Tugas Besar I (Tubes I) IF4020 Kriptografi

Pembuatan Web Based Chat App dengan Enkripsi End-to-End

Disusun oleh:

Farhan Raditya Aji	13522142
M. Zaidan Sa’dun Robbani	13522146

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025/2026

DAFTAR ISI

BAB 1
TEORI SINGKAT

1.1 Kriptografi Kunci Publik (Public Key Cryptography)

Kriptografi kunci publik, atau kriptografi asimetris, adalah sistem keamanan yang menggunakan sepasang kunci matematis yang saling berhubungan namun berbeda, yaitu kunci publik (public key) dan kunci privat (private key).

- **Kunci Publik** merupakan kunci yang disebarkan secara terbuka dan digunakan oleh pihak lain untuk mengenkripsi pesan atau memverifikasi tanda tangan digital.
- **Kunci Privat** merupakan kunci yang dijaga kerahasiaannya oleh pemilik dan digunakan untuk mendekripsi pesan atau membuat tanda tangan digital.

Sistem ini mengatasi masalah distribusi kunci yang terdapat pada kriptografi simetris, di mana pengirim dan penerima harus menyepakati kunci rahasia yang sama melalui saluran aman sebelum berkomunikasi.

1.2 Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) adalah pendekatan dalam kriptografi kunci publik yang didasarkan pada struktur aljabar kurva eliptik di atas medan hingga (*finite fields*). Keamanan ECC bergantung pada kesulitan masalah logaritma diskrit kurva eliptik (*Elliptic Curve Discrete Logarithm Problem / ECDLP*), yaitu sulitnya mencari nilai skalar k jika diketahui titik P dan hasil perkalian titik Q = k • P.

Kurva yang digunakan dalam standar industri (seperti Bitcoin dan tugas ini) adalah secp256k1, yang didefinisikan dengan persamaan:

$$y^2 = x^3 + 7 \bmod p$$

Keunggulan utama ECC dibandingkan algoritma non-ECC (seperti RSA) adalah efisiensi. ECC menawarkan tingkat keamanan yang setara dengan ukuran kunci yang

jauh lebih kecil (contoh: 256-bit ECC setara dengan 3072-bit RSA), sehingga menghemat bandwidth dan daya komputasi.

1.3 Fungsi Hash Kriptografis dan SHA-3

Fungsi hash adalah algoritma matematika yang memetakan data dengan ukuran berapapun menjadi untaian bit dengan ukuran tetap (digest). Fungsi hash kriptografis memiliki sifat-sifat krusial:

1. ***Pre-image Resistance*:** Sulit mencari input asli dari nilai *hash*-nya.
2. ***Collision Resistance*:** Sulit menemukan dua input berbeda yang menghasilkan *hash* yang sama.
3. **Efek Avalanche:** Perubahan kecil pada input mengubah output secara drastis.

SHA-3 (Secure Hash Algorithm 3) adalah standar fungsi hash terbaru dari NIST yang menggunakan konstruksi Keccak (struktur spons / sponge construction). Berbeda dengan pendahulunya (SHA-1 dan SHA-2), SHA-3 memiliki ketahanan yang lebih tinggi terhadap serangan kriptanalisis tertentu. Dalam sistem ini, SHA-3 digunakan untuk menjamin integritas data (memastikan data tidak berubah) dan sebagai dasar pembuatan kunci deterministik.

1.4 Tanda Tangan Digital (*Digital Signature*) dan ECDSA

Tanda tangan digital adalah skema matematis untuk membuktikan keaslian pesan atau dokumen digital. Skema ini menjamin:

- **Autentisitas:** Penerima yakin pesan berasal dari pengirim yang diklaim.
- **Integritas:** Pesan tidak diubah selama transmisi.
- **Nirsangkal (*Non-Repudiation*):** Pengirim tidak dapat menyangkal telah mengirim pesan tersebut.

ECDSA (*Elliptic Curve Digital Signature Algorithm*) adalah varian dari algoritma tanda tangan digital (DSA) yang menggunakan kurva eliptik. Prosesnya melibatkan:

1. ***Signing*** dimana pengirim membuat pasangan nilai (r, s) menggunakan kunci privatnya dan hash dari pesan.
2. ***Verifying*** dimana penerima menggunakan kunci publik pengirim, pesan asli, dan pasangan (r, s) untuk memvalidasi keabsahan tanda tangan.

1.5 Protokol Pertukaran Kunci (*Key Exchange*) dan ECDH

ECDH (*Elliptic Curve Diffie-Hellman*) adalah protokol pertukaran kunci yang memungkinkan dua pihak, yang masing-masing memiliki pasangan kunci ECC publik-privat, untuk menghasilkan shared secret yang sama tanpa pernah mempertukarkan kunci privat mereka. Secara matematis, jika Alice memiliki (d_A, Q_A) dan Bob memiliki (d_B, Q_B), maka:

$$SharedSecret = d_A \bullet Q_B = d_B \bullet Q_A$$

Titik hasil perkalian ini identik bagi kedua belah pihak dan digunakan sebagai dasar kunci enkripsi simetris.

