BAB II
LANDASAN TEORI

2.1 Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES), yang dikenal pula sebagai algoritma Rijndael, merupakan standar enkripsi simetris yang ditetapkan oleh NIST pada tahun 2001 sebagai pengganti Data Encryption Standard (DES) (Tuo, 2023). Dalam konteks sistem *end-to-end encryption* (E2EE), AES memegang peran sentral sebagai algoritma enkripsi data (*data encryption layer*), yakni komponen yang secara langsung mengubah *plaintext* baik berupa pesan teks maupun file media menjadi *ciphertext* yang tidak dapat dibaca oleh pihak ketiga, termasuk *server relay*. Dalam skema E2EE berbasis kriptografi hibrida, AES umumnya diposisikan sebagai algoritma simetris yang menerima kunci sesi (*session key*) dari hasil derivasi pertukaran kunci asimetris, sehingga mampu memenuhi kebutuhan *confidentiality*, *integrity*, dan *authenticity* secara simultan (Alatawi & Saxena, 2023; Knodel et al., 2024).

AES beroperasi pada blok data berukuran 128 bit dengan dukungan tiga panjang kunci, 128, 192, dan 256 bit menentukan jumlah ronde masing-masing sebesar 10, 12, dan 14 ronde. Dalam penelitian ini digunakan AES-256 yang menghasilkan 14 ronde transformasi, memberikan tingkat keamanan kriptografis tertinggi yang relevan terhadap kebutuhan perlindungan data sensitif dalam komunikasi *real-time* (Sarkar et al., 2024). Secara struktural, AES menggunakan jaringan substitution-permutation (SP network) yang memproses matriks state 4×4 *byte* melalui empat operasi per ronde, SubBytes, ShiftRows, MixColumns, dan

AddRoundKey. Operasi SubBytes menciptakan confusion melalui substitusi non-linier berbasis S-box, sementara MixColumns dan ShiftRows bersama-sama menciptakan diffusion bit yang tinggi di seluruh blok properti ini menjamin bahwa perubahan satu bit *plaintext* akan menyebar ke seluruh *ciphertext* (*avalanche effect*), yang merupakan syarat fundamental keamanan enkripsi pada sistem E2EE (Abikoye et al., 2019).

Dari perspektif performa, AES dirancang untuk efisien baik pada perangkat keras maupun perangkat lunak, menjadikannya kandidat yang tepat untuk enkripsi data dengan volume bervariasi dari pesan teks berukuran puluhan *byte* hingga file media berukuran beberapa megabyte. Faktor penentu performa AES dalam konteks penelitian ini meliputi panjang kunci yang digunakan, ukuran data yang diproses, serta mode operasi yang dipilih. Evaluasi performa pada lingkungan dengan keterbatasan sumber daya menunjukkan bahwa beban komputasi AES meningkat secara signifikan seiring bertambahnya ukuran data, sehingga pemilihan mode operasi yang tepat menjadi krusial untuk menjaga efisiensi sistem (El-Dalahmeh et al., 2024; Kim et al., 2020). Mode yang berbeda menghasilkan karakteristik keamanan dan performa yang berbeda secara signifikan, AES dalam mode CBC atau ECB, misalnya, tidak menyediakan autentikasi pesan sehingga tidak memadai untuk sistem E2EE yang mensyaratkan *confidentiality* sekaligus *integrity* dan *authenticity* secara bersamaan (Gueron et al., 2019).

Meskipun AES secara desain sangat aman terhadap serangan kriptanalisis konvensional, keamanan aktualnya dalam sistem E2EE sangat bergantung pada dua faktor, (1) kualitas kunci sesi yang digunakan, yang dalam penelitian ini dijamin

oleh mekanisme ECDH-HKDF, dan (2) mode operasi yang dipilih, yang menentukan ketahanan terhadap serangan seperti *nonce reuse* dan *chosen-plaintext attack*. Hal inilah yang melatarbelakangi pemilihan AES-GCM-SIV sebagai mode operasi dalam penelitian ini, yakni untuk mengatasi kelemahan inheren pada AES-GCM standar dalam skenario kesalahan pengelolaan *nonce* (*nonce misuse*) (Mattsson, 2024).

Gambar II.1 Arsitektur Algoritma AES-256 (Yang & Johansson, 2020)

Gambar 2.1 mengilustrasikan arsitektur algoritma AES-256 yang digunakan dalam penelitian ini. Proses enkripsi diawali dengan operasi AddRoundKey awal, di mana *plaintext* 128-bit di-XOR langsung dengan kunci ronde pertama sebelum memasuki tahap ronde utama. Pada ronde 1 hingga 13, setiap ronde menjalankan empat operasi secara berurutan: SubBytes, ShiftRows, MixColumns, dan AddRoundKey, di mana setiap kunci ronde dihasilkan secara bertahap melalui proses *key expansion* dari kunci utama 256-bit. Pada ronde terakhir (ronde 14), operasi MixColumns dihilangkan sehingga hanya SubBytes, ShiftRows, dan AddRoundKey yang dijalankan, menjaga simetri struktur enkripsi dan dekripsi. Hasil akhir dari seluruh proses transformasi ini adalah *ciphertext* 128-bit yang siap dikirimkan melalui kanal komunikasi dalam skema E2EE.

2.2 Authenticated Encrypted with Associated Data (AEAD)

Authenticated Encryption with *Associated Data* (AEAD) merupakan fondasi kriptografi utama pada sistem *end-to-end encryption* (E2EE) modern, termasuk aplikasi pesan instan. AEAD memungkinkan enkripsi sekaligus autentikasi pesan

dalam satu skema terpadu, sehingga sangat selaras dengan kebutuhan pertukaran pesan teks dan file media pada arsitektur *server relay* yang tidak dipercaya (Poettering & Rösler, 2020). AEAD didefinisikan sebagai skema kriptografi simetris yang, dengan satu primitif, sekaligus menjamin kerahasiaan dan keutuhan serta keaslian data. Fungsi enkripsi AEAD menerima kunci rahasia, *nonce*, *plaintext*, dan *associated data* (AD), lalu menghasilkan *ciphertext* dan tag autentikasi; fungsi dekripsi hanya mengembalikan *plaintext* jika tag valid, sehingga setiap modifikasi oleh pihak ketiga akan terdeteksi. AEAD dirancang untuk kasus di mana sebagian informasi harus dilindungi kerahasiaannya (isi pesan), sementara sebagian lain seperti *header* paket, metadata protokol, atau ID sesi tetap dikirim *in the clear* tetapi harus turut diautentikasi sebagai *associated data* (Poettering & Rösler, 2020).

Dalam skema E2EE, semua operasi enkripsi dan dekripsi dilakukan di endpoint pengguna (klien), sementara jaringan dan server hanya bertindak sebagai media pengiriman pesan (Alatawi & Saxena, 2023). AEAD di sini berperan sebagai kanal simetris yang melindungi isi pesan setelah kunci sesi diperoleh melalui protokol pertukaran kunci seperti Elliptic Curve Diffie-Hellman; setiap pesan teks maupun file media dienkripsi dengan AEAD sebelum dikirim melalui server (Gandhi et al., 2026). Dengan demikian, AEAD berfungsi sebagai lapisan keamanan simetris yang terikat kuat dengan konteks E2EE, memastikan bahwa setiap pesan hanya dapat didekripsi dan diverifikasi secara benar oleh endpoint yang memiliki kunci sesi yang sah.

