Implementasi Session-Based Forward Secure Field Encryption untuk Protocol Buffers dalam *Microservice* gRPC Architecture

Dyan Dananjaya Tejo Pamungkas[1], I Made Suartana[2]

[1,2] Program Studi Teknik Informatika, Universitas Negeri Surabaya
[1]dyan.22107@mhs.unesa.ac.id
[2]madesuartana@unesa.ac.id

***Abstrak*— Komunikasi antar layanan pada arsitektur *microservice* berbasis gRPC umumnya hanya mengandalkan Transport Layer Security (TLS). Mekanisme ini melindungi data selama berada dalam transmisi, tetapi tidak menjamin kerahasiaannya dalam jangka panjang karena kompromi kunci di kemudian hari berpotensi mengungkap seluruh komunikasi historis. Sifat enkripsi TLS yang menyeluruh (*all-or-nothing*) juga tidak memungkinkan perlindungan selektif terhadap *field* yang benar-benar sensitif. Berdasarkan permasalahan tersebut, penelitian ini merancang dan mengimplementasikan *framework* Session-Based Forward Secure Field Encryption (SBFSFE) menggunakan bahasa Go 1.25.0. *Framework* ini menggabungkan enkripsi pada tingkat *field* melalui anotasi Protobuf [(options.sensitive) = true], pertukaran kunci *ephemeral* ECDHE P-256 yang memberikan *forward secrecy* pada setiap sesi, serta integrasi transparan melalui gRPC *interceptor* sehingga logika bisnis yang telah berjalan tidak perlu diubah. Setiap *field* sensitif dienkripsi dengan AES-256-GCM menggunakan kunci independen hasil derivasi HKDF-SHA256, sedangkan konteks sesi disimpan secara terdistribusi pada Redis. Sepuluh skenario pengujian keamanan seluruhnya lolos, yang menunjukkan terpenuhinya *forward secrecy*, isolasi antar sesi, keamanan memori, dan ketahanan terhadap *replay attack*. Pengujian performa memperlihatkan *overhead* enkripsi yang besar pada *payload* kecil, namun pada *payload* besar (sekitar 100 KB dan 1 MB) sistem terenkripsi mencatat latensi lebih rendah dengan *throughput* 28 sampai 48% lebih tinggi dibanding *baseline*. Dengan demikian, *framework* ini paling sesuai diterapkan pada sistem medis, keuangan, maupun transfer data berukuran besar yang mengutamakan keamanan jangka panjang.**

***Kata Kunci*— *gRPC, microservice, forward secrecy, Protocol Buffers, field-level encryption, AES-256-GCM, ECDHE.*.**

I. PENDAHULUAN

Dalam satu dekade terakhir, arsitektur *microservice* telah mengubah cara aplikasi *enterprise* dibangun. Alih-alih dipertahankan sebagai satu basis kode monolitik, sistem dipecah menjadi sejumlah layanan kecil yang berdiri sendiri sehingga lebih mudah diskalakan, dikembangkan, dan dipelihara [1]. Namun keuntungan tersebut membawa konsekuensi tersendiri. Interaksi antar komponen yang sebelumnya cukup dilakukan melalui pemanggilan fungsi lokal kini berpindah ke jaringan, sehingga setiap titik komunikasi baru memperluas permukaan serangan dan data sensitif yang berpindah antar layanan menjadi rentan terhadap penyadapan maupun kebocoran. Oleh karena itu, keamanan jalur komunikasi antar layanan kerap disebut sebagai titik paling rawan dalam arsitektur ini [2].

Untuk mengamankan jalur tersebut, gRPC, yaitu kerangka komunikasi berbasis HTTP/2 dan Protocol Buffers yang banyak digunakan karena ringan dan cepat, pada umumnya mengandalkan TLS. Akan tetapi, TLS hanya melindungi data selama berada dalam transmisi. Setelah paket tiba di *server* dan didekripsi, isinya kembali menjadi *plaintext*, dan sifat perlindungannya yang *all-or-nothing* tidak memungkinkan pemilihan *field* tertentu saja untuk dilindungi. *Field-Level Encryption* hadir untuk menutup celah selektivitas tersebut, tetapi sebagian besar implementasinya belum memperhatikan satu properti penting, yaitu *forward secrecy*: jaminan bahwa komunikasi lama tetap tidak dapat dibaca sekalipun kunci bocor di kemudian hari [3]. Jaminan semacam ini umumnya diperoleh melalui kunci *ephemeral* yang berbeda pada tiap sesi menggunakan protokol Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) [4].

Sejumlah studi terdahulu telah menyentuh persoalan ini dari sudut pandang yang berbeda-beda. Perkasa et al. [5] menunjukkan bahwa enkripsi *end-to-end* berbasis Diffie-Hellman dan AES-256 layak diterapkan pada *microservice*, meskipun *forward secrecy* belum menjadi perhatian dalam penelitian tersebut. Khan dan Ahamad [6] mengevaluasi performa gRPC di atas HTTP/3, tetapi perlindungan yang dibahas tetap berhenti pada lapisan *transport*. Zdun et al. [7] menempuh pendekatan yang lebih konseptual dengan merumuskan metrik evaluasi keamanan *microservice* tanpa menyertakan implementasi nyata. Penelitian yang paling dekat dengan penelitian ini adalah Loechel et al. [8], yang memperlihatkan bahwa *interceptor* gRPC mampu memodifikasi pesan Protobuf secara transparan; gagasan itulah yang dikembangkan lebih jauh dalam penelitian ini.

