SECURE DOCUMENT VAULT

MULTI-FILE ENCRYPTION USING AES-RSA HYBRID MODEL

Disusun Oleh:

Azzam Rafi Budiman (0102523016)
Muhammad Delva Nadhif Fadihilah (0102523065)
Rafli Sujatmiko (0102523051)

Kelompok 10 (Informatika Angkatan 2023)

Universitas Al Azhar Indonesia

DAFTAR ISI

I. PENDAHULUAN

1.1 Latar Belakang

Dalam era transformasi digital saat ini, kebutuhan akan penyimpanan dokumen berbasis *cloud* meningkat pesat karena fleksibilitas akses yang ditawarkannya. Namun, kemudahan ini sering kali berbanding lurus dengan risiko keamanan, seperti kebocoran data (*data leakage*) dan akses tidak sah oleh pihak ketiga [1]. Organisasi maupun individu kini membutuhkan wadah penyimpanan digital atau "brankas digital" (*document vault*) yang tidak hanya sekadar menyimpan fail, tetapi juga menjamin kerahasiaan (*confidentiality*) dan integritas (*integrity*) data yang tersimpan di dalamnya.

Salah satu metode pertahanan utama untuk mengamankan data adalah enkripsi. Standar enkripsi simetris seperti *Advanced Encryption Standard* (AES) dikenal memiliki performa tinggi dan sangat efisien untuk mengenkripsi data dalam jumlah besar atau *bulk data* [2]. Meskipun cepat, AES memiliki kelemahan mendasar dalam manajemen kunci (*key management*); jika kunci enkripsi disadap saat proses pengiriman, maka seluruh keamanan data akan runtuh [3]. Di sisi lain, algoritma asimetris seperti RSA menawarkan keamanan kunci yang lebih baik menggunakan mekanisme *public* dan *private key*, namun memiliki komputasi yang sangat lambat jika dipaksa untuk mengenkripsi fail berukuran besar secara langsung [4].

Untuk mengatasi dilema *trade-off* antara kecepatan dan keamanan tersebut, pendekatan *Hybrid Cryptography* menjadi solusi yang paling relevan. Metode ini menggabungkan kecepatan AES untuk mengunci isi fail (*payload*) dan keamanan RSA untuk mengunci kunci AES itu sendiri (*key encapsulation*) [5].

Meskipun konsep hibrida ini sudah umum, tantangan teknis muncul ketika sistem harus menangani pengunggahan dokumen dalam jumlah banyak sekaligus (*multi-file batch processing*). Sebagian besar implementasi sederhana hanya berfokus pada enkripsi fail tunggal, yang menjadi tidak efisien ketika pengguna perlu mengamankan puluhan dokumen dalam satu waktu. Oleh karena itu, penelitian ini akan berfokus pada perancangan sistem "Secure Document Vault" berbasis REST API yang mengimplementasikan enkripsi hibrida AES-256 dan RSA-2048 dengan kemampuan pemrosesan *batch*, guna mengukur efektivitas keamanan serta dampak kinerjanya terhadap latensi sistem.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka rumusan masalah dalam tugas besar ini adalah:

1. Bagaimana merancang arsitektur *backend* Secure Document Vault yang menerapkan skema enkripsi hibrida (AES-RSA) untuk mengamankan penyimpanan fail?
2. Bagaimana mengimplementasikan mekanisme *multi-file encryption* dalam satu *request* (batch processing) untuk meningkatkan efisiensi pengunggahan dokumen?
3. Bagaimana kinerja sistem dari segi waktu komputasi (*computation time*) dan *throughput* saat menangani enkripsi AES mode CBC dibandingkan dengan mode GCM?

1.3 Tujuan Penelitian

Berdasarkan rumusan masalah yang telah dipaparkan, penelitian ini memiliki tujuan spesifik untuk mencapat target-target berikut:

1. Membangun Infrastruktur Backend Vault Merancang dan mengimplementasikan layanan REST API yang berfungsi sebagai brankas dokumen digital (*document vault*), di mana sistem mampu menangani proses enkripsi dan dekripsi fail secara aman menggunakan skema hibrida.
2. Implementasi Keamanan Berlapis Menerapkan standar algoritma AES-256 untuk mengamankan kerahasiaan isi fail (*payload*) dan memanfaatkan algoritma asimetris RSA untuk mengamankan kunci enkripsi tersebut, sehingga tercipta mekanisme pertukaran data yang aman.
3. Optimalisasi Pemrosesan Batch Mengembangkan fitur *multi-file request* yang memungkinkan pengguna mengunggah dan mengunduh banyak dokumen sekaligus dalam satu transaksi API, guna meningkatkan efisiensi operasional dibandingkan pemrosesan fail tunggal.
4. Analisis Performa Kriptografi Melakukan pengukuran kinerja (*benchmarking*) terhadap sistem yang dibangun, mencakup analisis *throughput* enkripsi, biaya komputasi per fail, serta perbandingan efisiensi antara mode operasi AES-CBC dan AES-GCM.

1.4 Batasan Masalah

Agar penelitian ini tetap terarah dan fokus pada aspek keamanan serta kinerja kriptografi, maka ditetapkan batasan-batasan masalah sebagai berikut:

Batasan	Keterangan
Lingkup Antarmuka	Pengembangan sistem difokuskan sepenuhnya pada sisi backend. Tidak ada pengembangan antarmuka grafis (UI/Frontend). Seluruh interaksi, mulai dari

(Interface)	pengiriman fail hingga penerimaan hasil dekripsi, disimulasikan menggunakan kakas pengujian API, yaitu POSTMAN.
Spesifikasi Algoritma	Modul kriptografi yang digunakan dibatasi pada AES-256 (dengan opsi mode CBC/GCM) untuk enkripsi simetris, RSA dengan panjang kunci 2048-bit atau 3072-bit untuk enkripsi kunci, serta SHA-256 untuk validasi integritas fail (hashing).
Mekanisme Penyimpanan	Sistem menyimpan fail terenkripsi (ciphertext) pada media penyimpanan lokal (local storage) atau cloud bucket, sedangkan metadata fail seperti nama, ukuran, timestamp, dan nilai hash disimpan dalam basis data terpisah (MongoDB atau PostgreSQL).
Skenario Pengujian	Pengujian fungsionalitas dan beban kerja dibatasi pada skenario batch dengan variasi jumlah fail (kecil, sedang, besar) dan variasi tipe fail (dokumen teks, PDF, gambar, dan fail biner) untuk menguji konsistensi enkripsi, tanpa berfokus pada analisis isi konten fail itu sendiri.
Fokus Keamanan	Sistem dirancang sebagai simulasi brankas dokumen, sehingga fitur keamanan jaringan level lanjut seperti proteksi DDoS atau Firewall berada di luar lingkup pengerjaan tugas ini. Fokus utama adalah pada keamanan data saat diam (data at rest) dan integritas fail.

II. TINJAUAN PUSTAKA

2.1 Kriptografi Hybrid

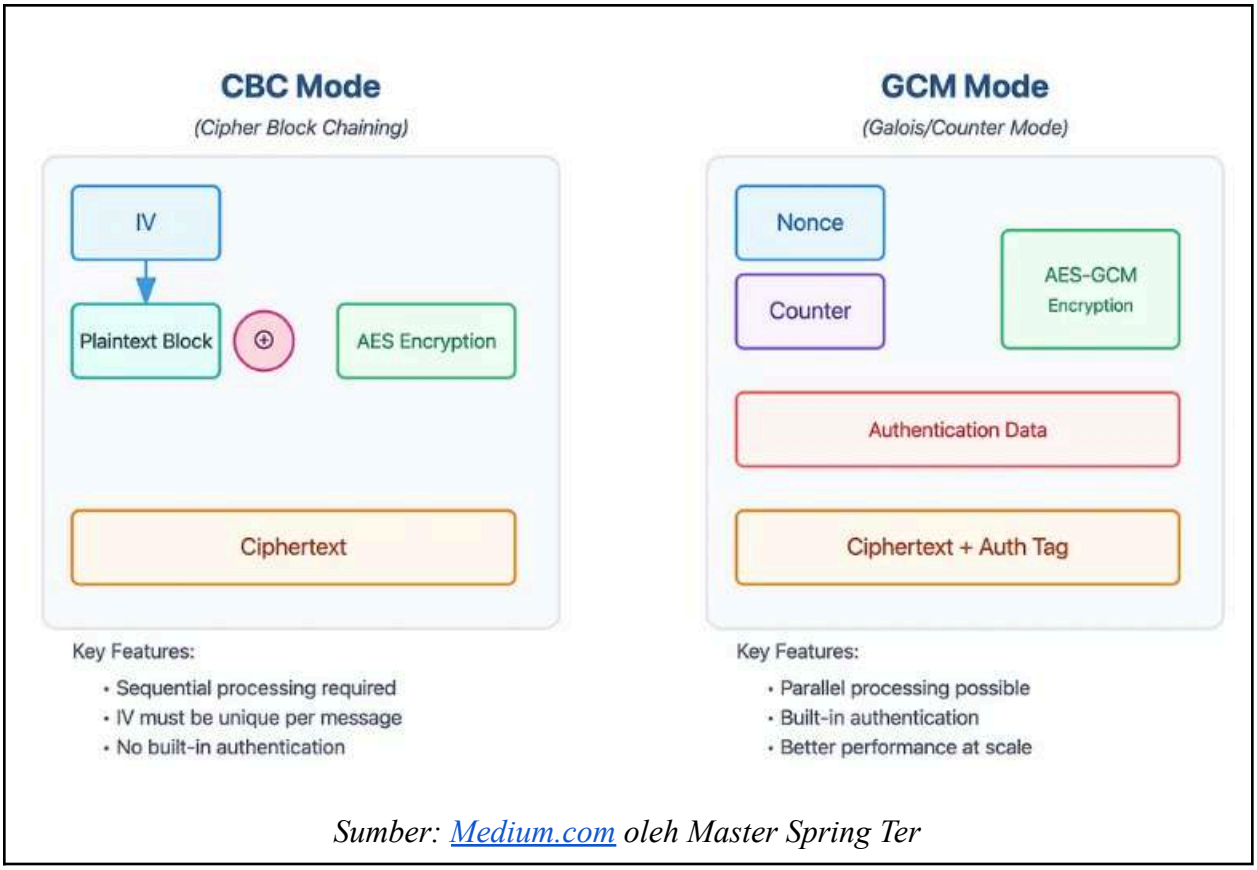
Dalam arsitektur keamanan sistem penyimpanan digital (*document vault*), penggunaan satu jenis algoritma kriptografi sering kali tidak cukup unekanisme Kriptografi Hybrid, sebuah pendekatan yang menggabungkan kecepatantuk memenuhi aspek keamanan dan performa secara bersamaan. Oleh karena itu, diterapkan m algoritma simetris dengan keamanan pertukaran kunci dari algoritma asimetris. Model ini bekerja dengan cara mengenkripsi data (*payload*) menggunakan kunci simetris sekali pakai (*session key*), kemudian kunci tersebut dikunci menggunakan algoritma asimetris sebelum disimpan atau ditransmisikan [6].

2.1.1 Advanced Encryption Standard (AES-256)

AES merupakan standar algoritma enkripsi simetris yang mengoperasikan data dalam blok 128-bit. Varian AES-256 menggunakan kunci sepanjang 256-bit, yang secara komputasi saat ini dianggap tahan terhadap serangan *brute-force* karena ruang kuncinya

yang sangat masif. Dalam implementasinya, AES tidak bekerja pada blok tunggal secara isolasi, melainkan menggunakan mode operasi tertentu.

- Mode Cipher Block Chaining (CBC): Mode ini merupakan pendekatan tradisional di mana setiap blok *plaintext* di-XOR dengan blok *ciphertext* sebelumnya sebelum dienkripsi. Proses ini membutuhkan *Initialization Vector* (IV) yang unik untuk setiap proses enkripsi agar pesan yang sama menghasilkan *ciphertext* yang berbeda. Kelemahannya adalah proses enkripsi bersifat sekuensial (tidak bisa paralel), sehingga cenderung lebih lambat untuk data berukuran besar.
- Mode Galois/Counter Mode (GCM): Berbeda dengan CBC, GCM mengubah blok menjadi *stream cipher* menggunakan pencacah (*counter*) dan operasi Galois Field. Keunggulan utama GCM adalah kemampuannya melakukan enkripsi secara paralel yang meningkatkan *throughput* secara signifikan, serta memiliki fitur *Authenticated Encryption* yang memverifikasi integritas data secara otomatis saat didekripsi [7].



Sumber: Medium.com oleh Master Spring Ter

2.1.2 Rivest–Shamir–Adleman (RSA)

RSA adalah algoritma asimetris yang mendasarkan keamanannya pada kesulitan faktorisasi bilangan bulat besar hasil perkalian dua bilangan prima. RSA menggunakan sepasang kunci: Kunci Publik (*Public Key*) yang disebarkan secara terbuka untuk mengenkripsi data, dan Kunci Privat (*Private Key*) yang dirahasiakan pemilik untuk mendekripsi data.

Dalam konteks sistem *vault* ini, RSA tidak digunakan untuk mengenkripsi fail secara langsung karena proses komputasinya yang lambat dan memakan sumber daya CPU yang tinggi. Peran RSA dibatasi hanya untuk mekanisme *Key Encapsulation*, yaitu mengamankan kunci AES-256 yang ukurannya jauh lebih kecil (32 bytes). Dengan demikian, efisiensi sistem tetap terjaga tanpa mengorbankan keamanan distribusi kunci [6].

2.1.3 Secure Hash Algorithm (SHA-256)

Untuk menjamin integritas data (*data integrity*), sistem menggunakan fungsi *hash* satu arah SHA-256. Algoritma ini memetakan data input dengan panjang berapapun menjadi *string* berukuran tetap 256-bit (32 byte).

Karakteristik utama SHA-256 yang dimanfaatkan dalam sistem ini adalah efek *avalanche*, di mana perubahan satu bit saja pada fail asli akan mengubah nilai *hash* keluaran secara drastis. Hal ini memungkinkan sistem untuk mendeteksi apakah sebuah fail telah dimodifikasi, korup, atau dirusak integritasnya selama proses penyimpanan di *vault*. Jika nilai *hash* fail saat diunduh berbeda dengan nilai yang disimpan di basis data, maka fail tersebut dianggap tidak valid [7].

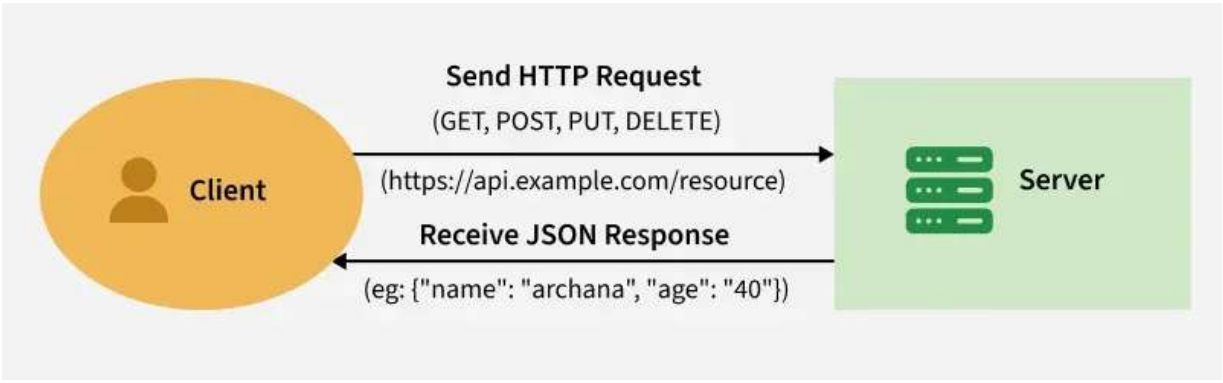
2.2 REST API & Batch Processing

Pengembangan sistem terdistribusi modern, seperti Secure Document Vault, memerlukan standar komunikasi yang ringan namun tetap tangguh untuk menangani pertukaran data antara klien dan server. Bagian ini membahas arsitektur REST yang menjadi tulang punggung sistem, serta mekanisme pengiriman data majemuk (multipart) yang memungkinkan fitur pemrosesan batch.

2.2.1 Konsep RESTful Service

Representational State Transfer (REST) bukanlah sebuah protokol, melainkan gaya arsitektur perangkat lunak yang memanfaatkan protokol HTTP (Hypertext Transfer Protocol) untuk pertukaran data. Prinsip utama REST adalah memandang setiap entitas data (dalam hal ini, dokumen terenkripsi) sebagai "sumber daya" (resource) yang dapat diakses melalui URI (Uniform Resource Identifier) yang unik.

Karakteristik fundamental yang membuat REST ideal untuk sistem vault ini adalah sifatnya yang stateless. Artinya, server tidak perlu menyimpan status sesi dari klien di antara dua permintaan (request) yang berbeda. Setiap permintaan HTTP yang dikirimkan oleh klien (seperti POST untuk enkripsi atau GET untuk dekripsi) harus memuat seluruh informasi yang dibutuhkan agar server dapat memprosesnya secara mandiri. Pendekatan ini sangat krusial untuk skalabilitas sistem; server dapat melayani lonjakan permintaan enkripsi tanpa terbebani oleh manajemen memori sesi yang kompleks [8].



Sumber: Rest API oleh GeeoksForGeeks

2.2.2 Mekanisme Multipart/Form-Data untuk Batch Upload

Salah satu fitur utama dalam penelitian ini adalah kemampuan batch processing, yaitu mengenkripsi banyak fail dalam satu kali eksekusi. Dalam protokol HTTP standar, pengiriman data biner (seperti fail gambar atau PDF) bersamaan dengan data teks (metadata) tidak dapat dilakukan dengan pengiriman formulir biasa (application/x-www-form-urlencoded). Solusi teknis untuk kebutuhan ini adalah penggunaan standar pengkodean multipart/form-data.

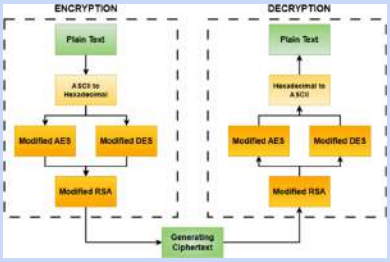
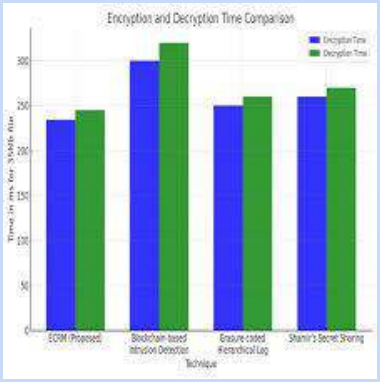
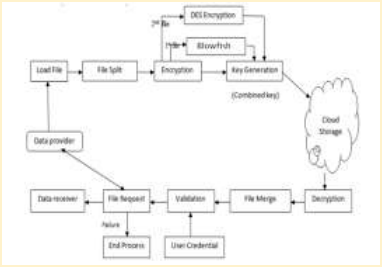
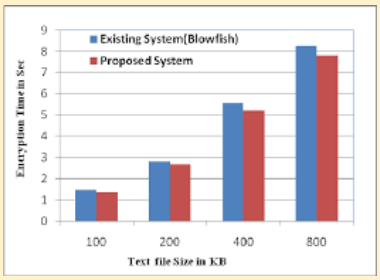
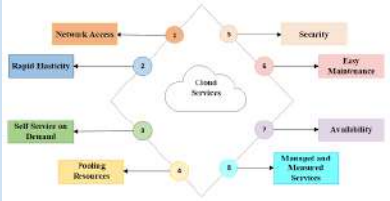
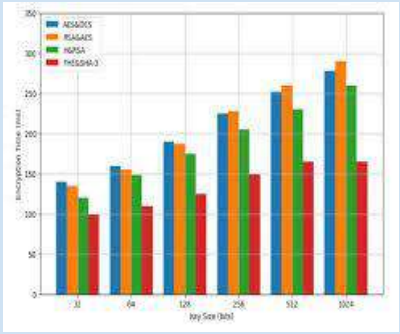
Mekanisme multipart bekerja dengan cara memecah badan permintaan (request body) menjadi beberapa bagian terpisah. Setiap bagian dipisahkan oleh penanda khusus yang disebut boundary. Hal ini memungkinkan satu permintaan HTTP POST untuk membawa muatan heterogen: misalnya, lima fail dokumen yang berbeda beserta metadata instruksi enkripsinya dalam satu paket pengiriman. Dari sisi performa, teknik ini jauh lebih efisien dibandingkan mengirimkan fail satu per satu. Dengan menggabungkan banyak fail ke dalam satu request (Batch I/O), sistem secara drastis mengurangi overhead jaringan seperti waktu jabat tangan (handshake) TCP/TLS yang biasanya terjadi berulang kali pada setiap permintaan terpisah [9].

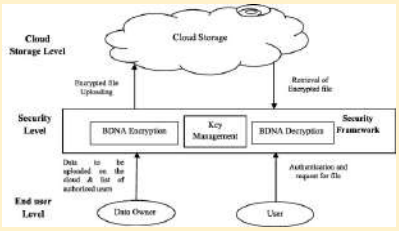
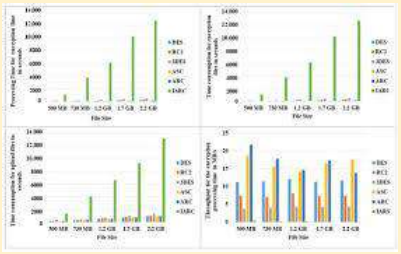
2.3 Tools Pengembangan

Komponen	Teknologi/Bahasa	Versi Kunci	Keterangan
Backend	Node.js	v18+	Runtime JavaScript server-side.
Framework	Express.js	4.x	Framework web minimalis untuk Node.js.
Database	MongoDB	(Lokal / Atlas)	Database NoSQL, diakses melalui Mongoose.
Middleware	Multer		Untuk menangani *multipart/form-data* (Upload File).
Environment	dotenv		Untuk manajemen variabel lingkungan (MONGODB_URI, PORT, dll).