Pada banyak layanan pesan instan, server hanya berperan sebagai *relay* yang menyimpan sementara dan meneruskan pesan antar klien. Model zero-trust ini mengasumsikan bahwa server dapat disusupi, bersikap jahat, atau berada di bawah pengawasan pihak ketiga, sehingga tidak boleh mendapatkan *plaintext* ataupun kunci kriptografi. Secara lebih konkret, dalam model ini server diasumsikan mampu melakukan tiga hal, (1) membaca seluruh *ciphertext* yang melintas; (2) memodifikasi atau menahan pesan secara aktif; dan (3) menyerahkan data tersimpan kepada pihak ketiga atas permintaan hukum maupun akibat peretasan. AEAD sangat sesuai dengan model ini karena semua operasi kriptografi berlangsung di sisi klien, sehingga server hanya menerima dan meneruskan *ciphertext* yang sudah memuat perlindungan kerahasiaan dan integritas sekaligus tanpa pernah memiliki akses terhadap *plaintext* maupun kunci sesi pengguna (Alatawi & Saxena, 2023).

Aplikasi chat E2EE harus menghadapi beragam ancaman yang dapat dikategorikan ke dalam tiga kelompok utama. Pertama, ancaman pasif berupa *eavesdropping*, yaitu pihak yang menyadap kanal komunikasi untuk merekonstruksi isi pesan. AEAD meniadakan manfaat praktis dari ancaman ini karena *ciphertext* dan tag yang dihasilkan tampak acak bagi pengintai, sehingga isi pesan teks maupun file media tidak dapat direkonstruksi tanpa kunci simetris yang benar. Kedua, ancaman aktif berupa modifikasi pesan dan penyisipan paket baru. Integritas yang terikat secara kriptografis pada *ciphertext* dan *associated data* mencegah seluruh bentuk manipulasi ini. Adapun *replay attack* ditangani melalui mekanisme tambahan pada level protokol, yaitu *replay cache* dan *timestamp* yang

disisipkan sebagai *associated data*, bukan melalui properti bawaan AEAD. Integritas yang terikat secara kriptografis pada *ciphertext* dan *associated data* mencegah seluruh bentuk manipulasi ini; perubahan bit tunggal sekalipun pada *ciphertext*, *nonce*, atau AD akan menyebabkan verifikasi tag gagal sehingga pesan ditolak oleh klien (Barbosa & Farshim, 2018). Ketiga, ancaman *man-in-the-middle* (MitM) aktif pada fase pertukaran kunci, di mana penyerang mencegat dan menggantikan kunci publik ECDH.

Dari seluruh skema AEAD yang tersedia, penelitian ini secara spesifik memilih AES-GCM-SIV sebagai implementasi AEAD. Keputusan ini didasarkan pada satu kelemahan kritis AES-GCM standar dalam konteks model ancaman dimana apabila *nonce* yang sama digunakan dua kali baik akibat kesalahan implementasi, *race condition* pada sistem konkuren, maupun kegagalan *random number generator* di sisi klien seluruh jaminan kerahasiaan dan integritas AES-GCM runtuh seketika. AES-GCM-SIV dirancang khusus untuk mengatasi kelemahan ini melalui mekanisme *nonce-misuse resistance*, sehingga menjadikannya pilihan AEAD yang paling sesuai untuk sistem E2EE yang beroperasi di lingkungan terdistribusi dengan banyak klien yang tidak dapat dijamin pengelolaan *nonce*-nya secara sempurna (Gueron et al., 2019).

2.3 AES-GCM Synthetic Initialization Vector (AES-GCM-SIV)

Sebelum membahas AES-GCM-SIV, diperlukan pemahaman terhadap AES-GCM sebagai dasar. AES Galois Counter Mode (AES-GCM) merupakan skema Authenticated Encryption with *Associated Data* (AEAD) yang menggabungkan mode Counter (CTR) untuk kerahasiaan dengan autentikasi berbasis perkalian di

medan hingga (Galois field), sehingga sekaligus menjamin konfidensialitas, integritas, dan autentikasi pesan dalam satu mekanisme. Pada AES-GCM, blok AES digunakan dalam mode counter sehingga berperilaku sebagai *stream cipher*. Dalam proses ini, nilai counter dienkripsi menggunakan kunci rahasia untuk menghasilkan *keystream*, yang kemudian dioperasikan dengan *plaintext* menggunakan XOR untuk menghasilkan *ciphertext*. Selain itu, tag autentikasi dihitung melalui operasi perkalian polinomial antara *ciphertext* dan data terkait menggunakan kunci *hash* pada medan GF(2^{128}), yang dikenal sebagai mekanisme GMAC (Kim et al., 2020).

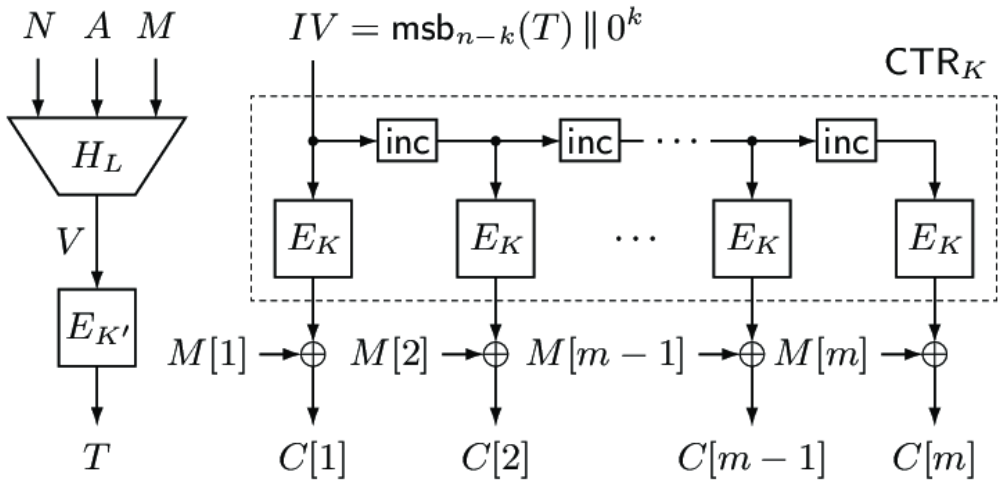
Namun demikian, AES-GCM memiliki kelemahan kritis yang menjadi alasan mendasar tidak digunakannya algoritma ini dalam penelitian ini, yaitu sensitivitas ekstrem terhadap *nonce reuse*. Apabila *nonce* yang sama digunakan dua kali dengan kunci yang sama bahkan hanya sekali maka *keystream* yang dihasilkan identik, sehingga penyerang dapat melakukan operasi XOR antara dua *ciphertext* untuk mengeliminasi *keystream* dan memperoleh informasi *plaintext* secara langsung. Lebih jauh, *reuse nonce* pada AES-GCM juga membocorkan kunci autentikasi (authentication key *H*), yang memungkinkan penyerang memalsukan tag autentikasi pada pesan-pesan berikutnya dan secara efektif meruntuhkan seluruh jaminan integritas sistem (Mattsson, 2024). Dalam lingkungan aplikasi *chatting real-time* yang menjadi konteks penelitian ini, risiko *nonce reuse* bukanlah sekadar skenario teoretis melainkan ancaman nyata yang dapat terjadi akibat kesalahan implementasi, kondisi *race condition* pada sistem konkuren, atau kegagalan *random number generator* pada perangkat klien.

