Implementasi dan Evaluasi Kinerja Modul Authenticated Encryption untuk Menjamin Integritas Audit Trail

Pradipta Rafa Mahesa and 13522162[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]author@itb.ac.id, [2]author@gmail.com

***Abstract*—Keamanan sistem audit trail sangat krusial untuk forensik digital, namun sering menjadi target serangan log tampering. Penelitian ini bertujuan untuk mengimplementasikan dan mengevaluasi kinerja modul Authenticated Encryption with Associated Data (AEAD) guna menjamin integritas dan otentikasi catatan aktivitas. Modul ini dibangun menggunakan bahasa pemrograman Rust dengan menguji tiga algoritma: AES-256-GCM, ChaCha20-Poly1305, dan metode legacy AES-CBC-HMAC. Evaluasi dilakukan melalui pengujian throughput statis pada dataset 10 MB dan simulasi latensi dinamis pada aliran 100 log real-time. Hasil pengujian menunjukkan bahwa AES-256-GCM memberikan performa tertinggi dengan throughput mencapai 519,89 MB/s dan latensi terendah sebesar 0,86 µs per log berkat dukungan instruksi hardware acceleration. ChaCha20-Poly1305 menunjukkan efisiensi throughput yang hampir menyamai GCM (513,39 MB/s), meskipun memiliki latensi lebih tinggi pada paket data kecil (2,79 µs). Sebaliknya, metode legacy AES-CBC-HMAC ditemukan paling tidak efisien dengan throughput hanya 248,63 MB/s akibat mekanisme pemrosesan dua fase. Penelitian ini menyimpulkan bahwa algoritma AEAD modern secara signifikan lebih unggul dibandingkan metode tradisional untuk mengamankan audit trail pada infrastruktur komputasi modern.**

***Keywords*— AEAD, AES-GCM, Audit Trail, ChaCha20-Poly1305, Rust.**

I. Latar Belakang

Di era transformasi digital saat ini, keamanan informasi telah menjadi fondasi utama dalam menjaga kepercayaan dan keberlangsungan operasional organisasi. Keamanan informasi tidak lagi hanya bergantung pada upaya pencegahan akses ilegal melalui firewall atau sistem deteksi intrusi, tetapi juga sangat bertumpu pada transparansi dan kemampuan sistem untuk melakukan audit terhadap setiap aktivitas yang telah terjadi. Catatan aktivitas atau audit trail yang akurat dan andal merupakan komponen kritikal, baik untuk pemenuhan kepatuhan regulasi (seperti GDPR atau ISO 27001) maupun sebagai instrumen utama dalam investigasi forensik digital pasca-insiden.

Namun, efektivitas dari sistem audit ini sering kali terancam oleh manipulasi data. Masalah utama yang sering dihadapi dalam manajemen log adalah serangan log tampering, di mana aktor ancaman yang berhasil menyusup ke dalam sistem akan berupaya menghapus, memodifikasi, atau menyuntikkan data palsu ke dalam catatan aktivitas guna menghilangkan jejak kejahatan mereka. Jika integritas log tidak terjaga, organisasi tidak hanya kehilangan kemampuan untuk melacak sumber serangan, tetapi juga berisiko menghadapi konsekuensi hukum akibat data audit yang tidak valid.

Secara tradisional, perlindungan log sering kali hanya berfokus pada enkripsi untuk menjaga kerahasiaan. Namun, enkripsi standar saja tidak cukup; diperlukan mekanisme yang secara simultan menjamin bahwa data tidak diubah (integritas) dan dipastikan berasal dari sumber yang sah (otentikasi). Pendekatan modern yang paling efektif untuk mengatasi tantangan ini adalah melalui skema Authenticated Encryption with Associated Data (AEAD). AEAD memungkinkan pengamanan konten log sekaligus melindungi metadata terkait, seperti timestamp dan identitas perangkat, tanpa harus mengenkripsi metadata tersebut sehingga tetap memudahkan proses pengindeksan.

