ciphertext*, maka hubungan keduanya dapat dinotasikan sebagai berikut [7]:

$$C = E_K(P) \tag{1}$$

$$P = D_K(C) \tag{2}$$

Sehingga berlaku sifat invertibilitas:

$$D_K(E_K(P)) = P \tag{3}$$

Kekuatan keamanan dari algoritma simetris sangat dipengaruhi oleh panjang kunci (dalam bit) yang digunakan. Semakin panjang kunci, semakin besar ruang kunci yang tersedia, yang secara langsung meningkatkan kompleksitas komputasi bagi penyerang untuk melakukan serangan *brute-force*. Untuk kunci dengan panjang n bit, terdapat 2^n kemungkinan kunci, sehingga probabilitas menebak kunci yang benar secara acak adalah $1/2^n$.

Dalam implementasinya, algoritma kriptografi simetris umumnya dikategorikan menjadi dua jenis utama berdasarkan cara pemrosesan datanya, yaitu *block cipher* dan *stream cipher*. *Block cipher* memproses data input dalam blok-blok berukuran tetap (misalnya 128 bit), sedangkan *stream cipher* memproses data secara kontinu bit-demi-bit atau byte-demi-byte dengan menggabungkan *plaintext* bersama aliran *keystream* yang dibangkitkan secara pseudo-acak.

B. Authenticated Encryption with Associated Data (AEAD)

Dalam praktik keamanan informasi modern, kerahasiaan saja sering kali tidak cukup untuk menjamin keamanan komunikasi. Skema enkripsi tradisional, seperti mode operasi *Cipher Block Chaining* (CBC) atau *Counter* (CTR) tanpa otentikasi tambahan, rentan terhadap serangan modifikasi atau *active attacks* lainnya. Penyerang dapat memodifikasi *ciphertext* di tengah jalan sedemikian rupa sehingga menghasilkan perubahan yang dapat diprediksi pada *plaintext* setelah didekripsi, tanpa diketahui oleh penerima. Oleh karena itu, diperlukan jaminan integritas dan otentikasi pesan secara bersamaan.

Authenticated Encryption (AE) adalah skema yang secara simultan memberikan jaminan kerahasiaan, integritas, dan otentikasi data. Pengembangan lebih lanjut dari konsep ini adalah *Authenticated Encryption with Associated Data* (AEAD). AEAD memungkinkan penerima untuk memverifikasi integritas dari data yang terenkripsi serta data tambahan yang tidak terenkripsi namun perlu diotentikasi, yang disebut sebagai *Associated Data* (AD) [5]. AD umumnya berupa *header* jaringan, metadata, atau informasi alamat yang harus tetap terbaca untuk keperluan perutean atau pemrosesan, namun tidak boleh dimodifikasi.

Secara formal, skema AEAD didefinisikan oleh dua fungsi utama: fungsi enkripsi terotentikasi $\mathcal{E}$ dan fungsi dekripsi terverifikasi $\mathcal{D}$. Misalkan K adalah kunci rahasia, N adalah *nonce* (angka unik yang hanya digunakan sekali), A adalah *associated data*, dan P adalah *plaintext*. Fungsi enkripsi menghasilkan pasangan *ciphertext* C dan *authentication tag* T:

$$(C, T) = \mathcal{E}_K(N, A, P) \tag{4}$$

Di mana T adalah nilai pendek (biasanya 128 bit) yang bergantung pada seluruh input (K, N, A, P). Fungsi dekripsi menerima input kunci, *nonce*, *associated data*, *ciphertext*, dan *tag*, lalu mengembalikan *plaintext* P jika dan hanya jika verifikasi integritas berhasil, atau simbol kesalahan $\perp$ jika verifikasi gagal [5]:

$$\mathcal{D}_K(N, A, C, T) = \begin{cases} P & \text{jika otentikasi valid} \\ \perp & \text{jika otentikasi gagal} \end{cases} \tag{5}$$

Prinsip keamanan AEAD menjamin bahwa jika penyerang mengubah satu bit pun pada C, A, atau N, maka proses dekripsi akan menghasilkan $\perp$, sehingga sistem tidak akan pernah memproses data yang telah dimanipulasi. Hal ini mencegah serangan *chosen-ciphertext attacks* yang sering menjadi kelemahan pada skema enkripsi tanpa otentikasi.

C. Algoritma Advanced Encryption Standard (AES-256)

Advanced Encryption Standard (AES) adalah standar enkripsi kunci simetris yang ditetapkan oleh *National Institute of Standards and Technology* (NIST) pada tahun 2001 untuk menggantikan *Data Encryption Standard* (DES) [2]. AES didasarkan pada keluarga *cipher* Rijndael dan beroperasi sebagai *block cipher* dengan ukuran blok tetap sebesar 128 bit. Algoritma ini mendukung panjang kunci 128, 192, dan 256 bit. Varian dengan kunci 256-bit, yang dikenal sebagai AES-256, dianggap sebagai standar emas untuk keamanan data sensitif karena ruang kuncinya yang sangat besar (2^{256}), yang membuatnya secara komputasi kebal terhadap serangan *brute-force* dengan teknologi komputasi saat ini maupun masa depan yang dapat diprediksi.