Bertolak dari celah tersebut, penelitian ini membangun *framework* Session-Based Forward Secure Field Encryption (SBFSFE) di atas ekosistem gRPC-Protobuf menggunakan bahasa Go. Melalui mekanisme *interceptor*, enkripsi pada level *field* beserta jaminan *forward secrecy* disisipkan tanpa mengubah logika bisnis yang telah ada. Terdapat tiga pertanyaan yang ingin dijawab, yaitu bagaimana merancang arsitektur *framework* agar kompatibel dengan Protocol Buffers, seberapa besar dampaknya terhadap *latency* dan *throughput*, serta sejauh mana ketahanannya menghadapi skenario *cryptographic compromise* dan *session hijacking*.

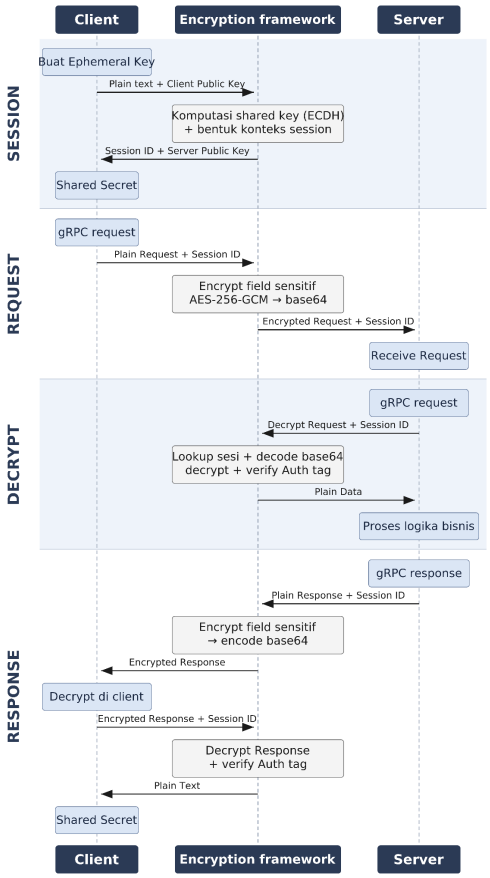
II. METODE PENELITIAN

Penelitian ini bersifat eksperimental dengan alur yang berurutan, dimulai dari analisis kebutuhan, perancangan arsitektur, implementasi dalam bahasa Go, hingga pengujian pada lingkungan yang terkontrol. Untuk mengukur beban yang ditimbulkan *framework*, performa komunikasi gRPC dengan enkripsi (*encrypted*) dibandingkan secara langsung terhadap *baseline* yang tidak menggunakan enkripsi tambahan.

A. Arsitektur Framework SBFSFE

Rancangan *framework* bertumpu pada pola *interceptor* gRPC yang memungkinkan logika enkripsi dan dekripsi disisipkan tanpa mengganggu jalannya aplikasi. Secara garis besar sistem terdiri atas tiga lapis, yaitu *Client*, *Server*, serta Encryption Framework yang berada di antara keduanya dan berperan sebagai *middleware*. Lapisan tengah ini menampung empat komponen utama: Crypto Module yang menangani ECDHE P-256, AES-256-GCM, dan HKDF-SHA256; Session Manager yang bersandar pada Redis terdistribusi; gRPC *Interceptor*; serta Field Encryptor yang bekerja melalui *reflection* Protobuf.

Framework menyediakan empat *endpoint* dengan penamaan bergaya jalur (*path*) seperti pada REST API. *Endpoint* /session/establish membuka sesi baru sekaligus melakukan pertukaran kunci ECDHE; /{service}/{method}, dengan bagian {service} dan {method} terisi otomatis sesuai layanan dan metode yang dipanggil (misalnya /UserService/GetProfile), merupakan jalur komunikasi terenkripsi; /session/renew memperbarui kunci tanpa memutus koneksi; dan /session/terminate menutup sesi sekaligus menghapus kunci dari memori. Seluruh *endpoint* diawasi *interceptor* secara otomatis: pada sisi *client interceptor* mengenkripsi data yang dikirim dan mendekripsi balasan, sedangkan pada sisi *server* berlaku sebaliknya. Alur lengkap ditunjukkan pada Gbr. 1.



Gbr. 1 Rancangan desain sistem SBFSFE pada komunikasi gRPC.

B. Mekanisme ECDHE dan Forward Secrecy

Pertukaran kunci memanfaatkan ECDHE P-256 dari paket crypto/ecdh bawaan Go. Kurva P-256 dipilih karena menawarkan tingkat keamanan setara kunci simetris 128-bit dengan beban komputasi yang relatif ringan. Pada saat sesi dibentuk, *client* dan *server* masing-masing membangkitkan pasangan kunci *ephemeral*, saling menukar kunci publik, kemudian menghitung *shared secret* di sisi masing-masing tanpa secret tersebut pernah melintas di jaringan. Setelah *shared secret* diperoleh, kunci privat *ephemeral* segera dimusnahkan oleh fungsi ZeroECDHPrivateKey() yang menimpa setiap *byte*-nya dengan nol. Mekanisme inilah yang menegakkan *forward secrecy*, sebab apabila memori *server* disalin setelah sesi berakhir, tidak ada lagi materi kunci yang dapat dimanfaatkan oleh penyerang.