1.6 Enkripsi Simetris Terautentikasi (AES-GCM)

AES (*Advanced Encryption Standard*) adalah algoritma enkripsi simetris blok yang menjadi standar global. Mode operasi GCM (Galois/Counter Mode) digunakan karena menyediakan *Authenticated Encryption with Associated Data* (AEAD). Artinya, AES-GCM tidak hanya menjamin kerahasiaan pesan, tetapi juga menjamin integritas pesan melalui authentication tag. Jika ciphertext dimodifikasi walau satu bit, proses dekripsi akan gagal, sehingga penerima tahu bahwa pesan telah dirusak (*tampered*).

1.7 *End-to-End Encryption* (E2EE)

End-to-End Encryption **(E2EE)** adalah paradigma komunikasi di mana data dienkripsi pada perangkat pengirim dan hanya didekripsi pada perangkat penerima. Pihak perantara, termasuk penyedia layanan internet (ISP) dan server aplikasi, hanya meneruskan data dalam bentuk terenkripsi (*ciphertext*) dan tidak memiliki kunci untuk mendekripsinya. Hal ini menjamin privasi mutlak bagi pengguna.

1.8 Zero-Knowledge Proof / Authentication

Konsep ini merujuk pada metode di mana satu pihak (pembukti) dapat membuktikan kepada pihak lain (verifikator) bahwa mereka mengetahui suatu rahasia (seperti *password* atau kunci privat) tanpa perlu mengungkapkan rahasia itu sendiri.

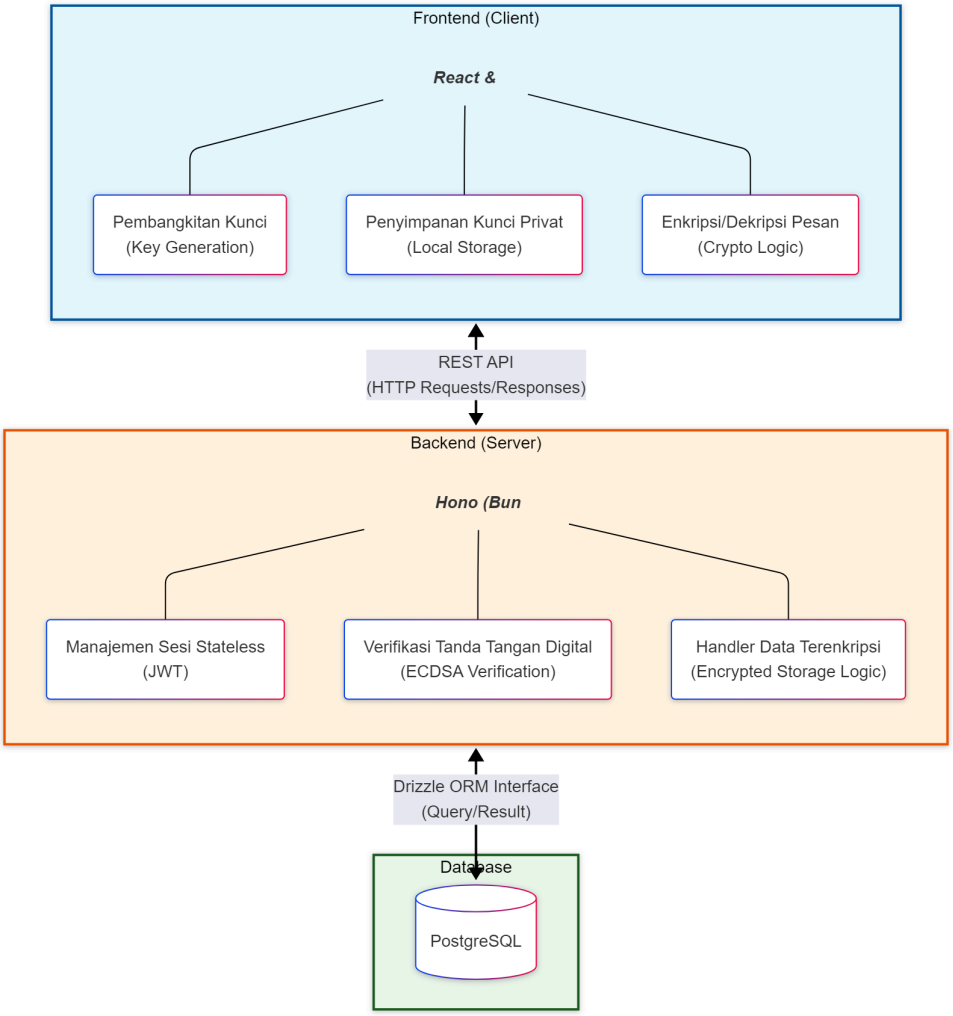
Dalam implementasi sistem chat, hal ini diterapkan melalui mekanisme Challenge-Response. Server mengirimkan tantangan acak (nonce), dan pengguna membuktikan identitasnya dengan menandatangani tantangan tersebut menggunakan kunci privat. Server memverifikasi tanda tangan menggunakan kunci publik tanpa pernah melihat atau menyimpan kunci privat/password pengguna .

BAB 2
PERANCANGAN DAN IMPLEMENTASI

2.1 Arsitektur Sistem

Sistem dibangun menggunakan arsitektur Client-Server berbasis REST API yang terpisah secara modular (*Decoupled Architecture*).