Berbeda dengan pendahulunya (DES) yang menggunakan struktur Feistel, AES menggunakan struktur *Substitution-Permutation Network* (SPN). Proses enkripsi dilakukan melalui serangkaian putaran transformasi data. Jumlah putaran (N_r) bergantung pada panjang kunci. Untuk AES-256, jumlah putaran ditetapkan sebanyak $N_r = 14$. Setiap putaran (kecuali putaran terakhir) terdiri dari empat transformasi non-linear dan linear yang beroperasi pada matriks status berukuran 4×4 byte:

1) *SubBytes*: Substitusi non-linear di mana setiap byte pada *state* diganti dengan byte lain berdasarkan tabel substitusi (*S-Box*).
2) *ShiftRows*: Transformasi permutasi di mana baris-baris pada *state matrix* digeser secara siklik (baris ke-i digeser sejauh i posisi).
3) *MixColumns*: Operasi pencampuran linear yang mengalikan setiap kolom *state* dengan polinomial tetap pada *Galois Field* $GF(2^8)$. Langkah ini dihilangkan pada putaran terakhir.
4) *AddRoundKey*: Operasi XOR antara *state* dengan *round key* yang diturunkan dari kunci utama melalui algoritma *Key Schedule*.

Untuk mengimplementasikan fungsionalitas AEAD pada AES, mode operasi yang paling umum digunakan adalah *Galois/Counter Mode* (GCM) [3]. AES-GCM menggabungkan mode *counter* untuk enkripsi berkecepatan tinggi dengan fungsi *universal hashing* di atas medan Galois biner $GF(2^{128})$ untuk menghasilkan tag otentikasi. Dalam mode ini, AES beroperasi layaknya *stream cipher*: blok enkripsi AES mengenkripsi sebuah *counter* yang nilainya bertambah secara inkremental untuk menghasilkan *keystream*, yang kemudian di-XOR dengan *plaintext* untuk menghasilkan *ciphertext*. Keunggulan utama GCM adalah efisiensi dan kemampuannya untuk diparalelisasi, yang memungkinkan kinerja tinggi pada perangkat keras modern yang mendukung instruksi pipelining.

D. Algoritma ChaCha20-Poly1305

ChaCha20-Poly1305 adalah algoritma enkripsi terotentikasi (AEAD) yang menggabungkan *stream cipher* ChaCha20 dengan *Message Authentication Code* (MAC) Poly1305. Algoritma ini distandarisasi dalam RFC 8439 dan dirancang untuk memberikan kinerja kriptografi yang sangat tinggi pada implementasi berbasis perangkat lunak, khususnya pada arsitektur prosesor umum yang tidak memiliki dukungan akselerasi perangkat keras khusus untuk kriptografi [1].

1) ChaCha20: ChaCha20 merupakan penyempurnaan dari algoritma Salsa20 yang dikembangkan oleh Daniel J. Bernstein [4]. Sebagai *stream cipher*, ChaCha20 bekerja dengan membangkitkan *keystream* yang kemudian dioperasikan dengan fungsi XOR terhadap *plaintext* untuk menghasilkan *ciphertext*. Keamanan dan efisiensi ChaCha20 bersumber dari desain *Add-Rotate-XOR* (ARX), yang hanya menggunakan operasi penjumlahan 32-bit, rotasi bit, dan XOR [4]. Operasi-operasi ini bersifat *CPU-friendly* dan dapat dieksekusi dalam waktu konstan, sehingga meminimalkan risiko serangan *side-channel* berbasis waktu.

Struktur internal ChaCha20 beroperasi pada *state matrix* berukuran 4×4 yang terdiri dari 16 kata 32-bit. Matriks ini diinisialisasi dengan konfigurasi berikut:

1) 4 kata konstanta (biasanya string ASCII "expand 32-byte k").
2) 8 kata untuk kunci rahasia 256-bit.
3) 1 kata untuk *counter* (penanda posisi blok).
4) 3 kata untuk *nonce* (angka acak unik).

Inti dari proses pengacakan data pada ChaCha20 terletak pada fungsi *quarter round*. Fungsi ini memproses empat kata (a, b, c, d) melalui serangkaian operasi ARX sebagai berikut [4]:

$$a = a + b \tag{6}$$
$$d = (d \oplus a) \lll 16 \tag{7}$$
$$c = c + d \tag{8}$$
$$b = (b \oplus c) \lll 12 \tag{9}$$
$$a = a + b \tag{10}$$
$$d = (d \oplus a) \lll 8 \tag{11}$$
$$c = c + d \tag{12}$$
$$b = (b \oplus c) \lll 7 \tag{13}$$

Dimana operator $\oplus$ melambangkan XOR dan $\lll n$ melambangkan rotasi bit ke kiri sebanyak n bit. Operasi ini diulang sebanyak 20 putaran untuk memastikan difusi bit yang sempurna sebelum hasil akhir dijumlahkan dengan *state* awal untuk menghasilkan blok *keystream*.