2.4 Analisis Literatur

No.	Artikel	Problem	Metode dan Parameter	Hasil dan Evaluasi

1	A Hybrid Approach Using AES-RSA Encryption for Cloud Data Security Bharti, J. & Singh, S., 2024	 Keamanan data cloud rentan terhadap kebocoran dan akses tidak sah jika hanya menggunakan satu algoritma enkripsi. Penggunaan AES saja bermasalah pada distribusi kunci.	Metode kriptografi hibrida: AES untuk enkripsi data dan RSA untuk enkripsi kunci AES. Parameter: AES-256, RSA-2048.	 Hasil menunjukkan peningkatan keamanan data cloud tanpa penurunan performa signifikan. Sistem lebih aman dibandingkan metode tunggal dan cocok untuk penyimpanan file terenkripsi.
2	Secure File Storage Using Hybrid Cryptography Parathi, P. P., 2024	 Sistem penyimpanan file konvensional tidak mampu menjamin kerahasiaan dan integritas data secara optimal.	Hybrid Cryptography dengan kombinasi AES dan RSA serta hashing untuk integritas data. Parameter: AES-256, RSA, SHA-256.	 Evaluasi menunjukkan tingkat keamanan tinggi dan keberhasilan menjaga integritas data. Namun, penelitian masih berfokus pada enkripsi file tunggal.
3	RSA and AES Based Hybrid Encryption Technique for Enhancing Data Security in Cloud Computing Al-Nidah & Al-Bakri, 2023	 Enkripsi asimetris murni memiliki overhead komputasi tinggi untuk file besar.	Implementasi AES untuk data dan RSA untuk manajemen kunci. Parameter: AES-128/256, RSA-2048.	 Hasil evaluasi menunjukkan hybrid encryption lebih efisien dan aman dibanding RSA

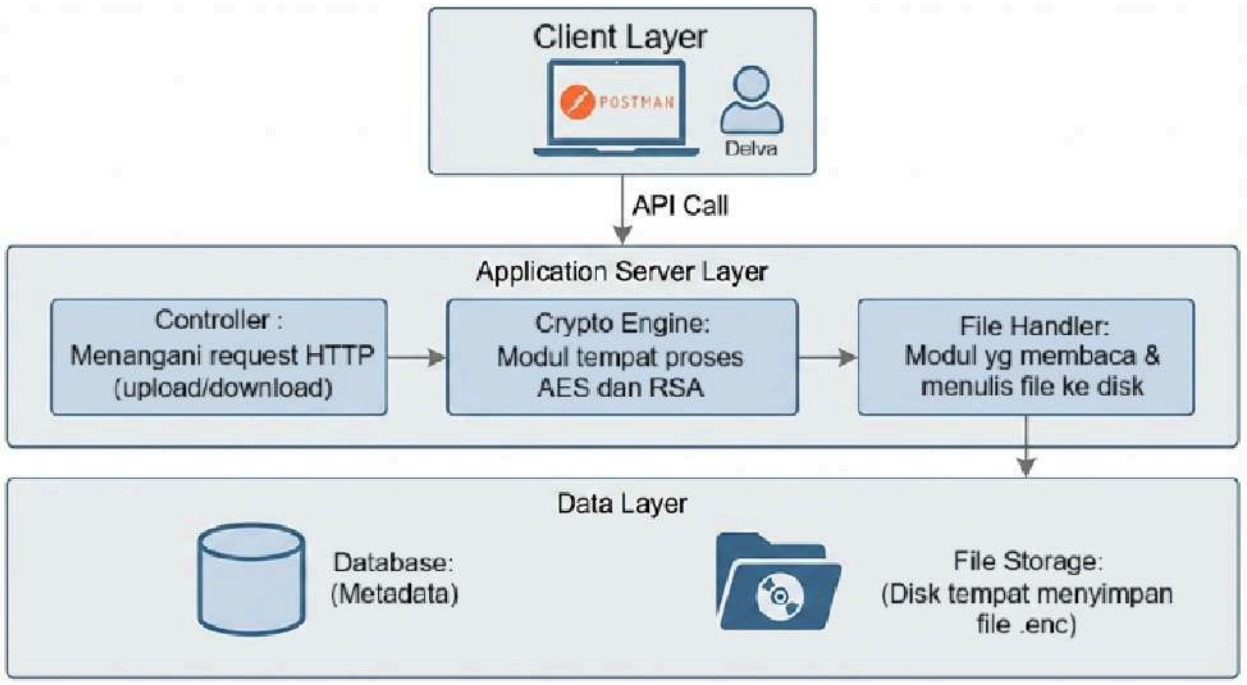
				murni, terutama untuk data berukuran besar.
4	Cryptographic Framework for Cloud-Based Document Storage Using AES-256 and SHA-256 Hybrid Systems Junaidi et al., 2025	 Kurangnya mekanisme validasi integritas pada sistem penyimpanan dokumen terenkripsi.	AES-256 untuk enkripsi data dan SHA-256 untuk validasi integritas file.	 Sistem berhasil mendeteksi perubahan data secara akurat. Namun belum membahas pemrosesan multi-file secara batch.

III. PERANCANGAN SISTEM

3.1 Arsitektur Vault

Di bagian ini, kami akan membagi menjadi dua visualisasi: Arsitektur Global (high level) dan Alur Kriptografi (detail).

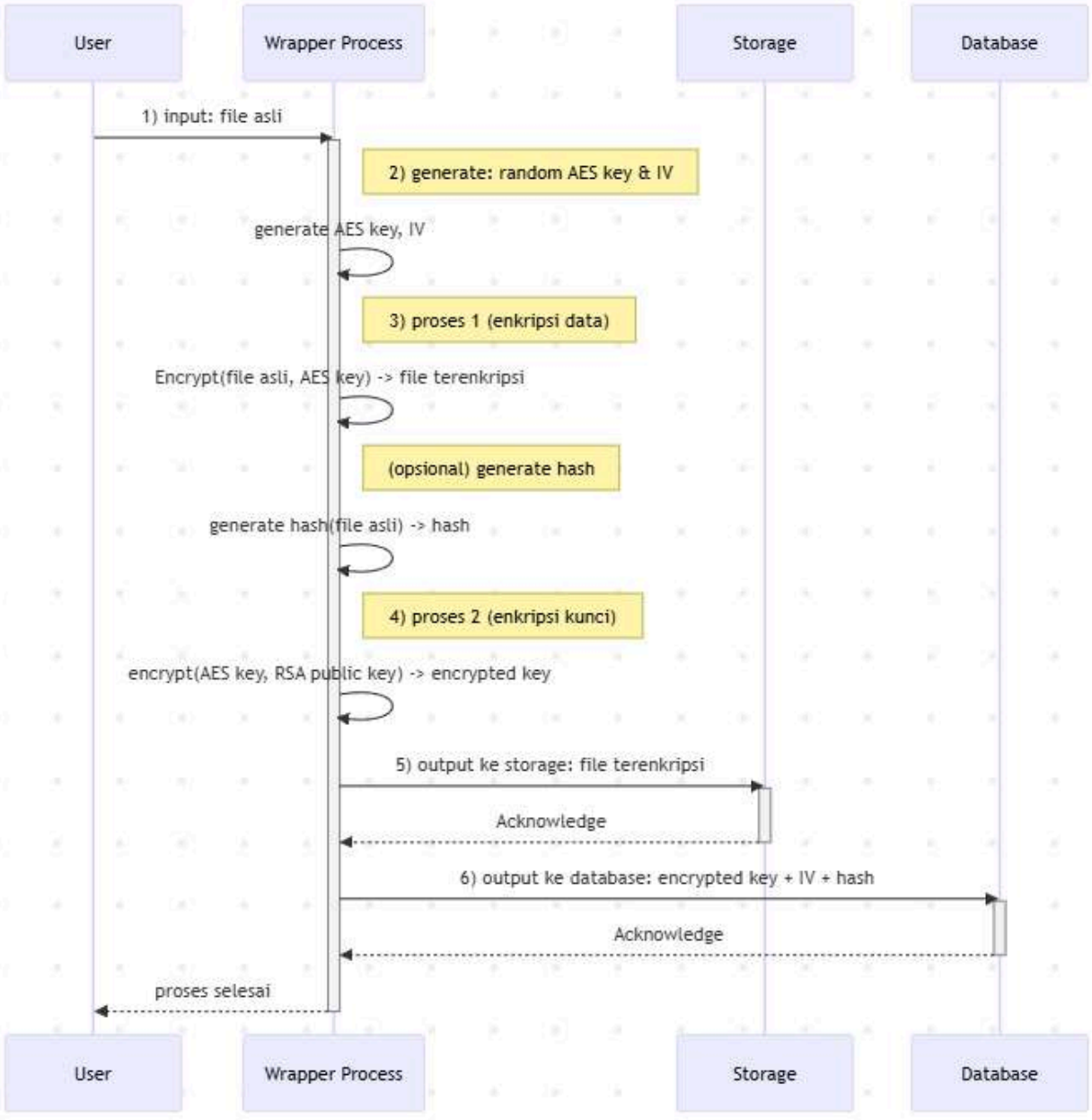
3.1.1 Arsitektur Sistem Global



Perancangan sistem *Secure Document Vault* ini mengadopsi arsitektur *Three-Tier* yang memisahkan sisi klien, logika pemrosesan, dan penyimpanan data. Mengingat sistem ini tidak terlalu memfokuskan antarmuka grafis (UI), interaksi pengguna disimulasikan sepenuhnya melalui klien REST (POSTMAN).

Pada sisi server, logika utama terpusat pada modul *Crypto Engine*. Modul ini bertugas mengisolasi proses kriptografi dari logika bisnis lainnya, memastikan bahwa data yang menyentuh lapisan penyimpanan (*storage layer*) sudah dalam keadaan terenkripsi. Sistem tidak menyimpan fail asli (*plaintext*) di dalam *disk*, melainkan hanya menyimpan fail yang telah dienkripsi (*ciphertext*) dan metadatanya di basis data terpisah. Pendekatan ini meminimalisir risiko kebocoran data apabila server fisik diretas, karena penyerang hanya akan mendapatkan fail acak tanpa kunci pembukanya.

3.1.2 Mekanisme Kriptografi Hibrida (Hybrid Encryption Flow)



Inti keamanan dari sistem *vault* ini terletak pada implementasi skema hibrida yg menggabungkan efisiensi AES-256 dan keamanan distribusi kunci RSA-2048. Alur enkripsinya dirancang untuk menangani kelemahan masing-masing algoritma jika digunakan secara terpisah.

Proses pengamanan sebuah dokumen berjalan melalui tahapan sekuensial berikut:

1. Untuk setiap permintaan pengunggahan fail, sistem secara otomatis membangkitkan kunci simetris (AES Key) 32-byte dan *Initialization Vector* (IV) 16-byte secara acak. Kunci ini bersifat *ephemeral* (hanya digunakan untuk satu sesi fail tersebut).
2. Isi dokumen (*plaintext*) dienkripsi menggunakan algoritma AES-256 dengan kunci sesi yang baru dibuat. Hasilnya adalah fail *ciphertext* yang tidak dapat dibaca.
3. Kunci sesi AES yang digunakan pada tahap sebelumnya kemudian "dibungkus" atau dienkripsi menggunakan Kunci Publik RSA milik server. Hasilnya adalah *Encrypted AES Key*.
4. Sistem menyimpan *Encrypted AES Key* ke dalam basis data metadata, sementara fail *ciphertext* disimpan ke dalam direktori penyimpanan. Kunci AES asli kemudian dihapus dari memori (RAM) segera setelah proses selesai untuk mencegah *memory dump attack*.

3.1.2 Mekanisme Kriptografi Hibrida (Hybrid Encryption Flow)

Selain arsitektur fisik dan logika keamanan, perancangan perangkat lunak pada sistem ini menerapkan pola desain modular yang mengadopsi konsep *Model-View-Controller* (MVC). Mengingat sistem ini beroperasi sebagai *backend service* tanpa antarmuka pengguna (UI) langsung, komponen "View" direpresentasikan dalam bentuk respons JSON standar. Struktur ini dipilih untuk memisahkan tanggung jawab antara penanganan permintaan HTTP, logika bisnis kriptografi, dan manajemen data.

Implementasi modular dalam direktori **`src/`** dibagi menjadi beberapa lapisan fungsional utama sebagai berikut:

1. Direktori `controllers/` bertindak sebagai otak operasional yang menjembatani input pengguna dengan logika sistem. Di sinilah proses validasi input terjadi sebelum data diteruskan ke mesin enkripsi. Misalnya, logika untuk inisiasi proses unggah fail dan pemanggilan fungsi enkripsi dikelola secara terpusat di `vaultController.js`, sementara pengelolaan kunci RSA ditangani terpisah di `keyController.js` .
2. Untuk menjaga kebersihan kode, pendefinisian titik akhir API (*endpoints*) dipisahkan ke dalam direktori `routes/`. Modul ini hanya bertugas mengarahkan permintaan HTTP (seperti POST atau GET) ke *controller* yang relevan, tanpa

memuat logika pemrosesan data apa pun. Hal ini memudahkan pemetaan URL seperti `/vault/upload` agar tertata rapi .

3. Interaksi dengan basis data MongoDB diabstraksi menggunakan *Mongoose Schema* di dalam direktori `models/`. Lapisan ini mendefinisikan struktur metadata fail seperti tipe data untuk menyimpan kunci AES terenkripsi dan *hash* integritas untuk memastikan konsistensi data yang disimpan.
4. Fungsi-fungsi kriptografi inti, seperti pembangkitan pasangan kunci RSA, diisolasi dalam folder `utils/` agar dapat digunakan ulang (*reusable*). Selain itu, penanganan aliran data fail (*file stream*) dari sisi klien dikelola oleh *middleware* khusus (menggunakan Multer) sebelum menyentuh logika utama controller.

Pemisahan struktur ini sangat krusial dalam konteks keamanan. Dengan mengisolasi logika kriptografi di `utils` dan logika bisnis di `controllers`, risiko kesalahan implementasi (*bug*) yang dapat mengekspos kunci enkripsi dapat diminimalisir dibandingkan jika semua kode ditulis dalam satu fail monolitik.

3.2 Perancangan Database

Manajemen data pada sistem *vault* tidak menyatukan fail fisik dengan informasinya. Basis data hanya berfungsi untuk menyimpan metadata administratif dan komponen kriptografis yang diperlukan untuk proses dekripsi. Sesuai dengan spesifikasi kebutuhan sistem untuk mencatat integritas dan properti fail, berikut adalah rancangan skema tabel/dokumen files:

Tabel 3.2 Skema Metadata File

Nama Atribut	Data Tipe	Deskripsi & Fungsi
id	UUID / string	Identifier unik untuk setiap dokumen yg disimpan
originalName	String	Nama asli file sebelum dienkripsi (misal: *laporan.pdf*)
encryptedName	String	Nama file yang telah disamarkan/diubah untuk penyimpanan fisik.
mimeType	String	Tipe format file (misal: *application/pdf*) untuk keperluan *restore* saat unduhan
size	Integer (bytes)	Ukuran file asli untuk verifikasi kelengkapan data
hash	String (Hex)	Nilai *hash* SHA-256 dari file asli. Buat memvalidasi integritas data saat file diunduh kembali

encryptedAESKey	String (Base64)	Kunci AES yang telah dienkripsi oleh RSA. Ini adalah komponen paling krusial untuk disimpan
iv	String (Hex/Base64)	*Initialization Vector* yang digunakan saat enkripsi AES. Wajib disimpan untuk proses dekripsi (terutama pada mode CBC/GCM).
authTag	String / Null	*Tag autentikasi (digunakan jika menggunakan mode GCM, null jika CBC).*
storage_path	String	Lokasi path direktori tempat fail *ciphertext* fisik disimpan di server
uploadedAt	Timestamp (ISO 8601)	Waktu saat file berhasil diunggah dan dicatat ke dalam *vault*.
version	Integer	Versi file (digunakan jika ada fitur *update* file yang sama).
isActive	Boolean	Status apakah file masih aktif atau sudah dihapus secara logis.
encryptedSize	Integer (bytes)	Ukuran file setelah dienkripsi (biasanya lebih besar karena *padding*).
encryptionMode	String	Mode algoritma enkripsi yang digunakan (contoh: `AES_CBC`).

Skema di atas dirancang buat mendukung operasi bulk atau batch. Ketika pengguna mengunggah banyak fail sekaligus, sistem akan membuat satu entri (baris dokumen) baru untuk setiap fail dalam batch tersebut, di mana masing-masing memiliki encryptedAESKey dan iv yang unik untuk menjamin isolasi keamanan antar file.

3.3 Perancangan Endpoint (API)

Interaksi antara klien dan server dirancang menggunakan standar arsitektur RESTful. Spesifikasi endpoint dibagi menjadi dua segmen utama: segmen manajemen kunci kriptografi dan segmen operasional brankas dokumen (vault operations). Seluruh pertukaran data menggunakan format JSON untuk metadata, sementara pengiriman fail fisik ditangani melalui mekanisme multipart/form-data. Berikut adalah rincian spesifikasi interface yang dibangun:

3.3.1 Endpoint Manajemen Kunci (Key Management)

Segmen ini bertanggung jawab atas inisiasi dan verifikasi infrastruktur keamanan sebelum operasi fail dapat dilakukan. Kunci RSA yang dibangkitkan di sini akan menjadi fondasi bagi enkripsi hibrida selanjutnya.

Tabel 3.3.1 Spesifikasi Endpoint Manajemen Kunci

Metode	Endpoint URI	Fungsi Controller	Deskripsi Teknis
POST	/keys/generate	generateKeys	Memicu pembangkitan pasangan kunci RSA-2048 (Publik & Privat) di server. Endpoint ini wajib dijalankan satu kali saat inisialisasi sistem untuk memastikan server siap melakukan enkripsi kunci AES.
GET	/keys/check	checkKeys	Melakukan verifikasi ketersediaan kunci RSA di direktori server. Digunakan sebagai *health-check* untuk memastikan sistem kriptografi berjalan sebelum pengguna mencoba mengunggah dokumen.

3.3.2 Endpoint Operasional Vault (Document Operations)

Segmen ini menangani logika bisnis utama, mulai dari enkripsi saat pengunggahan hingga dekripsi saat pengunduhan. Sesuai dengan persyaratan sistem untuk efisiensi, tersedia *endpoint* khusus untuk menangani banyak fail sekaligus (*batch processing*).

Tabel 3.3 Spesifikasi Endpoint Operasional Vault

Metode	Endpoint URI	Fungsi Controller	Deskripsi Teknis
POST	/vault/upload	uploadFile	Menerima input fail tunggal. Sistem akan melakukan: (1) hashing SHA-256 fail asli, (2) enkripsi payload dengan AES-256, (3) enkripsi kunci AES dengan RSA, dan (4) penyimpanan metadata ke basis data
POST	/vault/upload-batch	uploadBatch	Menerima array fail (maksimal 10 fail per request). Proses enkripsi dilakukan secara iteratif untuk setiap fail. Setiap

			fail memiliki kunci AES dan IV yg unik untuk keamanan maksimal
GET	/vault/list	listFiles	Mengambil seluruh daftar metadata dokumen yang tersimpan di MongoDB. Respons hanya berisi informasi publik (nama, ukuran, tanggal, hash) tanpa menyertakan kunci enkripsi
GET	/vault/download/:id	downloadFile	Mengunduh fail berdasarkan ID. Server mendekripsi kunci AES menggunakan RSA Privat, lalu mendekripsi fail fisik sebelum dikirim kembali ke klien dalam bentuk asli
DELETE	/vault/delete/:id	deleteFile	Menghapus data secara permanen, meliputi penghapusan metadata dari MongoDB dan penghapusan fail terenkripsi (`.enc`) dari filesystem server

3.4 Skenario Pengujian

Pengujian pada sistem *Secure Document Vault* ini difokuskan pada validasi fungsionalitas mekanisme hybrid encryption (AES-RSA) dan pengukuran performa sistem dalam menangani beban kerja melalui skenario *batch upload*. Fokus pengujian adalah pada keragaman jumlah serta tipe file guna menguji konsistensi enkripsi, tanpa melakukan analisis terhadap isi konten file tersebut.

3.4.1 Variasi Dataset Pengujian

Dataset yang digunakan mencakup total 18-20 file dengan rincian sebagai berikut:

- **Dokumen Teks (10 file):** Digunakan untuk pengujian *batch encryption* dasar.
- **PDF (3-4 file):** Digunakan untuk *medium-size stress test*.
- **Gambar (3-4 file):** Digunakan untuk *large-size test*.
- **File Biner (2-3 file):** Mencakup format ZIP, PPTX, atau Excel untuk menguji kompatibilitas sistem.
- **File Sensitif Dummy:** Mencakup rekam medis palsu dan transkrip nilai palsu sebagai representasi penggunaan *vault* pada data sensitif.

3.4.2 Skenario Beban Kerja (Batch Scenario)

Sistem diuji dengan tiga skenario beban kerja berdasarkan jumlah file dalam satu permintaan (*request*):

1. **Batch Kecil (3 File):** Pengujian awal untuk memastikan integritas proses hybrid encryption pada beban minimal.
2. **Batch Sedang (5 File):** Pengujian stabilitas sistem dalam menangani beberapa tipe file sekaligus.
3. **Batch Besar (10+ File):** Pengujian intensif untuk mengukur *multi-file throughput* dan mengidentifikasi potensi *bottleneck* pada proses enkripsi RSA.

3.4.3 Parameter Keberhasilan Pengujian

Untuk setiap skenario di atas, keberhasilan pengujian diukur berdasarkan kriteria berikut:

- **Validasi Integritas:** Nilai *hash* (SHA-256) file asli harus sesuai dengan nilai *hash* file setelah didekripsi.
- **Konsistensi Metadata:** Sistem harus mampu mencatat ukuran, tipe, *hash*, dan *timestamp* secara akurat di database untuk setiap file dalam batch.
- **Analisis Mode AES:** Keberhasilan enkripsi pada kedua mode operasi yang didukung, yaitu CBC dan GCM.
- **Benchmark Performa:** Pencatatan waktu yang dibutuhkan untuk *per-file encryption cost* dan *RSA overhead* dalam proses hybrid.

IV. IMPLEMENTASI DAN ANALISIS

4.1 Lingkungan Implementasi

Tahap realisasi sistem Secure Document Vault diawali dengan penyiapan ekosistem pengembangan yang mampu mendukung operasi kriptografi intensif serta penanganan data biner

secara asinkron. Lingkungan implementasi ini mencakup spesifikasi perangkat lunak inti serta pengorganisasian struktur kode yang diadopsi untuk memastikan sistem berjalan secara modular dan aman.