2.1.1 Diagram Arsitektur



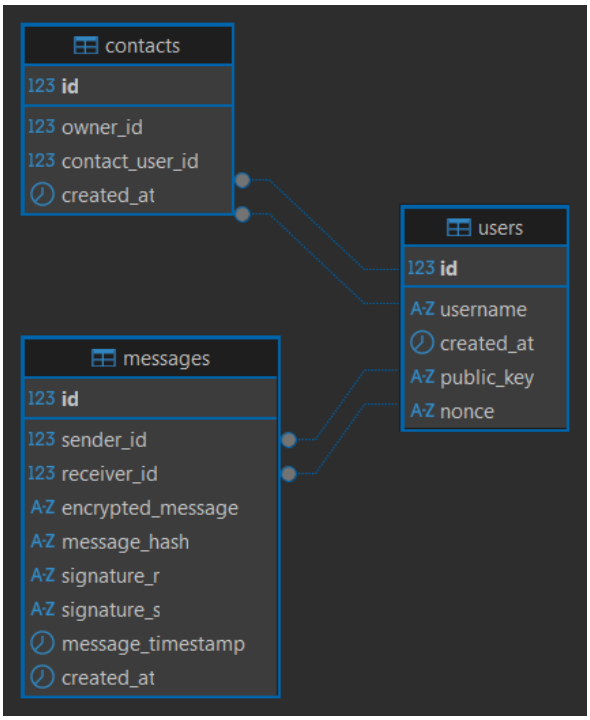
Gambar 2.1.1.1 Diagram Arsitektur

2.1.2 *Tech Stack* yang Digunakan

- **Bahasa Pemrograman**: TypeScript (Fullstack).

- **Backend**: Hono (Framework), Bun (Runtime), Zod (Validation).
- **Frontend**: React, Vite, Tailwind CSS.
- **Database**: PostgreSQL, Drizzle ORM.
- **Library Kriptografi**: elliptic (untuk kurva secp256k1), js-sha3 (untuk hashing), dan Web Crypto API (untuk AES-GCM).

2.2 Perancangan Basis Data



Gambar 2.2.1 Diagram Basis Data

Sistem basis data dirancang menggunakan PostgreSQL dengan skema relasional yang berfokus pada keamanan data dan prinsip Zero-Knowledge. Berdasarkan diagram skema (Gambar 2.2.1), basis data terdiri dari tiga entitas utama dengan fungsi sebagai berikut:

2.2.1 Tabel Pengguna (*users*)

Tabel ini berfungsi sebagai direktori identitas publik pengguna. Berbeda dengan sistem konvensional, tabel ini tidak menyimpan kata sandi (*password*) ataupun hash-nya. Secara kriptografi, tabel ini berperan penting untuk menyimpan public_key (dalam format Heksadesimal) yang digunakan oleh pengguna lain untuk mengenkripsi pesan yang ditujukan ke pengguna tersebut, serta menyediakan kolom nonce yang digunakan sebagai

penyimpanan sementara untuk tantangan acak saat proses autentikasi *Challenge-Response* berlangsung.

2.2.2 Tabel Pesan (*messages*)

Tabel ini menyimpan riwayat percakapan antar pengguna. Karena menerapkan *End-to-End Encryption*, server tidak dapat membaca isi pesan. Tabel ini dikhususkan untuk menyimpan komponen kriptografi yang meliputi encrypted_message (*ciphertext* hasil enkripsi AES-GCM), message_hash (SHA-3 digest untuk verifikasi integritas), serta signature_r dan signature_s (komponen tanda tangan ECDSA) yang berfungsi memvalidasi autentikasi pengirim pesan.

2.2.3 Tabel Kontak (*contacts*)

Tabel ini merepresentasikan relasi pertemanan antar pengguna. Tabel ini berfungsi untuk memetakan hubungan *many-to-many* antara pengguna, yang memungkinkan aplikasi untuk mengambil profil dan kunci publik lawan bicara secara cepat dan efisien saat sesi *chat* dimulai.

2.3 Implementasi Fitur Kriptografi

Bagian ini menjelaskan implementasi teknis dari fitur-fitur keamanan utama yang dikembangkan.

2.3.1 Key Generation

Sistem tidak menyimpan *password* pengguna. Sebagai gantinya, password digunakan sebagai seed untuk membangkitkan pasangan kunci secara deterministik di sisi klien.

- **Algoritma**

  SHA-3 digunakan sebagai *Key Derivation Function* (KDF) sederhana untuk mengubah kombinasi username dan password menjadi entropi 256-bit.

- **Implementasi**

  Hasil *hash* SHA-3 tersebut langsung digunakan sebagai *Private Key* untuk kurva **secp256k1**. Dari *Private Key* ini, *Public Key* diturunkan secara matematis.

- **Keamanan**

  Private Key hanya disimpan di localStorage peramban (*browser*) dan tidak pernah dikirim ke jaringan. Hanya *Public Key* yang dikirim ke server saat registrasi.