2) Poly1305: Poly1305 adalah fungsi *one-time authenticator* yang didesain untuk menjamin integritas data dan otentikasi pengirim. Poly1305 menerima masukan berupa kunci 32-byte (yang bersifat sekali pakai atau *one-time key*) dan pesan dengan panjang berapapun, kemudian menghasilkan *tag* otentikasi berukuran 16-byte (128-bit). Keunggulan utama Poly1305 dibandingkan skema MAC tradisional (seperti HMAC-SHA256) adalah komputasinya yang sangat ringan karena berbasis pada operasi aritmatika modular pada bilangan prima $2^{130} - 5$, serta sifat keamanannya yang deterministik jika kunci yang digunakan bersifat unik.

Arsitektur skema AEAD ChaCha20-Poly1305 yang menunjukkan alur koordinasi mekanisme otentikasi Poly1305 divisualisasikan pada Gambar 2

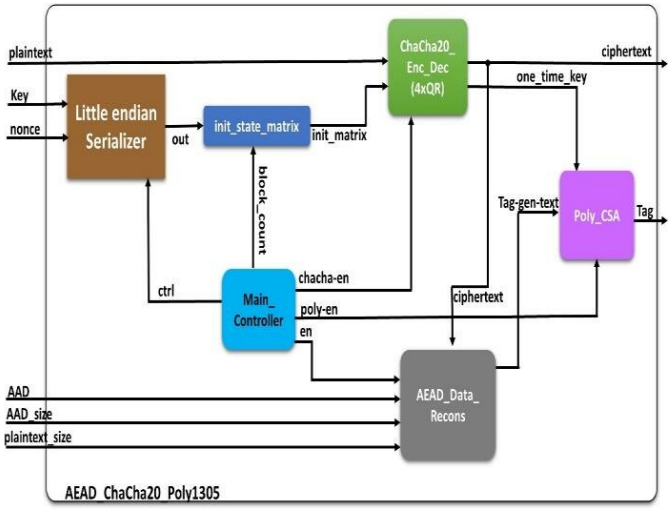

Fig. 2. Arsitektur Sistem AEAD ChaCha20-Poly1305 [8]

E. Karakteristik Berkas Multimedia dan Teknik Chunking

Berkas multimedia modern, seperti video beresolusi tinggi (4K/8K) dan arsip audio *lossless*, memiliki karakteristik unik dibandingkan dokumen teks atau data transaksional biasa. Karakteristik utama yang paling menonjol adalah ukuran data yang masif dan sifat akses yang sekuensial. Besarnya ukuran data tersebut sering kali melampaui kapasitas memori utama, sehingga menghadirkan tantangan teknis dalam menjaga efisiensi sistem serta manajemen sumber daya. Memuat seluruh berkas berukuran gigabyte ke dalam memori untuk proses enkripsi tunggal (*monolithic encryption*) tidak hanya tidak efisien, tetapi sering kali tidak mungkin dilakukan pada perangkat dengan sumber daya terbatas.

Untuk mengatasi kendala tersebut, diterapkan teknik segmentasi data yang dikenal sebagai *chunking*. Teknik ini melibatkan pemecahan aliran data kontinu menjadi blok-blok yang lebih kecil dan diskrit dengan ukuran tetap. Secara formal, jika sebuah berkas F memiliki ukuran total L byte, dan ukuran segmen ditentukan sebesar S byte, maka berkas tersebut akan dibagi menjadi n segmen, di mana:

$$n = \lceil L/S \rceil \tag{14}$$

Setiap segmen B_i (untuk $i = 0, 1, \ldots, n-1$) diproses secara independen dalam memori. Pendekatan ini mengubah kompleksitas penggunaan memori dari linear terhadap ukuran berkas $O(L)$ menjadi konstan $O(S)$, di mana $S \ll L$. Hal ini memungkinkan sistem untuk memproses berkas dengan ukuran tak terbatas (selama penyimpanan sekunder mencukupi) dengan penggunaan RAM yang tetap rendah dan stabil.

Dalam konteks kriptografi terotentikasi, penerapan teknik *chunking* sering disebut sebagai *streaming AEAD*. Alih-alih menghasilkan satu *tag* otentikasi untuk seluruh berkas di akhir proses, setiap *chunk* dienkripsi dan diberi *tag* otentikasi secara terpisah. Misalkan fungsi AEAD $\mathcal{E}$, kunci K, dan *nonce* dasar N. Untuk setiap *chunk* ke-i, digunakan *nonce* unik N_i yang diturunkan dari N dan indeks i, sehingga:

$$(C_i, T_i) = \mathcal{E}_K(N_i, \ldots, B_i) \tag{15}$$

Penerapan skema ini menawarkan keuntungan ganda sebagai berikut:

- Integritas granular: Kerusakan data dapat dideteksi secara spesifik pada tingkat *chunk*. Mekanisme ini mencegah terjadinya kegagalan fatal di mana kesalahan pada satu bit data tidak lagi mengharuskan pembatalan atau pembuangan seluruh hasil dekripsi berkas.
- Keamanan proaktif: Sistem mampu memitigasi risiko pemrosesan data ilegal dalam volume besar. Melalui verifikasi otentikasi bertahap yang dilakukan pada setiap segmen, anomali dapat dideteksi secara dini tanpa harus menunggu penyelesaian seluruh proses dekripsi berkas secara utuh.