C. Enkripsi Field AES-256-GCM dan Anotasi Protobuf

Penentuan *field* yang harus dienkripsi dilakukan oleh Field Encryptor dengan membaca anotasi [(options.sensitive)=true] pada berkas .proto melalui *reflection*. Pada pesan uji, misalnya CreateUserRequest, *field* name dan email dibiarkan sebagai *plaintext*, sedangkan password, address, credit_card, dan phone ditandai sensitif. Setiap *field* memperoleh kunci tersendiri yang diturunkan melalui HKDF-SHA256 dengan nama *field* sebagai *context info*; dengan demikian, keberhasilan membongkar satu *field* tidak memberikan keuntungan apa pun untuk membuka *field* lainnya. Proses enkripsi menggunakan AES-256-GCM, yaitu mode *authenticated encryption* yang menjaga kerahasiaan sekaligus integritas data. Karena setiap operasi memakai *nonce* acak sepanjang 12 *byte*, dua *plaintext* yang identik tetap menghasilkan *ciphertext* yang berbeda. Hasil akhir berupa gabungan *nonce*, *ciphertext*, dan GCM tag kemudian dikodekan ke *base64* agar dapat disimpan pada *field* Protobuf bertipe *string*.

D. Manajemen Sesi dan Dual Expiration

Siklus hidup sesi ditangani oleh Session Manager dengan Redis 7.x sebagai penyimpan terdistribusi, sehingga beberapa *instance* layanan dapat berbagi konteks sesi yang sama dan sistem tetap dapat diskalakan secara horizontal. Setiap konteks menyimpan *session ID*, *shared secret*, timestamp, penghitung pesan, dan identitas *client*. Sebuah sesi dapat berakhir melalui dua jalur yang berjalan paralel, yaitu habisnya masa hidup (*default* satu jam) atau tercapainya batas jumlah pesan (*default* 10.000); kondisi yang lebih dahulu terpenuhi akan memicu kedaluwarsa. Khusus pada saat *benchmark* performa, batas pesan dinaikkan hingga 100 juta agar sesi tidak kedaluwarsa selama pengukuran berlangsung. Ketika sesi ditutup, fungsi Zeroize() menimpa seluruh *byte shared secret* dengan nol. Adapun upaya *replay attack* dicegah oleh *ReplayCache* yang menolak *request-id* duplikat selama sesi masih aktif.

E. Environment dan Metodologi Pengujian

Spesifikasi lingkungan implementasi dirangkum pada Tabel I. Aspek keamanan diuji menggunakan Go testing framework melalui sepuluh skenario pada direktori security_test/, yang mencakup beragam aspek: panjang *shared secret* P-256, keacakan *ciphertext*, deteksi *tampering*, isolasi antar sesi, hilangnya kemampuan dekripsi setelah sesi dihapus, *zeroing* memori, penangkalan *replay attack*, hingga resistansi terhadap *timing attack*.

Aspek performa diukur dengan *benchmark* Go. Empat ukuran *payload* (1 KB, 10 KB, 100 KB, dan 1 MB) disilangkan dengan lima tingkat konkurensi (c10 sampai c1000); setiap

kombinasi dijalankan selama 60 detik dan diulang tiga kali, sehingga terkumpul 20 skenario atau 60 *run*. Metrik yang dicatat meliputi *latency* pada persentil P50, P95, dan P99, *throughput* dalam *requests per second* (RPS), serta konsumsi CPU dan memori *server*. Dua ambang batas ditetapkan sejak awal: *overhead* P99 *latency* tidak boleh melebihi 10% dan penurunan *throughput* tidak melampaui 15%.

TABEL I
SPESIFIKASI ENVIRONMENT IMPLEMENTASI

Komponen	Spesifikasi
Sistem Operasi	Windows 11 64-bit
Processor	AMD Ryzen 7 7735HS
RAM	16 GB LPDDR5
Go	go1.25.0 windows/amd64
gRPC	google.golang.org/grpc v1.79.0
Protocol Buffer	google.golang.org/protobuf v1.36.11
Redis	7.x (Docker / lokal)
Kripto	golang.org/x/crypto v0.48.0 (HKDF, ecdh)
Compiler	protoc + protoc-gen-go + protoc-gen-go-grpc

III. HASIL DAN PEMBAHASAN

Bagian ini memaparkan dua hal, yaitu hasil pengujian keamanan dan evaluasi performa *framework* SBFSFE. Seluruh implementasi ditulis dalam bahasa Go dan dapat dijalankan langsung melalui *docker compose*.

A. Hasil Pengujian Keamanan

Pengujian keamanan dilakukan melalui sepuluh skenario yang masing-masing dirancang untuk membuktikan satu properti keamanan secara terpisah. Rangkuman lengkapnya disajikan pada Tabel II, dan seluruh skenario tercatat PASS.