Untuk mengatasi kelemahan fundamental tersebut, dikembangkan AES-GCM-SIV yang mengadopsi konsep *Synthetic Initialization Vector* (SIV) sebagai mekanisme pertahanan utamanya. Berbeda dengan AES-GCM yang menggunakan *nonce* eksternal secara langsung sebagai IV, AES-GCM-SIV menghitung IV secara deterministik sebagai fungsi kriptografis dari kunci, *nonce*, pesan, dan *associated data*, sehingga IV yang dihasilkan bersifat unik terhadap kombinasi seluruh input tersebut (Gueron et al., 2019). Mekanisme ini menjadikan AES-GCM-SIV secara inheren *nonce-misuse resistant* bahkan jika *nonce* yang sama digunakan ulang, IV sintetis yang dihasilkan tetap berbeda selama *plaintext*-nya berbeda, sehingga *keystream* tidak pernah berulang dan kebocoran informasi dapat dicegah. Satu-satunya konsekuensi dari *nonce reuse* pada AES-GCM-SIV adalah terungkapnya fakta bahwa dua pesan identik telah dikirimkan sebuah kebocoran yang jauh lebih dapat ditoleransi dibandingkan runtuhnya kerahasiaan dan integritas secara menyeluruh seperti pada AES-GCM (Chung et al., 2025; Gueron et al., 2019).

Secara teknis, proses enkripsi AES-GCM-SIV berlangsung dalam dua tahap. Pertama, dihitung *message authentication code* (MAC) atas gabungan *associated data* dan *plaintext* menggunakan kunci turunan (*derived key*), yang kemudian dijadikan sebagai SIV sekaligus berfungsi sebagai tag autentikasi. Kedua, SIV tersebut digunakan sebagai IV pada AES-CTR untuk mengenkripsi *plaintext* menjadi *ciphertext*. Struktur dua tahap ini mengikuti paradigma *authenticate-then-encrypt*, MAC dihitung terlebih dahulu atas *plaintext* dan *associated data* untuk menghasilkan SIV, kemudian SIV tersebut digunakan sebagai IV deterministik pada tahap enkripsi. Pendekatan ini menjamin bahwa IV yang digunakan untuk enkripsi

selalu terikat secara kriptografis pada konten pesan, yang merupakan fondasi dari properti *nonce-misuse resistance*. (Chung et al., 2025). Sebagai konsekuensi dari desain ini, AES-GCM-SIV memerlukan dua *pass* atas data (satu untuk MAC, satu untuk enkripsi), sehingga *throughput*-nya lebih rendah dibandingkan AES-GCM yang bersifat *single-pass*. Namun dalam konteks penelitian ini, *trade-off* tersebut dinilai sangat sepadan mengingat jaminan keamanan yang jauh lebih robust, khususnya ketahanannya terhadap *nonce reuse* yang menjadi prioritas utama desain sistem.

Pemilihan AES-GCM-SIV dalam penelitian ini, dengan demikian, bukan sekadar preferensi teknis, melainkan keputusan desain yang didasarkan pada kebutuhan nyata sistem E2EE. sebuah sistem yang beroperasi di lingkungan terdistribusi dengan banyak klien, di mana kesempurnaan pengelolaan *nonce* tidak dapat dijamin sepenuhnya. AES-GCM-SIV memberikan lapisan keamanan tambahan (*defense in depth*) yang memastikan bahwa kesalahan implementasi pada level *nonce* tidak akan berujung pada kompromi keseluruhan sistem sebuah properti yang tidak dimiliki oleh AES-GCM standar maupun mode enkripsi konvensional lainnya (Gueron et al., 2019; Mattsson, 2024).



Gambar II.2 Algoritma Enkripsi AES-GCM (Iwata & Minematsu, 2016)

Gambar 2.2 menunjukkan alur kerja skema umum AES-GCM yang menggabungkan fungsi enkripsi berbasis mode Counter (CTR) dan mekanisme autentikasi menggunakan fungsi *hash* pada medan hingga. Pada bagian kiri gambar, nilai *nonce* (N), *additional authenticated data* (A), dan pesan (M) diproses melalui fungsi *hash* (H_L) untuk menghasilkan nilai perantara (V). Nilai ini kemudian dienkripsi menggunakan blok AES dengan kunci (K) untuk menghasilkan tag autentikasi (T).

Sementara itu, pada bagian kanan gambar ditunjukkan proses enkripsi pesan menggunakan mode *Counter* (CTR). Nilai *initialization vector* (IV) dibentuk dari sebagian bit paling signifikan dari tag (T), yaitu ($msb_{n-k}(t)$), yang kemudian dikonkatenasi dengan nol sebanyak (k) bit. Nilai IV ini selanjutnya digunakan sebagai counter awal. Counter tersebut akan mengalami proses increment (inc) untuk setiap blok pesan. Setiap nilai counter dienkripsi menggunakan AES dengan kunci (K) (ditunjukkan sebagai (E_K)) untuk menghasilkan *keystream*. *Keystream*

ini kemudian dioperasikan dengan masing-masing blok *plaintext* ($M[i]$) menggunakan operasi XOR, sehingga menghasilkan blok *ciphertext* ($C[i]$). Dengan demikian, proses pada gambar ini menunjukkan bahwa AES-GCM secara simultan menghasilkan *ciphertext* melalui mode CTR dan tag autentikasi melalui fungsi *hash* berbasis medan hingga, sehingga menjamin kerahasiaan dan integritas data dalam satu mekanisme.

2.4 Elliptic Curve Diffie Hellman (ECDH)

Elliptic Curve Diffie-Hellman (ECDH) adalah protokol pertukaran kunci publik yang memungkinkan dua pihak dimana dalam konteks penelitian ini, pengirim (Alice) dan penerima (Bob) membentuk *shared secret* secara aman melalui kanal yang dapat disadap, tanpa pernah mengirimkan kunci rahasia itu sendiri. ECDH dibangun di atas konsep dasar Diffie–Hellman, tetapi memindahkan operasi dari grup modulo bilangan prima ke grup titik-titik pada kurva eliptik di atas medan hingga; keamanan protokol ini bergantung pada kesulitan Elliptic Curve Discrete Logarithm Problem (ECDLP), yang saat ini dianggap tidak terpecahkan secara praktis dengan sumber daya komputasi yang wajar (Hadi & Neamah, 2024).

Secara matematis, kurva eliptik didefinisikan oleh persamaan kubik seperti

$$y^2 = x^3 + ax + b \tag{1}$$

pada medan hinggan F_q. Himpunan titik-titik yang memenuhi persamaan tersebut, dilengkapi dengan operasi penjumlahan titik dan perkalian skalar, membentuk sebuah grup abelian yang menjadi dasar operasi kriptografi (Shchur et al., 2023).