Selain pemilihan algoritma, efisiensi dan keamanan memori dari implementasi perangkat lunak juga menjadi faktor penentu. Banyak kerentanan sistem audit berakar pada masalah manajemen memori seperti buffer overflow. Oleh karena itu, penelitian ini menggunakan bahasa pemrograman Rust yang menawarkan performa tinggi setara C++ namun dengan jaminan keamanan memori di tingkat kompilasi. Makalah ini akan mengevaluasi implementasi berbagai modul AEAD dalam ekosistem Rust untuk menentukan solusi yang paling optimal dalam menjamin integritas audit trail dengan dampak minimal terhadap performa sistem.

II. Landasan Teori

A. Authenticated Encryption with Associated Data (AEAD)

Konsep Authenticated Encryption with Associated Data

(AEAD) secara formal diperkenalkan dan didefinisikan oleh Phillip Rogaway pada tahun 2002 sebagai solusi atas kebutuhan kriptografi untuk mengamankan pesan sekaligus mengautentikasi informasi tambahan yang tidak terenkripsi [1]. Dalam makalahnya, Rogaway menjelaskan bahwa banyak protokol komunikasi membutuhkan pengiriman data yang terdiri dari bagian rahasia (ciphertext) dan bagian publik (header atau associated data). Masalah utama yang diangkat adalah bagaimana menjamin bahwa associated data tersebut tetap terikat secara aman dengan ciphertext, sehingga penyerang tidak dapat mengubah metadata (seperti alamat IP atau nomor urut) tanpa merusak validitas seluruh paket data.

Secara teknis, AEAD merupakan skema enkripsi simetris yang menghasilkan tag autentikasi berdasarkan kunci rahasia, nonce, data teks terang (plaintext), dan data terkait (associated data). Rogaway merumuskan model keamanan di mana musuh (adversary) memiliki kendali penuh terhadap associated data, namun tetap tidak mampu memalsukan tag yang valid atau mendapatkan informasi tentang isi plaintext. Dalam konteks audit trail, skema ini sangat ideal karena memungkinkan penyimpanan log yang isinya terenkripsi demi kerahasiaan, sementara metadata log seperti timestamp atau kategori log tetap dapat dibaca tanpa enkripsi namun tetap terlindungi dari segala bentuk manipulasi integritas.

B. AES-GCM

AES-GCM (Galois/Counter Mode) adalah algoritma enkripsi terautentikasi (AEAD) yang menggabungkan dua primitif kriptografi dalam satu tahap operasi: mode Counter (CTR) untuk menjamin kerahasiaan data dan fungsi GHASH pada Galois field GF(2^128) untuk menjamin integritas. Dalam mekanisme kerjanya, AES-GCM menggunakan nilai nonce yang unik untuk setiap entri log guna menghasilkan aliran kunci (keystream) yang kemudian di-XOR dengan teks terang (plaintext). Secara simultan, hasil enkripsi tersebut diproses melalui fungsi GHASH untuk menghasilkan sebuah tag autentikasi berukuran 128-bit. Keberadaan tag ini memastikan bahwa pesan tidak mengalami modifikasi sedikit pun, karena setiap perubahan pada ciphertext akan menyebabkan kegagalan verifikasi saat proses dekripsi dilakukan.

Keunggulan utama dari AES-GCM terletak pada efisiensi komputasinya yang sangat tinggi karena mendukung pemrosesan secara paralel secara penuh. Berbeda dengan mode blok berantai tradisional yang bersifat sekuensial, setiap blok data dalam GCM dapat dienkripsi secara independen, sehingga memungkinkan utilisasi maksimal pada arsitektur prosesor modern. Selain itu, efisiensi ini semakin optimal pada CPU yang mendukung instruksi tingkat perangkat keras seperti AES-NI (Intel/AMD) dan CLMUL, yang secara khusus mempercepat operasi perkalian Galois field. Karakteristik ini menjadikan AES-GCM sebagai solusi ideal untuk mengamankan modul audit trail yang membutuhkan throughput data besar dengan latensi minimal, sehingga tidak mengganggu performa aplikasi utama saat melakukan pencatatan log secara real-time [2].