TABEL II
RINGKASAN HASIL PENGUJIAN KEAMANAN SBFSFE

No	Skenario	Tujuan	Hasil Aktual
1	SharedSecretIsP256Length	Shared secret ECDH P-256 harus 32 byte	Secret = 32 byte sesuai spesifikasi
2	DifferentSessionsDifferentSecret	Setiap sesi menghasilkan ephemeral secret berbeda	Dua sesi berturut-turut menghasilkan secret tidak sama
3	CannotDecryptAfterSessionDeleted	Penghapusan sesi memutus kemampuan dekripsi	Request setelah sesi dihapus ditolak (Unauthenticated)
4	EncryptionProducesUniqueCiphertext	Plaintext sama dienkripsi dua kali harus beda ciphertext	Dua enkripsi berturut menghasilkan ciphertext berbeda
5	TamperedCiphertextDetected	Modifikasi GCM auth tag harus terdeteksi	Server mengembalikan error Unauthenticated
6	SensitiveFieldsEncryptedOnWire	Field sensitif tidak boleh tampil plaintext di jaringan	credit_card dan password tidak ditemukan pada wire capture
7	CrossSessionDecryptionFails	Ciphertext sesi A tidak terbaca oleh kunci sesi B	Cross-session decryption gagal (Unauthenticated)
8	EphemeralKeyZeroedAfterTerminate	Key material di-zero setelah sesi berakhir	Area memori kunci berisi nol setelah terminasi
9	ReplayAttackPrevented	Request-id sama tidak boleh diterima lebih dari sekali	Request duplikat ditolak (replay attack terdeteksi)
10	TimingAttackResistance	Respons valid dan invalid tidak boleh beda secara waktu	Koefisien variasi < 30%, tidak ada pola timing bocor

Hasil pengujian keamanan pada Tabel II dapat dibaca secara berkelompok. Skenario 1 dan 2 menegaskan bahwa setiap sesi memakai kunci yang berbeda, sehingga kompromi pada satu sesi tidak berdampak pada sesi lainnya. Skenario 3 dan 7 memperlihatkan bahwa isolasi sesi tidak sekadar bergantung pada pencocokan ID, melainkan dijamin secara kriptografis. Pada skenario 5, perubahan satu bit pada *ciphertext* langsung terdeteksi oleh AES-GCM, sementara skenario 8 memastikan materi kunci benar-benar terhapus dari memori. Dua skenario terakhir, yaitu skenario 9 dan 10, masing-masing menutup celah *replay attack* dan memastikan waktu respons tidak membocorkan informasi apa pun

B. Evaluasi Latency

Benchmark membandingkan dua konfigurasi, yaitu *baseline* (gRPC tanpa enkripsi tambahan) dan *encrypted* (gRPC dengan *framework* aktif). Pembahasan *latency* difokuskan pada persentil P99. Nilai P99 menyatakan bahwa 99 persen permintaan selesai lebih cepat daripada angka tersebut dan hanya satu persen sisanya yang lebih lambat; dengan demikian P99 mewakili kondisi ekor (*tail latency*) terburuk yang masih dialami pengguna, dan persentil inilah yang dijadikan acuan *threshold*. Pada Tabel III sampai VI, kolom *Overhead* P99 dihitung sebagai (P99 *encrypted* − P99 *baseline*) / P99 *baseline* × 100%; tanda positif berarti *latency encrypted* lebih tinggi daripada *baseline*, sedangkan tanda negatif berarti lebih rendah.

Pada *payload* 1 KB (Tabel III), biaya enkripsi terlihat paling besar. Pada sepuluh pengguna serentak, *latency* P99 *encrypted* telah mencapai dua kali lipat *baseline*, dan selisihnya terus melebar seiring bertambahnya beban hingga mencapai 763,73 persen pada lima ratus pengguna. Penyebab utamanya bukan ukuran data, melainkan rangkaian operasi tetap yang harus dilalui setiap permintaan: pemeriksaan sesi ke Redis, derivasi kunci melalui HKDF, dan enkripsi dengan AES-GCM. Ketika muatan yang dikirim hanya satu *kilobyte*, rangkaian operasi itulah yang menghabiskan hampir seluruh waktu pemrosesan.

TABEL III
PERBANDINGAN LATENCY PAYLOAD SMALL (1 KB)

Skenario	Baseline P99 (ms)	Encrypted P99 (ms)	Overhead P99
c10	1,206	2,508	+107,96%
c50	2,868	12,785	+345,78%
c100	4,676	18,797	+301,99%
c500	13,621	117,648	+763,73%
c1000	56,166	137,432	+144,69%

Nilai dalam ms. Overhead P99 positif berarti *latency* encrypted lebih tinggi dari baseline.