III. METODOLOGI PENELITIAN

A. Arsitektur Pemrosesan Data Berbasis Chunk

Mengingat objek penelitian adalah berkas multimedia yang memiliki karakteristik ukuran besar, metode enkripsi monolitik dinilai tidak efisien. Oleh karena itu, penelitian ini menerapkan arsitektur pemrosesan berbasis aliran (*stream-based processing*) dengan teknik *chunking*.

Dalam skema ini, berkas masukan F dibagi menjadi n segmen $C_0, C_1, \ldots, C_{n-1}$ dengan ukuran tetap $S = 1$ MB. Proses enkripsi dilakukan secara iteratif pada setiap segmen independen, sehingga kompleksitas ruang memori dapat ditekan menjadi $O(1)$ relatif terhadap ukuran berkas. Alur kerja sistem dirancang sebagai berikut:

1) Sistem menginisialisasi parameter kriptografi utama, termasuk kunci rahasia 256-bit dan *Base Nonce* 96-bit.
2) Berkas dibaca per blok sebesar 1 MB.
3) Setiap blok diproses menggunakan fungsi enkripsi terotentikasi yang dipilih.
4) Hasil enkripsi (*ciphertext* dan *tag*) ditulis ke media penyimpanan secara sekuensial.

B. Protokol Keamanan dan Manajemen Nonce

Keamanan algoritma AEAD seperti GCM dan Poly1305 sangat bergantung pada keunikan *nonce*. Penggunaan ulang *nonce* pada kunci yang sama dapat menyebabkan hilangnya kerahasiaan data secara total. Untuk mengatasi tantangan ini dalam skema *multi-chunk*, penelitian ini mengusulkan mekanisme derivasi *nonce* deterministik. Alih-alih menyimpan *nonce* acak untuk setiap *chunk*, sistem hanya membangkitkan satu *base nonce* (N_{base}) di awal proses. *Nonce* operasional untuk setiap chunk ke-i (N_i) diturunkan menggunakan operasi XOR:

$$N_i = N_{base} \oplus (i \parallel \text{padding}) \tag{16}$$

Dimana i adalah indeks urutan chunk 64-bit *little-endian*. Mekanisme ini menjamin bahwa setiap panggilan fungsi enkripsi memiliki vektor inisialisasi yang unik, memenuhi syarat keamanan kriptografi tanpa mengorbankan efisiensi penyimpanan.

C. Rancangan Skenario Pengujian

Pengujian dilakukan secara komparatif dengan variabel bebas berupa algoritma enkripsi dan ukuran berkas multimedia. Skema pengujian dirancang untuk mengisolasi kinerja algoritma dari gangguan eksternal sistem operasi.

1) Variabel penelitian: Penelitian ini menetapkan parameter variabel sebagai berikut:

- Variabel bebas:
  1) Jenis algoritma: Perbandingan *head-to-head* antara AES-256-GCM dan ChaCha20-Poly1305.
  2) Ukuran berkas: Pengujian dilakukan terhadap tiga variasi ukuran berkas multimedia untuk merepresentasikan skenario penggunaan nyata:
     - Ukuran kecil (10 MB): Merepresentasikan klip audio pendek atau video resolusi rendah.
     - Ukuran menengah (500 MB): Merepresentasikan video resolusi HD (720p).
     - Ukuran besar (1 GB): Merepresentasikan video resolusi Full HD (1080p).
- Variabel terikat:
  1) Waktu eksekusi (*execution time* dalam detik).
  2) *Throughput* (MB/s).
  3) Penggunaan memori puncak (*peak memory usage* dalam MB).

2) Lingkungan pengujian: Perangkat lunak dibangun menggunakan bahasa pemrograman Python 3.10 dengan memanfaatkan pustaka `cryptography` sebagai *backend* untuk operasi primitif. Penggunaan pustaka ini memastikan bahwa implementasi algoritma (khususnya AES) dapat mengakses instruksi perangkat keras (AES-NI) jika tersedia, memberikan perbandingan yang adil dengan implementasi standar industri.

D. Metrik Evaluasi Kinerja

Kinerja kedua algoritma akan diukur berdasarkan tiga parameter utama:

1) Waktu komputasi (execution time): Waktu total yang dibutuhkan (T_{exec}) untuk menyelesaikan proses enkripsi atau dekripsi dari inisiasi hingga terminasi I/O. Pengukuran dilakukan menggunakan fungsi *high-resolution clock* dari sistem operasi untuk presisi tinggi.

