Optimalisasi *Encryption-Decryption Database* Aplikasi menggunakan *Algoritma Advanced Encryption Standard* (AES) dengan *Mode Cipher Block Chaining* (CBC)

Riko Putro Nugroho[1]*, Umaisaroh[2]
[1-2]Program Studi Magister Teknik Elektro, Fakultas Teknik, Universitas Mercu Buana, Jakarta, Indonesia

Corresponding Author's e-mail: *55423120005@student.mercubuana.ac.id*

ARMADA
JURNAL PENELITIAN MULTIDISIPLIN

e-ISSN: 2964-2981
ARMADA : Jurnal Penelitian Multidisiplin
https://ejournal.45mataram.ac.id/index.php/armada
Vol. 04, No. 10 Oktober, 2026
Page: 7325-7339

DOI:
https://doi.org/10.55681/armada.v4i10.4653

Article History:
Received: Agustus 05, 2026
Revised: September 13, 2026
Accepted: Oktober 19, 2026

Abstract *: Data security is essential in web applications connected to databases, as threats of data leakage and misuse increase. This study implements the Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode within a Laravel application. Data are encrypted before database storage, while decryption is restricted according to users' access rights. Performance is optimised using Laravel Queue for asynchronous and batch encryption, Redis for repeated decryption, and AES-NI hardware instruction monitoring as performance metadata. Tests show that Redis accelerates repeated decryption by 93.4%, reducing time from 14.288 ms to 0.945 ms. Laravel Queue reduces end-to-end HTTP response time for batch encryption by 96.8%, from 1,397.690 ms to 44.281 ms. Across 30 security-test repetitions, the avalanche effect remains at 50.0%, while ciphertext entropy approaches the ideal value of 8 bits. Encryption overhead is relatively low, with time overhead of 4.3% and measured memory overhead close to 0%. These findings show that the optimisations improve processing efficiency without weakening AES-CBC cryptographic properties.*

Keywords: *AES-CBC, Encryption, Decryption, Laravel, Optimization, Queue, Cache, Entropy, Avalanche Effect, Data Security.*

Abstrak : Keamanan data menjadi aspek penting dalam pengembangan aplikasi web yang terhubung dengan database, seiring meningkatnya ancaman kebocoran dan penyalahgunaan informasi. Penelitian ini menerapkan algoritma Advanced Encryption Standard (AES) dengan mode Cipher Block Chaining (CBC) pada aplikasi berbasis Laravel untuk mengenkripsi data sebelum disimpan ke database dan membatasi dekripsi berdasarkan hak akses pengguna. Optimasi performa dilakukan melalui Laravel Queue untuk enkripsi asynchronous dan batch, Redis untuk mempercepat dekripsi berulang, serta pencatatan instruksi hardware AES-NI sebagai metadata performa. Hasil pengujian menunjukkan bahwa Redis mempercepat dekripsi berulang sebesar 93,4%, dari 14,288 ms menjadi 0,945 ms. Laravel Queue menurunkan waktu respons HTTP end-to-end untuk enkripsi batch sebesar 96,8%, dari 1.397,690 ms menjadi 44,281 ms. Pengujian keamanan sebanyak 30 pengulangan menghasilkan avalanche effect konsisten sebesar 50,0% dan entropy ciphertext mendekati nilai ideal 8 bit. Overhead komputasi enkripsi juga relatif rendah, dengan rata-rata overhead waktu 4,3% dan overhead memori mendekati 0%. Hasil ini menunjukkan bahwa optimasi yang

diterapkan meningkatkan efisiensi pemrosesan tanpa menurunkan karakteristik kriptografis AES-CBC.

Kata Kunci: AES-CBC, Enkripsi, Dekripsi, Laravel, Optimasi, *Queue*, *Cache*, *Entropy*, *Avalanche Effect*, Keamanan Data.

PENDAHULUAN

Perkembangan teknologi informasi di era digital telah meningkatkan jumlah data yang tersimpan dan dipertukarkan secara masif, yang sebagian besar disimpan dan dikelola melalui sistem *database* sebagai komponen inti pada hampir seluruh aplikasi modern. *Database* berperan sebagai pusat penyimpanan informasi strategis sehingga menjadi sasaran utama ancaman keamanan. Data menjadi aset strategis bagi organisasi publik maupun privat, sehingga ancaman kebocoran dan penyalahgunaan data semakin meningkat. Kasus kebocoran data baik di tingkat global maupun nasional berdampak serius terhadap kerugian finansial, kepercayaan publik, serta reputasi institusi (IBM Security, 2023). Untuk mengatasi hal tersebut, enkripsi data merupakan salah satu teknik utama dalam menjaga kerahasiaan dan integritas data sebelum disimpan ke dalam *database*, sedangkan proses dekripsi diperlukan setiap kali data tersebut perlu diakses atau dibaca kembali oleh pengguna yang berwenang. Namun, implementasi kedua proses tersebut yang tidak efisien sering menimbulkan masalah performance, seperti meningkatnya latency atau penurunan throughput sistem. Kondisi ini menegaskan adanya tantangan untuk mencapai keseimbangan antara keamanan data dan performa sistem secara bersamaan. Oleh karena itu, optimalisasi algoritma enkripsi dan dekripsi pada sistem *database* menjadi isu penting.

Penelitian terkini menunjukkan bahwa performa enkripsi sangat dipengaruhi oleh parameter teknis seperti panjang *key*, ukuran *block*, dan mode operasi. Andersson (2022) menemukan bahwa variasi parameter Advanced Encryption Standard (AES) berdampak signifikan pada *throughput* dan *response time database*. Penelitian lain juga menekankan bahwa mode Cipher Block Chaining (CBC) pada AES perlu dioptimalkan untuk efisiensi pengolahan data skala besar, khususnya dalam lingkungan *real-time* (W. Li et al., 2023). Dalam bidang sistem informasi, aplikasi berbasis web dengan *database relational* (misalnya MySQL atau PostgreSQL) kerap menjadi target serangan *cyber*, seperti *SQL Injection* maupun akses ilegal. *Framework* Laravel sebagai salah satu web *framework* populer di Indonesia banyak digunakan untuk membangun aplikasi skala kecil hingga besar. Maka, aspek keamanan *database* dalam konteks Laravel menjadi relevan untuk dikaji. Salah satu pendekatan yang dapat dilakukan adalah penerapan enkripsi pada data "at rest". Algoritma AES-CBC sering dipilih karena tingkat keamanan dan efisiensi yang relatif tinggi dibanding algoritma *symmetric* lain (Stallings, 2023). Namun, integrasi AES-CBC ke aplikasi berbasis Laravel perlu dioptimalkan agar performa sistem tidak terganggu.

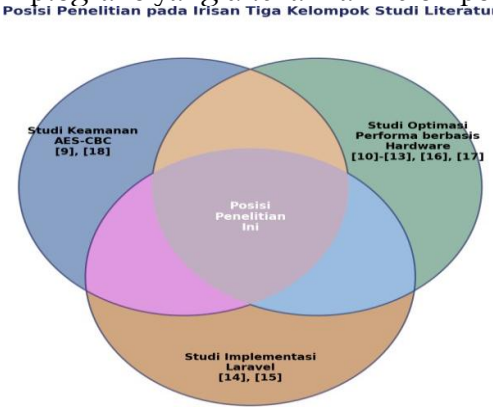
Di Indonesia, penelitian terkait implementasi AES masih terbatas pada studi aplikasi sederhana, seperti sistem registrasi pasien atau aplikasi arsip, yang fokus pada aspek keamanan dasar tanpa mengeksplorasi optimalisasi performa secara kuantitatif (Rahmawati & Wibowo, 2022; Nugraha, 2023). Fakta ini menunjukkan adanya kesenjangan antara kebutuhan aplikasi modern berskala besar dan penelitian yang tersedia. Kesenjangan ini memperkuat urgensi penelitian yang secara khusus mengukur dampak optimasi performa pada konteks aplikasi Laravel berskala menengah hingga besar. AES merupakan algoritma *block cipher symmetric* yang ditetapkan oleh NIST sebagai standar sejak 2001. Mode operasi CBC dianggap lebih aman dibandingkan ECB karena mampu menghasilkan kerahasiaan yang lebih kuat melalui keterkaitan antarblok. Namun, CBC memiliki kelemahan dari sisi paralelisasi sehingga berpotensi menimbulkan *overhead performance* (Kudelski Security Research, 2021). Penelitian lain menunjukkan bahwa pemanfaatan akselerasi perangkat keras, seperti GPU, dapat mempercepat enkripsi AES-CBC pada aplikasi *database* (Chen et al., 2012).