2.3.2 Autentikasi *Zero-Knowledge* (*Challenge-Response*)

Sistem menerapkan mekanisme login tanpa mengirimkan *password* atau kunci privat ke server. Implementasi dilakukan dalam dua tahap *round-trip*:

1. **Request Challenge:**
   a. Klien meminta tantangan login.
   b. Server membangkitkan string acak (nonce) dan menyimpannya sementara di kolom nonce pada tabel users.
2. **Verify Response:**
   a. Klien melakukan hashing pada nonce:

      h = SHA3(*nonce*)

   b. Klien menandatangani hash tersebut dengan kunci privatnya:

      sig = Sign(PrivKey, h)

   c. Server memverifikasi tanda tangan menggunakan kunci publik pengguna yang tersimpan di database.
   d. Jika valid, server menghapus nonce (untuk mencegah Replay Attack) dan mereturn token JWT ke klien.

2.3.3 Enkripsi Pesan (*End-to-End Encryption*)

Untuk menjamin kerahasiaan pesan, sistem menerapkan skema **Enkripsi Hibrida (*Hybrid Encryption*)** yang menggabungkan keunggulan ECC dan AES:

1. **Pertukaran Kunci (ECDH)**

   Sebelum mengirim pesan, pengirim menghitung *Shared Secret* dengan mengalikan *Private Key*-nya sendiri dengan *Public Key* penerima. Titik hasil perkalian ini (koordinat-x) kemudian di-*hash* dengan SHA-3 untuk menghasilkan kunci simetris 256-bit yang valid.

2. **Enkripsi Simetris (AES-GCM)**

   Isi pesan (*plaintext*) dienkripsi menggunakan algoritma **AES-256** dalam mode **GCM** (*Galois/Counter Mode*) dengan kunci simetris hasil ECDH. Mode GCM dipilih karena menghasilkan *Ciphertext* sekaligus *Authentication Tag*, yang menjamin pesan tidak dapat dibaca maupun dimodifikasi oleh pihak ketiga (termasuk server).

2.3.4 Penandatanganan dan Verifikasi (*Tamper Detection*)

Setiap pesan yang dikirim melalui sistem wajib melalui mekanisme *Digital Signature* untuk menjamin **Authenticity** dan **Integrity**. Mekanisme ini juga berfungsi sebagai sistem deteksi dini terhadap upaya manipulasi data (*Tamper Detection*).

- **Proses Penandatanganan (*client*):**
  Sebelum pesan dikirim, aplikasi klien membuat hash SHA-3 dari gabungan payload pesan (isi pesan, pengirim, penerima, dan timestamp). Hash ini kemudian ditandatangani menggunakan Kunci Privat pengirim dengan algoritma ECDSA, menghasilkan pasangan nilai (r, s). Tanda tangan ini disematkan dalam paket pesan JSON yang dikirim ke server.
- **Verifikasi Ganda:**
  - **Server**

Saat menerima pesan, server memvalidasi tanda tangan menggunakan Kunci Publik pengirim yang tersimpan di basis data. Server memastikan bahwa pengirim pesan adalah pemilik sah dari token sesi (JWT) yang digunakan. Jika tanda tangan tidak valid, pesan ditolak dan tidak disimpan.

- ***Client***

  Saat pesan diterima dan didekripsi, aplikasi klien melakukan perhitungan ulang terhadap *hash* pesan dan membandingkannya dengan *message hash* yang diterima.

- **Mekanisme Deteksi Manipulasi (Anti-Tampering):**
  - **Indikator Merah (Tampered):** Ditampilkan jika proses dekripsi AES-GCM gagal (menandakan *ciphertext* telah diubah) atau jika format pesan rusak.
  - **Indikator Kuning (Unverified):** Ditampilkan jika pesan berhasil didekripsi namun verifikasi tanda tangan ECDSA gagal (menandakan pesan berasal dari pengirim palsu atau *hash* tidak cocok).
  - **Indikator Hijau (Verified):** Ditampilkan hanya jika seluruh integritas kriptografis valid.

2.4 Implementasi Pesan *Real-time*

Untuk memenuhi kebutuhan komunikasi waktu nyata (*real-time*) tanpa mengorbankan stabilitas koneksi pada lingkungan *serverless*, sistem menggunakan pendekatan **HTTP Short Polling**.

- **Mekanisme**

  Klien melakukan permintaan HTTP GET secara berkala (interval 3 detik) ke *endpoint* /api/messages.

- **Efisiensi Bandwidth**

  Permintaan ini menyertakan parameter since (penanda waktu pesan terakhir diterima). Server hanya mengembalikan pesan-pesan yang lebih baru dari waktu

tersebut (*delta updates*), sehingga beban jaringan tetap minimal dan responsif layaknya aplikasi *chat* pada umumnya.

2.5 Implementasi Deployment (Bonus)

Sebagai bonus implementasi, sistem ini di-*deploy* ke layanan *cloud* publik menggunakan **Microsoft Azure**. Arsitektur produksi dirancang terpisah (*decoupled*) sebagai berikut:

1. **Frontend:** Di-*hosting* menggunakan layanan **Azure Static Web Apps**. Layanan ini terintegrasi dengan GitHub Actions untuk *deployment* otomatis.
2. **Backend:** Di-*hosting* menggunakan **Azure App Service** yang menjalankan *image* Docker dari *container registry*.