C. ChaCha20-Poly1305

ChaCha20-Poly1305 adalah skema AEAD yang menggabungkan stream cipher ChaCha20 dengan kode otentikasi pesan Poly1305. Berdasarkan analisis keamanan modern, skema ini dirancang untuk memberikan keamanan yang kuat bahkan dalam skenario multi-pengguna yang kompleks melalui penggunaan kunci 256-bit dan nonce 96-bit [3]. ChaCha20 beroperasi dengan serangkaian fungsi aritmatika Addition-Rotate-XOR (ARX) pada blok berukuran 512-bit untuk menghasilkan aliran kunci, sementara Poly1305 bertugas menghitung tag autentikasi untuk memastikan bahwa data terenkripsi dan metadata terkait tetap sinkron dan terlindungi dari segala bentuk pemalsuan.

Dari sisi performa, ChaCha20-Poly1305 sangat unggul dalam implementasi perangkat lunak murni, terutama pada perangkat yang tidak memiliki akselerasi perangkat keras khusus untuk AES. Desain berbasis ARX memungkinkan algoritma ini berjalan dengan waktu eksekusi yang konstan (constant-time), yang secara alami memberikan perlindungan terhadap serangan saluran samping berbasis waktu (timing attacks). Hal ini menjadikannya solusi ideal untuk mengamankan log audit pada perangkat seluler, sistem IoT, atau infrastruktur server lama, karena mampu memberikan kecepatan enkripsi yang konsisten dengan beban komputasi yang minimal tanpa mengorbankan integritas data [3].

C. Legacy AES-CBC-HMAC

Metode legacy AES-CBC-HMAC adalah sebuah skema kriptografi komposit yang menggabungkan tiga komponen utama: Advanced Encryption Standard (AES) sebagai algoritma enkripsi simetris, Cipher Block Chaining (CBC) sebagai mode operasi, dan Hash-based Message Authentication Code (HMAC) sebagai mekanisme integritas. AES berperan sebagai blok cipher dasar yang mengubah data log menjadi teks rahasia, sementara mode CBC memastikan bahwa setiap blok teks terang di-XOR dengan blok teks sandi sebelumnya sebelum dienkripsi. Untuk melengkapi aspek keamanan, HMAC ditambahkan menggunakan fungsi hash (seperti SHA-256) dan kunci rahasia guna menghasilkan nilai ringkasan yang menjamin bahwa data tersebut tidak dimanipulasi selama proses penyimpanan [4].

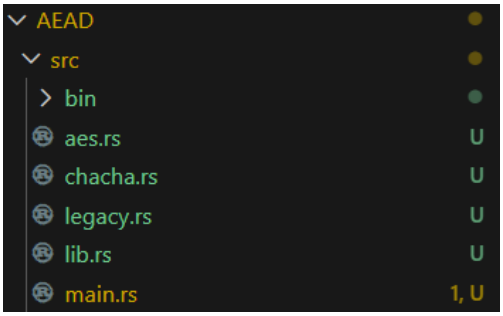
Penggabungan ketiga komponen ini dilakukan untuk mencapai tingkat keamanan yang kini dikenal sebagai Authenticated Encryption. Peran masing-masing sangat spesifik: AES dalam mode CBC bertanggung jawab penuh atas aspek kerahasiaan (confidentiality) agar isi log tidak dapat dibaca oleh pihak yang tidak berwenang, sedangkan HMAC memegang peranan krusial dalam menjamin integritas (integrity) dan otentikasi (authenticity) pesan. Tanpa HMAC, mode CBC murni sangat rentan terhadap serangan modifikasi bit yang dapat mengubah makna log tanpa terdeteksi. Meskipun kombinasi ini efektif dalam memberikan perlindungan menyeluruh, ia dianggap sebagai metode legacy karena membutuhkan dua kunci

berbeda dan dua fase pemrosesan yang terpisah, sehingga efisiensinya lebih rendah dibandingkan algoritma AEAD modern yang bersifat single-pass [4].