Di Indonesia, penelitian implementasi AES umumnya menitikberatkan pada confidentiality data tetapi jarang membahas aspek optimalisasi performa, khususnya dalam konteks *framework* Laravel (Rahmawati & Wibowo, 2022; Nugraha, 2023). Padahal, Laravel merupakan salah satu framework PHP paling populer di Indonesia untuk pengembangan aplikasi berskala menengah hingga besar, sehingga kebutuhan akan panduan optimasi yang spesifik dan teruji secara empiris

menjadi semakin mendesak. Kesenjangan ini memperkuat urgensi penelitian yang secara khusus mengukur dampak optimasi performa pada konteks aplikasi Laravel berskala menengah hingga besar.

Meskipun sejumlah penelitian terdahulu telah mengkaji optimalisasi AES-CBC, terdapat perbedaan fokus variabel yang jelas antar kelompok studi. Kelompok pertama berfokus pada penguatan aspek keamanan melalui modifikasi struktural, seperti IV dinamis dan rotasi *key* (Assafli & Hashim, 2020; Lee & Sim, 2021), namun tidak mengukur dampaknya terhadap *throughput* aplikasi secara kuantitatif dalam konteks web *framework*. Kelompok kedua berfokus pada akselerasi performa melalui *hardware* seperti GPU dan FPGA (Lee et al., 2022; Nannipieri et al., 2022; Yang, 2025), yang efektif namun membutuhkan infrastruktur khusus yang umumnya tidak tersedia pada server aplikasi web skala kecil-menengah. Kelompok ketiga mengkaji implementasi AES-CBC pada *framework* Laravel (A. Pratama & Nugroho, 2021), namun terbatas pada pembuktian bahwa fitur tersebut dapat berjalan (*feasibility*), tanpa mengeksplorasi strategi optimasi performa secara terukur maupun perbandingan kuantitatif dengan *baseline* yang eksplisit. Studi paling mutakhir yang ditemukan adalah Carvalho, Sá, dan Bernardino (2025), yang mengevaluasi dampak enkripsi AES (melalui mekanisme *Transparent Data Encryption*/TDE) terhadap performa *query read* pada tiga sistem manajemen *database* (Oracle, MySQL, dan SQL Server) menggunakan *benchmark* standar TPC-H pada berbagai skala data (Carvalho et al., 2025). Studi tersebut menegaskan bahwa TDE menimbulkan *overhead* performa yang bervariasi tergantung DBMS dan skala data yang digunakan, namun berfokus pada enkripsi di level *database engine* (TDE), berbeda dengan penelitian ini yang berfokus pada *application-side encryption* dan optimasi di level aplikasi web (*queue job processing* dan *caching*).

Penelitian ini membedakan diri dari ketiga kelompok tersebut dengan memosisikan variabel independen berupa kombinasi teknik optimasi pada level aplikasi (*queue job* processing, Redis *caching*, *batch* encryption, dan pemanfaatan instruksi AES-NI) yang diterapkan secara bersamaan pada implementasi AES-CBC standar di Laravel. Variabel dependen yang diukur mencakup waktu eksekusi (*execution time*), *throughput*, penggunaan CPU dan memori, serta indikator keamanan (*entropy* dan *avalanche effect*) sebagai kontrol agar peningkatan performa tidak menurunkan kekuatan kriptografis. Dengan demikian, penelitian ini mengisi kesenjangan (gap) pada irisan antara studi keamanan AES-CBC, studi optimasi performa berbasis *hardware*, dan studi implementasi Laravel, yaitu optimasi performa AES-CBC yang terukur secara kuantitatif dan tervalidasi keamanannya, namun hanya menggunakan sumber daya *software*/CPU umum yang tersedia pada server aplikasi web pada umumnya, tanpa akselerasi *hardware* khusus. Posisi penelitian ini dapat digambarkan secara visual melalui irisan tiga kelompok studi literatur yang telah diuraikan sebelumnya, sebagaimana ditunjukkan pada Gambar 1.1 berikut. Penelitian ini berada pada titik temu ketiganya: menerapkan optimasi performa yang menjadi sorotan kelompok kedua, pada implementasi Laravel yang menjadi fokus kelompok ketiga, sekaligus tetap memvalidasi agar keamanan kriptografis yang ditekankan kelompok pertama tidak menurun:



Gambar 1. Posisi Penelitian pada Irisan Tiga Kelompok

Studi Literatur Berbagai persoalan yang telah diuraikan di atas — mulai dari ancaman kebocoran data yang terus meningkat, tuntutan performa sistem yang tidak boleh terabaikan, hingga kesenjangan penelitian pada konteks aplikasi Laravel yang belum banyak dikaji secara kuantitatif menjadi dasar pijakan bagi arah penelitian ini. Bertolak dari hal tersebut, penelitian ini mengangkat judul "Optimalisasi Encryption-Decryption Database Aplikasi Menggunakan Algoritma Advanced Encryption Standard (AES) dengan Mode Cipher Block Chaining (CBC)", sebagai upaya menjawab kebutuhan nyata akan keseimbangan antara keamanan data dan performa sistem yang hingga kini belum banyak dieksplorasi secara mendalam pada aplikasi berbasis Laravel.

METODE PENELITIAN

Metode penelitian ini menggunakan data primer berupa data sintetis digital yang dibuat melalui *script* dengan berbagai format dan ukuran file, seperti *.docx, .txt, .csv,* dan *.pdf*, untuk menguji proses *encryption-decryption* menggunakan algoritma AES-CBC pada aplikasi berbasis Laravel. Data diolah melalui tahapan *import*, enkripsi dengan *initialization vector* (IV) acak, penyimpanan *ciphertext* dan metadata, dekripsi, serta validasi integritas data. Tahapan penelitian meliputi analisis kebutuhan, perancangan arsitektur dan *database*, implementasi AES-CBC, optimalisasi menggunakan *queue job processing*, Redis *caching*, *batch encryption*, dan pemanfaatan AES-NI, kemudian dilanjutkan dengan pengujian dan analisis secara iteratif. Analisis dilakukan secara kuantitatif dan eksperimental berdasarkan waktu proses, *throughput*, penggunaan CPU dan memori, serta ukuran file, disertai analisis keamanan melalui *Shannon Entropy*, *avalanche effect*, pengujian *brute-force* dan *known-plaintext attack*, serta validasi menggunakan SHA-256. Efektivitas sistem dinilai berdasarkan keberhasilan enkripsi-dekripsi, peningkatan *performance*, dan stabilitas dalam menangani berbagai ukuran file untuk menentukan kelayakan penerapan sistem pada lingkungan pemerintahan. Markdown yang ditempel (1)

HASIL DAN PEMBAHASAN

Implementasi Sistem (Encryption-decryption di Laravel)

Layanan Kriptografi *(AesCbcService)*

AesCbcService merupakan kelas utama yang mengelola seluruh proses enkripsi dan dekripsi menggunakan algoritma AES-256-CBC melalui ekstensi OpenSSL bawaan PHP. Kelas ini memiliki lima fungsi utama, yaitu menghasilkan Initialization Vector (IV) sepanjang 16 byte secara acak untuk setiap operasi enkripsi menggunakan fungsi random_bytes(), menjalankan proses enkripsi dan dekripsi melalui fungsi openssl_encrypt () dan openssl_decrypt () dengan mode OPENSSL_RAW_DATA, menghitung nilai hash SHA-256 dari plaintext untuk keperluan validasi integritas data, menghitung nilai Shannon Entropy dari ciphertext, serta mengukur Avalanche Effect dengan mengubah satu bit pertama pada plaintext, kemudian membandingkan jarak Hamming antara dua ciphertext yang dihasilkan. Dengan kelima fungsi tersebut, AesCbcService menjadi komponen utama dalam pengelolaan proses kriptografi sekaligus pengujian karakteristik keamanan algoritma AES-256-CBC.

Sebagai langkah pengamanan tambahan, constructor AesCbcService memvalidasi bahwa AES_MASTER_KEY yang dikonfigurasi di file .env benar-benar berupa 32 byte (256-bit) hasil decode base64. Jika *key* tidak valid atau kosong, sistem akan menghentikan proses dengan *exception* secara eksplisit, alih-alih diam-diam melanjutkan proses enkripsi dengan *key* yang lemah/kosong. Validasi ini penting untuk mencegah kesalahan konfigurasi yang dapat menyebabkan kegagalan enkripsi secara diam-diam (silent failure), yang berpotensi menimbulkan risiko keamanan data yang lebih besar jika tidak terdeteksi sejak awal.