2) Throughput: Kecepatan rata-rata pemrosesan data yang dinyatakan dalam Megabyte per detik (MB/s). Metrik ini menggambarkan seberapa cepat sistem dapat menangani aliran data multimedia.

$$\text{Throughput} = \frac{\text{Ukuran Berkas (MB)}}{T_{exec}\ \text{(detik)}} \tag{17}$$

3) Efisiensi memori: Penggunaan memori puncak (*peak memory usage*) diukur untuk memvalidasi efektivitas teknik *chunking*. Pengukuran dilakukan menggunakan modul pelacakan alokasi memori yaitu *memory tracer* yang mencatat selisih antara penggunaan memori awal dan penggunaan memori tertinggi selama proses berlangsung.

IV. IMPLEMENTASI SISTEM

A. Arsitektur Modul Program

Sistem yang dibangun dirancang dengan arsitektur modular untuk memastikan pemisahan tanggung jawab antara manajemen kunci, logika inti kriptografi, dan antarmuka pengujian. Struktur modul program didefinisikan sebagai berikut:

1) Modul manajemen kunci (`key_utils.py`): Modul ini bertanggung jawab atas seluruh operasi terkait kunci simetris. Fungsi utama di dalamnya adalah `generate_key_hex()` yang memanfaatkan generator angka acak kriptografis dari sistem operasi, serta fungsi `parse_key()` yang melakukan validasi ketat terhadap integritas format kunci heksadesimal sebelum digunakan dalam proses enkripsi.
2) Modul inti kriptografi (`engine.py`): Merupakan komponen sentral yang mengimplementasikan logika pemrosesan berkas. Modul ini menangani pembacaan berkas multimedia dalam bentuk segmen, pengelolaan *nonce* deterministik untuk setiap blok, serta pemanggilan API primitif AEAD.
3) Modul antarmuka dan metrik (`app.py`): Modul ini berfungsi sebagai *wrapper* yang mengintegrasikan seluruh sistem. Selain menyediakan antarmuka grafis untuk kemudahan pengujian, modul ini mengimplementasikan fungsi dekorator atau *wrapper* `time_and_memory()` yang bertugas mengisolasi dan mencatat penggunaan sumber daya sistem secara spesifik pada saat fungsi kriptografi berjalan.

B. Struktur Format Berkas Terenkripsi

Dalam sistem yang diimplementasikan, berkas keluaran tidak hanya berisi *ciphertext* mentah, melainkan dibungkus dalam sebuah format kontainer khusus. Hal ini diperlukan karena penggunaan teknik *chunking* mengharuskan sistem untuk menyimpan metadata tambahan agar proses dekripsi dapat dilakukan secara presisi dan mendukung verifikasi integritas pada setiap blok.

Struktur biner dari berkas terenkripsi terdiri dari dua komponen utama: *header* dan *payload* terfragmentasi.

1) Header berkas (16 Byte): Setiap berkas diawali dengan *header* tetap yang berfungsi sebagai identitas kriptografis:
   - *Base nonce* (12 Byte): Merupakan nilai acak unik yang dihasilkan oleh `os.urandom()`. Nilai ini menjadi basis derivasi *nonce* untuk seluruh *chunk* di dalam berkas tersebut.
   - *Chunk count* (4 Byte): *Unsigned integer* dalam format *little-endian* yang menyimpan total jumlah

segmen data di dalam berkas. Metadata ini krusial untuk mengontrol iterasi pada proses dekripsi.

2) Payload terfragmentasi: Setelah *header*, berkas berisi rangkaian blok data yang disusun secara sekuensial. Setiap unit blok memiliki struktur sebagai berikut:
   - *Chunk length* (4 Byte): Menunjukkan ukuran total dari data terenkripsi di dalam blok tersebut (termasuk ukuran *tag*).
   - *Ciphertext + auth tag*: Data multimedia yang telah terenkripsi diikuti oleh 16 byte kode otentikasi pesan. Penggabungan *tag* langsung setelah *ciphertext* pada setiap *chunk* memungkinkan deteksi kerusakan data secara granular tanpa harus membaca seluruh isi berkas.

Format ini memastikan bahwa meskipun ukuran berkas multimedia mencapai orde gigabyte, sistem hanya perlu melakukan *seeking* pada memori sebesar ukuran satu *chunk* (1 MB) ditambah metadata overhead yang sangat kecil.

C. Implementasi Algoritma AEAD Berbasis Aliran

Inti dari sistem ini terletak pada kemampuannya untuk melakukan enkripsi dan dekripsi pada aliran data yang besar tanpa membebani memori utama. Hal ini dicapai melalui implementasi algoritma AEAD berbasis aliran atau yang biasa dikenal dengan *streaming AEAD*. Seperti yang sudah dijelaskan pada Teori Dasar, algoritma ini membagi berkas multimedia menjadi segmen-segmen berukuran 1 MB.

1) Mekanisme derivasi nonce: Keamanan algoritma berbasis mode *Counter* seperti AES-GCM dan ChaCha20 sangat bergantung pada keunikan *nonce* untuk setiap operasi dengan kunci yang sama. Untuk menjamin hal ini pada skema *multi-chunk*, sistem mengimplementasikan fungsi `_derive_chunk_nonce()`.