Aplikasi saat ini dapat diakses secara publik melalui tautan berikut:

- **URL Aplikasi (Frontend):** https://jolly-flower-0c8425b00.3.azurestaticapps.net/
- **URL API (Backend):** https://kripto-e2e-chat-be-c7gsgab6g0gda4bq.malaysiawest-01.azurewebsites.net/

BAB 3
PENGUJIAN

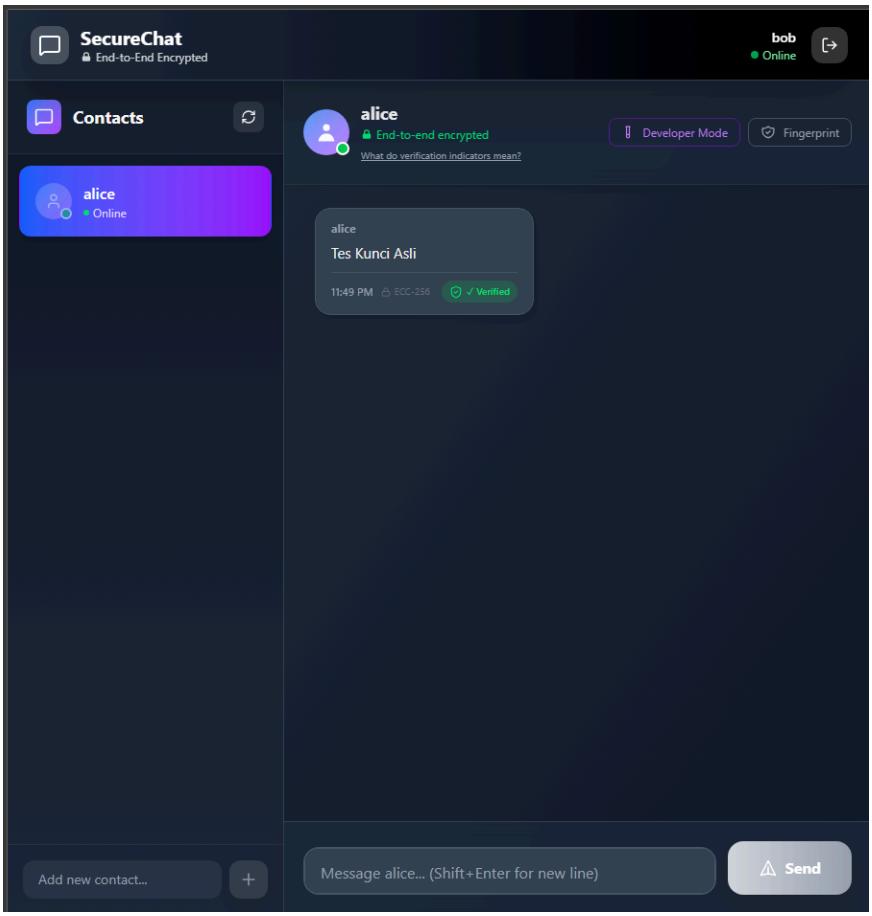
3.1 Pengujian Dekripsi dengan Kunci Privat yang Benar (*Normal Flow*)

Pengujian ini dilakukan untuk memverifikasi alur komunikasi normal di mana pengirim (Alice) dan penerima (Bob) memiliki pasangan kunci yang valid. Tujuannya adalah memastikan algoritma AES-GCM berhasil mendekripsi pesan dan algoritma ECDSA berhasil memverifikasi tanda tangan digital pengirim.