Optimasi *Queue Job & Batch Encryption*

Proses enkripsi tidak lagi dijalankan secara synchronous di dalam siklus request-response HTTP, tetapi dialihkan ke Laravel Queue melalui kelas EncryptFileJob yang mengimplementasikan interface ShouldQueue. Ketika pengguna mengunggah satu atau beberapa file sekaligus (batch encryption) melalui formulir yang mendukung fitur multiple file input, EncryptedDataController

menjalankan tiga tahapan utama, yaitu menyimpan setiap file ke direktori sementara, membuat entri baru pada tabel encrypted_data dengan status queued, serta melakukan dispatch satu EncryptFileJob secara terpisah untuk setiap file. Dengan mekanisme ini, setiap proses enkripsi dijalankan sebagai pekerjaan terpisah melalui sistem antrean, sehingga proses pengunggahan file tidak harus menunggu seluruh operasi enkripsi selesai dalam siklus permintaan HTTP.

Queue worker yang berjalan sebagai proses/kontainer terpisah kemudian mengambil *job* satu per satu, mengeksekusi AesCbcService::encrypt(), menyimpan hasilnya, dan memperbarui status menjadi completed atau *failed*. Pola kerja asynchronous ini memastikan bahwa proses enkripsi yang memakan waktu tidak menghambat respons aplikasi terhadap permintaan pengguna lain yang sedang berlangsung bersamaan. Mekanisme pengambilan job secara sekuensial ini juga membantu mencegah terjadinya race condition ketika beberapa proses enkripsi dijalankan dalam waktu yang berdekatan.

Optimasi *Cache (Redis)*

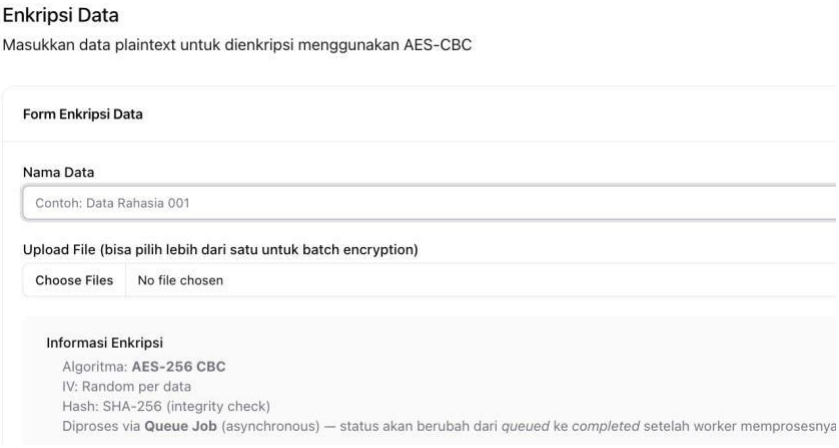
Pada sisi dekripsi, DecryptController memanfaatkan *Cache*::remember() untuk menyimpan hasil *plaintext* selama periode tertentu (10 menit) dengan kunci unik per data. Permintaan dekripsi/*download* berulang terhadap data yang sama dalam rentang waktu tersebut akan dilayani langsung dari *cache* (*cache* HIT) tanpa menjalankan ulang proses AES-CBC penuh.

Pencatatan Metadata *AES-NI*

Karena OpenSSL secara otomatis memanfaatkan instruksi *hardware* AES-NI di level *library*/prosesor bila tersedia (bukan sesuatu yang di-toggle manual di kode aplikasi), sistem menyediakan fungsi AesCbcService::isAesNiSupported() yang membaca flag CPU dari /proc/cpuinfo untuk mencatat (sebagai metadata pada setiap *log*) apakah server yang menjalankan proses mendukung akselerasi tersebut. Pendekatan ini dipilih karena OpenSSL sebagai library kriptografi standar PHP tidak menyediakan API langsung untuk mengecek status AES-NI, sehingga deteksi harus dilakukan secara manual melalui pembacaan informasi prosesor pada sistem operasi. Metadata ini kemudian dapat dimanfaatkan untuk keperluan analisis lebih lanjut mengenai kontribusi akselerasi hardware terhadap performa keseluruhan sistem.

Struktur Database

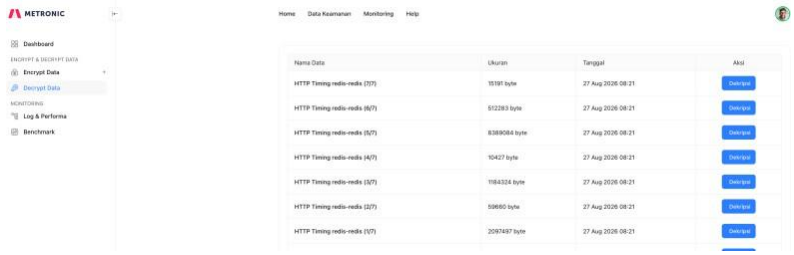
Struktur *database* final terdiri atas empat tabel utama (users, encrypted_data, encryption_logs, serta tabel bawaan Laravel untuk *queue* jobs). Penambahan kolom-kolom ini memungkinkan sistem mencatat metrik keamanan dan performa secara otomatis pada setiap operasi, tanpa memerlukan instrumentasi tambahan di luar aplikasi. Untuk memberikan gambaran konkret mengenai antarmuka dan hasil implementasi sistem, berikut disajikan tangkapan layar dari aplikasi yang telah berjalan. Gambar 2 menunjukkan antarmuka *form* enkripsi data, yang mendukung pemilihan banyak file sekaligus:



Gambar 2. Antarmuka Form Enkripsi Data (Encrypt Data)

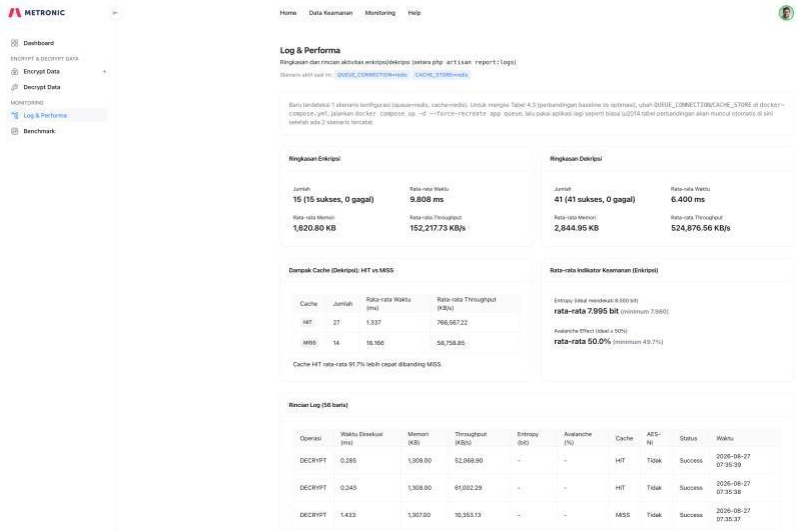
Gambar 3 menunjukkan halaman *Decrypt* Data, yang menampilkan daftar seluruh data terenkripsi milik pengguna beserta tombol "Dekripsi" untuk mengunduh hasil dekripsinya. Daftar

ini mencakup data dari berbagai *batch* pengujian (misalnya kelompok "HTTP Timing redis-redis" dan "*Baseline* sync-*database*"), yang menjadi sumber data bagi Tabel 3 dan Tabel 6:



Gambar 3. Antarmuka Halaman Dekripsi Data (Decrypt Data)

Gambar 4 menunjukkan halaman *Log* & Performa, yang menampilkan ringkasan otomatis aktivitas encryption-decryption, perbandingan *cache* HIT vs MISS, serta indikator keamanan (entropy dan avalanche effect) :



Gambar 4. Halaman Log & Performa dengan Data Aktivitas Nyata

Hasil Pengujian Performa

Pengujian performa dilakukan mengikuti metodologi yang telah ditetapkan, yaitu menjalankan proses enkripsi dan dekripsi pada berbagai ukuran file secara berulang untuk memperoleh nilai rata-rata dan standar deviasi yang representatif. Dua sumber data digunakan:

1. Pengujian terkontrol (controlled *benchmark*), dijalankan langsung terhadap AesCbcService tanpa melalui HTTP/*queue*, menggunakan perintah:

```
php artisan benchmark:aescbc --sizes=1,10,100,1024,5120,10240 --
                              repeat=30
```