Logika yang digunakan adalah melakukan operasi bitwise XOR antara *base nonce* (12 byte) dengan indeks *chunk* yang dikonversi ke dalam format *little-endian* 64-bit dan ditambahkan *padding* 32-bit agar sesuai dengan panjang *nonce* standar. Secara matematis, *nonce* untuk *chunk* ke-i (N_i) diturunkan melalui persamaan:

$$N_i = N_{base} \oplus (\text{struct.pack}(\text{"} < Q\text{"}, i) \parallel \texttt{0x00000000}) \quad (18)$$

Metode ini memastikan bahwa meskipun terdapat jutaan *chunk* dalam satu berkas video besar, setiap *chunk* akan dienkripsi dengan *nonce* yang berbeda secara deterministik, sehingga mencegah serangan *nonce-reuse*. Implementasi kode Python untuk logika tersebut ditunjukkan pada Gambar 3.

```python
def _derive_chunk_nonce(base_nonce: bytes, chunk_index: int) -> bytes:
    index_bytes = struct.pack("<Q", chunk_index) + b"\x00\x00\x00\x00"

    return bytes(a ^ b for a, b in zip(base_nonce, index_bytes))
```

Fig. 3. Fungsi _derive_chunk_nonce()

2) Siklus pemrosesan data: Siklus pemrosesan data merupakan implementasi konkret dari skema *streaming AEAD* yang diterapkan pada sistem penyimpanan multimedia. Prosedur ini dirancang untuk bekerja secara atomik dan efisien dalam penggunaan sumber daya melalui dua alur utama: enkripsi terotentikasi dan dekripsi terverifikasi.

a) Prosedur enkripsi terotentikasi: Prosedur enkripsi pada fungsi `encrypt_file()` diimplementasikan melalui tahapan teknis sistematis sebagai berikut:

1) Instansiasi primitif kriptografi: Berdasarkan parameter `algorithm`, sistem melakukan instansiasi objek `AESGCM` atau `ChaCha20Poly1305` dengan kunci 256-bit. Pada tahap ini, sebuah *base nonce* 12-byte dibangkitkan secara acak melalui sumber entropi sistem operasi (`os.urandom`).
2) Inisialisasi kontainer dan metadata: Sistem membuka *file handle* dalam mode biner (*write-binary*). Sebelum memproses data, *base nonce* dituliskan pada posisi awal berkas. Karena jumlah total *chunk* hanya diketahui setelah seluruh berkas terbaca, sistem menuliskan *placeholder* 4-byte untuk `chunk_count` menggunakan `struct.pack("<I", 0)`.
3) Iterasi segmen (*Chunking Loop*):
   - Untuk setiap segmen berukuran 1 MB, fungsi `_derive_chunk_nonce()` dipanggil untuk menghasilkan vektor inisialisasi unik.
   - Hasil enkripsi (yang secara otomatis menyertakan *Authentication Tag* 16-byte) dituliskan ke berkas dengan format *prefix* panjang segmen. Implementasi logika ini dapat dilihat pada Gambar 4.

```python
while True:
    plaintext = fin.read(chunk_size)
    if not plaintext:
        break

    chunk_nonce = _derive_chunk_nonce(base_nonce, chunk_index)
    ciphertext_with_tag = aead.encrypt(chunk_nonce, plaintext, None)

    fout.write(struct.pack("<I", len(ciphertext_with_tag)))
    fout.write(ciphertext_with_tag)
    chunk_index += 1

# Update chunk count
fout.seek(chunk_count_pos)
fout.write(struct.pack("<I", chunk_index))
```

Fig. 4. Implementasi Logika Perulangan pada Proses Enkripsi

b) Prosedur dekripsi terverifikasi: Prosedur dekripsi pada fungsi `decrypt_file()` dirancang dengan prinsip keamanan ketat yang memastikan integritas data diverifikasi secara granular sebelum dituliskan ke media penyimpanan:

1) Validasi struktur kontainer: Sistem melakukan verifikasi awal terhadap ukuran berkas untuk memastikan keberadaan *header* minimum. Jika valid, sistem mengekstraksi *base nonce* dan nilai `num_chunks` dari posisi awal berkas.
2) Iterasi verifikasi dan dekripsi:

- Sistem melakukan iterasi tepat sebanyak `num_chunks` kali.
- Pada setiap iterasi, sistem membaca 4 byte pertama untuk mengetahui panjang *payload* yang harus diekstraksi.
- Fungsi `aead.decrypt()` dijalankan dengan *nonce* yang diturunkan ulang secara deterministik. Pustaka kriptografi akan mengalkulasi *tag* secara internal dan membandingkannya dengan *tag* yang terlampir pada *ciphertext*. Logika ini ditunjukkan pada Gambar 5.

```
for chunk_index in range(num_chunks):
    # Read chunk length
    len_bytes = fin.read(4)
    chunk_len = struct.unpack("<I", len_bytes)[0]

    # Read ciphertext + derive
    ciphertext_with_tag = fin.read(chunk_len)
    chunk_nonce = _derive_chunk_nonce(base_nonce, chunk_index)

    # Decrypt
    plaintext = aead.decrypt(chunk_nonce, ciphertext_with_tag, None)
    fout.write(plaintext)
```

Fig. 5. Implementasi Logika Perulangan pada Proses Dekripsi

V. PENGUJIAN

A. Hasil Pengujian dan Analisis Metrik

Bagian ini menyajikan data hasil observasi performa algoritma AES-256-GCM dan ChaCha20-Poly1305. Pengujian dilakukan dengan mengukur tiga parameter utama: *execution time*, *throughput*, dan *peak memory usage*.