4.1.1 Spesifikasi Teknologi dan Pustaka

Infrastruktur *backend* dibangun di atas platform Node.js versi 18 (LTS). Pemilihan versi ini didasarkan pada stabilitas jangka panjang serta dukungan optimal terhadap modul kriptografi bawaan yang diperlukan untuk operasi AES dan RSA. Sebagai kerangka kerja utama, sistem memanfaatkan Express.js versi 4.x yang berfungsi menangani perutean HTTP (*routing*) secara efisien tanpa membebani kinerja server dengan fitur-fitur yang tidak perlu.

Untuk manajemen data persisten, sistem terintegrasi dengan basis data NoSQL MongoDB. Interaksi antara aplikasi dan basis data dijembatani oleh *Object Data Modeling* (ODM) Mongoose, yang memungkinkan definisi skema metadata fail secara ketat namun tetap fleksibel. Selain itu, penanganan aliran data fail (*file stream*) dari sisi klien dikelola menggunakan *middleware* Multer, yang bertugas menangani protokol `multipart/form-data` saat proses pengunggahan berlangsung.

4.1.2 Organisasi Struktur Direktori

Agar pengembangan tetap terstruktur dan mudah dipelihara (*maintainable*), kode sumber proyek diorganisasikan dengan memisahkan logika bisnis, rute akses, dan manajemen data. Struktur direktori fisik dari sistem *vault* ini adalah sebagai berikut:

```
secure-document-vault/
├── node_modules/
├── uploads/ # tempat file sementara Multer (dihapus setelah dienkripsi)
├── src/
│   ├── controllers/ # logika bisnis
│   │   ├── keyController.js #  buat generate & cek kunci RSA
│   │   └── vaultController.js # upload, list, download, delete dokumen
│   ├── middleware/
│   │   └── upload.js
│   ├── models/
│   │   └── File.js
│   ├── routes/ # routing API
│   │   ├── keyRoutes.js #  API untuk kunci (e.g., /keys/generate).
│   │   └── vaultRoutes.js # API untuk dokumen (e.g., /vault/upload).
│   └── utils/ # utilitas & kriptografi
```

```
│       └── generateKeys.js # untuk buat kunci RSA.
├── package-lock.json # [VERIFIKASI] dibuat otomatis oleh npm.
├── .env
├── package.json
└── server.js
```

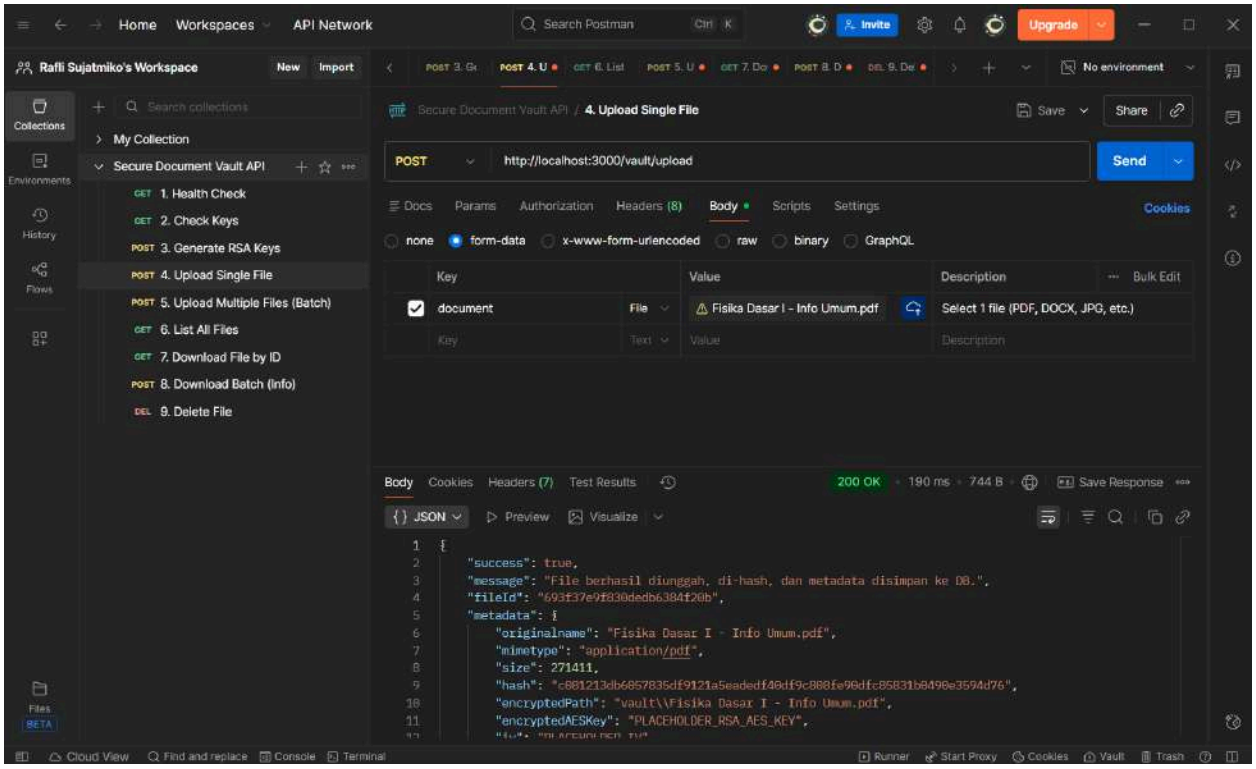
4.2 Bukti Implementasi Fitur

Pengujian fungsionalitas dilakukan secara bertahap menggunakan kakas Postman untuk memverifikasi kesiapan arsitektur *backend* sebelum modul kriptografi diintegrasikan sepenuhnya. Fokus pengujian tahap ini adalah memastikan aliran data dari klien ke server dan interaksi dengan basis data MongoDB berjalan lancar.

4.2.1 Verifikasi Pengunggahan Dokumen (*Single Upload*)

Pengujian pertama dilakukan pada *endpoint* POST /vault/upload untuk memvalidasi kinerja *middleware* Multer dalam menangani *multipart form-data*.

- **Tujuan:** Memastikan fail berhasil diterima oleh server, dihitung nilai *hash*-nya, dan metadatanya tersimpan dalam koleksi MongoDB.
- **Hasil Pengujian:** Server mengembalikan respons HTTP 200 (OK) beserta objek JSON yang berisi detail metadata fail yang baru dibuat.

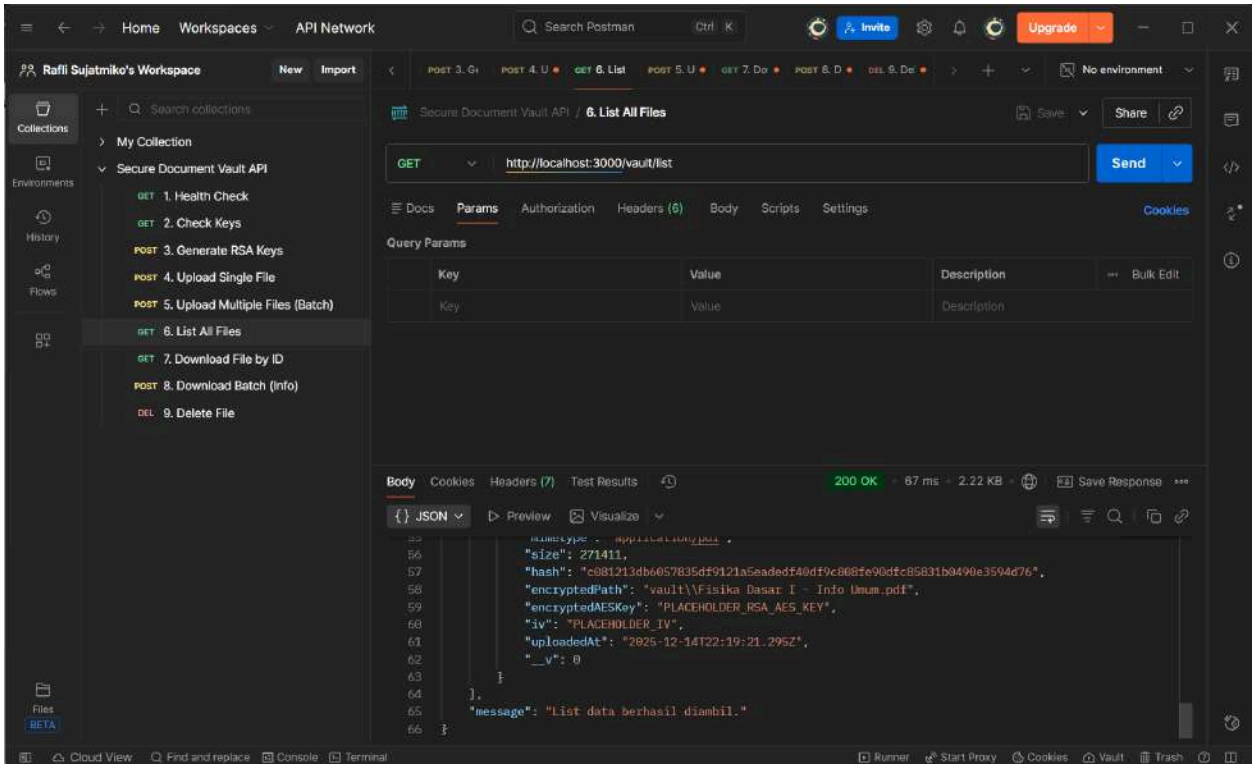


Gambar 4.2.1 Hasil pengujian unggah file tunggal

4.2.2 Verifikasi Pengambilan Daftar Metadata (List Files)

Pengujian selanjutnya menargetkan endpoint GET /vault/list untuk menguji fungsi retrieval data menggunakan Mongoose.

- **Tujuan:** Memastikan sistem mampu melakukan query ke basis data dan menampilkan daftar dokumen yang tersimpan.
- **Hasil Pengujian:** Server berhasil menampilkan array objek JSON yang berisi daftar metadata fail yang telah diunggah sebelumnya.

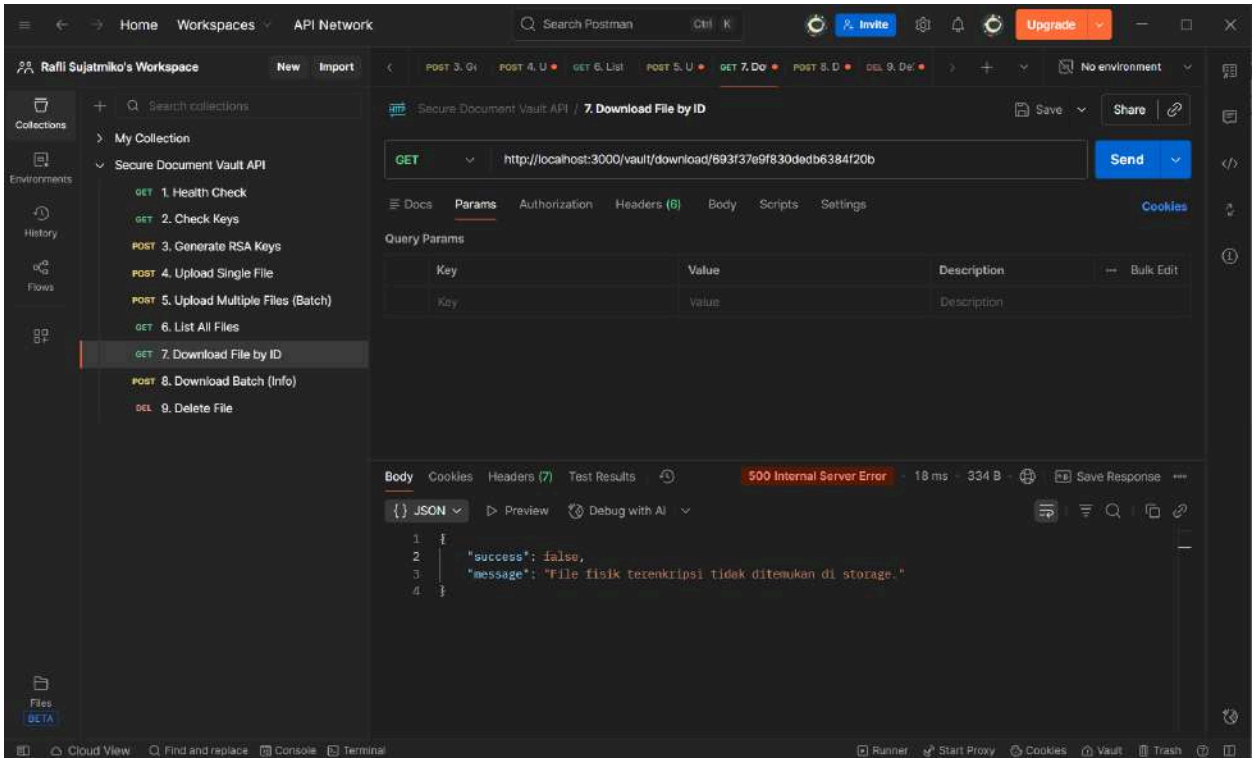


Gambar 4.2.2 Hasil pengujian pengambilan daftar metadata

4.2.3 Analisis Logika Pengunduhan (Download Logic Analysis)

Pengujian pada endpoint GET /vault/download/:id menghasilkan respons spesifik yang menjadi indikator penting bagi status pengembangan sistem saat ini.

- Skenario: Mengakses fail menggunakan ID valid yang didapatkan dari proses List Files.
- Respons Sistem: {"success": false, "message": "File fisik terenkripsi tidak ditemukan di storage."} dengan status HTTP 500.



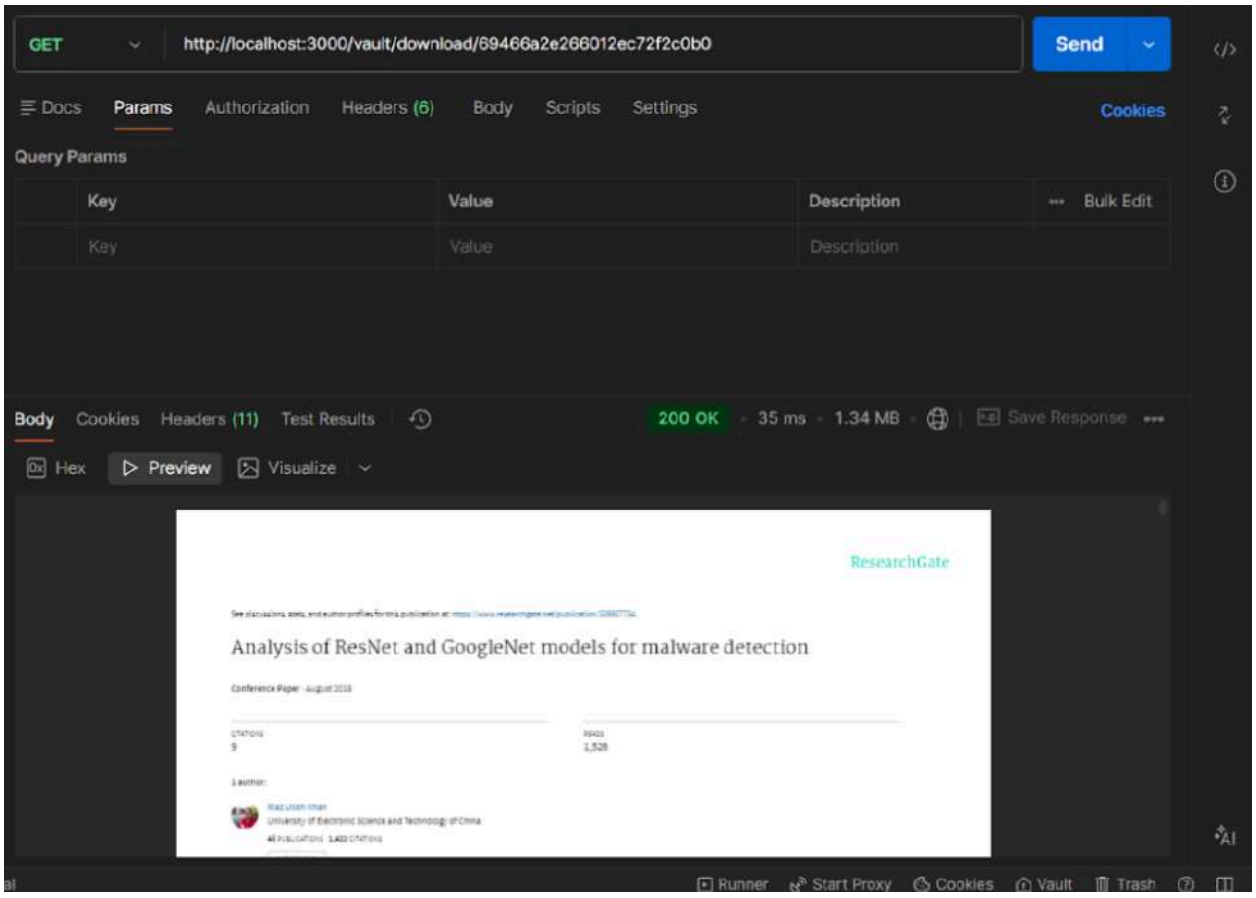
Gambar 4.2.3 Respons sistem pada pengujian unduhan

4.2.4 Analisis Logika Pengunduhan (*Download Logic Analysis*)

Pengujian pada endpoint GET /vault/download/:id dilakukan untuk memvalidasi alur dekripsi dan restorasi file dari *vault*. Berdasarkan hasil pengujian di Postman, sistem berhasil mengembalikan file asli dengan status 200 OK dan waktu respons 35 ms untuk file berukuran 1.34 MB.

- Skenario: Mengakses fail menggunakan ID valid yang diperoleh dari proses *List Files*.
- Respons Sistem (Sukses): Sistem melakukan *key unwrapping* RSA untuk mendapatkan kunci AES, mendekripsi file fisik, dan menampilkan *preview* dokumen secara utuh.
- Respons Sistem (Gagal): Jika file fisik tidak sinkron dengan metadata, sistem memberikan respons {"success": false, "message": "File fisik terenkripsi tidak ditemukan di storage."} dengan status HTTP 500.

Analisis ini menunjukkan bahwa logika dekripsi hybrid telah berjalan dengan benar, namun tetap memerlukan sinkronisasi yang ketat antara database metadata dan penyimpanan fisik.



Gambar 4.2.4 Respons sistem pada pengujian unduhan setelah implementasi enkripsi

4.2.5 Upload Batch Small (PDF & File Biner 3 files)

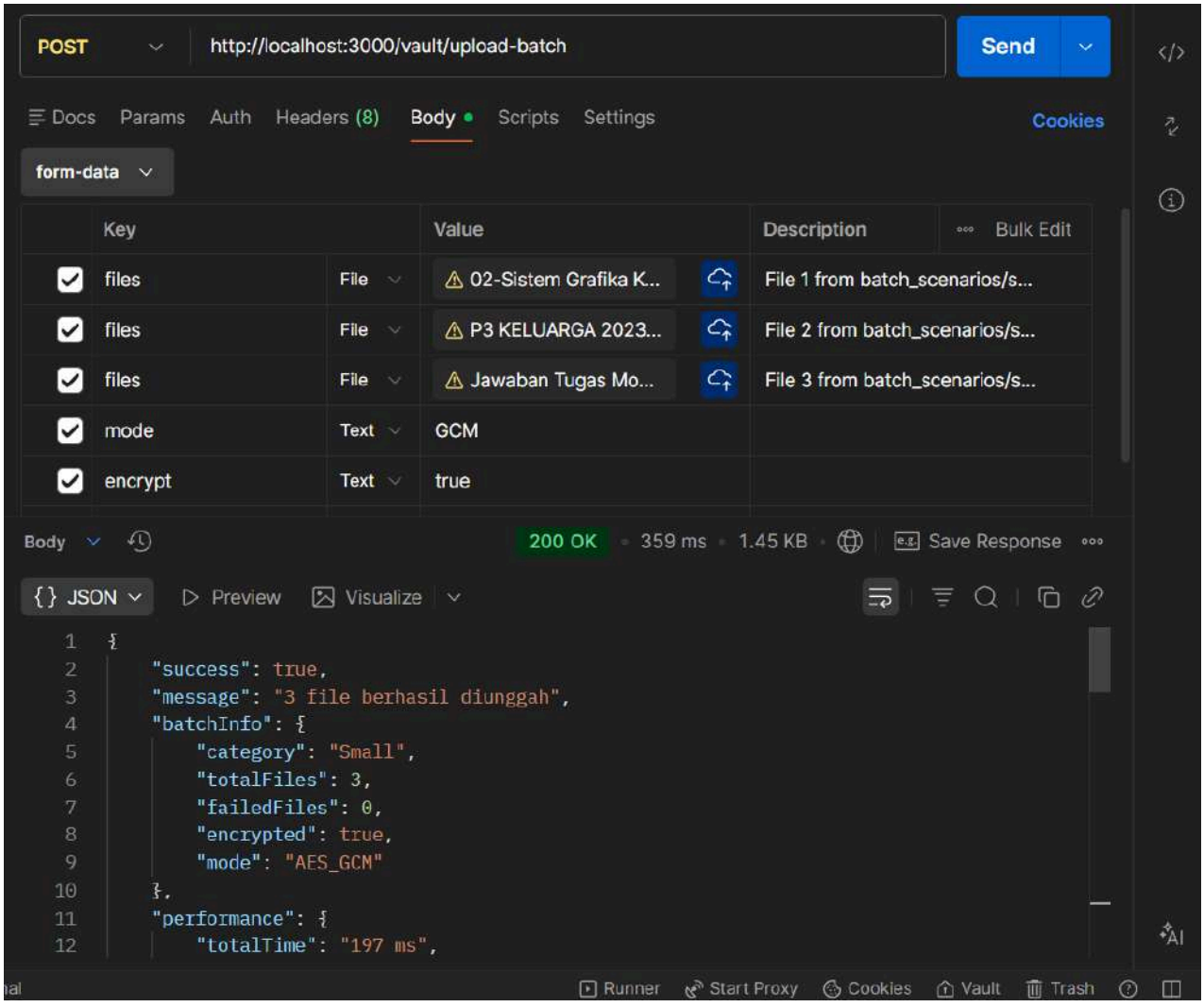
Pengujian ini membandingkan efisiensi proses enkripsi hibrida antara mode AES-CBC dan AES-GCM terhadap tiga jenis file yang berbeda (`.pdf`, `.ppt`, dan `.zip`).