Beban 10 KB menunjukkan pola serupa pada konkurensi rendah, tetapi memunculkan satu anomali pada seratus pengguna. Pada titik tersebut, P99 *encrypted* (22,453 ms) justru lebih rendah daripada *baseline* (25,567 ms), seolah-olah enkripsi mempercepat sistem, seperti terlihat pada Tabel IV. Penelusuran terhadap angka *baseline* memberikan penjelasan: P99 *baseline* melonjak secara tidak wajar ke 25,567 ms

dibandingkan skenario tetangganya pada c50 dan c500, sedangkan kurva *encrypted* tetap landai. Karena itu keunggulan 12,18 persen pada titik ini lebih tepat dibaca sebagai fluktuasi pengukuran pada ekor *latency baseline*, bukan bukti bahwa enkripsi benar-benar lebih cepat. Untuk memastikannya, pengujian perlu diulang dengan jumlah *run* yang lebih banyak.

TABEL IV
PERBANDINGAN LATENCY PAYLOAD MEDIUM (10 KB)

Skenario	Baseline P99 (ms)	Encrypted P99 (ms)	Overhead P99
c10	1,625	2,887	+77,66%
c50	3,445	12,751	+270,13%
c100	25,567	22,453	−12,18%
c500	58,677	80,150	+36,60%
c1000	96,497	150,683	+56,15%

Nilai dalam ms. *Overhead negatif* pada c100 diduga berasal dari outlier *tail-latency baseline* (lihat pembahasan).

Pola data berubah ketika *payload* mencapai 100 KB. P99 *encrypted* kini secara konsisten berada di bawah *baseline* pada semua tingkat konkurensi, dengan selisih sekitar 17 sampai 23 persen lebih cepat, seperti terlihat pada Tabel V. Pada ukuran ini, biaya kriptografi yang nilainya tetap menjadi hampir tidak berarti dibandingkan total waktu pengiriman. Selisih yang konsisten menguntungkan *encrypted* ini lebih tepat dipahami sebagai efek perbedaan cara kedua *server* mengelola *buffer frame* HTTP/2, bukan sebagai bukti langsung bahwa enkripsi mempercepat komunikasi.

TABEL V
PERBANDINGAN LATENCY PAYLOAD LARGE (100 KB)

Skenario	Baseline P99 (ms)	Encrypted P99 (ms)	Overhead P99
c10	8,540	6,633	−22,33%
c50	40,908	33,259	−18,70%
c100	82,342	63,037	−23,44%
c500	419,604	324,653	−22,63%
c1000	760,252	627,354	−17,48%

Nilai dalam ms. Overhead P99 negatif menandakan *latency* encrypted lebih rendah dari baseline.

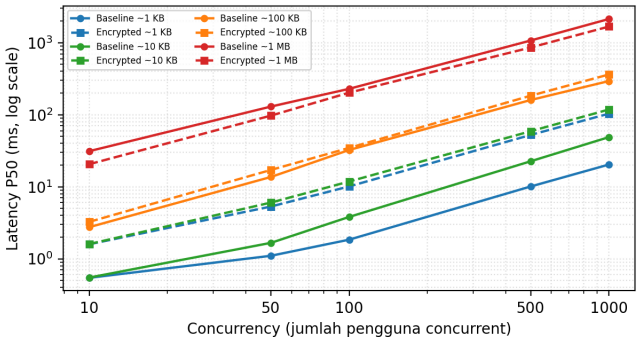
Pada *payload* 1 MB (Tabel VI) selisih tersebut semakin melebar dan tetap berpihak pada *encrypted*. Pada lima puluh pengguna (c50) sistem *encrypted* 59,27 persen lebih cepat, dan pada beban paling berat seribu pengguna (c1000) masih unggul 9,02 persen. Selama muatan cukup besar untuk mendominasi waktu komunikasi, kehadiran *framework* hampir tidak menambah biaya. Meski demikian, keunggulan ini sengaja tidak diinterpretasikan sebagai bukti bahwa enkripsi membuat sistem lebih cepat. Penafsiran yang lebih berhati-hati adalah bahwa biaya kriptografi pada *payload* besar berada di bawah rentang variasi pengukuran, sementara perbedaan konfigurasi *runtime* antar *server* belum sepenuhnya dikendalikan. Keterbatasan validitas internal ini dicatat sebagai bahan pengujian lanjutan.

TABEL VI
PERBANDINGAN LATENCY PAYLOAD VERY LARGE (1 MB)

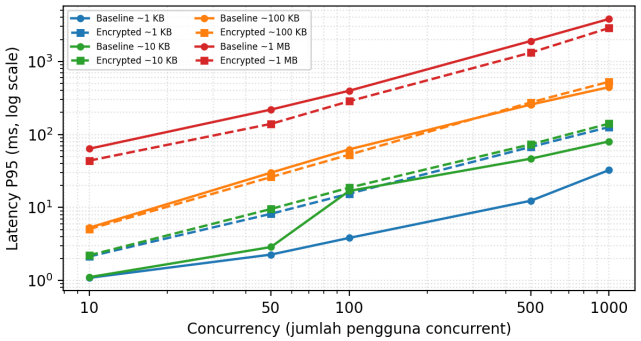
Skenario	Baseline P99 (ms)	Encrypted P99 (ms)	Overhead P99
c10	89,848	58,095	−35,34%
c50	403,535	164,341	−59,27%
c100	615,988	329,037	−46,58%
c500	9145,931	7016,481	−23,28%
c1000	9392,555	8544,884	−9,02%

Nilai dalam ms. Sistem encrypted lebih cepat di seluruh tingkat konkurensi pada P99. Lonjakan tajam P99 pada c500 dan c1000 (mencapai sekitar 9 detik) mencerminkan penumpukan *tail-latency* akibat antrean pada beban 1 MB berkonkurensi tinggi.