1) Data hasil pengujian: Tabel I dan Tabel II merangkum perbandingan kinerja pada skenario enkripsi dan dekripsi untuk berbagai ukuran berkas.

TABLE I
HASIL PENGUJIAN SKENARIO ENKRIPSI

Ukuran Berkas	Algoritma	Waktu (s)	Throughput (MB/s)	Memori (MB)
10 MB	AES-256-GCM	0,053	212,95	4,019
	ChaCha20-P1305	0,020	554,85	3,018
500 MB	AES-256-GCM	1,212	416,58	4,019
	ChaCha20-P1305	0,778	648,71	3,018
1 GB	AES-256-GCM	2,606	365,98	4,018
	ChaCha20-P1305	2,264	421,24	3,018

TABLE II
HASIL PENGUJIAN SKENARIO DEKRIPSI

Ukuran Berkas	Algoritma	Waktu (s)	Throughput (MB/s)	Memori (MB)
10 MB	AES-256-GCM	0,038	296,82	4,019
	ChaCha20-P1305	0,026	437,23	3,018
500 MB	AES-256-GCM	1,255	402,19	4,020
	ChaCha20-P1305	0,950	531,20	3,018
1 GB	AES-256-GCM	2,493	382,61	4,020
	ChaCha20-P1305	2,067	461,33	3,018

2) Tren throughput dan waktu komputasi: Berdasarkan hasil observasi, algoritma ChaCha20-Poly1305 secara konsisten menunjukkan *throughput* yang lebih tinggi dibandingkan AES-256-GCM pada seluruh variasi ukuran berkas. Tren data menunjukkan bahwa kecepatan puncak dicapai pada pemrosesan berkas 500 MB, di mana ChaCha20-Poly1305 mencapai 648,71 MB/s. Secara umum, proses dekripsi juga terpantau memiliki efisiensi waktu yang sedikit lebih baik dibandingkan proses enkripsi pada kedua algoritma.

3) Observasi konsumsi memori puncak: Metrik *peak memory usage* menunjukkan stabilitas yang signifikan. Penggunaan memori puncak AES-256-GCM secara konsisten berada pada angka ∼4,02 MB, sementara ChaCha20-Poly1305 berada pada angka ∼3,02 MB. Tidak ditemukan adanya lonjakan konsumsi memori seiring dengan meningkatnya ukuran berkas dari 10 MB hingga 1 GB, yang mengonfirmasi keberhasilan teknik *chunking* yang diterapkan.

B. Analisis Komprehensif dalam Konteks Sistem Multimedia

Setelah meninjau data performa pada sub-bab sebelumnya, bagian ini akan membahas implikasi hasil tersebut terhadap fungsionalitas dan skalabilitas sistem penyimpanan berkas multimedia dalam skenario penggunaan nyata.

1) Signifikansi performa terhadap pengalaman pengguna: Dalam ekosistem multimedia, kecepatan pemrosesan data secara langsung memengaruhi *latency* akses terhadap konten. Berdasarkan hasil pengujian pada berkas 1 GB, ChaCha20-Poly1305 mampu menyelesaikan proses dekripsi dalam waktu kurang lebih 2,067 detik, sementara AES-256-GCM membutuhkan waktu 2,493 detik.

Selisih waktu tersebut memberikan dampak yang signifikan apabila diakumulasikan pada pustaka multimedia yang menyimpan puluhan berkas beresolusi tinggi (seperti video 4K dengan ukuran rata-rata 20–50 GB). Penggunaan ChaCha20-Poly1305 memungkinkan sistem untuk memitigasi efek *buffering* dan memulai proses *playback* video secara lebih instan bagi pengguna akhir.

2) Analisis skalabilitas dan stabilitas sistem: Temuan paling krusial dalam penelitian ini adalah stabilitas penggunaan memori puncak yang konstan pada kisaran ∼3 MB untuk ChaCha20-Poly1305 dan ∼4 MB untuk AES-256-GCM, terlepas dari peningkatan ukuran berkas hingga 1 GB. Hal ini memberikan dua keuntungan strategis:

1) Penanganan berkas masif: Sistem secara teoritis mampu mengenkripsi berkas berukuran sangat besar (orde puluhan hingga ratusan gigabyte) tanpa risiko terjadinya *out of memory*, karena beban RAM diisolasi secara ketat oleh ukuran *chunk* (1 MB) yang didefinisikan pada Bab Implementasi Sistem.
2) Efisiensi multitasking: Konsumsi memori yang rendah dan stabil memungkinkan server penyimpanan atau perangkat *mobile* untuk menjalankan proses kriptografi di latar belakang dengan gangguan minimal terhadap aplikasi multimedia utama lainnya.