Mekanisme kerja ECDH berlangsung dalam tiga tahap. Pertama, kedua pihak menyepakati parameter domain kurva yang sama, mencakup medan hingga, persamaan kurva, titik dasar G, dan orde n. Kedua, masing-masing pihak memilih kunci privat acak (d_A, d_B) dan menghitung kunci publik masing-masing $(Q_A = d_A G)$ dan $(Q_B = d_B G)$, kemudian saling bertukar kunci publik tersebut. Ketiga, masing-masing pihak menghitung *shared secret* yang sama melalui perkalian skalar pada kurva

$$S = d_A Q_b = d_B Q_A \quad (2)$$

Nilai S ini tidak pernah ditransmisikan secara langsung melalui kanal komunikasi, melainkan diturunkan menjadi kunci simetris AES-GCM-SIV melalui fungsi derivasi kunci HKDF-SHA256 untuk mengenkripsi pesan teks maupun file media dalam skema *end-to-end encryption*.

Dibandingkan dengan Diffie-Hellman klasik dan RSA, ECDH memiliki keunggulan yang menjadikannya pilihan paling tepat untuk penelitian ini. Pertama, dibandingkan dengan Diffie-Hellman klasik berbasik logaritma diskrit pada grup $\mathbb{Z}_p^*$, ECDH mencapai tingkat keamanan yang setara dengan ukuran kunci yang jauh lebih kecil, dimana kunci 256-bit pada ECDH memberikan keamanan yang setara dengan kunci 3072-bit pada DH klasik, sehingga secara signifikan mengurangi beban komputasi, konsumsi energi, dan kebutuhan memori (Dar et al., 2021). Kedua, dibandingkan dengan RSA yang sering digunakan sebagai mekanisme enkripsi asimetris, ECDH dirancang secara spesifik sebagai protokol pertukaran kunci, bukan algoritma enkripsi. Penggunaan RSA untuk pembentukan *session key* menimbulkan kompleksitas implementasi yang lebih tinggi dan tidak menyediakan

forward secrecy secara native, sementara ECDH secara inheren menghasilkan kunci sesi yang bersifat *ephemeral* sehingga kompromi kunci jangka panjang tidak membahayakan sesi komunikasi sebelumnya (Shchur et al., 2023).

Dalam penelitian ini digunakan kurva Curve25519 yang telah terstandarisasi melalui RFC7748. Curve25519 beroperasi pada medan prima $p = 255^{19} - 19$ dan dipilih atas tiga pertimbangan utama yang relevan dengan penelitian ini. Pertama, Curve25519 dirancang khusus untuk meminimalkan risiko kesalahan implementasi melalui formulasi kurva Montgomery yang memungkinkan operasi perkalian skalar yang aman tanpa memerlukan penanganan kasus tepi yang kompleks. Kedua, terbukti tahan terhadap berbagai serangan *side-channel* yang relevan dengan lingkungan eksekusi berbasis perangkat lunak. Ketiga, digunakan secara luas oleh protokol keamanan modern seperti TLS 1.3 dan Signal Protocol, sehingga memiliki landasan kepercayaan implementasi yang kuat (Hafeez et al., 2025). Pemilihan Curve25519 dalam penelitian ini, dengan demikian, bukan sekadar preferensi teknis, melainkan keputusan desain yang didasarkan pada kebutuhan sistem E2EE yang aman, efisien, dan mudah direproduksi.

Meskipun ECDH menjamin kerahasiaan *shared secret* dari pihak yang menyadap kanal komunikasi secara pasif, terdapat keterbatasan mendasar yang perlu dipahami, ECDH murni tidak menyediakan mekanisme autentikasi kunci publik. Artinya, protokol ini tidak secara otomatis memverifikasi bahwa kunci publik yang diterima benar-benar berasal dari pihak yang diklaim. Keterbatasan ini membuka celah terhadap serangan *Man-in-the-Middle* (MitM), di mana penyerang dapat mencegat pertukaran kunci publik, menggantikannya dengan kunci publiknya

sendiri, dan membangun dua sesi ECDH terpisah dengan masing-masing pihak tanpa sepengetahuan keduanya (Perkasa et al., 2025). Dalam kondisi ini, meskipun enkripsi AES-GCM-SIV berjalan dengan benar secara kriptografis, seluruh komunikasi justru dienkripsi untuk penyerang, bukan untuk penerima yang sah.

Oleh karena itu, penggunaan ECDH dalam sistem E2EE yang aman tidak dapat berdiri sendiri, melainkan harus dikombinasikan dengan mekanisme autentikasi kunci publik yang memadai. Dalam penelitian ini, aspek tersebut diatasi melalui pengiriman kunci publik melalui kanal yang terlindungi oleh autentikasi server, sehingga substitusi kunci publik oleh penyerang dapat terdeteksi.

2.5 HMAC-based Key Derivation Function (HKDF)

HMAC-based *Key Derivation* Function (HKDF) adalah fungsi derivasi kunci yang dibangun di atas konstruksi HMAC dengan *hash* kriptografis (umumnya SHA-256) dan dirancang untuk mengekstraksi serta memperluas material kunci mentah menjadi kunci rahasia yang seragam dan *pseudorandom*. HKDF dikategorikan sebagai KDF multipurpose yang konservatif dalam pemakaian fungsi *hash* dan dimaksudkan dapat digunakan pada beragam protokol, mulai dari IKEv2, EAP hingga berbagai skema key-agreement modern (Krawczyk & Eronen, 2010). Secara kriptografis, HKDF dimodelkan sebagai KDF yang keamanannya didasarkan pada sifat ekstraktor komputasional dan *pseudorandomness* dari HMAC; analisis formal menunjukkan bahwa apabila HMAC (mis. HMAC-SHA256) memenuhi asumsi ekstraktor dan PRF yang wajar, maka HKDF memuaskan sifat-sifat KDF yang aman terhadap berbagai tipe sumber material kunci.

Dalam skema pertukaran kunci seperti (Elliptic Curve) Diffie–Hellman, output utama protokol adalah *shared secret* yang masih berupa material kunci mentah dan tidak langsung layak dipakai sebagai kunci kriptografi tingkat aplikasi. HKDF berperan mengubah *shared secret* tersebut menjadi satu atau lebih kunci simetris dengan distribusi yang mendekati seragam dan terpisah secara kriptografis (*key separation*), sehingga satu kebocoran kunci tidak serta-merta membocorkan keseluruhan material kunci awal (Senthilkumar et al., 2024).

Shared secret yang dihasilkan oleh protokol ECDH secara matematis aman terhadap pemecahan langsung, namun nilai tersebut tidak didesain sebagai kunci simetris siap pakai; struktur aljabar dan kemungkinan kebocoran informasi (misalnya panjang, distribusi, atau reuse) membuka ruang serangan jika digunakan langsung sebagai key AES maupun kunci MAC (Krawczyk & Eronen, 2010). Dalam konteks E2EE modern, HKDF memungkinkan pemisahan peran kunci (*key separation*) antara kunci enkripsi, kunci autentikasi, dan kunci derivasi lanjutan; hal ini terbukti krusial dalam skema yang memerlukan banyak kunci jangka pendek (*session key*s) dari satu *shared secret*, seperti pengiriman pesan beruntun atau enkripsi banyak berkas/media dalam satu sesi (Zhiqiang et al., 2025).