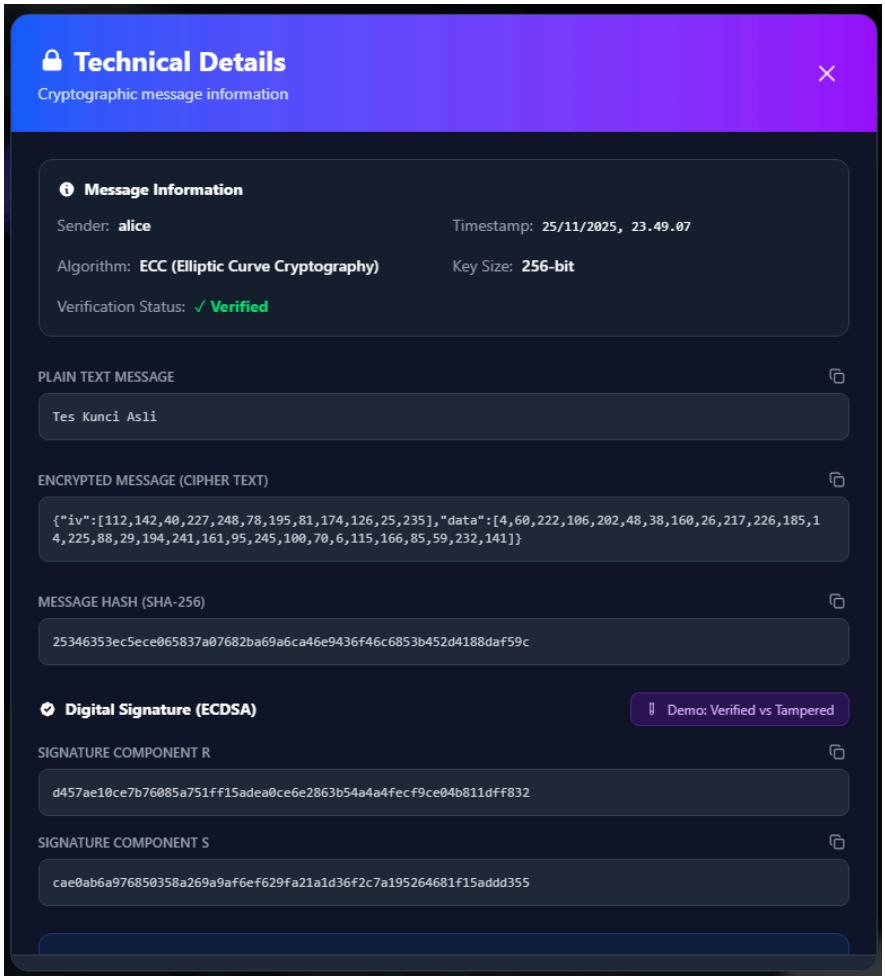
3.1.1 Modifikasi yang dilakukan

- **Status**: Tidak ada modifikasi (Kondisi Normal).
- **Data**: Menggunakan kunci asli yang dihasilkan saat registrasi.

3.1.2 Hasil Pengujian



Gambar 3.1.2.1 Tampilan Penerima (Bob) - Pesan Terbaca & Verified



Gambar 3.1.2.2 Detail Teknis Kriptografi - Status Hijau

3.1.3 Analisis

Seperti terlihat pada Gambar 3.1.2.1, pesan "Tes Kunci Asli" berhasil didekripsi menjadi plaintext yang dapat dibaca. Indikator keamanan menunjukkan status Hijau (*Verified*), yang menandakan bahwa:

- **Confidentiality**: Kunci privat penerima cocok dengan kunci publik yang digunakan pengirim.
- **Authenticity**: Tanda tangan digital valid (pesan benar-benar dari Alice).
- **Integrity**: Pesan tidak mengalami perubahan selama transmisi.

3.2 Pengujian Dekripsi dengan Kunci Privat yang Salah (*Wrong Private Key*)

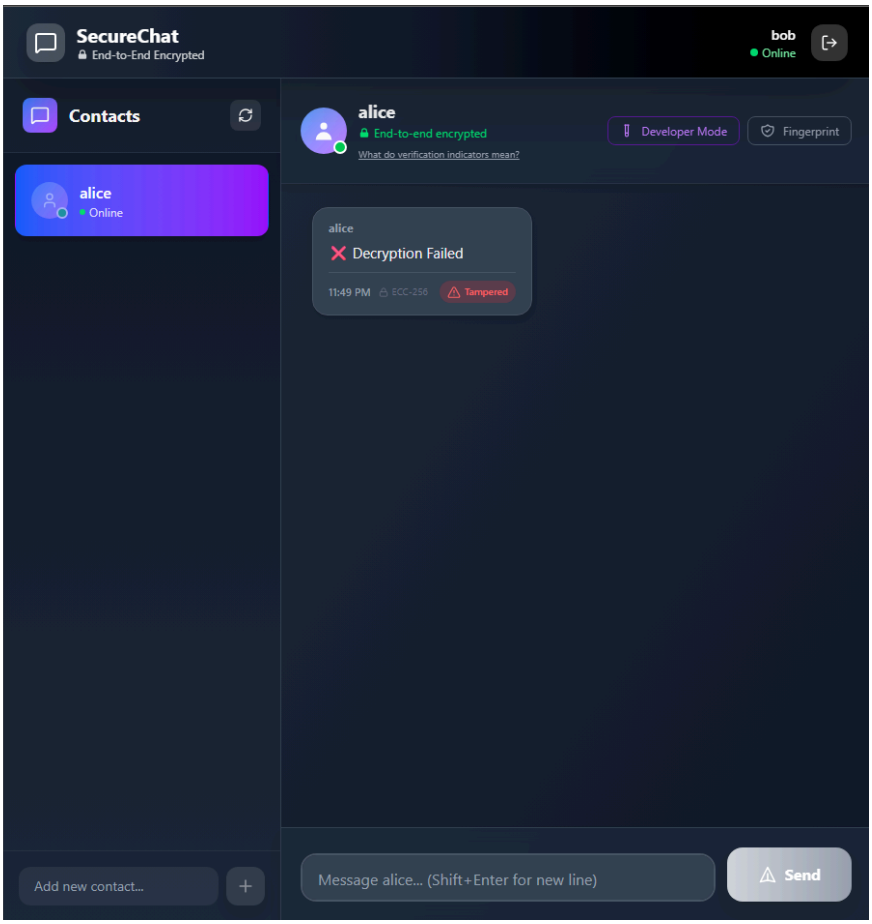
Pengujian ini mensimulasikan skenario di mana kunci privat pengguna rusak, hilang, atau seseorang mencoba memaksa membuka pesan menggunakan kunci yang tidak sesuai. Pengujian dilakukan dengan memanipulasi Local Storage browser secara manual.