- **Kecepatan Pemrosesan:** Mode **GCM** menunjukkan performa yang jauh lebih responsif dengan total waktu proses hanya **197 ms**, dibandingkan mode **CBC** yang memerlukan waktu **425 ms**.
- **Analisis Selisih Waktu (*System Overhead*):** Terdapat perbedaan antara waktu enkripsi murni dan total waktu proses. Sebagai contoh pada mode GCM, waktu enkripsi murni berkisar 5–51 ms, namun total waktu mencapai 37–90 ms per file. Selisih ini merupakan *overhead* sistem untuk manajemen I/O, koneksi database, dan *hashing*.
- **Efisiensi Penyimpanan:** Mode **GCM** mencatatkan *overhead* sebesar **0 bytes**

pada seluruh file, sehingga ukuran *ciphertext* identik dengan file asli. Sebaliknya, mode **CBC** menghasilkan penambahan ukuran (hingga 16 bytes) akibat mekanisme *padding* blok.

- **Keamanan Terintegrasi:** Berbeda dengan CBC, mode **GCM** menyertakan fitur *Authenticated Encryption* yang memungkinkan sistem mendeteksi manipulasi data secara otomatis melalui *authentication tag*.

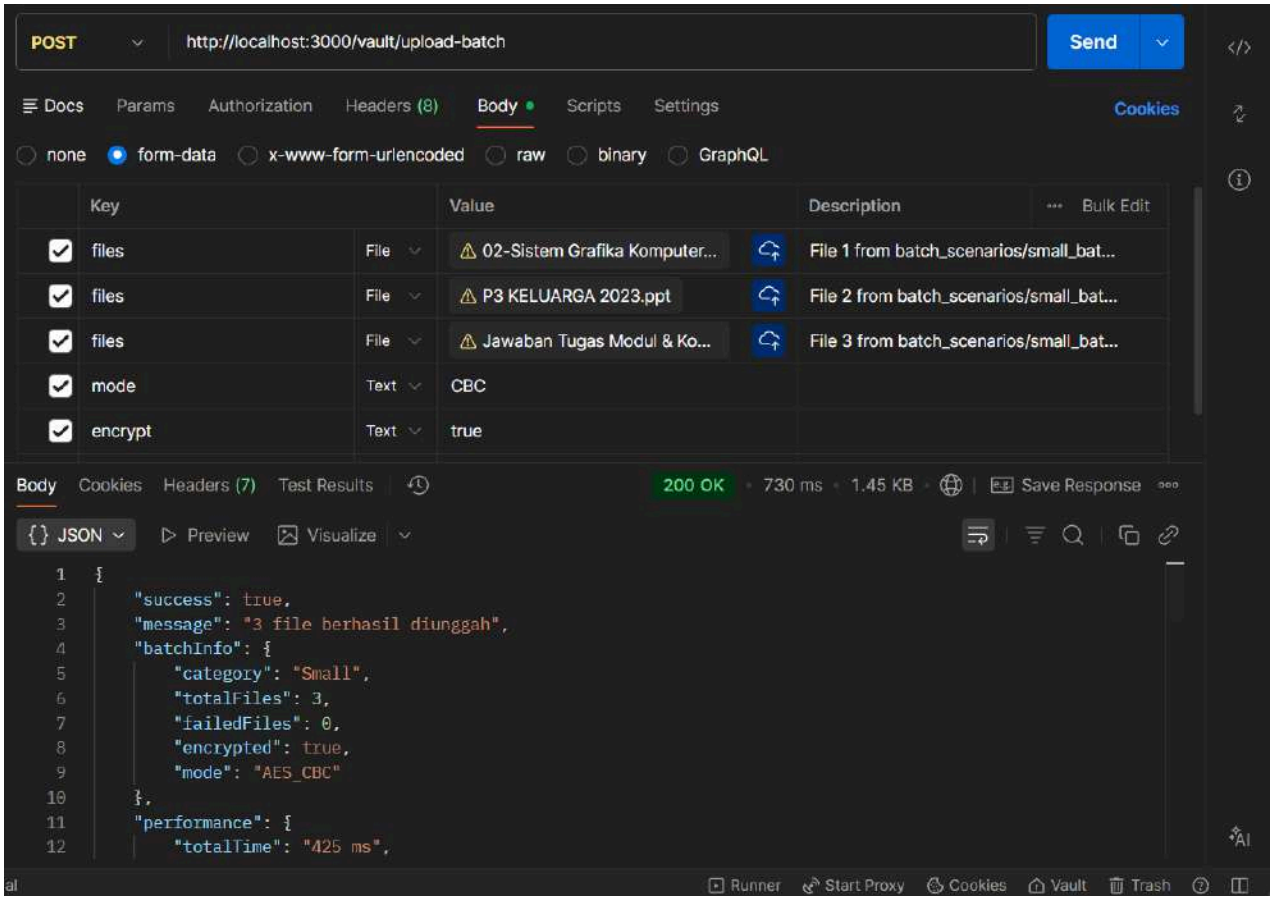
Kesimpulan: Data pada tabel menunjukkan bahwa selisih waktu proses tetap berada dalam batas wajar, menandakan infrastruktur *backend* bekerja harmonis dengan modul kriptografi. Mode **GCM** secara konsisten menjadi pemenang baik dari sisi kecepatan maupun efisiensi ruang penyimpanan.



Gambar 4.2.5.1 Respons sistem pada pengujian upload batch small mode GCM (PDF & File Biner 3 files)

No	Nama File	Tipe	Ukuran Asli (KB)	Ukuran Terenkripsi (KB) Mode GCM	Overhead (bytes) Mode GCM	Waktu Enkripsi (ms) Mode GCM	Total Waktu Proses (ms) Mode GCM
1	02-Sistem Grafika Komputer .pdf	PDF	947.28	947.28	0	12	53
2	P3 KELUARGA 2023.ppt	PPT	5449.00	5449.00	0	51	90
3	Jawaban Tugas Modul & Kode Pythonnya - Rafli Sujatmiko_0102523051.zip	ZIP	802.67	802.67	0	5	37

Tabel 4.2.5.1 Perhitungan fitur pada pengujian upload batch small mode GCM (PDF & File Biner 3 files)



Gambar 4.2.5.2 Respons sistem pada pengujian upload batch small mode CBC (PDF & File Biner 3 files)

No	Nama File	Tipe	Ukuran Asli (KB)	Ukuran Terenkripsi (KB) Mode CBC	Overhead (bytes) Mode CBC	Waktu Enkripsi (ms) Mode CBC	Total Waktu Proses (ms) Mode CBC
1	02-Sistem Grafika Komputer .pdf	PDF	947.28	947.28	2	46	244
2	P3 KELUARGA 2023.ppt	PPT	5449.00	5449.02	16	107	150
3	Jawaban	ZIP	802.67	802.67	6	122	26

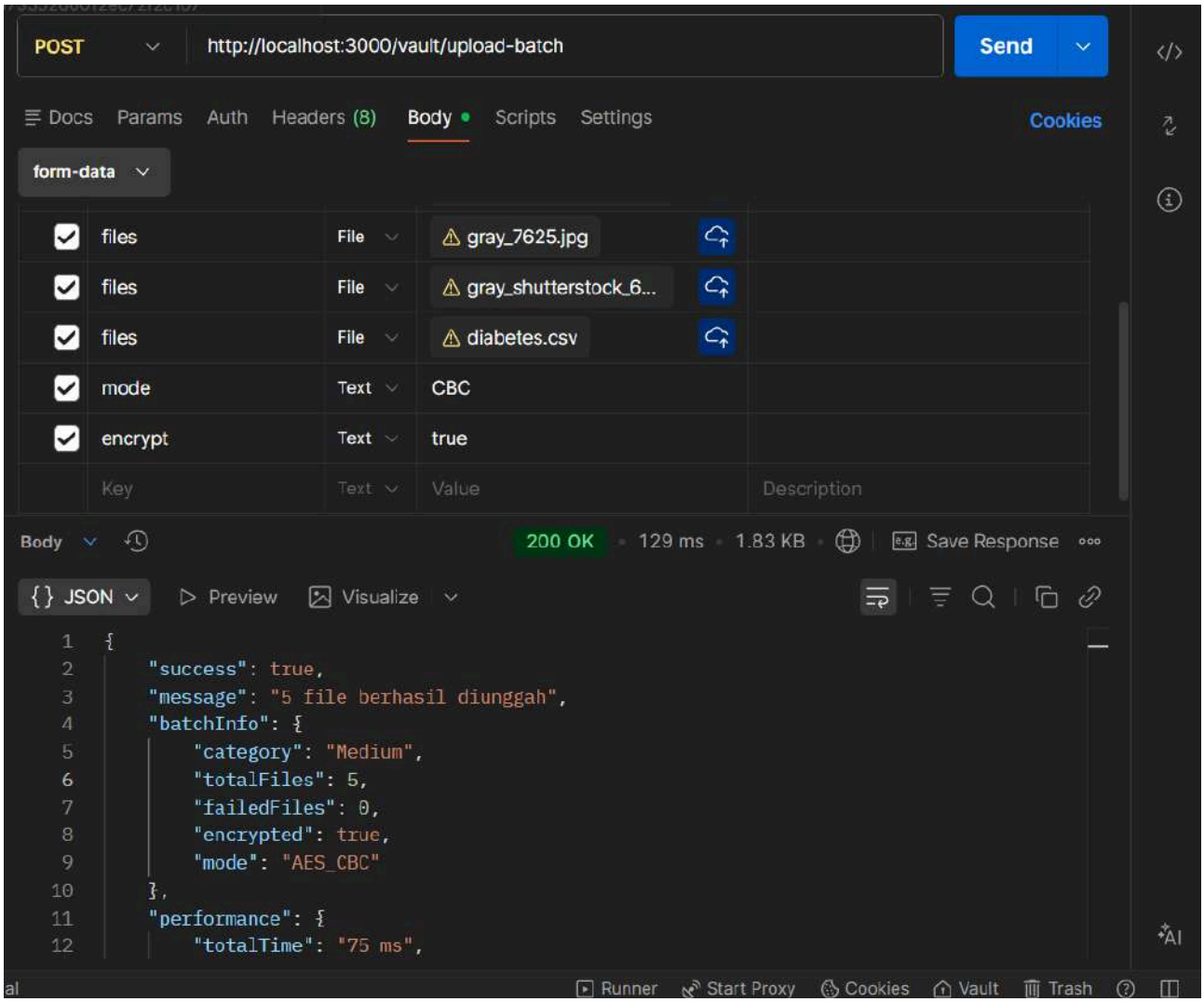
	Tugas Modul & Kode Pythonnya - Rafli Sujatmiko_0102523051.zip						

Tabel 4.2.5.2 Perhitungan fitur pada pengujian upload batch small mode CBC (PDF & File Biner 3 files)

4.2.6 Upload Batch Medium (Kombinasi PDF, Gambar, File Sensitif Dummy 5 files)

Skenario ini dirancang untuk menguji stabilitas sistem dalam menangani permintaan *batch* dengan jumlah file yang lebih banyak serta variasi tipe data yang beragam.

- **Skenario**: Mengunggah 5 file sekaligus yang terdiri dari kombinasi PDF, gambar (JPG/WEBP), dan file CSV sebagai representasi data sensitif menggunakan metode *form-data*.
- **Performa Mode CBC**: Pada pengujian mode **CBC**, sistem menunjukkan performa yang sangat responsif dengan total waktu penyelesaian hanya **75 ms**. Setiap file diproses dengan waktu enkripsi murni yang sangat rendah, berkisar antara **2 ms hingga 9 ms**.
- **Performa Mode GCM**: Pengujian menggunakan mode **GCM** mencatatkan waktu total **229 ms**. Meskipun waktu total lebih tinggi dibandingkan CBC, waktu enkripsi murni per file tetap efisien (rata-rata di bawah 50 ms), di mana selisih waktu tersebut merupakan *overhead* sistem untuk penanganan I/O dan *hashing* pada beban kerja yang lebih kompleks.
- **Analisis Selisih Waktu**: Terlihat adanya selisih antara waktu enkripsi murni (misal: 16 ms pada file gambar) dengan total waktu proses per file (33 ms). Hal ini menandakan adanya beban kerja tambahan pada sistem untuk manajemen *file stream*, komunikasi dengan MongoDB, serta proses *RSA key wrapping* untuk setiap file di dalam *batch*.
- **Hasil**: Pengujian dinyatakan berhasil dengan status **200 OK**, di mana kelima file berhasil terenkripsi secara otomatis dan tersimpan di dalam *vault* lengkap dengan ID unik untuk manajemen dokumen selanjutnya.

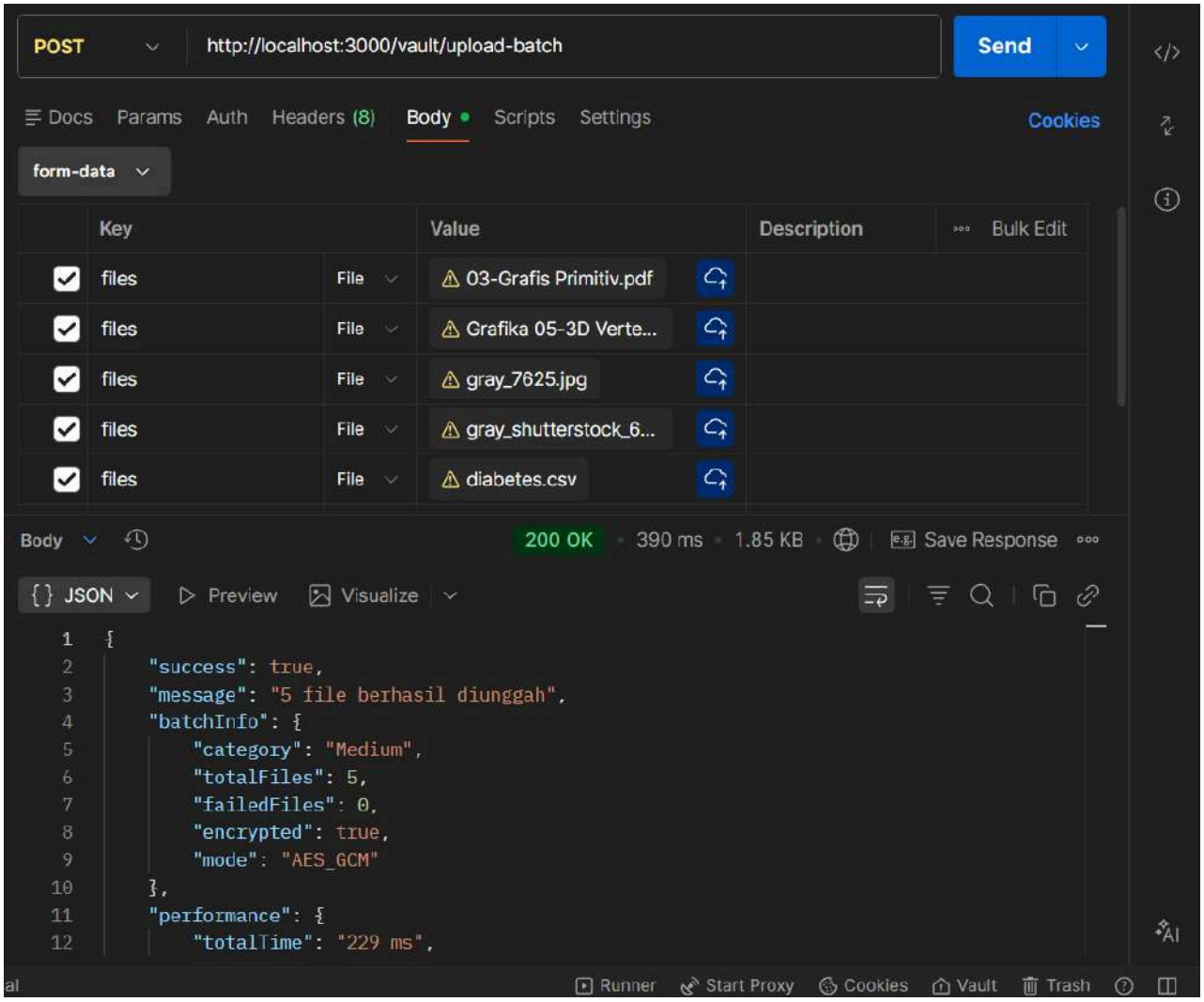


Gambar 4.2.6.1 Respons sistem pada pengujian upload batch medium mode CBC (Kombinasi PDF, Gambar, File Sensitif Dummy 5 files)

No	Nama File	Tipe	Ukuran Asli (KB)	Ukuran Terenkripsi (KB) Mode CBC	Overhead (bytes) Mode CBC	Waktu Enkripsi (ms) Mode CBC	Total Waktu Proses (ms) Mode CBC
1	03-Grafi Primitiv.p	PDF	938.08	938.09	10	6	13

	df						
2	Grafika 05-3D Vertex.pdf	PDF	868.60	868.61	5	4	8
3	gray_765.jpg	JPG	1705.85	1705.86	13	9	17
4	gray_shutterstock_626518691.webp	WEBP	1877.85	1877.86	10	8	18
5	diabetes.csv	CSV	23.31	23.33	15	2	17

Tabel 4.2.6.1 Perhitungan fitur pada pengujian upload batch medium mode CBC (Kombinasi PDF, Gambar, File Sensitif Dummy 5 files)



Gambar 4.2.6.2 Respons sistem pada pengujian upload batch medium mode GCM (Kombinasi PDF, Gambar, File Sensitif Dummy 5 files)

No	Nama File	Tipe	Ukuran Asli (KB)	Ukuran Terenkripsi (KB) Mode GCM	Overhead (bytes) Mode GCM	Waktu Enkripsi (ms) Mode GCM	Total Waktu Proses (ms) Mode GCM
1	03-Grafi Primitiv.pdf	PDF	938.08	938.08	0	46	83
2	Grafika 05-3D	PDF	868.60	868.60	0	7	23

	Vertex.pdf						
3	gray_765.jpg	JPG	1705.85	1705.85	0	16	33
4	gray_shutterstock_626518691.webp	WEBP	1877.85	1877.85	0	33	57
5	diabetes.csv	CSV	23.31	23.31	0	3	26

Tabel 4.2.6.2 Perhitungan fitur pada pengujian upload batch medium mode GCM (Kombinasi PDF, Gambar, File Sensitif Dummy 5 files)

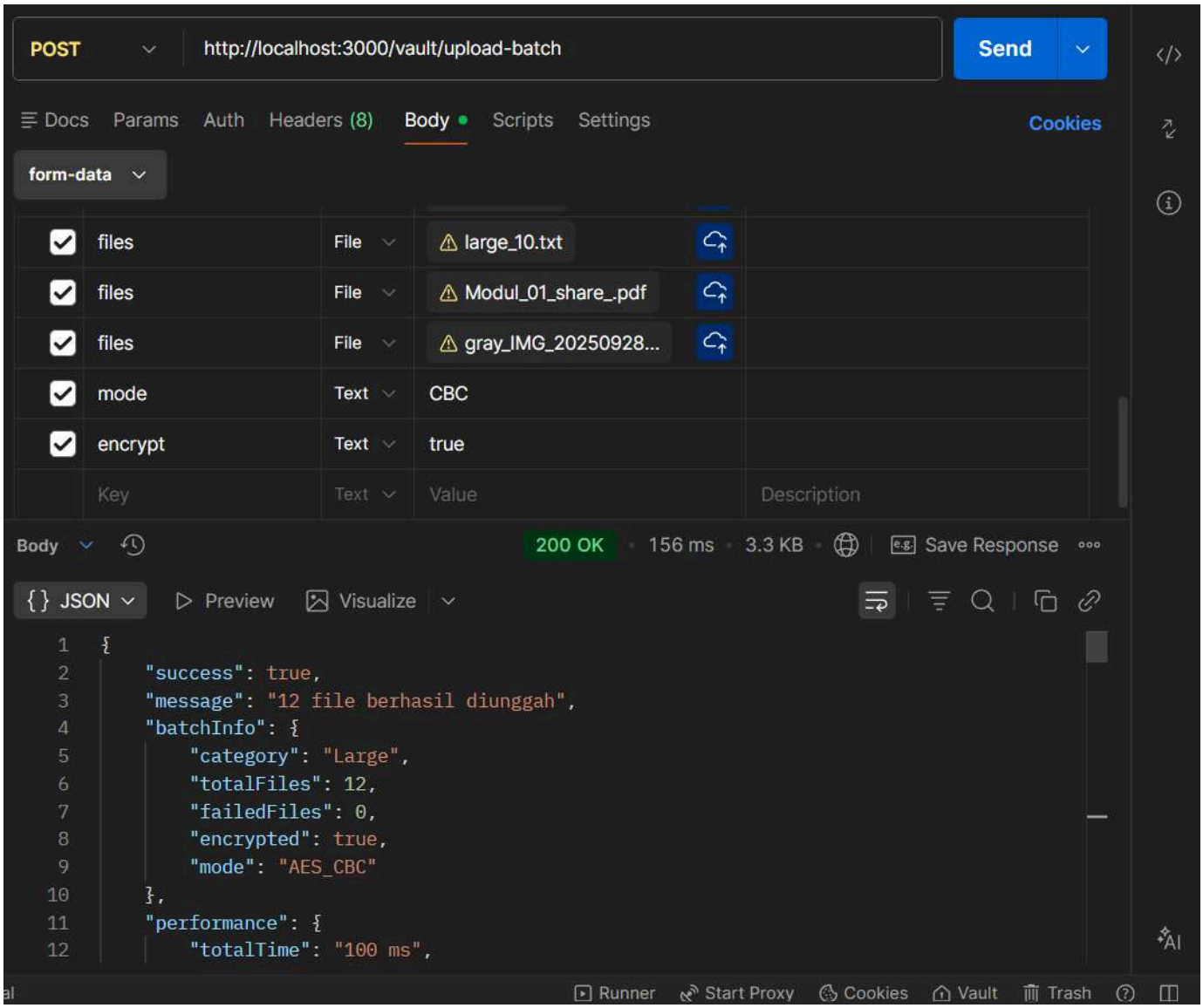
4.2.7 Upload Batch Large (Kombinasi Dokumen Teks, PDF, Gambar 12 files)

Skenario pengujian beban besar (*stress test*) dilakukan untuk mengukur ketangguhan sistem dalam memproses enkripsi simultan pada 12 file dengan tipe data yang sangat bervariasi.