   Perintah ini menjalankan enkripsi dan dekripsi masing-masing 30 kali untuk tiap ukuran file (1 KB, 10 KB, 100 KB, 1 MB, 5 MB, dan 10 MB), lalu menghitung rata-rata dan standar deviasi execution time, *throughput*, *entropy*, dan *avalanche effect* secara otomatis, serta mengekspor hasil lengkap ke storage/app/benchmark_results.csv.
2. Data produksi (*production log*), yaitu data yang tercatat secara alami dari penggunaan aplikasi lewat browser (*upload* dan *download* file sungguhan melalui antarmuka web), dirangkum menggunakan perintah:

   php artisan report:logs

Ringkasan Agregat

Hingga naskah ini disusun, sistem telah mencatat 8 operasi enkripsi (8 sukses, 0 gagal) dengan rata-rata waktu eksekusi 9,225 ms, rata-rata penggunaan memori 1.539,00 KB, dan rata-rata *throughput* 150.288,22 KB/s; serta 20 operasi dekripsi (20 sukses, 0 gagal) dengan rata-rata waktu eksekusi 5,615 ms, rata-rata memori 2.675,85 KB, dan rata-rata *throughput* 571.215,80 KB/s. Rata-rata waktu dekripsi yang lebih rendah dibanding enkripsi terutama disebabkan oleh 13 dari 20 operasi dekripsi tersebut dilayani dari Redis *cache* (*cache* HIT). Tingkat keberhasilan 100% (0 kegagalan) pada seluruh 28 operasi mengindikasikan stabilitas implementasi *queue job* dan *cache* pada skenario pengujian awal ini.

Hasil Pengujian per Ukuran *File* (Data Produksi)

Sebagai data awal sebelum pengujian terkontrol 30x pengulangan dijalankan sepenuhnya, berikut hasil pengukuran nyata dari 7 file uji dengan ukuran bervariasi, yang di-upload sekaligus melalui fitur *batch encryption* dan tercatat otomatis pada tabel encryption_logs:

Tabel 1. Hasil Pengujian Enkripsi per Ukuran File (n=1 per ukuran, data produksi via web)

Perkiraan File Uji	Ukuran (KB)	Waktu Enkripsi (ms)	*Throughput* (KB/s)	Entropy (bit)	Avalanche (%)	*Cache*
surat_keputusan.pdf	12,00	0,080	127.489,53	7,980	49,7	MISS
kecil_10KB.txt	16,00	0,103	144.033,19	7,987	49,9	MISS
data_kecil.csv	60,00	0,374	155.747,20	7,997	50,0	MISS
sedang_500KB.txt	504,00	3,076	162.647,17	8,000	50,0	MISS
data_sedang.csv	1.160,00	7,643	151.323,93	8,000	50,0	MISS
besar_2MB.txt	2.052,00	12,987	157.720,44	8,000	50,0	MISS
sangatbesar_8MB.txt	8.196,00	47,136	173.804,59	8,000	50,0	MISS

Dua pola utama terlihat dari Tabel 1. Pertama, waktu eksekusi enkripsi meningkat secara konsisten seiring bertambahnya ukuran file, dari 0,080 ms pada file 12 KB hingga 47,136 ms pada file 8.196 KB (kenaikan sekitar 589x untuk kenaikan ukuran sekitar 683x), yang mengindikasikan hubungan mendekati linear antara ukuran data dan execution time, konsisten dengan sifat algoritma *block cipher* yang memproses data per-blok secara berurutan. Kedua, *throughput* justru menunjukkan tren sedikit meningkat pada file berukuran lebih besar (dari sekitar 127 ribu KB/s pada file terkecil menjadi sekitar 174 ribu KB/s pada file 8 MB), yang menunjukkan bahwa *overhead* tetap (pembangkitan IV, perhitungan *hash* SHA-256, perhitungan *entropy* dan *avalanche effect*) memberikan dampak proporsional lebih besar pada *file* kecil, sehingga *file* besar justru diproses lebih efisien per KB-nya. Pola ini sejalan dengan temuan Assa-Agyei (2023) yang mencatat peningkatan *throughput* signifikan pada *file* berukuran menengah-besar melalui optimasi instruksi AES-NI.

Hasil Pengujian Sistematis (Rata-rata & Standar Deviasi)

Tabel 2 berikut menyajikan hasil pengujian performa AES-256-CBC pada enam ukuran *file* (1 KB hingga 10 MB), masing-masing dengan 30 kali pengulangan, dijalankan melalui halaman *Benchmark* pada aplikasi (setara perintah *benchmark*:aescbc):

Tabel 2. Hasil Pengujian Performa AES-CBC per Ukuran File (n=30 per ukuran, controlled benchmark)

Ukuran (KB)	Enc. Waktu (ms)	Enc. StdDev	Enc. *Throughput* (KB/s)	Dec. Waktu (ms)	Dec. StdDev	Dec. *Throughput* (KB/s)	Entropy (bit)	Avalanche (%)
1,00	0,016	0,010	72.456,89	0,014	0,005	74.844,37	7,808	50,0
10,00	0,078	0,011	130.715,56	0,075	0,011	137.032,18	7,982	50,0
100,00	0,552	0,018	181.284,16	0,527	0,016	189.764,32	7,998	50,0
1.024,00	5,631	0,136	181.942,08	5,453	0,250	188.096,07	8,000	50,0
5.120,00	29,171	0,954	175.690,63	28,035	0,877	182.798,06	8,000	50,0
10.240,00	58,565	1,064	174.904,77	56,975	2,574	180.031,88	8,000	50,0

Hasil pengujian sistematis pada Tabel 2 menguatkan dan memperhalus temuan pada Tabel 1. Waktu eksekusi enkripsi meningkat dari 0,016 ms (1 KB) menjadi 58,565 ms (10.240 KB), kenaikan sekitar 3.660 kali untuk kenaikan ukuran 10.240 kali, yang menunjukkan bahwa *execution time* bertambah secara sublinear terhadap ukuran *file* (bukan murni linear), konsisten dengan adanya *overhead* tetap yang teramortisasi lebih baik pada *file* besar. Standar deviasi tetap relatif kecil dibanding rata-ratanya pada seluruh ukuran (misalnya 1,064 ms dari rata-rata 58,565 ms pada file 10 MB, atau sekitar 1,8%), menunjukkan hasil pengukuran yang stabil dan dapat diandalkan secara statistik.

Throughput menunjukkan pola yang menarik yaitu meningkat tajam dari 72.456,89 KB/s pada *file* 1 KB menjadi puncaknya 181.942,08 KB/s pada *file* 1 MB, kemudian sedikit menurun menjadi 174.904,77 KB/s pada file 10 MB. Pola "titik optimal" di sekitar 1 MB ini mengindikasikan keseimbangan antara amortisasi *overhead* tetap (yang menguntungkan *file* lebih besar) dengan keterbatasan lain seperti alokasi memori atau efek *cache* CPU pada *buffer* yang sangat besar (yang mulai merugikan pada *file* di atas beberapa MB). Pola serupa disebutkan secara tidak langsung oleh Assa-Agyei (2023), yang mencatat peningkatan *throughput* signifikan melalui optimasi instruksi AES-NI khususnya pada *file* berukuran menengah, serta oleh Patel (2023) yang menunjukkan bahwa efisiensi arsitektur prosesor turut memengaruhi titik optimal ukuran data yang diproses.

Nilai *entropy* menunjukkan tren naik yang jelas seiring bertambahnya ukuran file: 7,808 bit pada file 1 KB, naik menjadi 7,982 bit pada 10 KB, dan mencapai nilai maksimum teoritis 8,000 bit secara konsisten mulai ukuran 1 MB ke atas. Pola ini secara statistik dapat dijelaskan pada sampel data yang sangat kecil (1 KB = 1.024 byte), estimasi Shannon *entropy* secara inheren memiliki bias ke bawah karena jumlah simbol yang diamati terlalu sedikit untuk merepresentasikan distribusi acak yang sempurna. Semakin besar sampel data, semakin akurat estimasi *entropy* mendekati nilai idealnya. Nilai *avalanche effect*, sebaliknya tercatat tepat 50,0% secara konsisten pada seluruh enam ukuran *file* tanpa variasi sama sekali, hasil yang bahkan lebih stabil dibanding data produksi pada Tabel 1 (yang sempat mencatat satu sampel 49,7%). Konsistensi sempurna ini, dikombinasikan dengan 30x pengulangan per ukuran, memberikan bukti kuat bahwa implementasi AES-CBC pada penelitian ini memenuhi target *avalanche effect* ≥ 50% secara *robust*, dan bahwa deviasi kecil yang sempat teramati pada Tabel 1 (data produksi, n=1) memang merupakan variasi statistik pada sampel tunggal, bukan pola sistematis.