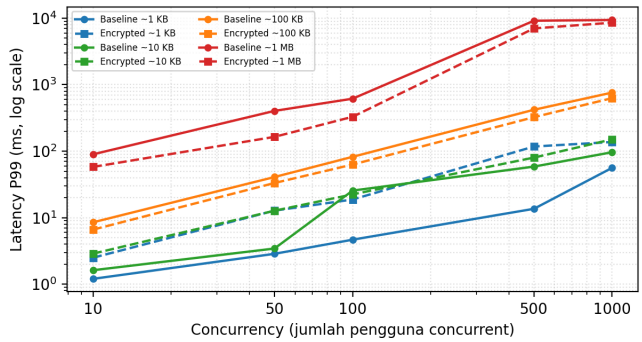
Sebagai pelengkap tabel, sebaran *latency* seluruh skenario pada tiga persentil ditampilkan pada Gbr. 2 sampai Gbr. 4. P50 (median) mewakili pengalaman tipikal mayoritas permintaan, P95 menggambarkan kondisi mendekati terburuk, dan P99 merupakan ekor paling ekstrem yang menjadi acuan *threshold*. Ketiganya bergerak dengan pola yang serupa: *overhead* enkripsi besar pada *payload* kecil, kemudian menyusut seiring membesarnya *payload* hingga berbalik menjadi lebih cepat daripada *baseline* pada 100 KB dan 1 MB. Gbr. 5 merangkum *overhead* P99 tiap skenario terhadap ambang 10 persen.



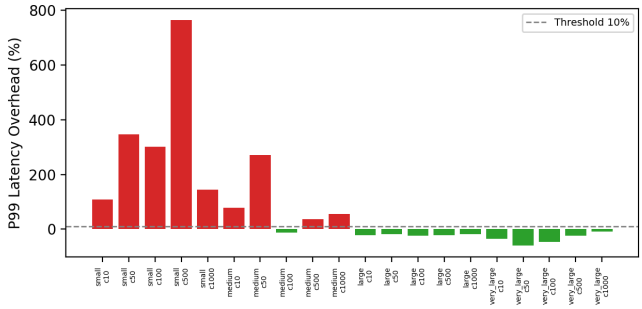
Gbr. 2 Line chart P50 *latency baseline* dan encrypted untuk seluruh skenario.



Gbr. 3 Line chart P95 *latency baseline* dan encrypted untuk seluruh skenario.



Gbr. 4 Line chart P99 latency baseline dan encrypted untuk seluruh skenario.



Gbr. 5 Bar chart P99 overhead (%) per skenario terhadap threshold 10%.

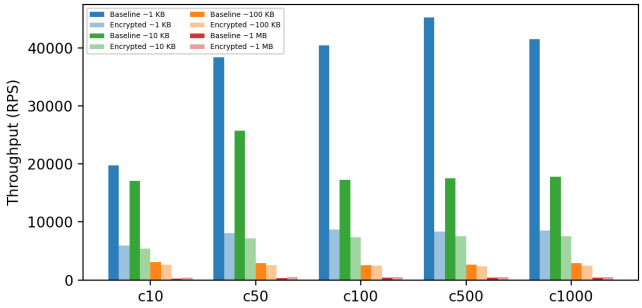
C. Evaluasi Throughput

Throughput mengikuti pola yang sama dengan *latency*. Pada *payload* kecil dan sedang, jumlah permintaan per detik turun pada kisaran 57 sampai 82 persen, jauh melampaui ambang 15 persen, karena setiap permintaan menanggung biaya kriptografi yang sama besar. Pada beban 100 KB kondisinya jauh lebih longgar: hampir semua skenario berada di dalam batas, dan hanya konkurensi seribu pengguna yang sedikit melampauinya pada angka 16,96 persen. Pada *payload* 1 MB keadaan justru berbalik, yaitu *throughput encrypted* 28 sampai 48 persen lebih tinggi daripada *baseline*. Perlu dicatat bahwa besaran *throughput* absolut yang terukur tergolong rendah dibandingkan kapasitas teoretis gRPC. Hal ini lebih mencerminkan keterbatasan *harness* pengujian lokal, tempat *client* dan *server* berbagi satu mesin, sehingga yang layak diperbandingkan adalah selisih relatif antara *baseline* dan *encrypted*, bukan angka mutlaknya. Data selengkapnya disajikan pada Tabel VII; Gbr. 6 menyandingkan *throughput* mentah kedua konfigurasi, sedangkan Gbr. 7 menampilkan laju penurunan *throughput* terhadap konkurensi dengan garis putus-putus sebagai penanda ambang 15 persen.