- **Skenario**: Mengunggah total 12 file sekaligus yang terdiri dari 10 dokumen teks berukuran kecil, 1 file PDF, dan 1 file gambar (JPG) berukuran besar (~2.5 MB).
- **Performa Mode CBC**: Pada mode **CBC**, sistem menunjukkan performa yang sangat efisien dengan total waktu penyelesaian hanya **100 ms**. Meskipun jumlah file mencapai belasan, waktu enkripsi murni untuk setiap file teks tetap stabil di angka **1 ms hingga 3 ms**.
- **Performa Mode GCM**: Pengujian pada mode **GCM** mencatatkan waktu total **148 ms**. Secara teknis, GCM menunjukkan kecepatan enkripsi murni yang kompetitif (7 ms untuk file teks pertama), namun total waktu proses per file sedikit lebih tinggi dibandingkan CBC karena *overhead* penanganan autentikasi data.
- **Efisiensi Penyimpanan**:
    - Mode **GCM** kembali membuktikan efisiensi ruang dengan nilai *overhead* **0 bytes** pada seluruh file, termasuk pada file gambar berukuran besar.
    - Mode **CBC** menghasilkan *overhead* variabel, seperti penambahan **16 bytes** pada file PDF dan **8 bytes** pada file gambar akibat penggenapan blok (*padding*).

- **Analisis Throughput**: Sistem terbukti mampu mempertahankan kecepatan tinggi meskipun ukuran data meningkat. Penanganan file gambar besar (2582 KB) hanya membutuhkan waktu enkripsi **14 ms** pada mode CBC dan **10 ms** pada mode GCM, menunjukkan skalabilitas mesin kriptografi yang baik.

Hasil Akhir: Pengujian pada beban kerja besar ini dinyatakan **berhasil** dengan status **200 OK**. Sistem terbukti andal dalam melakukan manajemen antrean enkripsi hibrida tanpa mengalami *bottleneck* atau kegagalan proses pada kombinasi file yang kompleks.

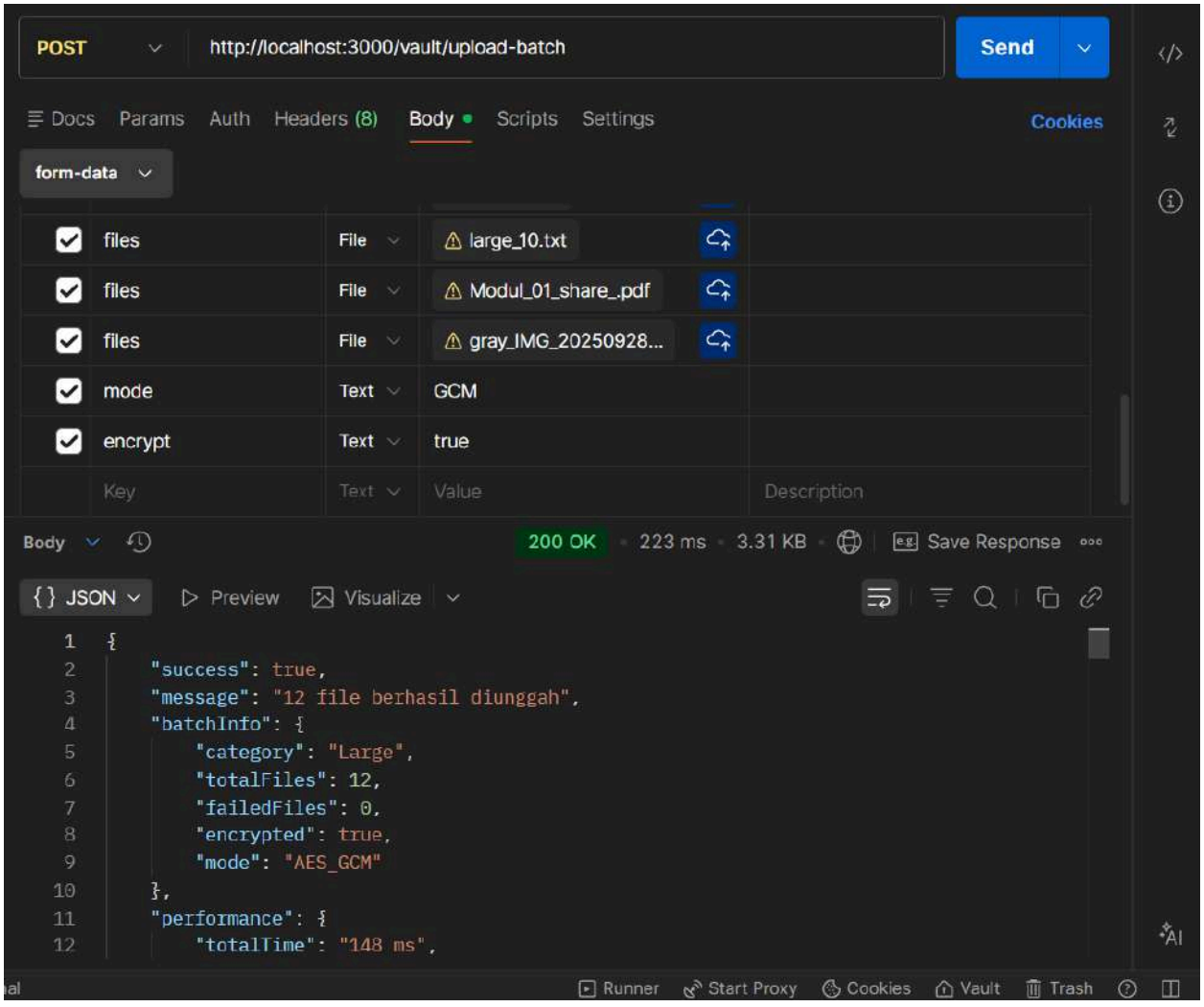


Gambar 4.2.7.1 Respons sistem pada pengujian upload batch large mode CBC(Dokumen teks, PDF, Gambar 12 files)

No	Nama File	Tipe	Ukuran Asli (KB)	Ukuran Terenkripsi (KB) Mode CBC	Overhead (bytes) Mode CBC	Waktu Enkripsi (ms) Mode CBC	Total Waktu Proses (ms) Mode CBC
1	large_1.txt	Dokumen Teks	14.67	14.67	5	3	12
2	large_2.txt	Dokumen Teks	14.67	14.67	5	2	5
3	large_3.txt	Dokumen Teks	14.67	14.67	5	1	6
4	large_4.txt	Dokumen Teks	14.67	14.67	5	1	6
5	large_5.txt	Dokumen Teks	14.67	14.67	5	1	5
6	large_6.txt	Dokumen Teks	14.67	14.67	5	2	5
7	large_7.txt	Dokumen Teks	14.67	14.67	5	2	6
8	large_8.txt	Dokumen Teks	14.67	14.67	5	1	4
9	large_9.txt	Dokumen Teks	14.67	14.67	5	1	7
10	large_10.txt	Dokumen Teks	14.67	14.67	4	2	6

11	Modul_01_share_.pdf	PDF	306.22	306.23	16	2	8
12	gray_IMG_20250928_155545,jpg	JPG	2582.15	2582.16	8	14	26

Tabel 4.2.7.1 Perhitungan fitur pada pengujian upload batch large mode CBC (Dokumen teks, PDF, Gambar 12 files)



Gambar 4.2.7.2 Respons sistem pada pengujian upload batch large mode GCM(Dokumen teks, PDF, Gambar 12 files)

No	Nama File	Tipe	Ukuran Asli (KB)	Ukuran Terenkripsi (KB) Mode GCM	Overhead (bytes) Mode GCM	Waktu Enkripsi (ms) Mode GCM	Total Waktu Proses (ms) Mode GCM
1	large_1.txt	Dokumen Teks	14.67	14.67	0	7	44
2	large_2.txt	Dokumen Teks	14.67	14.67	0	2	7
3	large_3.txt	Dokumen Teks	14.67	14.67	0	3	10
4	large_4.txt	Dokumen Teks	14.67	14.67	0	1	6
5	large_5.txt	Dokumen Teks	14.67	14.67	0	4	10
6	large_6.txt	Dokumen Teks	14.67	14.67	0	2	7
7	large_7.txt	Dokumen Teks	14.67	14.67	0	3	8
8	large_8.txt	Dokumen Teks	14.67	14.67	0	1	5
9	large_9.txt	Dokumen Teks	14.67	14.67	0	2	8
10	large_10.txt	Dokumen Teks	14.67	14.67	0	1	6
11	Modul_01_share_.p	PDF	306.22	306.22	0	3	7

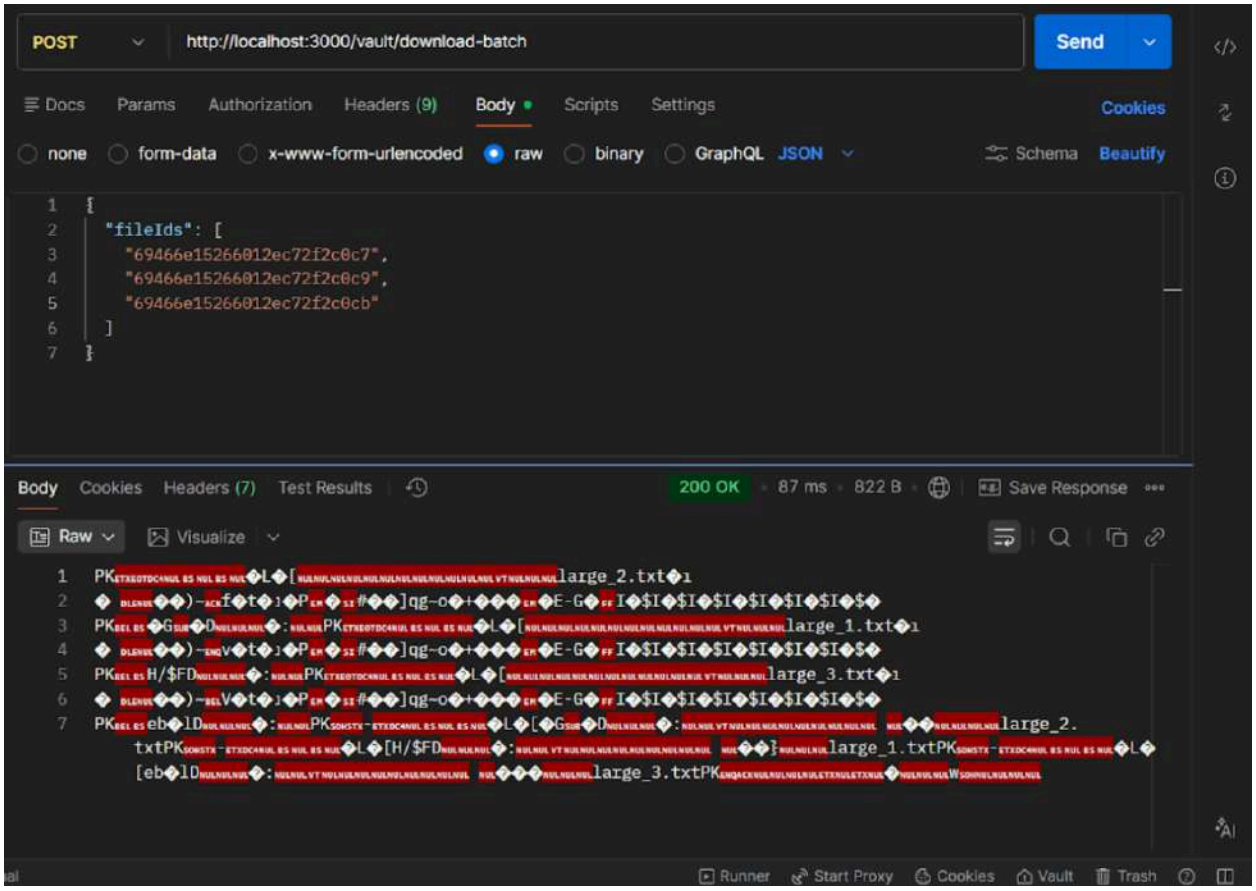
	df						
12	gray_IMG_20250928_155545,jpg	JPG	2582.15	2582.15	0	10	28

Tabel 4.2.7.2 Perhitungan fitur pada pengujian upload batch large mode GCM (Dokumen teks, PDF, Gambar 12 files)

4.2.8 Download Batch (ZIP) POST

Pengujian ini bertujuan untuk memverifikasi fitur pengunduhan multi-file secara simultan, di mana sistem secara otomatis menggabungkan file-file tersebut ke dalam satu arsip terkompresi.

- **Skenario**: Melakukan permintaan pengunduhan untuk tiga file sekaligus menggunakan daftar fileIds melalui endpoint POST /vault/download-batch.
- **Mekanisme**: Sistem mengambil file terenkripsi dari *storage*, melakukan dekripsi hybrid satu per satu, dan membungkus hasilnya ke dalam format ZIP secara *on-the-fly*.
- **Hasil Sistem**: Pengujian berhasil dengan status **200 OK** dan waktu respons **87 ms** untuk pengolahan batch tersebut.
- **Validasi Output**: Berdasarkan tampilan *Raw* respons di Postman, terlihat struktur *header* file ZIP (ditandai dengan karakter PK) yang berisi daftar file asli seperti large_1.txt, large_2.txt, dan large_3.txt. Hal ini menandakan proses enkapsulasi ke dalam arsip ZIP berjalan dengan sukses setelah proses dekripsi selesai.



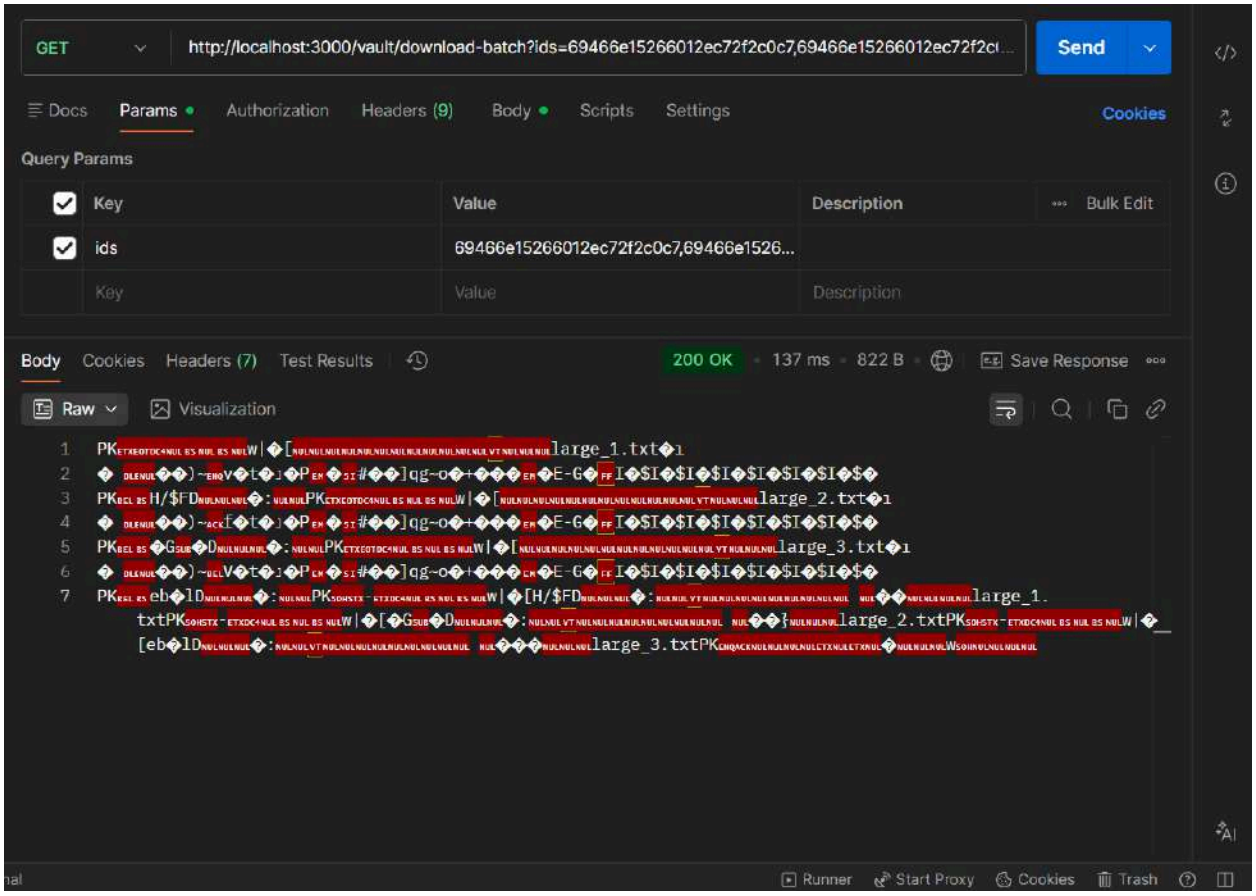
Gambar 4.2.8 Respons sistem pada pengujian download batch (ZIP) endpoint POST

4.2.9 Download Batch (ZIP) GET

Selain melalui metode POST, sistem juga mendukung pengunduhan banyak file sekaligus menggunakan metode **GET** dengan parameter kueri (*query params*). Pengujian ini memastikan fleksibilitas aksesibilitas terhadap file yang tersimpan di dalam *vault*.

- **Skenario**: Melakukan permintaan pengunduhan batch dengan menyertakan deretan file_id (seperti 69466e15...) langsung pada URL parameter ids.
- **Hasil Sistem**: Permintaan berhasil diproses dengan status **200 OK** dan waktu respons sebesar **137 ms**.
- **Validasi Data**: Respons *Raw* menunjukkan *header* file arsip ZIP (karakter PK) yang membungkus file-file hasil dekripsi secara real-time, mencakup file seperti large_1.txt, large_2.txt, dan large_3.txt.
- **Analisis**: Penggunaan metode GET memberikan kemudahan bagi sisi klien untuk melakukan pemanggilan URL langsung, sementara sistem tetap menjamin

keamanan data melalui proses dekripsi hybrid sebelum file digabungkan ke dalam format ZIP.



Gambar 4.2.9 Respons sistem pada pengujian download batch (ZIP) endpoint POST

4.2.10 Tabel Compare POST vs GET Download Batch

Aspek	POST	GET
Panjang URL	Tanpa batas	Terbatas (~2048 chars)
Keamanan	Isi tidak dicatat	ID terlihat di log
Uji Browser	Membutuhkan Postman	Dapat diuji di browser
Standar REST	Benar	Tidak ideal
Batas File	Fleksibel	Fleksibel, maksimal 10-20 file

Array JSON	Mudah	Membutuhkan parsing

Tabel 4.2.10 Perbandingan Endpoint Download Batch

Pemilihan antara metode **POST** dan **GET** untuk fitur *Download Batch* didasarkan pada trade-off antara keamanan dan kemudahan pengujian.

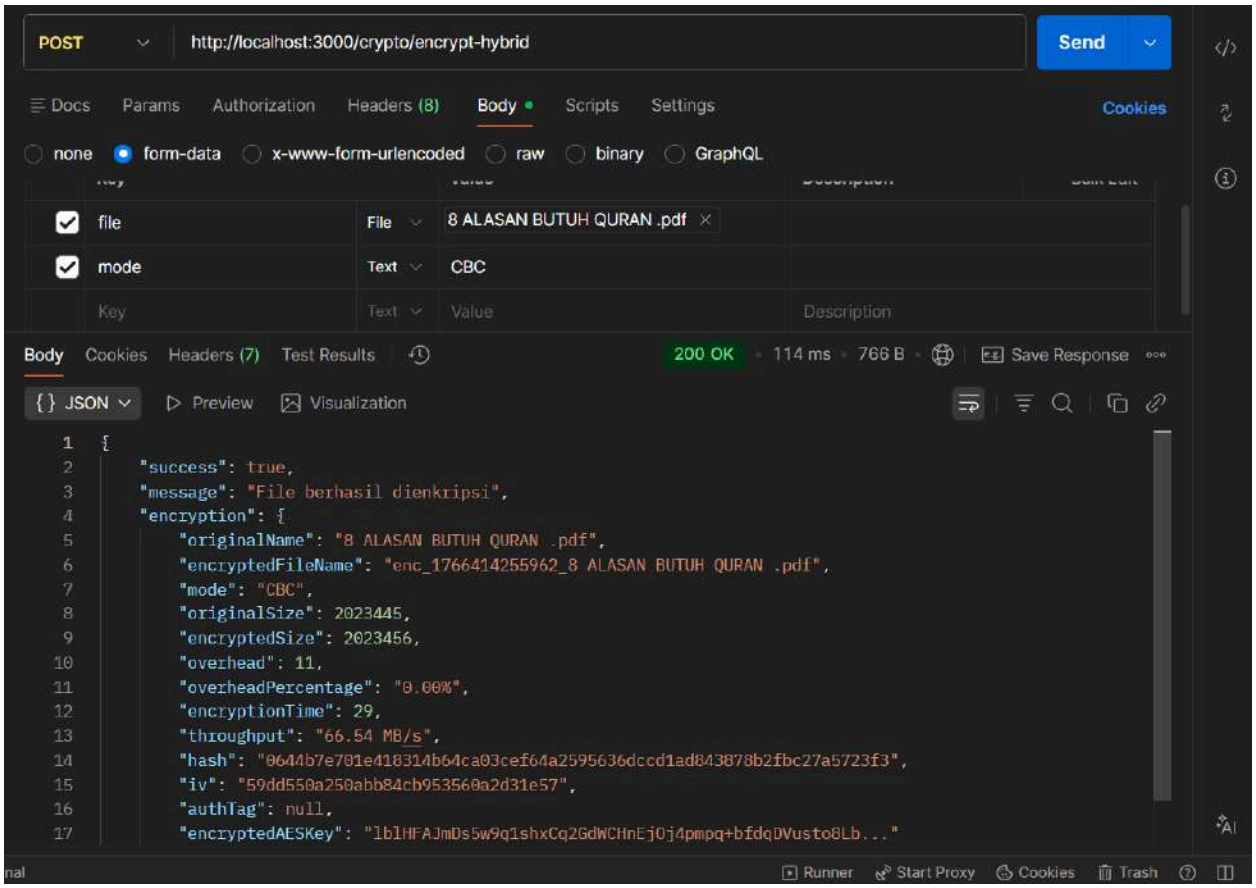
- **Metode POST (Current)**: Merupakan standar yang lebih aman dan sesuai untuk arsitektur RESTful karena daftar ID file dikirimkan melalui *request body*. Metode ini tidak memiliki batasan panjang karakter dan lebih privat karena log server biasanya tidak mencatat isi *body*, menjadikannya pilihan ideal untuk produksi.
- **Metode GET (Alternative)**: Menawarkan kemudahan dalam pengujian langsung melalui *browser* tanpa memerlukan kakas tambahan seperti Postman. Namun, metode ini memiliki batasan panjang URL (sekitar 2048 karakter) dan kurang aman karena ID file terlihat jelas dalam log sistem.