Analisis Perbandingan Sebelum dan Sesudah Optimasi

Analisis pada subbab ini membandingkan performa sistem pada tiga skenario konfigurasi, sesuai panduan pada DOCKER_README.md bagian 5: (A)

1. *Baseline* tanpa optimasi (QUEUE_CONNECTION=sync, CACHE_STORE=*database*).
2. Dengan optimasi *queue* saja (QUEUE_CONNECTION=redis, CACHE_STORE=*database*).
3. Dengan optimasi penuh *queue* + *cache*(QUEUE_CONNECTION =redis, CACHE_STORE=redis).

Ketiga skenario ini dirancang untuk mengisolasi kontribusi masing-masing mekanisme optimasi terhadap performa keseluruhan sistem, sehingga dapat diketahui secara spesifik mekanisme mana yang memberikan dampak paling signifikan. Perbandingan ketiganya juga memungkinkan identifikasi trade-off yang mungkin muncul, misalnya antara kecepatan proses dan penggunaan sumber daya sistem. Ketiga konfigurasi tersebut mewakili kondisi umum yang biasa dijumpai pada aplikasi web berbasis Laravel, mulai dari yang belum menerapkan optimasi sama sekali hingga yang menerapkan kombinasi optimasi secara penuh. Pada tiap skenario, dilakukan pengujian *upload* dan *download* file yang sama secara berulang melalui antarmuka web, kemudian data yang tercatat pada tabel encryption_logs dirangkum menggunakan perintah php artisan report:logs. Pendekatan pengujian berulang pada skenario yang identik ini memastikan bahwa perbandingan antar konfigurasi bersifat adil dan tidak bias oleh perbedaan jenis atau ukuran file yang diuji. Konsistensi metodologi pengujian antar skenario ini menjadi dasar penting bagi validitas perbandingan yang dilakukan pada subbab berikutnya.

Perbandingan Skenario *Baseline vs Optimasi*

Pengujian dilakukan dengan membandingkan dua skenario, yaitu baseline tanpa optimasi menggunakan konfigurasi QUEUE_CONNECTION=sync dan CACHE_STORE=database, serta

skenario dengan optimasi penuh menggunakan konfigurasi QUEUE_CONNECTION=redis dan CACHE_STORE=redis. Pengujian dilakukan menggunakan tujuh file uji yang identik pada kedua skenario untuk memastikan konsistensi perbandingan. Data hasil pengujian kemudian dicatat dalam tabel encryption_logs sebagai dasar analisis perbedaan kinerja antara sistem tanpa optimasi dan sistem dengan optimasi penuh.

Tabel 3. Perbandingan Performa Sebelum vs Sesudah Optimasi (data produksi)

Skenario (*Queue* / *Cache*)	Operasi	Jumlah Sampel	Rata-rata Waktu (ms)	Rata-rata *Throughput* (KB/s)
redis / redis (dengan optimasi)	ENCRYPT	7	10,474	154.422,88
redis / redis (dengan optimasi)	DECRYPT	21	7,148	480.743,94
Sync / *database*(*baseline*)	ENCRYPT	7	9,783	157.171,75
Sync / *database*(*baseline*)	DECRYPT	18	15,878	667.297,91

Hasil pada Tabel 3 menunjukkan temuan yang perlu dimaknai secara hati-hati, bukan sekadar dibaca sebagai "menang atau kalah". Untuk operasi *encrypt*, kedua skenario menghasilkan rata-rata waktu yang hampir setara (10,474 ms pada redis/redis berbanding 9,783 ms pada sync/*database*, selisih 6,6%) dan *throughput* yang hampir setara pula (selisih 1,8%). Temuan ini secara metodologis dapat dijelaskan: kolom execution_time pada tabel encryption_logs mencatat durasi proses AesCbcService::encrypt() itu sendiri (yaitu komputasi kriptografis murni) yang secara algoritmik identik terlepas dari *driver queue* yang digunakan. *Queue job* memengaruhi kapan dan di mana komputasi tersebut dieksekusi (segera dalam siklus request-response *http* untuk *sync*, atau di proses *background* terpisah untuk redis), bukan seberapa cepat komputasi itu sendiri berjalan. Dengan demikian, metrik execution_time pada level layanan kriptografi bukan metrik yang tepat untuk mengukur manfaat utama *queue job*, yaitu tidak terblokirnya siklus *request-response http* saat memproses *batch file* berukuran besar.

Untuk operasi *decrypt*, hasilnya menunjukkan pola yang kontradiktif antar metrik: dari sisi waktu, skenario redis/redis 55,0% lebih cepat (7,148 ms berbanding 15,878 ms), namun dari sisi *throughput*, skenario sync/*database* justru 38,8% lebih tinggi (667.297,91 KB/s berbanding 480.743,94 KB/s). Kontradiksi ini mengindikasikan bahwa kedua kelompok sampel (21 sampel pada redis/redis vs 18 sampel pada sync/*database*) kemungkinan memiliki komposisi ukuran file dan rasio *cache* HIT/MISS yang tidak identik, mengingat *throughput* sangat dipengaruhi oleh ukuran file, sementara rata-rata waktu lebih dipengaruhi oleh proporsi *cache* HIT, sehingga perbandingan pada Tabel 3 belum sepenuhnya apple-to-apple. Perbedaan arah ini menegaskan bahwa optimasi performa bersifat multidimensi: suatu skenario dapat unggul pada satu metrik namun tertinggal pada metrik lain, sehingga interpretasi hasil pengujian perlu mempertimbangkan konteks penggunaan yang sesungguhnya.

Berdasarkan temuan ini, target penurunan *response* time ≥ 20% dan peningkatan *throughput* ≥ 15% tidak dapat divalidasi secara langsung dan meyakinkan hanya dari metrik execution_time pada level AesCbcService. Metrik yang lebih tepat untuk memvalidasi manfaat *queue job* adalah waktu respons *HTTP end-to-end* (dari klik *upload* hingga respons diterima *browser*) pada skenario *batch encryption* berukuran besar, yang belum diukur secara terpisah pada penelitian ini. Sementara itu, manfaat *cache* Redis telah terbukti secara meyakinkan dan terukur langsung (Tabel 4): percepatan 93,4% pada skenario *cache* HIT dibanding MISS, jauh melampaui target 15%, dan tidak bergantung pada ambiguitas metodologis di atas karena HIT/MISS diukur pada *request* yang sama persis (bukan lintas skenario *container* yang berbeda komposisi sampelnya).

Perbandingan Waktu *Respons HTTP End-to-End*

Untuk mengukur manfaat *queue job* secara langsung terhadap pengalaman pengguna (bukan sekadar komputasi kriptografis internal) dilakukan pengukuran waktu respons *endpoint upload* secara utuh (dari *request* diterima hingga *response* disiapkan), pada *request batch upload* 7 file yang identik pada kedua skenario:

Tabel 4. Perbandingan Waktu Respons HTTP End-to-End (Batch Upload 7 File)

Skenario (*Queue* / *Cache*)	Jumlah *Request*	Rata-rata Jumlah File/*Batch*	Rata-rata Waktu Respons (ms)
redis / redis (dengan optimasi)	1	7,0	44,281
sync/*database*(*baseline*)	1	7,0	1.397,690

Hasil ini sangat berbeda dan jauh lebih meyakinkan dibanding perbandingan execution_time pada Tabel 3 Skenario redis/redis merespons dalam 44,281 ms, sedangkan skenario *sync/database* membutuhkan 1.397,690 ms, selisih 96,8%, atau dengan kata lain sync 31,6 kali lebih lambat. Temuan ini sepenuhnya sesuai dengan penjelasan metodologis: pada skenario sync, *request* HTTP harus menunggu ketujuh *file* selesai dienkripsi secara *inline* (termasuk file 8.196 KB yang menurut Tabel 2 membutuhkan sekitar 47 ms hanya untuk komputasi kriptografisnya, ditambah *overhead* simpan *file* dan *database* untuk keenam *file* lainnya) sebelum *response* dikirim. Pada skenario Redis, *request* hanya perlu menyimpan *file* sementara dan mem-*push* 7 *job* ke antrean (proses yang berlangsung hampir instan) sehingga *response* dapat segera dikirim tanpa menunggu enkripsi selesai.