3) Keunggulan ChaCha20-Poly1305 pada arsitektur heterogen: Meskipun pengujian dilakukan pada prosesor modern yang mendukung instruksi AES-NI, ChaCha20-Poly1305 tetap menunjukkan keunggulan kecepatan. Hal ini mengindikasikan bahwa ChaCha20-Poly1305 adalah opsi yang lebih konsisten untuk sistem penyimpanan yang bersifat heterogen.

Sistem penyimpanan multimedia yang diakses melalui berbagai perangkat (seperti *smartphone*, tablet, dan PC lama) akan lebih diuntungkan oleh ChaCha20. Hal ini disebabkan tidak semua perangkat tersebut memiliki akselerasi perangkat keras untuk AES yang optimal, sementara ChaCha20 didesain khusus untuk memberikan performa tinggi melalui implementasi perangkat lunak murni tanpa mengandalkan instruksi prosesor yang spesifik.

VI. KESIMPULAN

A. Kesimpulan

Berdasarkan hasil implementasi, pengujian, dan analisis komparatif yang telah dilakukan terhadap algoritma AES-256-GCM dan ChaCha20-Poly1305 pada sistem penyimpanan berkas multimedia, dapat ditarik beberapa kesimpulan utama sebagai berikut:

1) Keunggulan performa komputasi: Algoritma ChaCha20-Poly1305 secara konsisten menunjukkan kinerja yang lebih unggul dibandingkan AES-256-GCM dalam implementasi berbasis perangkat lunak. Meskipun pengujian dilakukan pada prosesor modern yang mendukung instruksi AES-NI, ChaCha20-Poly1305 menghasilkan *throughput* yang lebih tinggi di seluruh skenario ukuran berkas, dengan kecepatan puncak mencapai 648,71 MB/s.
2) Efisiensi dan skalabilitas memori: Penerapan teknik *chunking* terbukti sangat efektif dalam menjaga stabilitas sistem. Penggunaan memori puncak tetap konstan pada kisaran ∼3,02 MB untuk ChaCha20-Poly1305 dan ∼4,02 MB untuk AES-256-GCM, terlepas dari peningkatan ukuran berkas hingga 1 GB. Hal ini membuktikan bahwa sistem memiliki skalabilitas tinggi untuk menangani berkas multimedia berkapasitas besar tanpa risiko kegagalan memori.
3) Optimalisasi sumber daya: ChaCha20-Poly1305 terbukti lebih efisien dengan konsumsi memori yang secara konsisten lebih rendah sebesar 1 MB dibandingkan AES-256-GCM. Karakteristik ini menjadikannya algoritma yang ideal untuk diimplementasikan pada perangkat penyimpanan dengan sumber daya terbatas atau arsitektur sistem yang heterogen.
4) Relevansi terhadap layanan multimedia: Kecepatan dekripsi ChaCha20-Poly1305 yang tinggi (mencapai 461,33 MB/s pada berkas 1 GB) sangat mendukung kebutuhan sistem multimedia modern yang menuntut akses data instan dan *latency* rendah, seperti pada layanan *streaming* video resolusi tinggi.

VII. LAMPIRAN

Seluruh kode sumber dan video demonstrasi pengujian sistem tersedia secara publik pada tautan GitHub dan YouTube berikut :

- Repositori GitHub: https://github.com/ChrisCS50X/Makalah-Kripto
- Video Demonstrasi: https://youtu.be/kTVLovwX-Jw

VIII. UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa, karena atas berkat dan karunia-Nya, makalah ini dapat diselesaikan dengan baik.

Penulis juga ingin menyampaikan terima kasih yang sebesar-besarnya kepada Bapak Dr. Ir. Rinaldi, M.T., selaku dosen pengampu mata kuliah Kriptografi atas ilmu, wawasan, dan bimbingan yang telah beliau berikan. Selain itu, rasa terima kasih penulis persembahkan kepada kedua orang tua, sahabat, dan teman-teman yang senantiasa memberikan dukungan, motivasi, serta doa kepada penulis.

REFERENSI

[1] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC 8439, June 2018.
[2] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)," FIPS Publication 197, November 2001.
[3] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," NIST Special Publication 800-38D, November 2007.
[4] D. J. Bernstein, "ChaCha, a variant of Salsa20," in Workshop Record of SASC, 2008.
[5] P. Rogaway, "Authenticated-encryption with associated-data," in Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS), 2002, pp. 98–107.
[6] S. Gueron, "Intel Advanced Encryption Standard (AES) New Instructions Set," Intel Corporation, 2010.
[7] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography. CRC Press, 1996.
[8] G. Kanda and K. Ryoo, "Design of an Integrated Cryptographic SoC Architecture for Resource-Constrained Devices," International Journal of Electrical and Electronics Research, vol. 10, pp. 230-244, 2022.
[9] V. Sarker, T. Nguyen gia, I. Ben Dhaou, and T. Westerlund, "Smart Parking System with Dynamic Pricing, Edge-Cloud Computing and LoRa," Sensors, vol. 20, no. 17, 4669, 2020.

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

Christian Justin Hendrawan
13522135