III. IMPLEMENTASI

A. Lingkungan Pengembangan dan Abstraksi

Sistem keamanan log audit ini diimplementasikan menggunakan bahasa pemrograman Rust (edisi 2021). Pemilihan Rust didasarkan pada kebutuhan sistem audit yang memerlukan performa tinggi serta keamanan memori yang ketat untuk mencegah celah keamanan seperti buffer overflow. Arsitektur perangkat lunak dirancang secara modular, di mana logika utama dipisahkan ke dalam beberapa modul independen yang dikoordinasikan melalui berkas lib.rs.



Abstraksi tingkat tinggi dilakukan dengan mendefinisikan trait AuditEncryptor. Trait ini berfungsi sebagai kontrak atau antarmuka (interface) yang memastikan setiap algoritma memiliki metode yang konsisten untuk proses enkripsi dan dekripsi. Dengan menggunakan polymorphism, sistem dapat memanggil berbagai algoritma tanpa mengubah logika pada aplikasi utama.

B. Implementasi Modul Kriptografi

Setiap algoritma diimplementasikan dalam berkas terpisah yang mengacu pada trait AuditEncryptor. Berikut adalah detail implementasi untuk masing-masing modul:

1). Implementasi AES-256-GCM:

Modul AesGcmModule memanfaatkan pustaka aes-gcm untuk menyediakan enkripsi terautentikasi. Dalam metode encrypt_log, sistem membangkitkan nonce 12-byte secara acak menggunakan OsRng sebelum melakukan enkripsi. Hasil akhir dari proses ini adalah sebuah vektor byte yang menggabungkan nonce di bagian awal diikuti oleh ciphertext, sehingga memudahkan modul dekripsi untuk mengekstraksi nilai nonce yang sama guna melakukan validasi integritas dan dekripsi data.

2). Implementasi ChaCha20-Poly1305:

Implementasi pada modul `ChaChaModule` memiliki kemiripan struktur dengan GCM, namun menggunakan algoritma *stream cipher* ChaCha20 dan otentikator Poly1305. Kode program menunjukkan penggunaan kunci 256-bit dan penanganan *error* yang ketat; jika proses dekripsi mendeteksi adanya ketidakcocokan pada *tag* autentikasi, fungsi akan mengembalikan pesan kesalahan. Hal ini menjamin bahwa log audit yang telah dimanipulasi tidak akan pernah bisa didekripsi ke dalam bentuk teks.

3). Implementasi AES-CBC-HMAC

Berbeda dengan dua modul sebelumnya, modul

AesCbcHmacModule mengimplementasikan skema Encrypt-then-MAC secara manual. Kode program memperlihatkan alur kerja di mana data pertama-tama diberikan padding PKCS7 agar sesuai dengan ukuran blok AES, kemudian dienkripsi menggunakan mode CBC. Setelah ciphertext terbentuk, sistem menghitung nilai HMAC-SHA256 yang mencakup IV dan seluruh ciphertext. Saat proses dekripsi, kode secara eksplisit melakukan verifikasi HMAC terlebih dahulu (mac.verify_slice) sebelum melakukan dekripsi blok, sesuai dengan praktik terbaik keamanan untuk mencegah serangan padding oracle.