3.2.1 Modifikasi yang dilakukan

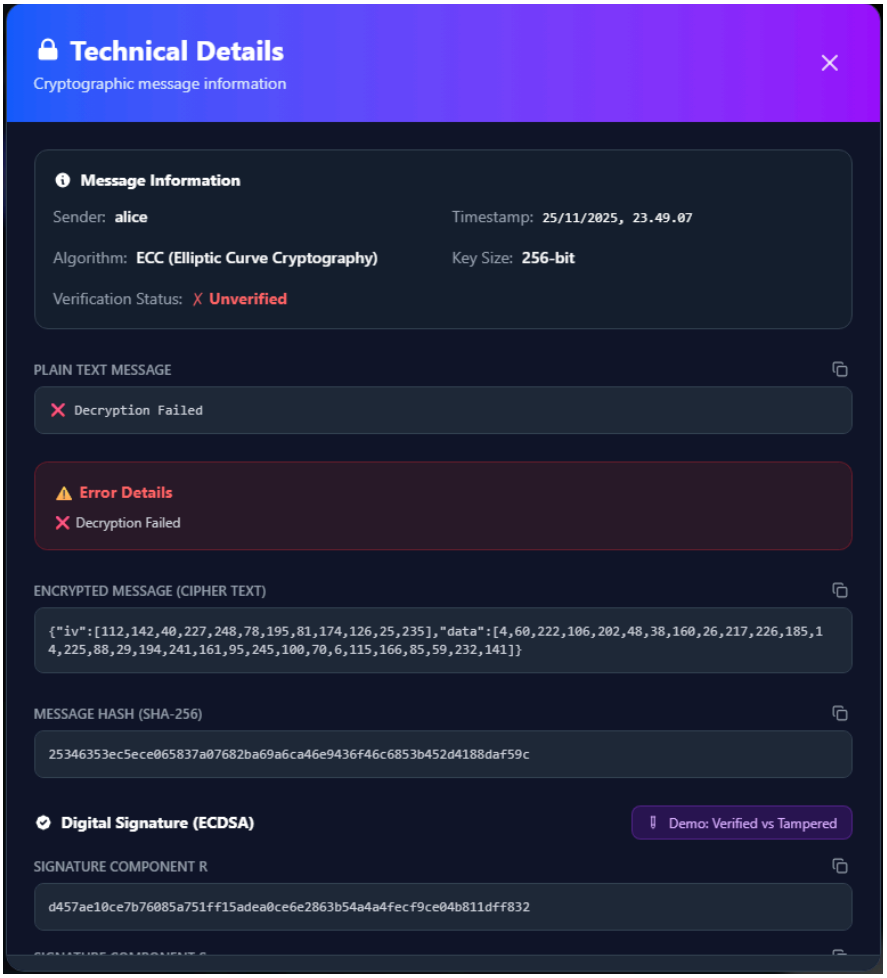
Perubahan dilakukan pada Local Storage browser penerima (Bob) pada key priv_bob.

- **Kunci Asli:**
  8e6cf3e9d880a230410a22fce89b50258b14e9cfcb0008aa1799d31908a07440
- **Dimodifikasi**:
  8d6cf3e9d880a230410a22fce89b50258b14e9cfcb0008aa1799d31908a07440

3.2.2 Hasil Pengujian



Gambar 3.2.2.1 Tampilan Penerima (Bob) - Gagal Dekripsi



Gambar 3.2.2.2 Detail Teknis - *Error Decryption Failed*

3.2.3 Analisis

Pada Gambar 3.2.2.1, sistem menampilkan pesan error **"Decryption Failed"** dengan label **Tampered**. Hal ini terjadi karena algoritma AES-GCM mendeteksi ketidakcocokan *authentication tag* saat mencoba mendekripsi *ciphertext* menggunakan kunci yang salah. Hasil ini membuktikan aspek **Confidentiality** (Kerahasiaan) terjaga; tanpa kunci privat yang spesifik dan benar, pesan tetap terkunci dan tidak dapat dibaca.

3.3 Pengujian Integritas Pesan dengan Interceptor (Burp Suite)

Pengujian ini mensimulasikan serangan *Man-in-the-Middle* (MITM) di mana penyerang mencegat paket data di jaringan sebelum sampai ke server. Pengujian menggunakan *tools*

Burp Suite untuk menahan *request* pengiriman pesan dan memodifikasi isinya (*tampering*).

3.3.1 Modifikasi yang dilakukan

Perubahan dilakukan pada *payload* JSON saat *request* POST /api/messages sedang dalam status *intercept*.