Secara konseptual, HKDF terdiri atas dua tahap utama, yaitu HKDF-Extract dan HKDF-Expand, yang mengikuti paradigma *extract-then-expand*. Pada tahap *extract*, *shared secret* yang dihasilkan dari proses ECDH digunakan sebagai *initial keying material* (IKM) dan dikombinasikan dengan *pre-shared key* atau *salt* melalui fungsi HMAC untuk menghasilkan *pseudorandom key* (PRK). Tahap ini berfungsi

untuk mengekstraksi dan menormalkan entropi dari IKM sehingga menghasilkan kunci antara yang memiliki distribusi lebih seragam

Selanjutnya, pada tahap *expand*, PRK digunakan bersama dengan informasi konteks (*shared context info*) untuk menghasilkan satu atau lebih *output keying material* (OKM). Proses ini dapat dilakukan beberapa kali untuk menghasilkan turunan kunci yang berbeda sesuai kebutuhan. Dalam konteks penelitian ini, tahap *expand* menghasilkan beberapa parameter kriptografi, seperti kunci enkripsi (*AEAD key*) dan *nonce* yang digunakan dalam algoritma AES-GCM-SIV. Dengan demikian, HKDF tidak hanya menghasilkan kunci utama, tetapi juga memungkinkan pemisahan fungsi (*key separation*) melalui derivasi beberapa nilai kriptografi dari satu sumber kunci yang sama.

Secara umum, HKDF memproses tiga input utama, (1) IKM (*initial keying material*), dalam konteks E2EE biasanya berupa *shared secret* hasil ECDH; (2) salt, nilai publik (namun rahasia sangat dianjurkan) yang menambah entropi dan mencegah reuse output ketika IKM serupa digunakan pada banyak sesi; dan (3) info, string konteks yang menyematkan informasi penggunaan kunci, identitas pihak, atau label protokol. Berdasarkan ketiga input tersebut, tahap HKDF-Extract dirumuskan sebagai berikut:

$$PRK = HMAC - SHA256(salt, IKM) \quad (3)$$

Di mana PRK adalah *pseudorandom* key dengan panjang tetap 256-bit yang menjadi input tahap berikutnya. Selanjutnya, tahap HKDF-Expand menggunakan PRK tersebut untuk memproduksi OKM dengan panjang yang diinginkan:

$$OKM = HKDF - Expand(PRK, info, L) \quad (4)$$

Di mana *L* adalah panjang output yang dibutuhkan dalam *byte*, dan *info* adalah string konteks yang memastikan kunci yang dihasilkan unik untuk setiap tujuan penggunaan. Dari OKM ini, sistem memotong dan membagi hasilnya menjadi beberapa kunci simetris sesuai kebutuhan, misalnya kunci enkripsi AES-256-GCM-SIV dan kunci autentikasi (Krawczyk & Eronen, 2010).

Dalam konteks penelitian ini, HKDF-SHA256 diterapkan dalam rantai implementasi ECDH–HKDF–AES-GCM-SIV sebagai berikut, protokol ECDH dengan kurva Curve25519 dieksekusi untuk menghasilkan *shared secret S*, nilai *S* diproses menggunakan HKDF-SHA256 melalui persamaan (3) dan (4) untuk menurunkan kunci simetris AES-256 dan *nonce* yang digunakan pada proses enkripsi dan kunci tersebut digunakan langsung pada algoritma AES-GCM-SIV. Dengan demikian, HKDF-SHA256 tidak hanya berperan sebagai jembatan antara ECDH dan AES-GCM-SIV, tetapi juga menjamin bahwa setiap sesi komunikasi menghasilkan kunci yang unik, terisolasi, dan tidak dapat diturunkan kembali ke *shared secret* asal, sehingga sifat *forward secrecy* sistem E2EE yang dibangun dapat terpenuhi (Zhiqiang et al., 2025).

2.6 Pesan Teks dan File Media

Pesan teks merupakan bentuk data uji pertama yang digunakan dalam simulasi penelitian ini, merepresentasikan pertukaran informasi berbasis karakter antara pengirim (Alice) dan penerima (Bob) dalam skema E2EE. Dalam lingkup komunikasi digital yang aman, kriptografi menjadi komponen penting untuk melindungi sumber daya data online dari perspektif integritas, kerahasiaan, dan keamanan dari potensi serangan seperti penyadapan dan *brute force* (Ansh Goel et

al., 2024). Pesan teks memiliki karakteristik yang membedakannya dari jenis data lain, yakni ukurannya yang relatif kecil, berbasis karakter, dan ditransmisikan hampir secara instan. Dalam sistem ASCII, 16 karakter alfabet dapat direpresentasikan dalam 128 bit, sehingga algoritma enkripsi seperti AES mampu memproses 16 karakter Latin sekaligus dalam satu blok enkripsi; namun untuk skrip Unicode yang membutuhkan 2 *byte* per karakter, hanya 8 karakter yang dapat dienkripsi dalam satu blok, menjadikan waktu pemrosesan enkripsi Unicode lebih lambat dibandingkan representasi ASCII (Rose et al., 2023).

Landasan teknis dari pesan teks dalam komunikasi digital adalah sistem encoding karakter. Keamanan jaringan kini menjadi isu utama seiring berkembangnya pertukaran data elektronik, di mana kriptografi berperan penting dalam melindungi sumber daya data online dari aspek integritas, kerahasiaan, dan keamanan termasuk perlindungan terhadap serangan *eavesdropping* melalui proses encoding karakter berdasarkan nilai ASCII. Standar encoding yang paling umum digunakan adalah ASCII (American Standard Code for Information Interchange) dan UTF-8. ASCII merepresentasikan 95 karakter yang dapat dicetak dan 33 karakter kontrol dalam rentang nilai 0 hingga 127, yang tersimpan sebagai bilangan bulat 7-bit. Sementara itu, UTF-8 sebagai pengembangan dari Unicode mampu merepresentasikan karakter dari berbagai bahasa di dunia dengan menggunakan 1 hingga 4 *byte* per karakter. Pemrosesan enkripsi pada Unicode memerlukan waktu dua kali lebih lama atau bahkan lebih, dibandingkan enkripsi karakter dalam representasi ASCII, karena kebutuhan bit yang lebih besar untuk merepresentasikan setiap karakter non-Latin (Jirjees et al., 2023).

File media merujuk pada berkas digital yang merepresentasikan konten multimedia seperti gambar, audio, dan video, yang dalam penelitian ini berfungsi sebagai data uji kedua dalam simulasi untuk mengevaluasi performa dan ketahanan keamanan skema E2EE pada berbagai ukuran dan jenis data. Video merupakan salah satu bentuk media digital yang paling dominan digunakan untuk komunikasi, penyebaran informasi, dan pemantauan; namun penggunaan yang luas ini sekaligus meningkatkan risiko akses tidak sah dan manipulasi, yang menimbulkan tantangan keamanan signifikan (Asghar et al., 2024). File gambar umumnya menggunakan format JPEG, di mana JPEG menerapkan kompresi lossy yang menghasilkan ukuran file lebih kecil namun dengan sedikit kehilangan kualitas. File audio menggunakan format seperti MP3 (kompresi lossy). Adapun file video seperti MP4 merupakan format container yang menggabungkan aliran data audio dan visual dengan tingkat kompresi tertentu.

Pengamanan file media menghadirkan tantangan teknis yang jauh lebih kompleks dibandingkan pengamanan pesan teks biasa. Mengingat pesatnya kemajuan dalam enkripsi multimedia yang bertujuan mencapai keamanan lebih tinggi dengan biaya komputasional lebih rendah, pengejar keseimbangan antara keamanan dan efisiensi telah menjadi isu kritis dalam riset enkripsi (Go & Lee, 2025). Tantangan utama dalam pengamanan file media meliputi dua aspek. Pertama, ukuran file yang besar menyebabkan waktu enkripsi dan dekripsi yang lebih panjang dibandingkan pesan teks. Kedua, kebutuhan sumber daya komputasi, baik memori maupun CPU, turut meningkat seiring bertambahnya ukuran file media yang diproses.