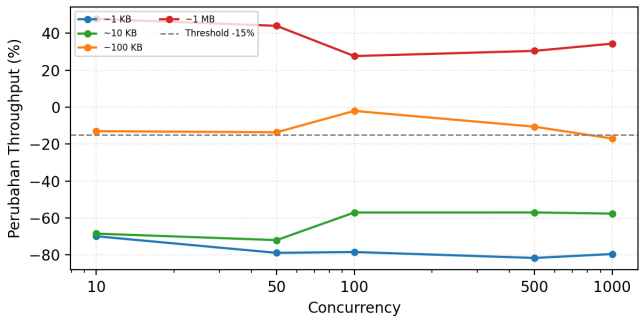
TABEL VII
PERBANDINGAN THROUGHPUT BASELINE DAN ENCRYPTED

Skenario	Baseline (RPS)	Encrypted (RPS)	Selisih (%)
small_c10	19.728	5.939	−69,89%
small_c50	38.406	8.108	−78,89%
small_c100	40.474	8.733	−78,42%
small_c500	45.249	8.306	−81,64%
small_c1000	41.574	8.520	−79,51%
medium_c10	17.101	5.390	−68,48%
medium_c50	25.773	7.221	−71,98%
medium_c100	17.241	7.408	−57,03%
medium_c500	17.534	7.541	−56,99%
medium_c1000	17.762	7.526	−57,63%
large_c10	3.071	2.672	−12,98%
large_c50	2.929	2.531	−13,59%
large_c100	2.546	2.495	−2,00%
large_c500	2.669	2.387	−10,56%
large_c1000	2.942	2.443	−16,96%
very_large_c10	269	397	+47,81%
very_large_c50	341	491	+44,07%
very_large_c100	371	473	+27,70%
very_large_c500	395	516	+30,51%
very_large_c1000	388	522	+34,43%

Nilai dalam RPS. Tanda negatif berarti throughput encrypted lebih rendah dari baseline, tanda positif berarti lebih tinggi (kebalikan interpretasi kolom Overhead pada Tabel III–VI).



Gbr. 4 Bar chart throughput (RPS) baseline dan encrypted per kategori payload.



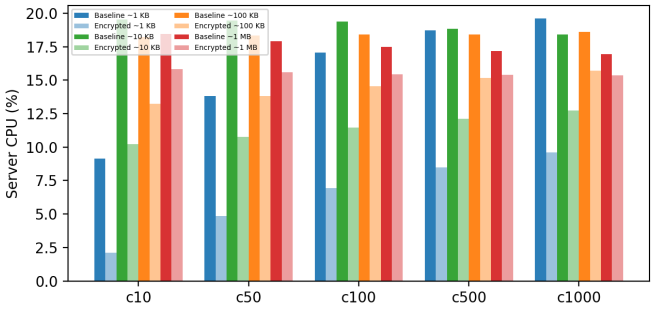
Gbr. 5 Line chart degradasi throughput (%) terhadap concurrency untuk tiap kategori payload.

D. Penggunaan CPU dan Memori

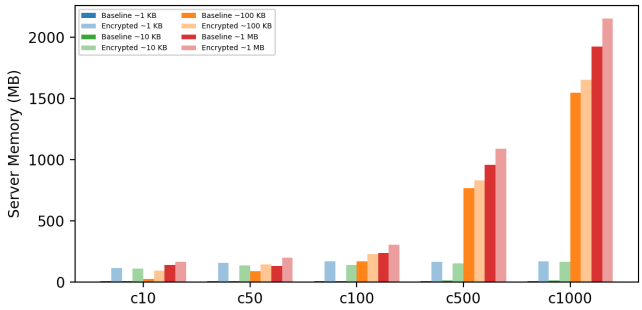
Salah satu temuan yang menarik perhatian adalah penggunaan CPU *server* pada konfigurasi *encrypted* yang justru selalu lebih rendah daripada *baseline*, dengan selisih 1,58 sampai 10,11 *percentage point*. Penjelasannya berada di sisi *client*. Karena *client* memerlukan waktu tambahan untuk mengenkripsi setiap permintaan sebelum mengirim, laju permintaan yang tiba di *server* ikut tertahan sehingga beban kerja *server* berkurang. Efek pembatasan laju dari hulu ini juga menjelaskan mengapa sejumlah metrik *server*, termasuk *latency* pada *payload* besar, tampak lebih baik pada konfigurasi *encrypted*: yang terukur sebagian merupakan efek pembatasan laju dari sisi *client*, bukan murni efisiensi *server*. Memori menunjukkan hasil sebaliknya. *Server encrypted* mengonsumsi RAM jauh lebih banyak, hingga 162 MB pada *payload* kecil,

sebagai konsekuensi penyimpanan konteks tiap sesi pada Redis berupa *shared secret*, metadata, dan penghitung pesan. Selisih tersebut mengecil pada *payload* besar karena dalam rentang waktu yang sama lebih sedikit sesi yang sempat dibuka. Perbandingan penggunaan CPU dan memori disajikan pada Gbr. 8 dan Gbr. 9.

.



Gbr. 6 Bar chart penggunaan CPU server (%) baseline dan encrypted untuk seluruh skenario.



Gbr. 7 Bar chart penggunaan memori server (MB) baseline dan encrypted untuk seluruh skenario.