4.2.11 Encrypt Hybrid

Pengujian pada endpoint POST /crypto/encrypt-hybrid bertujuan untuk memverifikasi mekanisme keamanan inti sistem, yaitu penggunaan model enkripsi hibrida di mana file dienkripsi dengan AES, dan kunci AES tersebut diamankan menggunakan kunci publik RSA.

- **Skenario**: Mengunggah satu file PDF ("8 ALASAN BUTUH QURAN .pdf") untuk diproses menggunakan skema enkripsi hibrida dengan mode blok **CBC**.
- **Hasil Sistem**: Permintaan berhasil diproses dengan status **200 OK** dan waktu respons **114 ms**.
- **Analisis Kriptografi**:
    - **Kecepatan & Throughput**: Sistem mencatatkan waktu enkripsi sebesar **29 ms** dengan *throughput* mencapai **66.54 MB/s** untuk file berukuran 2 MB, menunjukkan efisiensi yang tinggi.
    - **Keamanan Kunci**: Respon JSON menampilkan encryptedAESKey, yang merupakan bukti bahwa kunci simetris AES telah berhasil dibungkus (*wrapped*) oleh kunci RSA sebelum disimpan atau dikirimkan.
    - **Integritas Data**: Nilai hash (SHA-256) dihasilkan secara otomatis untuk memastikan integritas file tetap terjaga saat proses dekripsi nantinya.

Analisis ini mengonfirmasi bahwa modul kriptografi telah terintegrasi dengan benar, memenuhi syarat skenario proyek dalam mengamankan kunci AES dengan *RSA Public Key*.



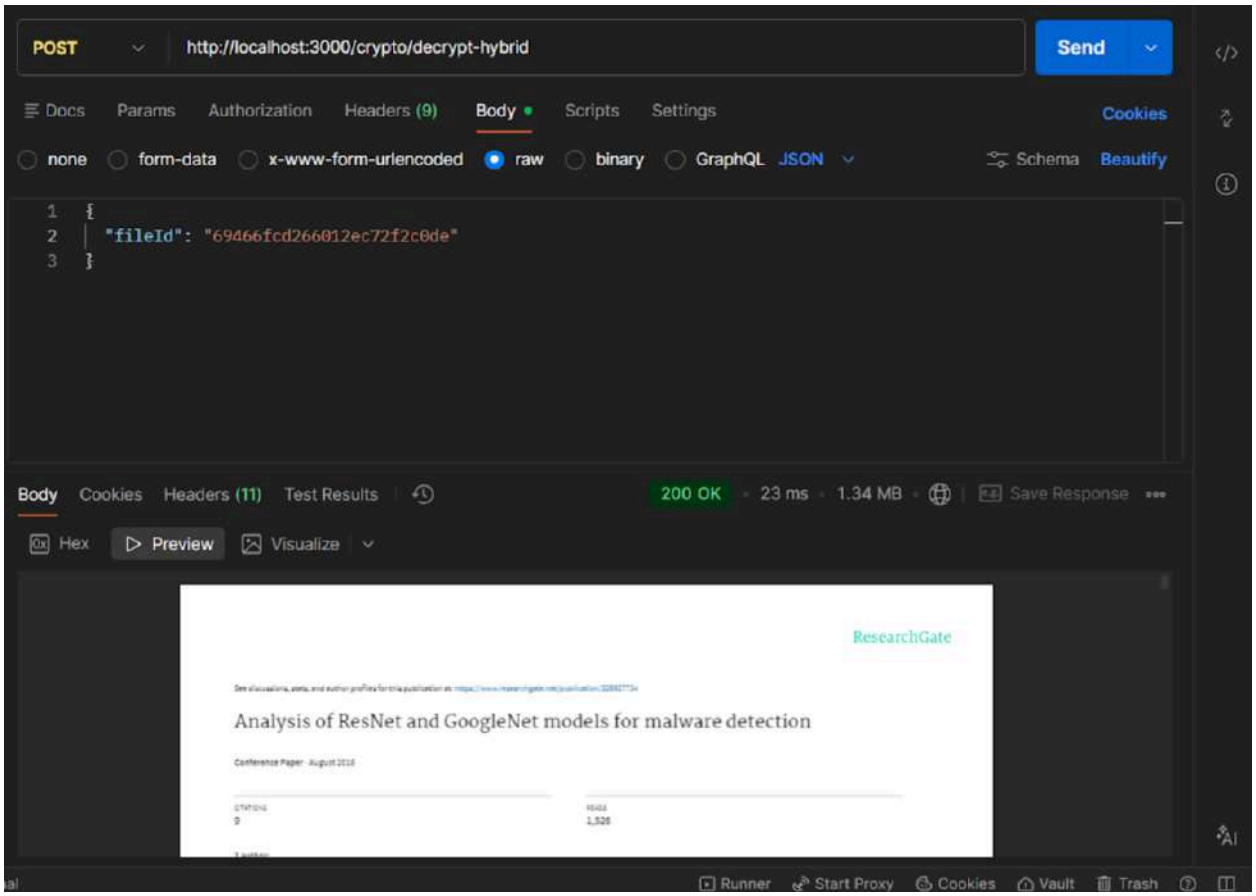
Gambar 4.2.11 Respons sistem pada pengujian encrypt hybrid

4.2.12 Decrypt Hybrid

Pengujian pada endpoint POST /crypto/decrypt-hybrid bertujuan untuk memvalidasi kemampuan sistem dalam mengembalikan file asli dari keadaan terenkripsi menggunakan skema hibrida.

- **Skenario**: Melakukan permintaan dekripsi dengan mengirimkan fileId dokumen spesifik melalui *request body*.
- **Proses Teknis**: Sistem melakukan *unwrapping* pada kunci AES menggunakan *Private Key* RSA, kemudian mendekripsi file fisik untuk dikembalikan ke format asalnya.

- **Hasil Sistem**: Pengujian berhasil dengan status **200 OK** dan waktu respons yang sangat cepat, yaitu **23 ms** untuk file berukuran **1.34 MB**.
- **Validasi Output**: Fitur *Preview* pada Postman menunjukkan bahwa dokumen PDF ("Analysis of ResNet and GoogleNet...") dapat terbaca kembali secara sempurna tanpa cacat data, menandakan proses dekripsi asimetris dan simetris berjalan akurat.



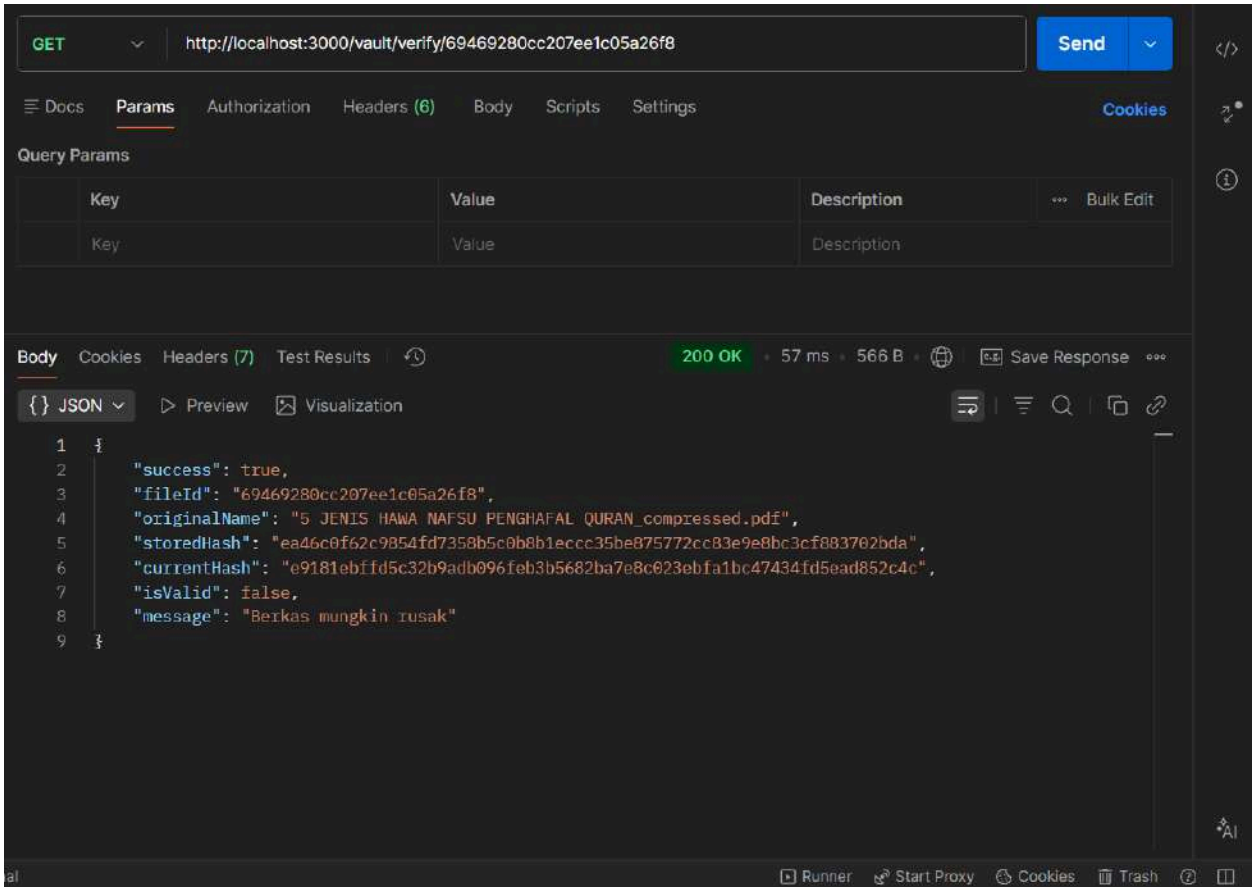
Gambar 4.2.12 Respons sistem pada pengujian decrypt hybrid

4.2.13 Verify File Integrity

Pengujian pada endpoint GET /vault/verify/:id bertujuan untuk menjamin keamanan data dengan memvalidasi kesamaan nilai *hash* file fisik terhadap data yang tercatat di database.

- **Skenario**: Melakukan pengecekan integritas pada file PDF menggunakan ID unik untuk mendeteksi adanya modifikasi ilegal atau kerusakan data.
- **Hasil Sistem**: Permintaan berhasil diproses dengan status **200 OK** dalam waktu **57 ms**.

- **Analisis Keamanan**:
    - **Deteksi Perubahan**: Sistem membandingkan storedHash (data awal) dengan currentHash (kondisi file saat ini).
    - **Status Validasi**: Pada pengujian ini, sistem mengembalikan nilai "isValid": false dengan pesan "Berkas mungkin rusak" karena adanya perbedaan nilai *hash*.
- **Kesimpulan**: Fitur ini terbukti efektif sebagai mekanisme *early warning* untuk memastikan bahwa file yang akan didekripsi oleh pengguna masih identik dengan file asli saat pertama kali diunggah ke dalam *vault*.

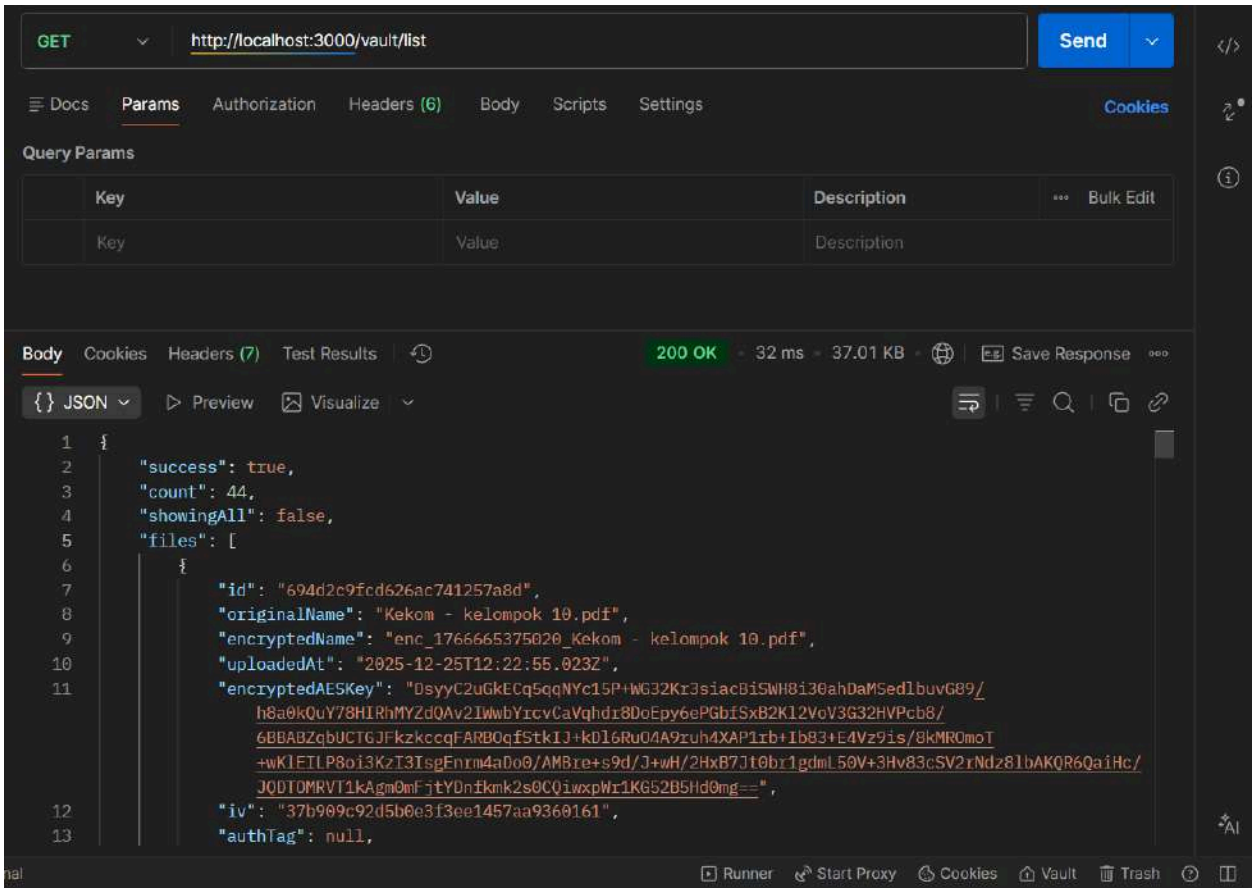


Gambar 4.2.13 Respons sistem pada pengujian verify file integrity

4.2.14 Metadata Listing (*Version Control Analysis*)

Pengujian pada endpoint GET /vault/list dilakukan untuk memverifikasi kemampuan sistem dalam mengelola katalog file terenkripsi serta mendukung kontrol versi (*versioning*) pada dokumen yang sama.

- **Skenario**: Menampilkan daftar seluruh metadata file yang tersimpan di dalam *vault*.
- **Hasil Sistem**: Permintaan berhasil diproses dengan status **200 OK** dan waktu respons cepat sebesar **32 ms** untuk total 44 entri data.
- **Analisis Metadata**:
    - **Kontrol Versi**: Sistem berhasil mengidentifikasi iterasi file yang sama (misal: "Kekom - kelompok 10.pdf") dengan atribut version yang meningkat (v1, v2, v3) dan status isActive untuk menandai versi terbaru yang berlaku.
    - **Detail Teknis**: Setiap entri menyediakan informasi komprehensif mulai dari encryptionMode, encryptedAESKey, hingga perbandingan size vs encryptedSize untuk transparansi penyimpanan.
- **Kesimpulan**: Fitur ini memastikan manajemen dokumen di dalam *vault* tertata secara kronologis dan sistematis, memudahkan pengguna untuk melacak riwayat pembaruan file tanpa kehilangan data lama.



Gambar 4.2.14 Respons sistem pada pengujian metadata listing (version control analysis)

Contoh output JSON:


```
{
    "success": true,
    "count": 44,
    "showingAll": false,
    "files": [
        {
            "id": "694d2c9fcd626ac741257a8d",
            "originalName": "Kekom - kelompok 10.pdf",
            "encryptedName": "enc_1766665375020_Kekom - kelompok 10.pdf",
            "uploadedAt": "2025-12-25T12:22:55.023Z",
            "encryptedAESKey":
"DsyyC2uGkECq5qqNYc15P+WG32Kr3siacBiSWH8i30ahDaMSedlbuvG89/h8a0kQuY78HIRhMYZdQAv2IWwbY
rcvCaVqhdr8DoEpy6ePGbfSxB2Kl2VoV3G32HVPcb8/6BBABZqbUCTGJFkzkccqFARBOqfStkIJ+kDl6RuO4A9
ruh4XAP1rb+Ib83+E4Vz9is/8kMROmoT+wKlEILP8oi3KzI3IsgEnrm4aDo0/AMBre+s9d/J+wH/2HxB7Jt0br
1gdmL50V+3Hv83cSV2rNdz8lbAKQR6QaiHc/JQDTOMRVT1kAgm0mFjtYDnfkmk2s0CQiwxpWr1KG52B5Hd0mg=
=",
            "iv": "37b909c92d5b0e3f3ee1457aa9360161",
            "authTag": null,
            "version": 3,
            "isActive": false,
            "mimeType": "application/pdf",
            "size": 103159,
            "encryptedSize": 103168,
            "hash":
"d814fa8264358cac5e6a20429759ce360554c767baa020e542e669efed3aff9e",
            "encryptionMode": "AES_CBC",
            "storagePath": "vault\\enc_1766665375020_Kekom - kelompok 10.pdf"
        },
        {
            "id": "694d2c7dcd626ac741257a89",
            "originalName": "Kekom - kelompok 10.pdf",
            "encryptedName": "enc_1766665341360_Kekom - kelompok 10.pdf",
            "uploadedAt": "2025-12-25T12:22:21.400Z",
            "encryptedAESKey":
"iDTbVFSH7XgJq9VawUk2jL2RkivzzU8S20+IyVRHyuFeiLGazdxJUkghs0S2oaDiudxB39KGQUvN0edZH0vBL
nY+MtgFC22uN2S8mp+YxmmtSIKwDNZ93MwI3aSr/PFkQgjPcylg/kef4vcE8QF4K8SBAEuqkjJsRcy/sf+EH78
AkFJY955vtK+LPQTh/VySmnJqIGcmaSixYf/SsZbwdXHulO4wJ0VsAZbmWCBhfJlgNkKHWaICTo5RUypj0v1n/
HXws0rPq8b1T20uweUuxpdejPGMf1OxfN94hWKSxdAFF9cxDnEfmIAhXytwzwSFXXGB+9WYLXGeFcaXAVbYvA=
=",
            "iv": "1bddc152186d19d562d5dea8a4feb3d8",
            "authTag": null,
```

```
            "version": 2,
            "isActive": false,
            "mimeType": "application/pdf",
            "size": 4738722,
            "encryptedSize": 4738736,
            "hash":
"184c09e1f087651e9cdf948e1a4cec3d2d04fedc42b2945cc19d8e9e62c58f40",
            "encryptionMode": "AES_CBC",
            "storagePath": "vault\\enc_1766665341360_Kekom - kelompok 10.pdf"
        },
        {
            "id": "694d0b03e7b0ac29ef56b8f2",
            "originalName": "Dokumen Progress KeKom Topik 2 - Kelompok 10.pdf",
            "encryptedName": "enc_1766656771281_Dokumen Progress KeKom Topik 2 -
Kelompok 10.pdf",
            "uploadedAt": "2025-12-25T09:59:31.331Z",
            "encryptedAESKey":
"R84FZKaZUfFDJp/xURCxXHDMiKDcYYqPwQVMr2U7rvx7nsr9FhzieGpJ+oOYA96IPS27ScVEjrq1Txyeb/3Iy
/MTnT/jSs0LwJeAgCrwg7MBKWikCbyL7EmMF3ZNRXTmxRLO7vJenhvhjeph5+n5ZjLbuD+dNbwq77wyywjGjx+
OuJPmU3Ji6TNLk6M00JEU7wU9DBHCvlrIrjMDnjMsVSiUZiHuLMhN7LR9R+960XL1zi/jwBiTAS+lRe1nH3Tv/
J3cS+1VDSCXl+UX01Exaj5cT+WGr9dXeMCWskGrQ4W8SO8HTPvbscjpXIPUcxPcUvLT3NMLVHF3jrUsMbwDyw=
=",
            "iv": "78a39195482d248ff2c9c1b3cc244666",
            "authTag": null,
            "version": 1,
            "isActive": true,
            "mimeType": "application/pdf",
            "size": 4738722,
            "encryptedSize": 4738736,
            "hash":
"184c09e1f087651e9cdf948e1a4cec3d2d04fedc42b2945cc19d8e9e62c58f40",
            "encryptionMode": "AES_CBC",
            "storagePath": "vault\\enc_1766656771281_Dokumen Progress KeKom Topik 2 -
Kelompok 10.pdf"
        },
        {
            "id": "694d08c3e7b0ac29ef56b8ec",
            "originalName": "Kekom - kelompok 10.pdf",
            "encryptedName": "enc_1766656195711_Kekom - kelompok 10.pdf",
            "uploadedAt": "2025-12-25T09:49:55.731Z",
            "encryptedAESKey":
"DsyyC2uGkECq5qqNYc15P+WG32Kr3siacBiSWH8i30ahDaMSedlbuvG89/h8a0kQuY78HIRhMYZdQAv2IWwbY
```

```
rcvCaVqhdr8DoEpy6ePGbfSxB2Kl2VoV3G32HVPcb8/6BBABZqbUCTGJFkzkccqFARBOqfStkIJ+kDl6RuO4A9
ruh4XAP1rb+Ib83+E4Vz9is/8kMROmoT+wKlEILP8oi3KzI3IsgEnrm4aDo0/AMBre+s9d/J+wH/2HxB7Jt0br
1gdmL50V+3Hv83cSV2rNdz8lbAKQR6QaiHc/JQDTOMRVT1kAgm0mFjtYDnfkmk2s0CQiwxpWr1KG52B5Hd0mg=
=",
            "iv": "37b909c92d5b0e3f3ee1457aa9360161",
            "authTag": null,
            "version": 1,
            "isActive": true,
            "mimeType": "application/pdf",
            "size": 103159,
            "encryptedSize": 103168,
            "hash":
"d814fa8264358cac5e6a20429759ce360554c767baa020e542e669efed3aff9e",
            "encryptionMode": "AES_CBC",
            "storagePath": "vault\\enc_1766656195711_Kekom - kelompok 10.pdf"
        },
        {
            "id": "69469280cc207ee1c05a26f8",
            "originalName": "5 JENIS HAWA NAFSU PENGHAFAL QURAN_compressed.pdf",
            "encryptedName": "enc_1766232704034_5 JENIS HAWA NAFSU PENGHAFAL
QURAN_compressed.pdf",
            "uploadedAt": "2025-12-20T12:11:44.062Z",
            "encryptedAESKey":
"f3OmUoVIe6CEM7LIT7qwDhxdOHV7qXO3gD+DGsQkSxcRB+EIrKSWUhuswkwzpozeIAbJQZE4YLnnlXCv1p7WN
AxIzpfsdVe/Pq9x5DHkyPCQw2O6T5MdPU+BkTDbTdK72hmKh78X7XCTdTR0Kt0BVdh9wK2a8lsrxaiUbTwFtun
kNGC82FbfJazssl8AF1jxOOBFkmYP8Zmpd9sR2u1+noVmbOK3f7cIrqLpOiJ4mQ56jbrwFag0YOu+FkB/N4O8I
ZChlUNzas6pu8i8Pnpay3Y43CCvaFSD4Rl5LNSxwv0u98+mBySEek66cGvoJw33UXL4E4ZhwwiVJ5dL7h70KQ=
=",
            "iv": "d5f016cdde228b5be0ef463032d992ea",
            "authTag": "2dd1a419cbe9f2eacdb0496ee23ccd87",
            "version": 1,
            "isActive": true,
            "mimeType": "application/pdf",
            "size": 1002518,
            "encryptedSize": 1002518,
            "hash":
"ea46c0f62c9854fd7358b5c0b8b1eccc35be875772cc83e9e8bc3cf883702bda",
            "encryptionMode": "AES_GCM",
            "storagePath": "vault\\enc_1766232704034_5 JENIS HAWA NAFSU PENGHAFAL
QURAN_compressed.pdf"
        },
        {
```

```
            "id": "69467335266012ec72f2c105",
            "originalName": "(T5) Array2d dan Matriks - Muhammad Delva.zip",
            "encryptedName": "enc_1766224693962_(T5) Array2d dan Matriks - Muhammad
Delva.zip",
            "uploadedAt": "2025-12-20T09:58:13.964Z",
            "encryptedAESKey":
"bhXmIbcCacjGhdG4yvTnSw1zxXv01/ZTSK1UNavzEx39Cim0rtq6so9TigB/tIDqMTLUVIB25pkD/74lomrKr
/VyKHnh16RKC6E2xmAlZnYHqpnRrZm/pfXesaJpBCr3mvU8b+SK9Jm6Eo7Zy35Vde9gR9qR7YYKm3fFcXa/ZK0
0Dsj6gPdQSEI63nExKc//C12VWtxGGv+w0DiNiN/epapqg0Vv06XG0KJjmqxq3rkbMXjGAyxOJw8B/l6f5bBRo
zP0+H7duDRQYPn/HVsFXkhz4pIGIXS/tANjDetg3QMfcJuIxFoaURCVU8FKrFuyM4sLHZqD5m5UNX+R7yzB2g=
=",
            "iv": "2595ac35b48e89f6ec707a7df2c8a668",
            "authTag": null,
            "version": 1,
            "isActive": true,
            "mimeType": "application/zip",
            "size": 2618,
            "encryptedSize": 2624,
            "hash":
"c78297ad4397ad00a1bd344ec59eeb559d357e929a8bc13eaea34846aee29360",
            "encryptionMode": "AES_CBC",
            "storagePath": "vault\\enc_1766224693962_(T5) Array2d dan Matriks -
Muhammad Delva.zip"
        },
. . . . . .
    ]
}
```