Solusi keamanan tradisional tidak sesuai untuk lingkungan dengan sumber daya terbatas akibat sifat resource-constrained yang tidak dapat mengakomodasi pemrosesan enkripsi yang berat, sehingga diperlukan solusi keamanan yang efektif dan mampu menyeimbangkan desain yang ringan dengan tindakan keamanan yang kuat (Mehmood et al., 2025). Dalam konteks simulasi penelitian ini, file media dibaca langsung dalam representasi biner dan diproses melalui *pipeline* kriptografi tanpa melalui tahap transmisi jaringan fisik, sehingga pengukuran performa yang dihasilkan murni mencerminkan karakteristik komputasi algoritma kriptografi tanpa noise latensi jaringan (Utama Siahaan, 2024).

Keamanan dalam pertukaran pesan teks maupun file media bertumpu pada tiga pilar utama, kerahasiaan (*confidentiality*), integritas data, dan autentikasi. Kriptografi memanfaatkan algoritma matematika dan kunci untuk mengubah data *plaintext* menjadi *ciphertext*, menjadikannya tidak terbaca oleh pihak yang tidak berwenang dan hanya individu yang memiliki kunci dekripsi yang tepat dapat mengembalikan pesan terenkripsi ke bentuk aslinya, menjaga kerahasiaannya (Ansh Goel et al., 2024). Integritas data memastikan bahwa konten yang diterima tidak mengalami perubahan selama transmisi. Autentikasi pesan merupakan mekanisme atau layanan yang digunakan untuk memverifikasi integritas pesan, guna memastikan bahwa data yang diterima adalah persis seperti yang dikirimkan tanpa modifikasi, penyisipan, penghapusan, atau replay serta bahwa identitas pengirim yang diklaim adalah valid. Algoritma AES-GCM-SIV menggabungkan ketiga aspek ini secara simultan dalam satu primitif AEAD terpadu. Konfidensialitas dijamin melalui enkripsi berbasis mode counter, sementara

integritas dan autentikasi dijamin melalui *authentication tag* 16-*byte* yang dihitung menggunakan fungsi POLYVAL atas seluruh *ciphertext* dan *associated data*, sebagai bagian dari konstruksi Synthetic IV (SIV) sesuai RFC 8452. Setiap perubahan pada *ciphertext* selama transmisi, sekecil apapun, akan menyebabkan verifikasi tag gagal di sisi penerima sehingga pesan ditolak (Iwata & Minematsu, 2016). Dalam penelitian ini, pesan teks dan file media digunakan sebagai objek pengujian dalam simulasi untuk mengukur secara kuantitatif bagaimana karakteristik masing-masing jenis data memengaruhi kinerja algoritma AES-GCM-SIV dan AES-GCM pada setiap parameter performa yang telah ditetapkan.

2.7 Parameter Pengujian Performa

Parameter pengukuran performa merupakan aspek penting dalam penelitian ini karena digunakan untuk mengevaluasi dan membandingkan efisiensi algoritma AES-GCM-SIV dan AES-GCM secara objektif dalam lingkungan simulasi. Pengukuran performa dilakukan dengan pendekatan kuantitatif berdasarkan metrik yang dapat diukur secara langsung pada proses kriptografi dalam *pipeline* simulasi, tanpa melibatkan latensi jaringan fisik.

Waktu enkripsi adalah total durasi yang dibutuhkan sistem untuk mengubah *plaintext* menjadi *ciphertext*, sedangkan waktu dekripsi adalah kebalikannya. Waktu enkripsi merupakan total waktu yang dibutuhkan untuk menghasilkan *ciphertext* dari *plaintext*, dan waktu enkripsi ini selanjutnya digunakan untuk menghitung *throughput* dari skema enkripsi yang diuji.

Rata-rata waktu enkripsi dihitung dari sejumlah n iterasi pengujian menggunakan rumus berikut:

$$\bar{\mathrm{T}}_{\mathrm{enc}} = \frac{1}{n}\sum_{i=1}^{n} t_{enc,i} \quad (5)$$

Di mana $\bar{\mathrm{T}}_{\mathrm{enc}}$ adalah adalah rata-rata waktu enkripsi (dalam milidetik atau mikrodetik), $t_{enc,i}$ adalah waktu enkripsi pada iterasi ke-i, dan $n$adalah jumlah total iterasi pengujian. Demikian pula untuk dekripsi:

$$\overline{T}_{dec} = \frac{1}{n}\sum_{i=1}^{n} t_{dec,i} \quad (6)$$

Untuk mengukur stabilitas hasil pengujian, digunakan pula standar deviasi waktu eksekusi:

$$\sigma_T = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(t_i - \overline{T})^2} \quad (7)$$

Throughput didefinisikan sebagai jumlah data yang berhasil diproses oleh sistem enkripsi per satuan waktu. *Throughput* suatu algoritma dihitung sebagai rasio antara total ukuran data terhadap total waktu eksekusi selama proses enkripsi atau dekripsi.

Rumus *throughput* enkripsi adalah:

$$Tp_{enc} = \frac{S_{plaintext}}{\overline{T}_{enc}} \quad (8)$$

Rumus *throughput* dekripsi adalah:

$$Tp_{dec} = \frac{S_{plaintext}}{\overline{T}_{dec}} \quad (9)$$

Di mana $T_{p_{enc}}$dan $T_{p_{dec}}$masing-masing adalah *throughput* enkripsi dan dekripsi dalam satuan MB/s atau Mbps, $S_{plaintext}$adalah ukuran data *plaintext* (dalam *byte*

atau megabyte), $S_{ciphertext}$adalah ukuran data *ciphertext*, dan $\bar{T}_{enc}$, $\bar{T}_{dec}$ adalah rata-rata waktu enkripsi dan dekripsi dalam detik.

Konsumsi memori selama proses kriptografi diukur sebagai selisih antara alokasi memori saat proses kriptografi aktif berjalan dan alokasi memori *baseline* sistem dalam kondisi idle, diukur menggunakan pustaka stats_alloc yang menginstrumentasi *heap* allocator Rust secara langsung. Dalam konteks simulasi penelitian ini, pengukuran dilakukan pada lingkungan eksekusi yang terisolasi dan konsisten, sehingga hasil yang diperoleh bersifat reproducible dan bebas dari variasi yang disebabkan oleh faktor jaringan atau antarmuka pengguna.

Rumus penggunaan memori aktif proses kriptografi:

$$M_{crypto} = M_{active} - M_{baseline} \tag{10}$$

Di mana M_{crypto} adalah memori yang dikonsumsi oleh proses kriptografi (dalam KB atau MB), M_{active} adalah total memori yang terpakai saat proses enkripsi atau dekripsi berlangsung, dan $M_{baseline}$ adalah memori yang terpakai sistem sebelum proses kriptografi dimulai.