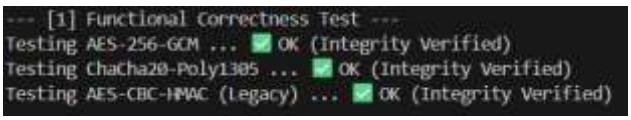
IV. Hasil Pengujian Dan Analisis

A. Skenario

Evaluasi dilakukan untuk membandingkan performa tiga algoritma dalam mengamankan data *audit trail*. Pengujian menggunakan dua skenario utama: (1) *Static Throughput Test* menggunakan dataset log sebesar 10 MB untuk menyimulasi pengarsipan log massal, dan (2) *Dynamic Latency Simulation* menggunakan aliran 100 log acak untuk menyimulasi beban kerja sistem *real-time*. Seluruh pengujian dijalankan pada mode *release* menggunakan bahasa Rust untuk mendapatkan hasil optimasi kompilator yang maksimal.

B. Uji Kebenaran Fungsional (Correctness Test)

Tahap awal pengujian dilakukan melalui *sanity check* untuk memastikan integritas data. Berdasarkan hasil keluaran sistem, ketiga algoritma (AES-256-GCM, ChaCha20-Poly1305, dan AES-CBC-HMAC) berhasil melakukan proses enkripsi dan dekripsi dengan sempurna (Tanda Centang dan message OK). Hal ini membuktikan bahwa setiap algoritma mampu menjaga integritas pesan dan memvalidasi bahwa data tidak mengalami perubahan (*tampering*) selama proses penyimpanan.



C. Analisis Throughput (Data Statis)

Pengujian *throughput* bertujuan untuk melihat seberapa cepat algoritma memproses data log dalam volume besar. Hasil pengujian ditunjukkan pada Tabel I.

Tabel I Hasil Pengujian Throughput

Algoritma	Rata-rata Waktu (detik)	Throughput (MB/s)
AES-GCM	0,01923 s	519,89 MB/s
ChaCha20-Poly1305	0,01948 s	513,39 MB/s
AES-CBC-HMAC (Legacy)	0,04022 s	248,63 MB/s

Berdasarkan Tabel I, AES-256-GCM menunjukkan performa tertinggi dengan kecepatan mencapai 519,89 MB/s, disusul sangat ketat oleh ChaCha20-Poly1305. Kecepatan tinggi pada kedua algoritma ini dikarenakan desain AEAD yang bersifat single-pass, di mana proses enkripsi dan autentikasi dilakukan secara simultan. Sementara itu, AES-CBC-HMAC memiliki throughput paling rendah (hanya sekitar 47,8% dari kecepatan GCM). Hal ini disebabkan oleh mekanisme two-pass (Encrypt-then-MAC) yang secara teoritis memerlukan waktu pemrosesan dua kali lebih lama karena harus memindai data secara sekuensial untuk enkripsi dan menghitung hash HMAC secara terpisah.

C. Analisis Latensi (Simulasi Real-time)

Skenario kedua mengukur latensi rata-rata yang dibutuhkan untuk mengamankan satu entri log pada aliran data yang cepat. Hasilnya dirangkum dalam Tabel II.

Tabel II Hasil Pengujian Latensi

Algoritma	Rata-rata Waktu (detik)
AES-GCM	0,86 µs
ChaCha20-Poly1305	1,47 µs
AES-CBC-HMAC (Legacy)	1,47 µss

Hasil latensi menunjukkan fenomena yang menarik. AES-256-GCM mencatatkan waktu tercepat yaitu 0,86 µs per log. Rendahnya latensi ini mengindikasikan bahwa prosesor yang digunakan kemungkinan besar memiliki dukungan instruksi hardware acceleration (seperti AES-NI) yang sangat efisien dalam menangani enkripsi blok kecil.

Hal yang menonjol adalah ChaCha20-Poly1305 memiliki latensi tertinggi (2,79 µs) pada pengujian data kecil ini. Meskipun ChaCha20 sangat cepat pada data besar, namun pada implementasi software-based untuk paket data yang sangat kecil, terdapat overhead inisialisasi state algoritma yang sedikit lebih besar dibandingkan AES yang dibantu oleh perangkat keras. Namun, semua algoritma masih berada di bawah ambang batas 3 µs, yang berarti tidak akan memberikan beban signifikan pada kinerja aplikasi utama saat melakukan pencatatan log secara real-time.