E. Evaluasi terhadap Threshold

bang yang ditetapkan di awal, yaitu *overhead* P99 *latency* maksimal 10 persen dan penurunan *throughput* maksimal 15 persen, polanya cukup jelas: keduanya hanya terpenuhi secara konsisten pada *payload* besar.

Pada 1 KB dan 10 KB kedua ambang jauh terlampaui, dengan *latency* membengkak dari sekitar dua kali lipat hingga hampir delapan kali *baseline* dan *throughput* menyusut 57 sampai 82 persen. Pada 100 KB ambang *latency* P99 telah terpenuhi karena *encrypted* justru lebih cepat, sementara *throughput* hanya sedikit di luar batas pada konkurensi tertinggi. Pada 1 MB keduanya terpenuhi dengan longgar, yaitu *latency* P99 lebih rendah dari *baseline* dan *throughput* 28 sampai 48 persen lebih tinggi. Terlampauinya ambang pada *payload* kecil dan sedang bukan merupakan indikasi kegagalan *framework*, melainkan karakteristik bawaan *field-level encryption* yang biayanya melekat pada setiap *field* dan setiap permintaan, bukan tumbuh mengikuti ukuran data.

IV. KESIMPULAN

Penelitian ini menghasilkan *framework* Session-Based Forward Secure Field Encryption (SBFSFE) untuk Protocol Buffers pada lingkungan *microservice* gRPC. Di dalamnya, ECDHE P-256, AES-256-GCM, dan HKDF-SHA256 dipadukan dengan manajemen sesi terdistribusi berbasis Redis beserta mekanisme *dual expiration*, dan seluruhnya berjalan secara transparan melalui gRPC *interceptor* tanpa perlu mengubah *handler* bisnis.

Dari sisi keamanan, kesepuluh skenario pengujian seluruhnya PASS, sehingga *forward secrecy*, isolasi sesi, keamanan memori, dan ketahanan terhadap *replay attack* terbukti terpenuhi. Dari sisi performa, hasilnya bergantung pada ukuran data: pada *payload* 100 KB *latency* P99 sistem *encrypted* 17 sampai 23% lebih rendah dari *baseline*, dan pada 1 MB *throughput*-nya 28 sampai 48% lebih tinggi, sementara pada *payload* kecil sampai sedang *overhead*-nya justru melampaui ambang yang ditetapkan. Perlu ditegaskan bahwa keunggulan pada *payload* besar diperlakukan sebagai indikasi bahwa *overhead* enkripsi berada di bawah margin variasi pengukuran, bukan sebagai klaim bahwa enkripsi mempercepat sistem; pengendalian konfigurasi *runtime* antar *server* menjadi pekerjaan lanjutan. Dengan catatan tersebut, *framework* ini paling sesuai digunakan oleh layanan yang memang memindahkan data dalam jumlah besar, misalnya transfer berkas, aliran data sensor, atau pertukaran data antar *data warehouse*.

Beberapa arah pengembangan layak ditempuh selanjutnya. *Overhead* pada *payload* kecil berpotensi ditekan dengan menambahkan *in-process session cache*; pengujian pada *deployment* Kubernetes dengan trafik yang realistis akan memberikan gambaran yang lebih dekat dengan kondisi produksi; migrasi ke ECDHE X25519 atau algoritma *post-quantum* patut dipertimbangkan demi ketahanan jangka panjang; serta dukungan untuk *streaming* gRPC dapat memperluas cakupan *framework*.

REFERENSI

[1] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed. Sebastopol, CA: O'Reilly Media, 2021.

[2] R. Chandramouli, Security Strategies for Microservices-Based Application Systems, NIST Special Publication 800-204. Gaithersburg, MD: National Institute of Standards and Technology, 2019.

[3] W. Diffie, P. C. Van Oorschot, dan M. J. Wiener, "Authentication and authenticated key exchanges," Designs, Codes and Cryptography, vol. 2, no. 2, hal. 107–125, 1992.

[4] C. Boyd dan A. Mathuria, Protocols for Authentication and Key Establishment, 2nd ed. Berlin, Germany: Springer, 2020.

[5] M. J. P. Perkasa, H. M. Ramdan, A. J. Maliki, dan Somantri, "Implementation of secure end-to-end encrypted chat application using Diffie–Hellman key exchange and AES-256 in a microservice architecture," Engineering Proceedings, vol. 107, no. 1, Art. 98, 2025.

[6] I. Khan dan M. K. Ahamad, "Enhancing security and performance of gRPC-based microservices using HTTP/3 and AES-256 encryption," J. Information Systems Engineering and Management, vol. 10, no. 42s, 2025.

[7] U. Zdun, P. Queval, G. Simhandl, R. Scandariato, S. Chakravarty, M. Jelic, dan A. Jovanovic, "Microservice security metrics for secure communication, identity management, and observability," ACM Trans. Softw. Eng. Methodol., vol. 32, no. 1, Art. 16, 2023.

[8] L. Loechel, S.-R. Akbayin, E. Grünewald, J. Kiesel, I. Strelnikova, T. Janke, dan F. Pallas, "Hook-in privacy techniques for gRPC-based microservice communication," in Proc. 24th Int. Conf. Web Engineering (ICWE), 2024 (arXiv:2404.05598).