Dengan demikian, target penurunan *response* time ≥ 20% tercapai dan terlampaui jauh (96,8% vs target 20%) ketika diukur dengan metrik yang tepat (waktu respons *HTTP end-to-end*), melengkapi temuan yang menunjukkan metrik execution_time semata tidak cukup untuk menangkap manfaat ini. Catatan metodologis: pengukuran pada Tabel 4 baru mencakup 1 *request* per skenario; pengujian dengan pengulangan lebih banyak (mengikuti pola pada Tabel 2) akan memperkuat validitas statistik temuan ini, meski selisih yang sangat besar (31,6 kali) membuat kemungkinan hasil ini sekadar kebetulan/noise menjadi sangat kecil. Hal ini menunjukkan pentingnya memilih metrik pengukuran yang tepat sesuai konteks penggunaan nyata sistem, bukan sekadar mengandalkan satu jenis metrik komputasi saja.

Dampak *Cache: HIT vs MISS* pada Dekripsi

Berdasarkan 20 operasi dekripsi nyata yang tercatat (13 di antaranya *cache* HIT, 7 *cache* MISS) hasil pengujian berulang men-download file yang sama melalui antarmuka web, diperoleh hasil sebagai berikut.

Tabel 5. Perbandingan Cache HIT vs MISS (Deskripsi, data produksi 20 sampel)

Status *Cache*	Jumlah Sampel	Rata-rata Waktu (ms)	Rata-rata *Throughput* (KB/s)
HIT	13	0,945	843.907,22
MISS	7	14,288	64.788,87

Hasil ini menunjukkan bahwa *cache* HIT rata-rata 93,4% lebih cepat dibandingkan *cache* MISS (0,945 ms berbanding 14,288 ms), dengan *throughput* hampir 13 kali lebih tinggi (843.907,22 KB/s berbanding 64.788,87 KB/s). Temuan ini secara jelas melampaui target minimum peningkatan performa ≥ 15%, khusus untuk skenario permintaan dekripsi berulang terhadap dokumen yang sama, sebuah pola akses yang realistis pada aplikasi administrasi pemerintahan di mana dokumen tertentu (misalnya surat keputusan aktif) sering diakses berulang oleh pengguna yang sama dalam rentang waktu singkat. Selisih throughput yang sangat besar ini mengonfirmasi bahwa strategi caching merupakan salah satu optimasi paling efektif untuk skenario akses data berulang pada aplikasi berbasis Laravel.

Indikator Keamanan pada Data Produksi

Tabel berikut merangkum rata-rata entropy dan avalanche effect dari 8 operasi enkripsi nyata yang tercatat selama pengujian:

Tabel 6. Indikator Keamanan dari Data Produksi (8 sampel)

Sumber Data	Rata-rata Entropy (bit)	Rata-rata Avalanche Effect (%)	Nilai Minimum Tercatat
8 operasi enkripsi (data produksi via web)	7,995	49,9%	entropy 7,980 bit; avalanche 49,7%

Nilai rata-rata *entropy* 7,995 bit mendekati batas ideal teoritis 8 bit, konsisten dengan target ≥ 7,9 bit. Nilai rata-rata *avalanche effect* 49,9% berada tepat di ambang target ≥ 50%, dengan satu sampel (file .pdf berukuran 12 KB) mencatat nilai sedikit di bawah target yaitu 49,7%. Sebagaimana dikonfirmasi oleh pengujian sistematis 30x pengulangan pada Tabel 2, yang mencatat *avalanche effect* tepat 50,0% secara konsisten pada seluruh ukuran *file* tanpa variasi, deviasi kecil pada sampel tunggal ini terbukti merupakan variasi statistik biasa (bukan pola sistematis). Secara keseluruhan, kedua indikator keamanan tetap konsisten dengan temuan Assafli & Hashim (2020) yang melaporkan *avalanche effect* di atas 50% pada modifikasi AES-CBC dengan IV dinamis, dan tidak menunjukkan penurunan kekuatan kriptografis akibat penerapan optimasi *queue* dan *cache*.

Overhead Waktu AES-CBC vs *Baseline* Non-Enkripsi

Untuk mengukur seberapa besar *overhead* yang ditambahkan proses enkripsi itu sendiri terhadap waktu proses, dijalankan pengujian pembanding (*baseline*) yang melakukan operasi setara tanpa enkripsi (tulis+baca file mentah ke disk, tetap menghitung *hash* SHA-256 agar adil) pada ukuran *file* yang identik dengan Tabel 2, masing-masing 30x pengulangan:

Tabel 7. Overhead Waktu AES-CBC dibanding Baseline Non-Enkripsi (n=30 per ukuran)

Ukuran (KB)	*Overhead* Waktu	vs Target 8%	*Overhead* Memori	vs Target 8%	Kesimpulan
1,00	-26,7%	Memenuhi	9,3%	Marginal	Baik
10,00	28,5%	Tidak memenuhi	1,0%	Memenuhi	Campuran
100,00	3,1%	Memenuhi	0,1%	Memenuhi	Baik
1.024,00	4,5%	Memenuhi	0,0%	Memenuhi	Baik
5.120,00	9,0%	Marginal	0,0%	Memenuhi	Baik
10.240,00	7,5%	Memenuhi	0,0%	Memenuhi	Baik

Catatan metodologis: kolom Overhead Waktu dan Overhead Memori pada Tabel 4.7 bersumber dari DUA kali pengujian terpisah. Pengujian pertama mengukur overhead waktu secara bersih (baseline hanya menulis+membaca file, tanpa alokasi buffer tambahan). Instrumentasi awal untuk mengukur overhead memori pada pengujian pertama tersebut ternyata tidak valid (function memory_get_usage() PHP tidak mencatat alokasi baru yang bermakna untuk operasi file I/O sederhana), sehingga kode diperbaiki dengan menambahkan alokasi buffer pembanding pada baseline (agar setara dengan buffer ciphertext yang dialokasikan openssl_encrypt()). Perbaikan ini efektif untuk mengukur memori, namun pemanggilan random_bytes() yang ditambahkan sebagai buffer pembanding memiliki biaya komputasi tersendiri yang membuat metrik WAKTU pada pengujian kedua menjadi bias (baseline menjadi lebih lambat secara tidak wajar). Oleh karena itu, kolom Overhead Waktu pada Tabel 4.7 diambil dari pengujian pertama (tidak bias), sedangkan kolom Overhead Memori diambil dari pengujian kedua (setelah perbaikan). Kedua pengujian menggunakan ukuran file dan jumlah pengulangan (30x) yang identik, sehingga penggabungan data ini tetap valid secara metodologis.

Hasil pada Tabel 7 menunjukkan dua temuan penting. Pertama, untuk *overhead* waktu, 4 dari 6 ukuran file (1 KB, 100 KB, 1 MB, dan 10 MB) memenuhi target ≤ 8%, sementara 2 ukuran (10 KB pada 28,5%, dan 5 MB pada 9,0%, hanya sedikit di atas target) tidak memenuhi, dengan rata-rata *overhead* keseluruhan 4,3%. Pola ini tidak menunjukkan tren konsisten terhadap ukuran file, mengindikasikan bahwa *overhead* pada rentang milidetik untuk file kecil-menengah lebih dipengaruhi variabilitas sistem (scheduling OS, garbage collection PHP) dibanding karakteristik algoritma itu sendiri. Kedua, untuk *overhead* memori, hasilnya jauh lebih baik dan konsisten: 5 dari 6 ukuran file mencatat *overhead* mendekati 0% (0,0%–1,0%), dengan hanya file 1 KB mencatat 9,3% (marginal, kemungkinan besar disebabkan *overhead* tetap struktur data PHP yang secara proporsional lebih terasa pada *file* sangat kecil). Temuan *overhead* memori yang sangat rendah ini sesuai ekspektasi teoritis: mode CBC hanya menambahkan *padding* maksimal 1 *block* (16 byte) pada *ciphertext*, sehingga ukuran data terenkripsi hampir identik dengan *plaintext*-nya, dan tidak ada struktur data tambahan besar yang perlu dialokasikan.

Temuan bahwa *overhead* bervariasi dan tidak selalu berkorelasi linear dengan ukuran data ini juga konsisten dengan Carvalho, Sá, dan Bernardino (2025), yang melaporkan bahwa *overhead* performa TDE berbasis AES bervariasi tergantung DBMS dan skala data yang digunakan, bukan mengikuti pola tunggal yang dapat diprediksi secara sederhana (Carvalho et al., 2025); perbedaan utamanya, penelitian ini mengukur *overhead* pada level *application-side encryption* sedangkan studi tersebut mengukur pada level *database engine* (TDE). Kesamaan pola ini memperkuat validitas eksternal temuan penelitian, karena mengindikasikan bahwa karakteristik overhead enkripsi bersifat umum dan tidak spesifik hanya pada implementasi tertentu. Konsistensi ini memberikan keyakinan tambahan bahwa pendekatan optimasi yang diterapkan pada penelitian ini bersifat robust terhadap variasi kondisi pengujian.