Overhead ukuran data (data size *overhead*) mengacu pada pertambahan ukuran data akibat penambahan metadata kriptografi seperti *nonce*, *authentication tag*, dan header protokol pada setiap pesan yang dienkripsi. Transformasi *plaintext* menjadi *ciphertext* mengakibatkan peningkatan ukuran data akibat penambahan metadata kriptografi, dan perbandingan antara ukuran data terenkripsi dan tidak terenkripsi menjadi salah satu objek analisis utama dalam evaluasi simulasi enkripsi pada penelitian ini.

Rumus *overhead* ukuran data absolut:

$$O_{size} = S_{ciphertext} - S_{plaintext} \quad (11)$$

Rumus *overhead* ukuran data relatif (persentase):

$$O_{size(\%)} = \frac{S_{ciphertext} - S_{plaintext}}{S_{plaintext}} \times 100\% \quad (12)$$

Rumus expansion ratio (rasio ekspansi *ciphertext*):

$$R_{exp} = \frac{S_{ciphertext}}{S_{plaintext}} \quad (13)$$

Di mana O_{size} adalah *overhead* ukuran data absolut (dalam *byte*), $S_{ciphertext}$ adalah ukuran *ciphertext* setelah enkripsi, $S_{plaintext}$ adalah ukuran *plaintext* asli, dan R_{exp} adalah rasio ekspansi *ciphertext* terhadap *plaintext*.

2.8 Penelitian Terkait

Kajian terhadap penelitian-penelitian terdahulu dilakukan untuk mengidentifikasi perkembangan terkini dalam implementasi skema enkripsi pada sistem komunikasi digital, sekaligus memetakan celah penelitian yang belum terjawab. Tabel 2.1 merangkum dua belas penelitian yang relevan berdasarkan tiga dimensi analisis, metode kriptografi yang digunakan, cakupan data yang diamankan, dan kelengkapan evaluasi performa serta keamanan yang dilakukan.

Tabel II.1 State-of-the-Art (SOTA)

No	Judul Penelitian	Hasil / Metode	Gap Penelitian
1	Design of an Encryption Chat Application Based on ChaCha20-Poly1305 (Feng, 2024)	Merancang aplikasi chat dengan enkripsi ChaCha20-Poly1305 + HKDF + ECDH. Sistem terbukti mampu mencegah serangan kriptografi umum (differential, linear, related-key). Metode: implementasi arsitektur	Belum membahas pengamanan file media. Sinkronisasi data aman antar perangkat berbeda belum dieksplorasi. Fokus hanya pada pesan teks.

No	Judul Penelitian	Hasil / Metode	Gap Penelitian
		aplikasi chat dengan salted hashing pada *database*.	
2	An image encryption method based on modified elliptic curve Diffie-Hellman key exchange protocol and Hill Cipher (Hadi & Neamah, 2024)	Sistem kriptosistem ECDHHC (ECDH + Hill Cipher) dengan matriks self-invertible 8x8. Key space mencapai 2^768, entropi mendekati 8, NPCR >99%, UACI >33%. Metode: kombinasi ECDH termodifikasi dengan Hill Cipher untuk enkripsi citra grayscale.	Hanya diuji pada citra grayscale, belum mencakup citra berwarna. Analisis kompleksitas komputasi dibanding metode non-matriks belum mendalam.
3	Implementation of Secure *End-to-End* Encrypted Chat Application Using Diffie-Hellman Key Exchange and AES-256 in a Microservice Architecture (Perkasa et al., 2025)	Aplikasi chat E2E dengan AES-256 + DH dalam arsitektur mikroservis. Berhasil meningkatkan skalabilitas via load balancing dan scaling horizontal. Metode: pemisahan layanan (autentikasi, *WebSocket*, obrolan) dengan enkripsi per layanan.	Rentan serangan *man-in-the-middle* karena kurang autentikasi dalam fase pertukaran kunci. Belum mengeksplorasi enkripsi multimedia dan varian ECDH yang lebih modern.
4	Development and Implementation of an Efficient *Chatting* Application for Mobile Devices (A & Chauhan, 2023)	Aplikasi 'The Inner Circle' dengan UI dioptimalkan untuk mobile (bubble chat lateral). Mampu menangani volume pesan besar dengan rekam jejak percakapan yang rapi. Metode: pengembangan berbasis Firebase dengan manajemen kontak dan status online.	Tidak membahas protokol keamanan data secara mendalam. Belum ada enkripsi *end-to-end* yang kuat (ECDH/ChaCha20-Poly1305). Fokus dominan pada UX bukan keamanan.

No	Judul Penelitian	Hasil / Metode	Gap Penelitian
5	File Encryption and Decryption Application using Elliptic Curve Diffie-Hellman Algorithm and Chaotic Map Function (Heru et al., 2024)	Sistem enkripsi berkas (teks, citra, audio, video) dengan ECDH + Chaotic Map berbasis web. Waktu enkripsi/dekripsi lebih cepat dibanding metode terdahulu. Metode: pemanfaatan sensitivitas chaotic map terhadap nilai awal untuk meningkatkan kompleksitas keamanan.	Optimasi khusus untuk citra berukuran besar belum mendalam. Belum mengeksplorasi sinkronisasi data *real-time* untuk aplikasi chat. Dampak variasi parameter chaotic function belum dikaji mendalam.
6	Enkripsi Pesan Chat Menggunakan Algoritma ChaCha20 pada Aplikasi Komunikasi *Real-Time* (Darmansyah & Hasugian, 2025)	Implementasi ChaCha20 pada aplikasi Android (Flutter + Firebase). ChaCha20 terbukti lebih cepat dan *throughput* lebih tinggi dibanding AES-CBC. Metode: *stream cipher* ChaCha20 untuk enkripsi pesan teks *real-time*.	Fokus hanya pada pesan teks, belum mengeksplorasi enkripsi citra/multimedia. Belum mengintegrasikan protokol pertukaran kunci ECDH. Analisis resistensi serangan kriptografi belum mendalam.
7	Implementation of a Hybrid Cryptosystem Using ChaCha20 and ECC for Image Encryption in an Android Application (Zai et al., 2025)	Aplikasi Android dengan kriptografi hibrida ChaCha20 (enkripsi citra) + ECC (proteksi kunci). Berhasil menjaga kerahasiaan dan integritas citra selama transmisi. Metode: kunci *ephemeral* + pembagian tugas antara enkripsi simetris dan asimetris.	Belum mengeksplorasi transmisi citra *real-time* atau volume masif secara simultan. Protokol ECDH untuk skenario chat dua arah belum dijelaskan. Belum ada analisis efisiensi baterai/CPU pada berbagai spesifikasi perangkat.
8	A Python-Based Simulation and Security Analysis of ChaCha20 and AES for Secure Data	ChaCha20 secara konsisten mengungguli AES dalam kecepatan dan efisiensi memori	Simulasi hanya membandingkan ChaCha20 dan AES tanpa mengintegrasikan