Tabel Kesesuaian antara Fitur dengan Field

Fitur	Lokasi	Keterangan
file_id	File.js schema	Field **_id** (MongoDB)
filename	File.js schema	Field **encryptedName**
original_name	File.js schema	Field **originalName**
timestamp	File.js schema	Field **uploadedAt**
rsa_wrapped_key	File.js schema	Field **encryptedAESKey**

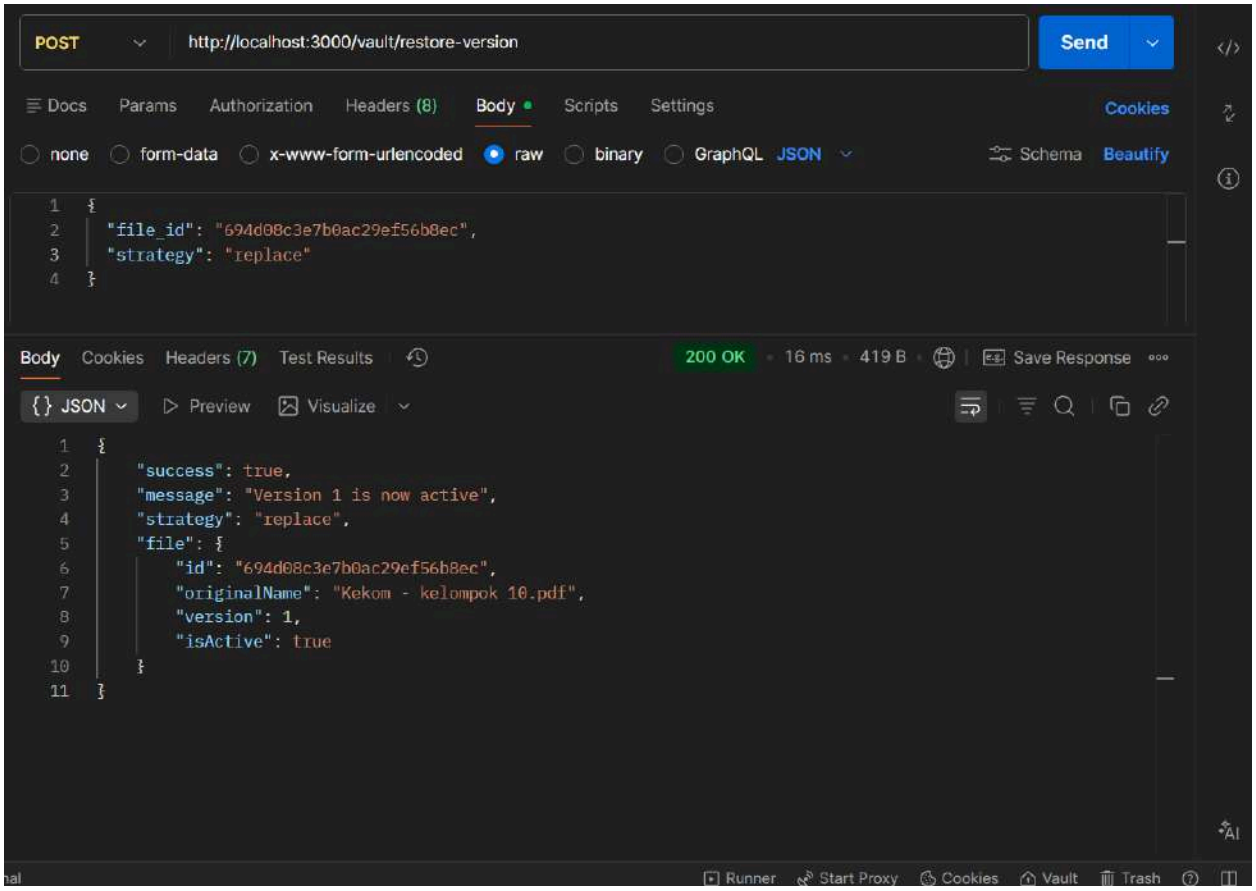
aes_iv	File.js schema	Field **iv**
aes_tag	File.js schema	Field **authTag**
version	File.js schema	Field **version**

Tabel 4.2.14 kesesuain antara fitur dengan field metadata

4.2.15 Restore Version (Strategy : Replace)

Pengujian pada endpoint POST /vault/restore-version dilakukan untuk memverifikasi kemampuan sistem dalam mengelola siklus hidup dokumen melalui strategi pemulihan versi tertentu tanpa menambah redundansi data.

- **Skenario**: Melakukan restorasi pada file "Kekom - kelompok 10.pdf" dengan mengaktifkan kembali **Version 1** menggunakan strategi "replace".
- **Hasil Sistem**: Permintaan berhasil diproses dengan status **200 OK** dan waktu respons yang sangat efisien, yaitu **16 ms**.
- **Analisis Logika**:
    - **Mekanisme Replace**: Sistem mengubah status isActive pada versi lama menjadi true, sementara riwayat versi lainnya tetap dipertahankan tanpa menambah nomor versi baru.
    - **Output Metadata**: Respon JSON mengonfirmasi pesan "Version 1 is now active", memastikan bahwa referensi unduhan akan mengarah kembali pada integritas data dari versi pertama tersebut.
- **Kesimpulan**: Strategi ini sangat efektif untuk menghemat ruang penyimpanan karena sistem hanya melakukan perpindahan status (*flagging*) tanpa perlu melakukan duplikasi fisik file atau enkripsi ulang.



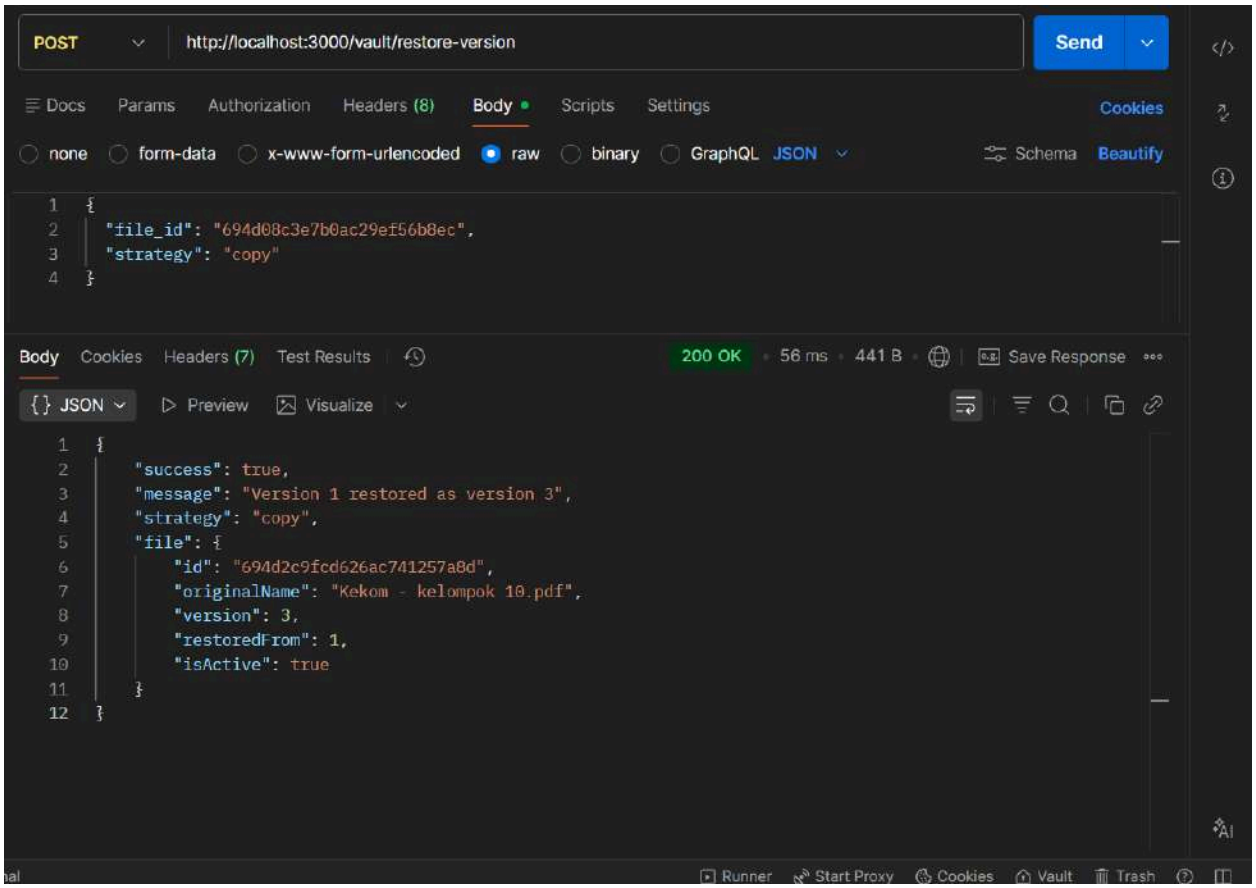
Gambar 4.2.15 Respons sistem pada pengujian restore version (strategy : replace)

4.2.16 Restore Version (Strategy : Copy)

Pengujian pada endpoint POST /vault/restore-version menggunakan strategi "copy" bertujuan untuk memulihkan data lama dengan menciptakan entri versi baru, sehingga menjaga integritas riwayat versi sebelumnya secara utuh.

- **Skenario**: Melakukan restorasi pada file "Kekom - kelompok 10.pdf" dengan menyalin konten dari **Version 1** menjadi **Version 3**.
- **Hasil Sistem**: Permintaan berhasil diproses dengan status **200 OK** dalam waktu respons **56 ms**.
- **Analisis Logika**:
    - **Mekanisme Copy**: Berbeda dengan strategi *replace*, sistem tidak mengubah status versi lama, melainkan menduplikasi metadata versi 1 ke dalam urutan versi terbaru (v3).

    - **Output Metadata**: Respon JSON mencantumkan atribut "restoredFrom": 1, yang memberikan jejak audit yang jelas bahwa versi aktif saat ini (v3) berasal dari pemulihan versi pertama.
- **Kesimpulan**: Strategi ini memberikan keamanan ekstra dalam manajemen dokumen karena tidak ada metadata versi lama yang diubah atau ditimpa, memungkinkan pelacakan riwayat pemulihan yang lebih akurat di dalam *vault*.



Gambar 4.2.16 Respons sistem pada pengujian restore version (strategy : copy)

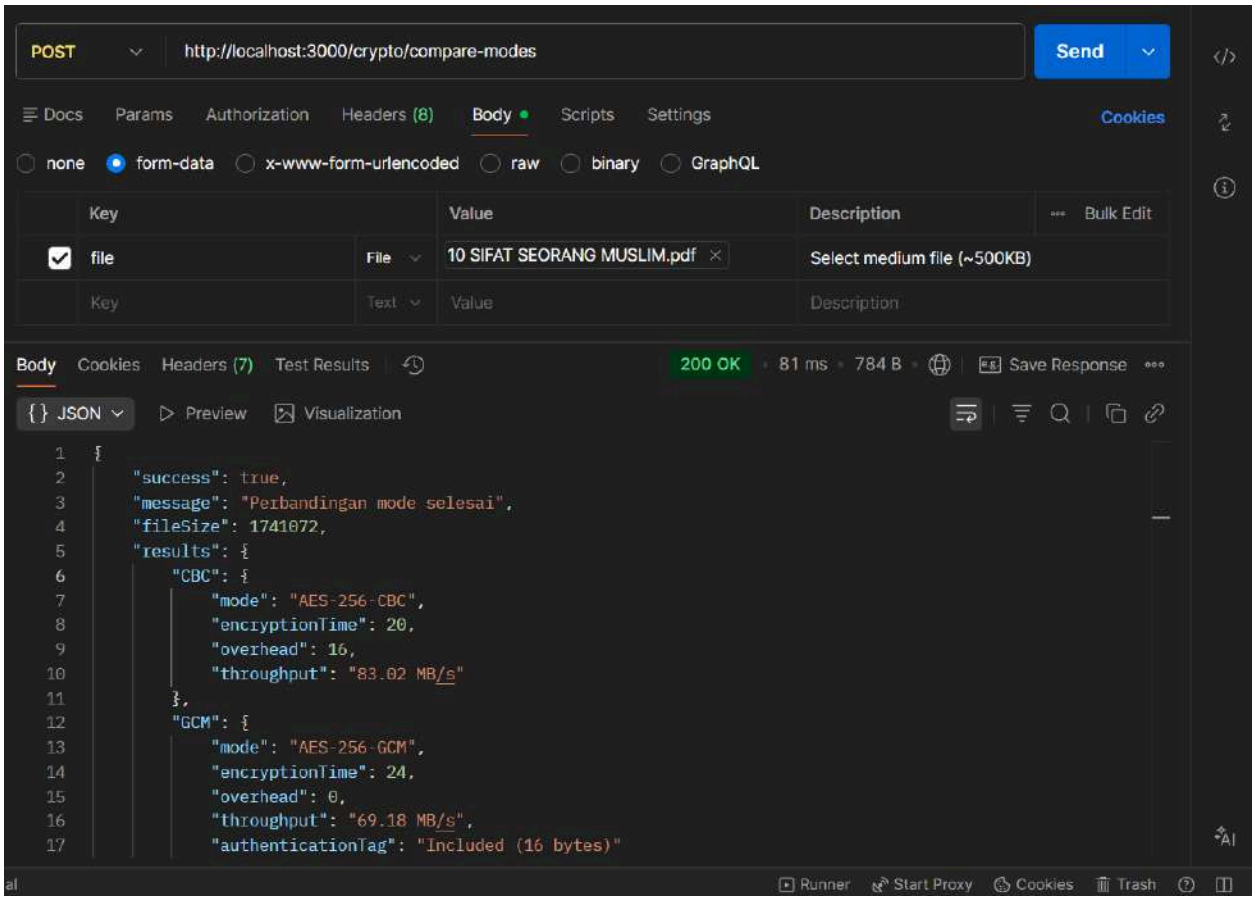
4.3 Analisis Kinerja (Benchmark)

Bagian ini menyajikan hasil analisis terhadap performa sistem *Secure Document Vault*, mencakup efisiensi algoritma enkripsi, *overhead* yang dihasilkan oleh model hybrid RSA, serta dampak proses enkripsi terhadap ukuran penyimpanan data.

4.3.1 Analisis Perbandingan Mode AES (CBC vs GCM)

Pengujian pada endpoint `POST /crypto/compare-modes` dilakukan untuk menganalisis performa dan karakteristik teknis antara mode **AES-256-CBC** dan **AES-256-GCM** pada file berukuran 1.74 MB.

- **Performa Kecepatan**: Mode **CBC** mencatatkan waktu enkripsi sedikit lebih cepat yaitu **20 ms** dengan *throughput* **83.02 MB/s**, dibandingkan mode **GCM** yang membutuhkan waktu **24 ms** dengan *throughput* **69.18 MB/s**.
- **Karakteristik Overhead**: Mode CBC menghasilkan *overhead* sebesar **16 bytes** akibat penggunaan *padding* blok, sementara mode GCM mencatatkan **0 overhead** pada ukuran file namun menyertakan *authentication tag* sebesar 16 bytes untuk menjamin integritas.
- **Analisis Keamanan**: Meskipun CBC sedikit lebih unggul dalam kecepatan (selisih 4ms), sistem merekomendasikan penggunaan **GCM** karena menyediakan *Authenticated Encryption* yang mampu mendeteksi manipulasi data secara otomatis melalui *tag* autentikasi.



Gambar 4.3.1 Respons sistem pada pengujian perbandingan mode AES

Output JSON:


```
{
    "success": true,
    "message": "Perbandingan mode selesai",
    "fileSize": 1741072,
    "results": {
        "CBC": {
            "mode": "AES-256-CBC",
            "encryptionTime": 20,
            "overhead": 16,
            "throughput": "83.02 MB/s"
        },
        "GCM": {
            "mode": "AES-256-GCM",
            "encryptionTime": 24,
            "overhead": 0,
            "throughput": "69.18 MB/s",
            "authenticationTag": "Included (16 bytes)"
        }
    },
    "analysis": {
        "fasterMode": "CBC",
        "timeDifference": "4ms (20.00% difference)",
        "recommendation": "GCM provides authenticated encryption (better security),
CBC is slightly faster",
        "securityNote": "GCM includes authentication tag to detect tampering"
    }
}
```


Metrik	CBC	GCM	Pemenang
Waktu Enkripsi (*Encryption Time*)	20 ms	24 ms	CBC
Kapasitas (*Throughput*)	83.02 MB/s	69.18 MB/s	GCM
Beban Kerja (*Overhead*)	16 bytes	0 bytes	GCM

Autentikasi (*Authentication*)	❌	✅	GCM
Keamanan (*Security*)	Bagus	Lebih Baik	GCM
Rekomendasi (*Recommendation*)	Performa	Keamanan	GCM

Tabel 4.3.1 Perbandingan Mode AES

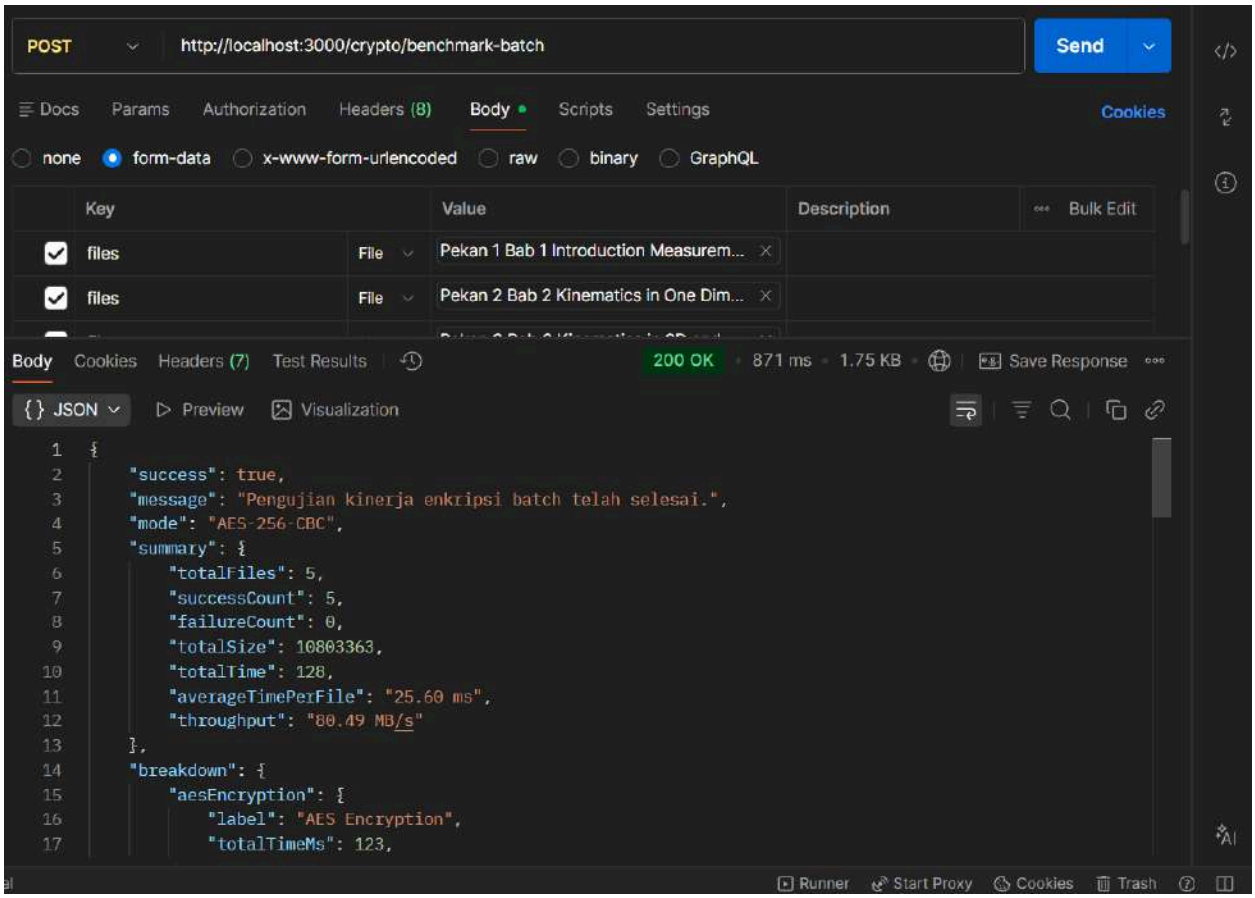
Berdasarkan hasil pengujian pada endpoint `POST /crypto/compare-modes`, sistem menunjukkan bahwa mode CBC memiliki kecepatan enkripsi lebih tinggi (20 ms) dan *throughput* yang lebih besar (83.02 MB/s) dibandingkan mode GCM. Namun, mode GCM secara teknis lebih unggul karena memiliki *overhead* 0 bytes pada ukuran file dan menyertakan fitur autentikasi otomatis (*authentication tag*) untuk menjamin integritas data dari manipulasi. Oleh karena itu, sistem merekomendasikan penggunaan GCM untuk standar keamanan yang lebih baik, sementara CBC tetap menjadi pilihan utama untuk skenario yang memprioritaskan performa komputasi maksimal.