V. KESIMPULAN

Secara keseluruhan, algoritma AEAD modern (GCM dan ChaCha20) terbukti jauh lebih efisien untuk sistem *audit trail* dibandingkan metode *legacy*. AES-256-GCM adalah pilihan terbaik untuk infrastruktur server modern yang memiliki akselerasi perangkat keras karena memberikan keseimbangan terbaik antara *throughput* tinggi dan latensi rendah. Di sisi lain, ChaCha20-Poly1305 tetap menjadi alternatif yang sangat kuat dengan stabilitas *throughput* yang hampir menyamai GCM, menjadikannya pilihan ideal untuk sistem terdistribusi atau perangkat seluler yang tidak memiliki unit perangkat keras khusus AES.

VI. SARAN

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, terdapat beberapa saran untuk pengembangan penelitian selanjutnya guna menutupi celah teknis yang ada:

1. Pengujian dengan Dataset Log Riil

   Pengujian dalam penelitian ini masih menggunakan data dummy statis yang dibangkitkan langsung melalui fungsi main. Untuk meningkatkan validitas hasil, penelitian selanjutnya disarankan menggunakan dataset log dari sistem riil (seperti web server logs atau system events) yang memiliki variasi panjang karakter dan entropi data yang lebih kompleks, sehingga performa algoritma dapat diukur dalam skenario beban kerja yang lebih alami.

2. Integrasi dengan Media Penyimpanan Persisten

   Implementasi saat ini baru mencakup proses kriptografi di level memori. Disarankan untuk mengintegrasikan modul ini dengan sistem penyimpanan data persisten seperti database (PostgreSQL, SQLite, atau NoSQL). Hal ini penting untuk mengevaluasi dampak I/O overhead saat data log yang telah dienkripsi ditulis ke dalam disk, serta untuk menguji efisiensi indeks pada metadata log yang tetap dibiarkan dalam bentuk teks terang (Additional Authenticated Data).

3. Perluasan Implementasi untuk Keamanan Transport

   Penelitian ini difokuskan pada perlindungan data saat diam (encryption at rest). Saran untuk pengembangan berikutnya adalah menerapkan algoritma AEAD ini untuk mengamankan data log saat dikirimkan melalui jaringan (encryption in transit), misalnya dari client agent ke server log terpusat. Dengan demikian, efisiensi algoritma dapat diuji dalam menangani network latency dan memastikan integritas data log tetap terjaga dari ancaman man-in-the-middle attack.

REFERENCES

[1] P. Rogaway, "Authenticated-encryption with associated-data," in Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS '02), Washington, DC, USA, 2002, pp. 98–107. doi: 10.1145/586110.586125.

[2] S. Gueron and L. Krasnov, "Practical Challenges with AES-GCM," in Third Workshop on Block Cipher Modes of Operation, National Institute of Standards and Technology (NIST), 2023. https://csrc.nist.gov/csrc/media/Events/2023/third-workshop-on-block-cipher-modes-of-operation/documents/accepted-papers/Practical%20Challenges%20with%20AES-GCM.pdf

[3] J. P. Degabriele, J. Govinden, G. Gunner, dan K. G. Paterson, "The Security of ChaCha20-Poly1305 in the Multi-User Setting," dalam *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*, 2021, hlm. 1981–2003. doi: 10.1145/3460120.3484814.

[4] R. M. S. El-Din, "Performance Evaluation of AES-CBC and AES-GCM Algorithms for Data Security," *Earthline Journal of Mathematical Sciences*, vol. 14, no. 1, pp. 11-20, 2024. https://earthlinepublishers.com/index.php/ejms/article/view/1110

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

Pradipta Rafa Mahesa 13522162