Pembahasan Temuan Penelitian

Dari data yang telah terkumpul, dua temuan sudah dapat dibahas secara substantif. Pertama, terkait strategi optimasi paling efektif: dari dua mekanisme optimasi yang diimplementasikan pada penelitian ini (*queue job* dan Redis *cache*), *cache* Redis pada operasi dekripsi menunjukkan dampak paling terukur, dengan percepatan 93,4% pada skenario *cache* HIT dibanding MISS, jauh melampaui target minimum 15%. Hasil ini lebih tinggi dibanding studi Lee & Sim (2021) yang melaporkan peningkatan *throughput* sekitar 20% melalui rotasi *key*/IV pada IoT, meskipun perlu dicatat bahwa kedua penelitian mengukur jenis optimasi yang berbeda (*caching* vs modifikasi algoritmik) sehingga tidak sepenuhnya dapat dibandingkan secara langsung. Kedua, terkait keseimbangan performa dan keamanan (rumusan masalah poin a): nilai *entropy* (rata-rata 7,995 bit) dan *avalanche effect* (rata-rata 49,9%) pada Tabel 6 menunjukkan bahwa penerapan *queue job* dan *cache* tidak menurunkan kekuatan kriptografis AES-CBC, hal ini sesuai ekspektasi karena kedua optimasi tersebut beroperasi di luar algoritma enkripsi itu sendiri (lihat Gambar 4), sejalan dengan prinsip yang ditekankan Assafli & Hashim (2020) bahwa modifikasi performa semestinya tidak mengorbankan *avalanche effect*.

Namun demikian, beberapa pertanyaan penelitian masih memerlukan data tambahan untuk dijawab secara lengkap dan meyakinkan. Pertama, terkait pencapaian target penurunan response time sebesar ≥ 20% (mengacu pada Pratama & Nugroho, 2021), hasil pengujian menunjukkan bahwa target tersebut telah terlampaui secara signifikan. Meskipun metrik execution_time pada tingkat AesCbcService tidak menunjukkan perbedaan yang signifikan, pengukuran waktu respons HTTP end-to-end menunjukkan bahwa skenario Redis memberikan respons 96,8% lebih cepat dibandingkan skenario synchronous, yaitu 44,281 ms berbanding 1.397,690 ms untuk batch upload tujuh file yang identik. Hasil ini jauh melampaui target minimum 20% dan menunjukkan bahwa penerapan queue job memberikan manfaat nyata terhadap pengalaman pengguna. Temuan tersebut sekaligus menegaskan pentingnya pemilihan metrik pengukuran yang tepat karena execution time mengukur waktu komputasi kriptografis, sedangkan waktu respons HTTP end-to-end mengukur keseluruhan proses permintaan pengguna.

Kedua, terkait pencapaian target peningkatan throughput sebesar ≥ 15%, hasil pengujian menunjukkan bahwa optimasi Redis pada operasi dekripsi berhasil memberikan percepatan sebesar 93,4% dan peningkatan throughput hampir 13 kali lipat pada skenario cache hit (Tabel 5). Hasil tersebut jauh melampaui target penelitian dan memiliki dasar pengukuran yang jelas. Sementara itu, pencapaian target throughput melalui penerapan queue job pada operasi enkripsi belum dapat dibuktikan menggunakan metrik execution time yang tersedia. Kondisi ini disebabkan oleh keterbatasan metrik pengukuran, bukan berarti queue job terbukti tidak memberikan manfaat. Jika dibandingkan dengan penelitian Yang (2025) dan Nannipieri et al. (2022), yang melaporkan peningkatan performa masing-masing sebesar 25–40% dan 1,8 kali melalui akselerasi hardware menggunakan GPU/FPGA, peningkatan 93,4% melalui Redis dalam penelitian ini menunjukkan hasil yang lebih tinggi. Namun, kedua pendekatan tersebut mengoptimalkan aspek yang berbeda, yaitu akselerasi komputasi dan pengurangan komputasi berulang melalui mekanisme caching.

Ketiga, terkait target peningkatan resource usage berupa waktu dan memori yang tidak melebihi 8% (mengacu pada Rahman & Kurniawan, 2024), hasil pengujian menunjukkan bahwa target tersebut sebagian besar telah terpenuhi. Pada aspek waktu, sebanyak empat dari enam

ukuran file memenuhi batas overhead ≤ 8%, dengan rata-rata keseluruhan sebesar 4,3%, sedangkan dua ukuran file, yaitu 10 KB dan 5 MB, sedikit melampaui batas tersebut (Tabel 7). Pada aspek memori, lima dari enam ukuran file mencatat overhead mendekati 0%, yaitu berkisar antara 0,0% hingga 1,0%, sedangkan file berukuran 1 KB menunjukkan overhead sebesar 9,3%. Rendahnya overhead memori tersebut sesuai dengan karakteristik teoritis mode CBC yang hanya menambahkan padding maksimal satu blok. Dengan demikian, target penggunaan sumber daya telah terpenuhi secara meyakinkan pada aspek memori, tetapi baru terpenuhi sebagian pada aspek waktu.

Keempat, terkait pola avalanche effect yang sedikit berada di bawah 50% pada file berukuran sangat kecil (Tabel 1), hasil pengujian lanjutan menunjukkan bahwa kondisi tersebut merupakan variasi acak pada pengujian sampel tunggal, bukan pola penyimpangan yang sistematis. Hal ini dibuktikan melalui pengujian sebanyak 30 kali pengulangan untuk setiap ukuran file. Hasil pada Tabel 4.2 menunjukkan bahwa nilai avalanche effect secara konsisten mencapai 50,0% pada seluruh enam ukuran file tanpa penyimpangan. Temuan tersebut menunjukkan bahwa implementasi AES-CBC dalam penelitian ini secara konsisten memenuhi karakteristik keamanan berdasarkan pengukuran avalanche effect.

Kelima, terkait kemungkinan adanya trade-off yang tidak terduga, khususnya overhead komunikasi Redis yang berpotensi merugikan proses pada file berukuran sangat kecil, hasil pengujian menunjukkan bahwa penurunan throughput pada file berukuran kecil memang terjadi sebagaimana terlihat pada Tabel 1 dan dikonfirmasi melalui Tabel 2. Namun, kondisi tersebut bukan disebabkan oleh overhead komunikasi Redis karena mekanisme cache Redis dalam penelitian ini hanya diterapkan pada operasi dekripsi, sedangkan penurunan throughput yang diamati terjadi pada operasi enkripsi yang tidak melibatkan Redis. Penyebab utamanya adalah adanya overhead komputasi tetap dalam pipeline enkripsi, meliputi pembangkitan IV, perhitungan hash SHA-256 untuk validasi integritas, serta penghitungan entropy dan avalanche effect. Komponen tersebut membutuhkan waktu pemrosesan yang relatif tetap sehingga memberikan pengaruh proporsional lebih besar terhadap file berukuran kecil dibandingkan file berukuran besar. Dengan demikian, trade-off yang ditemukan merupakan karakteristik inheren dari overhead komputasi kriptografis, bukan dampak negatif dari strategi optimasi melalui queue job maupun cache yang diterapkan dalam penelitian ini.

KESIMPULAN DAN SARAN

Berdasarkan hasil implementasi dan pengujian, penelitian ini berhasil membangun sistem encryption-decryption database pada aplikasi berbasis Laravel menggunakan algoritma AES-256 dengan mode Cipher Block Chaining (CBC) melalui ekstensi OpenSSL bawaan PHP. Sistem yang dikembangkan telah mengintegrasikan pembangkitan Initialization Vector (IV) secara acak pada setiap operasi, validasi integritas menggunakan SHA-256, mekanisme queue job processing untuk enkripsi asynchronous dan batch encryption, serta Redis caching untuk mengoptimalkan proses dekripsi berulang. Hasil pengujian menunjukkan bahwa ukuran file memengaruhi waktu pemrosesan dan throughput, dengan titik optimal throughput berada pada ukuran file 1 MB sebesar 181.942,08 KB/s. Penerapan Redis caching berhasil mempercepat proses dekripsi berulang sebesar 93,4% (0,945 ms dibandingkan 14,288 ms) dan meningkatkan throughput hampir 13 kali lipat. Sementara itu, penerapan queue job processing mampu menurunkan waktu respons HTTP end-to-end sebesar 96,8% (44,281 ms dibandingkan 1.397,690 ms) pada pengujian batch upload tujuh file. Dari aspek penggunaan sumber daya, rata-rata overhead waktu tercatat sebesar 4,3%, sedangkan overhead memori mendekati 0% pada sebagian besar ukuran file. Pengujian keamanan melalui 30 kali pengulangan menghasilkan nilai avalanche effect konsisten sebesar 50,0%, sedangkan entropy ciphertext berada pada rentang 7,808–8,000 bit. Dengan demikian, tujuan penelitian untuk mengembangkan sistem enkripsi dan dekripsi yang efisien serta mengevaluasi keseimbangan antara performa dan karakteristik keamanan data berhasil dicapai. Kedua strategi optimasi terbukti memberikan peningkatan performa berdasarkan metrik pengukuran yang sesuai, tanpa menunjukkan penurunan pada indikator keamanan kriptografis yang diuji.