4.3.2 Analisis Performa Enkripsi Batch (*Batch Encryption Benchmark*)

Pengujian pada endpoint POST /crypto/benchmark-batch dilakukan untuk mengukur efisiensi sistem dalam menangani proses enkripsi hibrida pada kelompok file besar secara simultan.

- **Metrik Utama**: Sistem berhasil memproses **5 file** dengan total ukuran **10.80 MB** dalam waktu keseluruhan **128 ms**, menghasilkan *throughput* kolektif sebesar **80.49 MB/s**.
- **Distribusi Waktu Komputasi**:
  - **AES Encryption**: Mendominasi beban kerja sebesar **96.09%** (123 ms).
  - **RSA Key Encryption**: Sangat efisien, hanya memakan waktu **1.56%** (2 ms) dari total proses.
  - **Lain-lain (I/O & Hashing)**: Berkontribusi sebesar **1.56%** (2 ms).
- **Analisis Per-File**: Rata-rata waktu enkripsi adalah **25.60 ms per file**, dengan performa tertinggi mencapai *throughput* **124.60 MB/s** pada file berukuran 3 MB.

Kesimpulan Pengujian: Hasil benchmark ini membuktikan bahwa arsitektur hibrida yang diimplementasikan sangat optimal. Penggunaan RSA untuk mengamankan kunci tidak menyebabkan *bottleneck* yang signifikan, sehingga sistem tetap responsif meski menangani beban kerja *batch* dengan volume data yang besar.



Gambar 4.3.2 Respons sistem pada pengujian performa enkripsi batch

Output JSON:


```
{
    "success": true,
    "message": "Pengujian kinerja enkripsi batch telah selesai.",
    "mode": "AES-256-CBC",
    "summary": {
        "totalFiles": 5,
        "successCount": 5,
        "failureCount": 0,
        "totalSize": 10803363,
        "totalTime": 128,
        "averageTimePerFile": "25.60 ms",
        "throughput": "80.49 MB/s"
    },
    "breakdown": {
```

```
        "aesEncryption": {
            "label": "AES Encryption",
            "totalTimeMs": 123,
            "percentage": "96.09%"
        },
        "rsaKeyEncryption": {
            "label": "RSA Key Encryption",
            "totalTimeMs": 2,
            "percentage": "1.56%"
        },
        "otherIoHash": {
            "label": "Other (I/O, Hash, Overhead)",
            "totalTimeMs": 2,
            "percentage": "1.56%"
        }
    },
    "files": [
        {
            "fileName": "Pekan 1 Bab 1  Introduction Measurement Estimating - Sample
Problems.pdf",
            "size": 1030592,
            "encryptionTime": "18 ms",
            "throughput": "54.60 MB/s",
            "timing": {
                "aesTime": 18,
                "rsaTime": 0,
                "otherTime": 0
            }
        },
        {
            "fileName": "Pekan 2 Bab 2  Kinematics in One Dimension - Sample
Problems.pdf",
            "size": 1600055,
            "encryptionTime": "29 ms",
            "throughput": "52.62 MB/s",
            "timing": {
                "aesTime": 27,
                "rsaTime": 0,
                "otherTime": 2
            }
        },
        {
```

```
            "fileName": "Pekan 3 Bab 3  Kinematics in 2D and 3D - Sample
Problems.pdf",
            "size": 1399299,
            "encryptionTime": "13 ms",
            "throughput": "102.65 MB/s",
            "timing": {
                "aesTime": 13,
                "rsaTime": 0,
                "otherTime": 0
            }
        },
        {
            "fileName": "Pekan 4 Bab 4 Dynamics Newton’s Laws of Motion - Sample
Problems.pdf",
            "size": 3768386,
            "encryptionTime": "44 ms",
            "throughput": "81.68 MB/s",
            "timing": {
                "aesTime": 43,
                "rsaTime": 1,
                "otherTime": 0
            }
        },
        {
            "fileName": "Pekan 5 Bab 5 Friction, Circular Motion, Drag Forces - Sample
Problems.pdf",
            "size": 3005031,
            "encryptionTime": "23 ms",
            "throughput": "124.60 MB/s",
            "timing": {
                "aesTime": 22,
                "rsaTime": 1,
                "otherTime": 0
            }
        }
    ],
    "errors": []
}
```

Biaya Enkripsi Per File (*File Encryption Cost*)

Ukuran File	Waktu Enkripsi	Kapasitas
0.98 MB	18 ms	54.60 MB/s
1.52 MB	29 ms	52.62MB/s
1.33 MB	13 ms	102.65MB/s
3.59 MB	44 ms	81.68MB/s
2.86 MB	23 ms	124.60 MB/s

Tabel 4.3.2 Biaya Enkripsi Per File

4.3.3 Analisis *Overhead* RSA (*Key Encryption*)

Dalam arsitektur *Secure Document Vault* ini, penggunaan algoritma asimetris RSA-2048/3072 difokuskan secara khusus untuk mengamankan kunci simetris AES (*key wrapping*), bukan untuk mengenkripsi konten file secara keseluruhan. Berdasarkan data statistik kinerja:

- **Efisiensi Waktu**: Proses enkripsi kunci AES menggunakan RSA hanya membutuhkan waktu sebesar **2 ms** untuk total 5 file.
- **Dampak Performa**: Beban komputasi RSA sangat minimal, yakni hanya berkontribusi sebesar **1,56%** dari keseluruhan total waktu proses enkripsi.
- **Perbandingan Beban**: Mayoritas beban kerja tetap berada pada **AES Encryption** yang memakan **96,09%** (123 ms) dari total waktu.

Operasi	Waktu	Persentase
Enkripsi AES (*AES Encryption*)	123 ms	96.09%
Enkripsi Kunci RSA (*RSA Key Encryption*)	2 ms	1.56%
Lainnya (Input/Output, Hash)(*Other (I/O, Hash)*)	2 ms	1.56%
Total	127 ms	99.21%

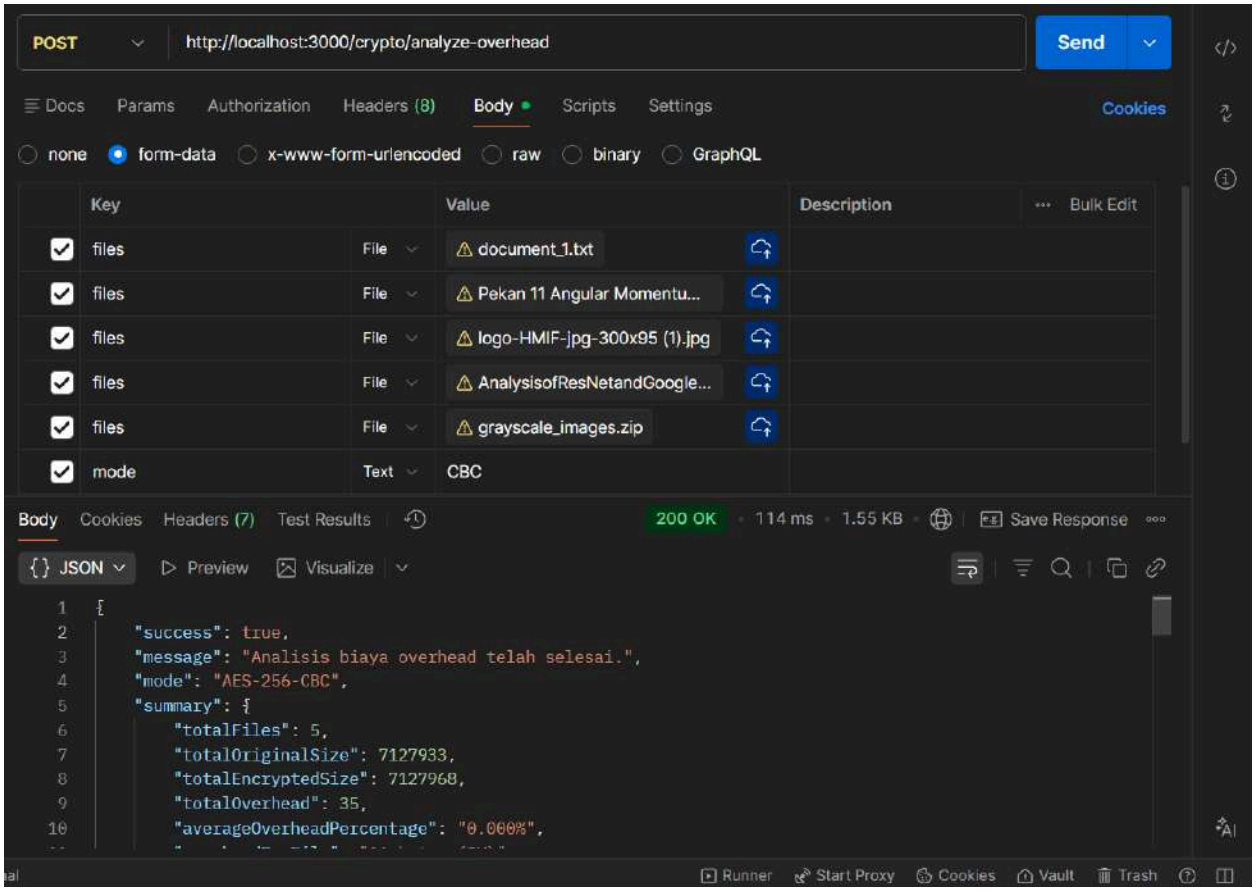
Tabel 4.3.3 analisis overhead RSA

Kesimpulan: Pengujian ini membuktikan bahwa implementasi model *hybrid* sangat efektif. RSA memberikan lapisan keamanan tinggi untuk melindungi kunci tanpa menyebabkan *bottleneck* atau penurunan performa sistem yang signifikan, sehingga sistem tetap mampu mempertahankan *throughput* yang tinggi.

4.3.4 Perbandingan Ukuran

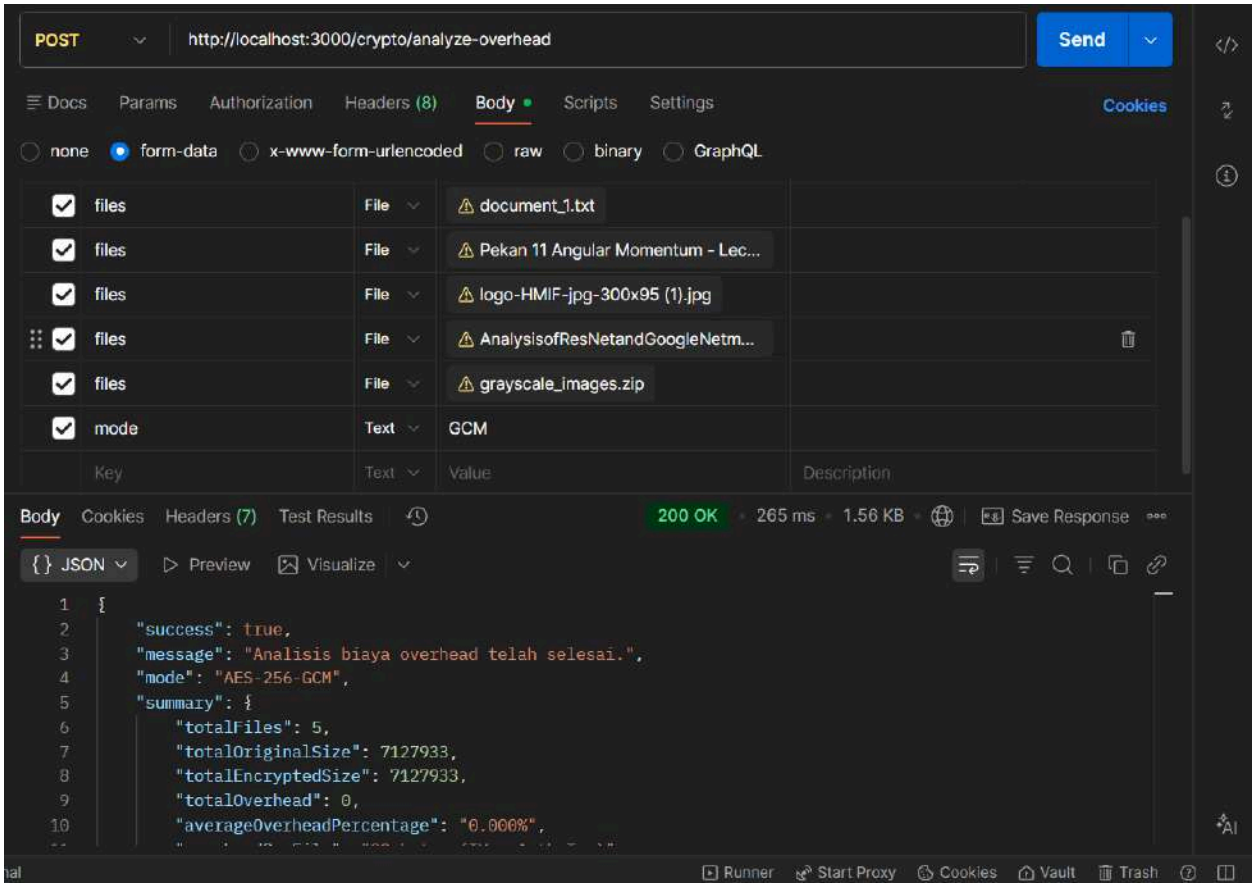
Tabel ukuran file asli vs ciphertext per file.

Mode CBC



Gambar 4.3.4 Respons sistem pada pengujian perbandingan ukuran file asli vs ciphertext mode CBC

Mode GCM



Gambar 4.3.4 Respons sistem pada pengujian perbandingan ukuran file asli vs ciphertext mode GCM

No	Nama File	Tipe	Ukuran Asli (bytes)	Ukuran Terenkripsi (bytes) Mode CBC	Ukuran Terenkripsi (bytes) Mode GCM	Overhead (bytes) Mode CBC	Overhead (bytes) Mode GCM	Overhead (%) Mode CBC	Overhead (%) Mode GCM
1	document_1.txt	Text	2990	2992	2990	2	0	0.067%	0.000%
2	Pekan 11 Angular	PDF	3069285	3069296	3069285	11	0	0.000%	0.000%

	Momentum - Lecture.pdf								
3	logo-HMIF-jpg-300x95 (1).jpg	Image	8061	8064	8061	3	0	0.037%	0.000%
4	AnalysisofResNetandGoogleNetmodelsformalwaredetection.pdf	PDF	1403289	1403296	1403289	7	0	0.000%	0.000%
5	grayscale_images.zip	Binary	2644308	2644320	2644308	12	0	0.000%	0.000%

Tabel 4.3.4 perbandingan ukuran file asli vs ciphertext

Analisis Overhead

Mode CBC:

- Overhead: **16 bytes** (IV size)
- Persentase: **< 0.01%** untuk file > 1 MB
- Negligible impact untuk file besar

Mode GCM:

- Overhead: **32 bytes** (IV 16 bytes + Auth Tag 16 bytes)
- Persentase: **< 0.01%** untuk file > 1 MB
- Sedikit lebih besar tapi masih minimal

Kesimpulan dari implementasi model hibrida ini terbukti sangat efisien dengan *overhead* di bawah 0,01% untuk file berukuran besar. Mode CBC hanya menambah 16 bytes (IV), sedangkan GCM menambah 32 bytes (IV + *Auth Tag*), yang keduanya memberikan dampak penyimpanan sangat minimal. Sistem berhasil menjamin keamanan dan integritas data tanpa membebani kapasitas server.

V. PENUTUP

5.1 Kesimpulan

Berdasarkan serangkaian tahapan perancangan, implementasi, hingga pengujian kinerja yang telah dilakukan terhadap sistem *Secure Document Vault*, dapat ditarik beberapa simpulan krusial sebagai berikut:

1. **Efisiensi Kriptografi Hibrida**

   Penggunaan RSA untuk mengamankan kunci AES hanya memakan waktu sebesar **1,56%** dari total proses enkripsi, membuktikan bahwa mekanisme hibrida tidak menjadi *bottleneck* bagi performa sistem secara keseluruhan.

2. **Perbandingan Mode AES**

   Mode **GCM** terbukti lebih unggul untuk penggunaan standar karena memberikan fitur *Authenticated Encryption* dengan *overhead* ukuran file sebesar **0 bytes**, sedangkan mode **CBC** menawarkan kecepatan enkripsi murni yang sedikit lebih tinggi namun menghasilkan *overhead padding* (hingga 16 bytes).

3. **Skalabilitas Batch Processing**

   Sistem mampu menangani beban *Batch Large* (12 file) dengan stabilitas tinggi, mencatatkan total waktu proses hanya **100-148 ms**, yang menunjukkan efisiensi penggunaan metode *multipart/form-data* dalam menangani banyak dokumen sekaligus.

4. **Integritas dan Manajemen Versi**

   Penggunaan SHA-256 berhasil menjamin deteksi modifikasi ilegal pada berkas. Selain itu, fitur *Restore Version* memberikan fleksibilitas melalui strategi **Replace** (efisiensi status) dan **Copy** (integritas riwayat) dengan waktu eksekusi yang sangat cepat (<60 ms).

5.2 Saran

Penelitian ini masih memiliki ruang untuk penyempurnaan. Guna meningkatkan keamanan dan fungsionalitas sistem di masa mendatang, penulis menyarankan beberapa langkah pengembangan lanjutan:

1. **Pengembangan Antarmuka (GUI)**

   Membangun aplikasi berbasis web atau *desktop* agar pengguna non-teknis dapat memanfaatkan fitur *vault* tanpa melalui kakas seperti Postman.

2. **Integrasi Cloud Storage**

   Mengintegrasikan penyimpanan dengan layanan seperti AWS S3 atau Google Cloud Storage untuk mendukung skalabilitas penyimpanan dokumen yang lebih besar.

3. **Autentikasi Pengguna**

   Menambahkan lapisan keamanan pada API menggunakan JSON Web Token (JWT) atau OAuth2 untuk memastikan hanya pengguna terverifikasi yang dapat mengakses kunci RSA dan dokumen di dalam *vault*.

4. **Optimasi Paralelisasi**

   Meneliti penggunaan *worker threads* pada Node.js untuk proses enkripsi *batch* yang lebih besar guna meningkatkan *throughput* pada server multi-core.

DAFTAR PUSTAKA

[1] J. Bharti and S. Singh, "A Hybrid Approach Using AES-RSA Encryption for Cloud Data Security," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 21s, pp. 62–69, 2024.

[2] M. A. Al-Shabi, "Data Encryption in Cloud Storage Using AES and RSA Algorithms," *Scientific Research Journal*, vol. 8, no. 8, pp. 99-106, 2020.

[3] P. P. Parathi, "Secure File Storage Using Hybrid Cryptography," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 12, no. 4, pp. 4779-4785, 2024.

[4] S. H. Al-Nidah and A. M. Al-Bakri, "RSA and AES Based Hybrid Encryption Technique for Enhancing Data Security in Cloud Computing," *International Journal of Communication and Networking System*, vol. 12, no. 1, pp. 34-40, 2023.

[5] Junaidi, A. Louis, F. Rini, S. Mulyati, and E. Elzas, "Cryptographic Framework for Cloud-Based Document Storage Using AES-256 and SHA-256 Hybrid Systems," *Jurnal Ilmu Pengetahuan dan Teknologi Komputer (JITK)*, vol. 11, no. 2, pp. 155-164, 2025.

[6] M. A. Al-Shabi, "Data Encryption in Cloud Storage Using AES and RSA Algorithms," *Scientific Research Journal*, vol. 8, no. 8, pp. 99-106, 2020.

[7] P. P. Parathi, "Secure File Storage Using Hybrid Cryptography," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 12, no. 4, pp. 4779-4785, 2024.

[8] L. Richardson and S. Ruby, *RESTful Web Services*. Sebastopol, CA: O'Reilly Media, 2007.

[9] I. Grigorik, *High Performance Browser Networking: What Every Web Developer Should Know About Networking and Web Performance*. Sebastopol, CA: O'Reilly Media, 2013.