No	Judul Penelitian	Hasil / Metode	Gap Penelitian
	Transmission in Resource-Constrained Environments (Dungog, 2025)	pada simulasi Python. AES menunjukkan peningkatan beban memori drastis seiring bertambahnya data. Metode: analisis komparatif waktu enkripsi, dekripsi, dan peak memory consumption.	ECDH sebagai mekanisme pertukaran kunci. Tidak mengevaluasi ketahanan terhadap skenario *nonce reuse*. Belum menggabungkan AES-GCM-SIV dalam *pipeline* simulasi E2EE yang mencakup pesan teks dan file media secara bersamaan.
9	ChaCha20-Poly1305 Authenticated Encryption with Additional Data for Transport Layer Security 1.3 (Serrano et al., 2022)	ChaCha20-Poly1305 sebagai mekanisme AEAD dalam TLS 1.3 memberikan keseimbangan optimal antara keamanan dan performa. Lebih tahan serangan timing dibanding AES-GCM. Metode: evaluasi implementasi AEAD dengan *Associated Data* pada protokol TLS 1.3.	Fokus pada level protokol transportasi, belum diimplementasikan pada aplikasi chat level pengguna. Tidak mengeksplorasi optimasi untuk multimedia besar (citra/video). Belum mengintegrasikan ECDH untuk konteks *chatting* E2E.
10	A Secure Cloud Based *Chatting* Application with Hybrid Cryptography (Awachat et al., 2021)	Sistem obrolan berbasis awan dengan kriptografi hibrida yang lebih aman dibanding algoritma tunggal. Enkripsi/dekripsi data ke cloud berjalan efektif. Metode: integrasi beberapa algoritma enkripsi secara bersamaan untuk E2EE antara pengguna dan server awan.	Belum mengoptimalkan untuk perangkat sumber daya terbatas. Tidak mengeksplorasi ChaCha20 atau ECDH secara spesifik. Belum menyajikan analisis performa komprehensif untuk pengamanan citra/multimedia berkapasitas besar.

Berdasarkan kajian terhadap seluruh penelitian yang telah dipaparkan pada Tabel 2.1, terdapat tiga celah penelitian yang secara konsisten belum terjawab oleh

literatur yang ada. Pertama, berdasarkan kajian literatur yang dianalisis pada Tabel 2.1, belum ditemukan penelitian yang mengintegrasikan AES-GCM-SIV secara penuh dalam skenario komunikasi *chat* dua arah (*real-time*) yang sekaligus mencakup pesan teks dan file media (gambar, audio, video) dalam satu sistem. Penelitian yang membangun aplikasi *chat* seperti (Perkasa et al., 2025) dan (Alkhyeli et al., 2023) masih menggunakan AES-GCM standar yang rentan terhadap *nonce reuse*. Kedua, dari seluruh penelitian yang menerapkan AES-GCM-SIV maupun ECDH, dari seluruh literatur yang dikaji, belum ditemukan penelitian yang menggabungkan keduanya dalam satu skema E2EE terpadu sekaligus menguji ketahanannya terhadap skenario *nonce reuse* secara eksplisit dalam lingkungan eksperimen yang terkontrol. Ketiga, pengukuran performa yang komprehensif di sisi klien (*client-side*) mencakup waktu enkripsi, waktu dekripsi, *throughput*, konsumsi memori, dan *overhead* ukuran data untuk skema AES-GCM-SIV pada konteks *chat real-time* juga belum ditemukan dalam literatur yang dikaji.

Berdasarkan tiga celah penelitian di atas, kebaruan (novelty) penelitian ini terletak pada dua hal yang saling melengkapi. Pertama, penelitian ini berdasarkan kajian literatur yang dilakukan, belum ditemukan simulasi skema E2EE berbasis integrasi ECDH–HKDF–AES-GCM-SIV yang mencakup pertukaran pesan teks dan file media (gambar, audio, video) secara bersamaan dalam satu *pipeline* kriptografi terpadu. Kedua, penelitian ini menyusun dan menjalankan skenario pengujian terstruktur yang secara khusus mencakup uji *nonce reuse* dan pengukuran performa secara menyeluruh, sehingga menghasilkan data empiris yang belum ditemukan dalam literatur yang dikaji. Dengan demikian, kebaruan penelitian ini

bukan pada perancangan algoritma kriptografi baru, melainkan pada level integrasi, implementasi, dan evaluasi empiris skema yang sudah ada ke dalam konteks simulasi komunikasi yang terstruktur dan dapat direproduksi.

Penelitian ini memosisikan diri sebagai jembatan antara dua kelompok penelitian yang selama ini berjalan secara terpisah: (1) penelitian yang membangun aplikasi *chat* E2EE tetapi masih menggunakan algoritma yang rentan terhadap *nonce reuse* (Alkhyeli et al., 2023; Darmansyah & Hasugian, 2025; Perkasa et al., 2025), dan (2) penelitian yang mengimplementasikan AES-GCM-SIV tetapi terbatas pada pengamanan berkas statis tanpa mekanisme pertukaran kunci yang aman (Hernandi & Christian Chandra, 2024). Penelitian ini mengambil kelebihan dari keduanya. ketahanan AES-GCM-SIV terhadap *nonce reuse* dari kelompok kedua, dan konteks aplikasi *chat real-time* dari kelompok pertama, lalu menyatukannya dengan menambahkan ECDH sebagai mekanisme pertukaran kunci yang aman. Secara lebih luas, penelitian ini juga merespons temuan komparatif dari (Dungog, 2025) dan (Serrano et al., 2022) yang menegaskan pentingnya pemilihan mode AEAD yang tepat, dengan menawarkan bukti empiris bahwa AES-GCM-SIV layak dan efisien diterapkan pada berbagai jenis dan ukuran data yang relevan dengan konteks komunikasi digital modern.

Tabel II.2 Matriks Penelitian

Peneliti	Jenis Media				Algoritma Enkripsi				Algoritma Pertukaran Kunci		Parameter Pengujian		
	Teks	Gambar	Video	Audio	CC2P	AES-256	AES-GCM	AES-GCM-SIV	ECDH	RSA	Enkripsi	Dekripsi	Penggunaan Memori
(Awachat et al., 2021)	✓	x	x	x	x	✓	x	x	x	✓	✓	✓	x
(Serrano et al., 2022)	✓	x	x	x	✓	x	x	x	x	x	✓	✓	x
(Alkhyeli et al., 2023)	✓	x	x	x	x	x	✓	x	x	x	✓	✓	x
(A & Chauhan, 2023)	✓	x	x	x	x	x	x	x	x	x	x	x	x
(Feng, 2024)	✓	x	x	x	✓	x	x	x	✓	x	x	x	x
(Hadi & Neamah, 2024)	x	✓	x	x	x	x	x	x	✓	x	x	x	x
(Heru et al., 2024)	✓	✓	✓	✓	x	x	x	x	✓	x	✓	✓	x
(Hernandi & Christian Chandra, 2024)	✓	x	x	x	x	✓	✓	x	x	x	✓	✓	x
(Perkasa et al., 2025)	✓	x	x	x	x	✓	x	x	✓	x	✓	✓	x
(Darmansyah & Hasugian, 2025)	✓	x	x	x	✓	x	x	x	x	x	✓	✓	x

Peneliti	Jenis Media				Algoritma Enkripsi				Algoritma Pertukaran Kunci		Parameter Pengujian		
	Teks	Gambar	Video	Audio	CC2P	AES-256	AES-GCM	AES-GCM-SIV	ECDH	RSA	Enkripsi	Dekripsi	Penggunaan Memori
(Zai et al., 2025)	✓	✓	x	x	✓	x	x	x	✓	x	✓	✓	✓
(Dungog, 2025)	✓	x	x	x	✓	✓	x	x	x	x	✓	✓	✓
(Ours, 2026)	✓	✓	✓	✓	x	x	x	✓	✓	x	✓	✓	✓