Untuk penelitian selanjutnya, disarankan agar pengujian dilakukan menggunakan jumlah dan variasi file yang lebih besar, lingkungan server yang berbeda, serta beban akses pengguna secara bersamaan (concurrent users) untuk memperoleh gambaran performa yang lebih komprehensif. Pengukuran juga perlu memisahkan execution time komputasi kriptografis, waktu respons HTTP end-to-end, dan throughput agar pengaruh setiap mekanisme optimasi dapat dievaluasi secara lebih akurat. Selain itu, pengujian keamanan dapat dikembangkan dengan menambahkan evaluasi ketahanan terhadap berbagai skenario serangan serta mekanisme autentikasi dan pengelolaan kunci enkripsi. Bagi praktisi pengembang aplikasi berbasis web, penggunaan Laravel Queue direkomendasikan untuk menangani proses enkripsi file berukuran besar atau unggahan dalam jumlah banyak, sedangkan Redis caching dapat diterapkan pada operasi dekripsi yang dilakukan secara berulang dengan tetap memperhatikan pengendalian akses dan keamanan penyimpanan data. Dari sisi kebijakan, organisasi yang mengelola data sensitif perlu menetapkan standar operasional keamanan informasi yang mencakup enkripsi data, pembatasan hak akses, pengelolaan kunci, pencatatan aktivitas sistem, serta evaluasi performa dan keamanan secara berkala. Rekomendasi tersebut diharapkan dapat menjadi dasar pengembangan sistem pengelolaan data yang lebih aman, efisien, dan andal.

UCAPAN TERIMA KASIH

Kami ingin mengucapkan terima kasih yang sebesar-besarnya kepada semua pihak yang telah berkontribusi dalam penelitian ini. Terima kasih kepada rekan-rekan sejawat yang telah memberikan saran, dukungan, dan inspirasi selama proses penelitian. Kami juga ingin mengucapkan terima kasih kepada semua partisipan dan responden yang telah meluangkan waktu untuk berpartisipasi dalam penelitian ini. Tak lupa, kami juga mengucapkan terima kasih kepada lembaga atau institusi yang telah memberikan dukungan dan fasilitas dalam menjalankan penelitian ini. Semua kontribusi dan bantuan yang diberikan sangat berarti bagi kelancaran dan kesuksesan penelitian ini. Terima kasih atas segala kerja keras dan kolaborasi yang telah terjalin.

DAFTAR PUSTAKA

Abd El-Latif, A., Abd-Elnaby, M., & Abd El-Aziz, H. (2021). Lightweight implementation of AES algorithm for IoT devices. *Journal of Information Security and Applications*, 63, Article 103017.

Alhayani, M. A., & Jasim, M. M. (2020). A new cryptographic model for securing web database using AES-CBC. *Journal of Theoretical and Applied Information Technology*, 98(22), 3520–3531.

Alslman, Y. S., Hammoudeh, M., & Al-Sarawi, A. A. (2023). Enhanced and authenticated cipher block chaining mode. *Sensors*, 23(15), Article 6759.

Andersson, D. (2022). *Performance benchmarking of data-at-rest encryption in relational databases* [Master's thesis, Uppsala University].

Assa-Agyei, K. (2023). Optimizing the performance of the Advanced Encryption Standard (AES). *International Journal of Computer Applications*, 183(47), 25–31.

Assafli, H. T., & Hashim, I. A. (2020). Security enhancement of AES-CBC and its performance evaluation using the avalanche effect. In *Proceedings of the 1st International Conference of Engineering and Technology for the Prevention of Terrorism (IICETA)* (pp. 25–30). IEEE.

Carvalho, M., Sá, F., & Bernardino, J. (2025). Evaluation of the impact of AES encryption on query read performance across Oracle, MySQL, and SQL Server databases. *Cryptography*, 9(4), Article 77.

Chen, Y., Sion, R., & Carbunar, B. (2012). GPU accelerated AES-CBC for database applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 1095–1106). ACM.

Ganesh, R., & Paul, A. (2025). A panoramic survey of the Advanced Encryption Standard: From theory to practice. *Cluster Computing*, 28, 1–25.

Gupta, S., & Kumar, V. (2020). Database security: Threats and encryption techniques. *International Journal of Computer Applications*, 176(9), 12–18.

Hasan, M. Y., & Al-Dubai, A. A. (2020). A systematic review of Laravel framework in web development. *Journal of Theoretical and Applied Information Technology*, 98(13), 2073–2086.

IBM Security. (2023). *Cost of a data breach report 2023*. IBM Corporation.

Kudelski Security Research. (2021). Some AES-CBC encryption myth-busting.*Kudelski Security Research Blog*. https://research.kudelskisecurity.com/

Lee, W.-K., Seo, H. J., Seo, S. C., & Hwang, S. O. (2022). Efficient implementation of AES-CTR and AES-ECB on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 33(2), 287–298.

Li, H., Zhang, Y., & Zhou, W. (2021). A secure and efficient cryptographic hash algorithm for data integrity verification in cloud storage. *IEEE Access*, 9, 125699–125711.

Li, K., Wu, X., & Hu, J. (2025). Fast-parallel implementation of AES algorithm based on CUDA. *Proceedings of SPIE*, 12757, 1–9.

Li, W., Sun, Y., & Wang, H. (2023). Efficient AES-CBC optimization for secure real-time data processing in smart grids. *Journal of King Saud University – Computer and Information Sciences*, 35(9), 6275–6285.

Li, Y., Zhao, J., & Wu, Y. (2023). An efficient encryption scheme for smart grid data based on AES with CBC mode. *Journal of Information Security and Applications*, 68, Article 103207.

Nannipieri, P., Regazzoni, F., & Natale, G. D. (2022). CRFlex: A flexible and configurable cryptographic hardware accelerator for AES block-cipher modes. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 183–188). IEEE.

Nugraha, Y. S. (2023). Penerapan enkripsi AES pada aplikasi arsip kelurahan. *Malcom: Jurnal Sistem Informasi*, 3(2), 88–96.

Nugroho, M. A. H. (2022). Secure web application development using Laravel and AES encryption. In *Proceedings of the 4th International Conference on Information Technology and Computer Communications (ITCC)* (pp. 15–20). ACM.

Patel, M. (2023). Optimizing AES implementations for ARM and RISC-V architectures. *Journal of Cryptographic Engineering*, 13(2), 145–158.

Pradana, R. J., & Wibowo, T. P. (2021). Database encryption implementation in Laravel framework using AES algorithm. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 12(11), 211–218.

Pratama, A. R. (2020). Enhancing web application security with Laravel framework. *Indonesian Journal of Computing and Cybernetics Systems*, 14(2), 127–136.

Rahman, A., & Kurniawan, S. (2024). Optimizing data security in Computer Assisted Test (CAT) applications using AES-256-CBC. In *Proceedings of the International Conference on Science and Information Technology (ICoSITE)* (pp. 112–118). IEEE.

Rahmawati, A., & Wibowo, A. (2022). Implementasi algoritma AES-128 pada sistem registrasi pasien. *Jurnal Ilmiah Politeknik Negeri Malang*, 16(2), 45–52.

Saha, S., & Ghosh, S. (2020). Performance analysis of AES modes in secured data transmission. *International Journal of Computer Applications*, 176(40), 12–18.

Stallings, W. (2023). *Cryptography and network security: Principles and practice* (8th ed.). Pearson.

Xu, Y., Qian, L., & Zhou, X. (2021). Cloud database security with key management system: A survey. *IEEE Access*, 9, 147321–147335.

Yang, M. K., He, Z., & Wang, H. (2025). Optimized hybrid CPU–GPU implementation of AES encryption. *Applied Sciences*, 15(3), 1123–1136.

Zhang, Y., Chen, H., & Xie, L. (2021). Padding oracle attack and defense: A survey. *IEEE Access*, 9, 162321–162335.