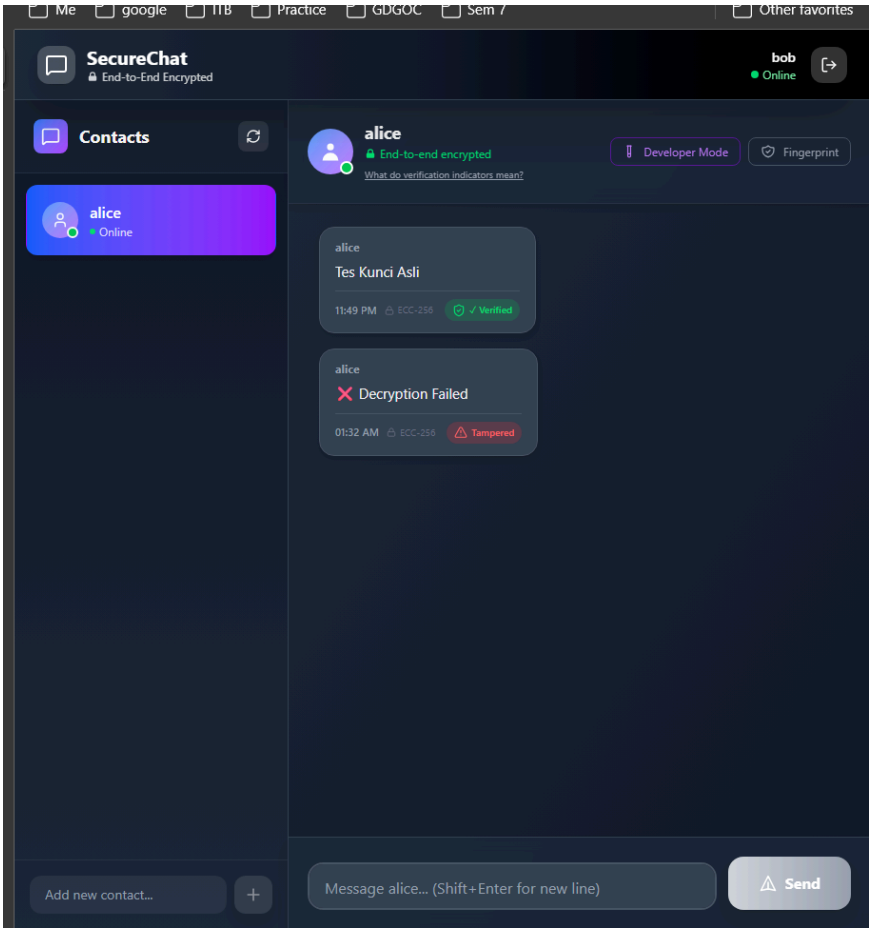
- **Target:** Field encrypted_message (bagian array data).
- **Data asli:**

  {\"iv\":[207,8,255,97,80,23,35,114,49,254,36,150],\"data\":[200,121,153,64,109,189,99,202,138,3,174,47,36,181,103,12,23,207,73,10,14,200,109,143,159,100,31,172,50,78,200,80]}

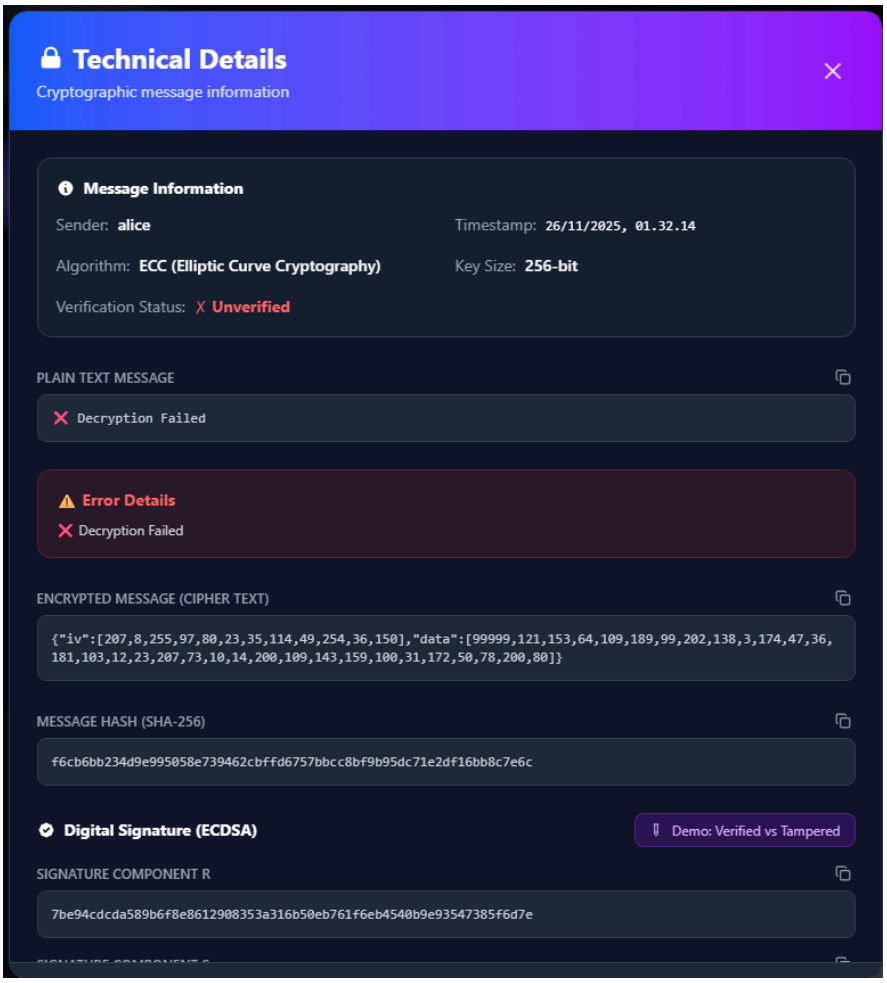
- ***Tampered*:**

  {\"iv\":[207,8,255,97,80,23,35,114,49,254,36,150],\"data\":[99999,121,153,64,109,189,99,202,138,3,174,47,36,181,103,12,23,207,73,10,14,200,109,143,159,100,31,172,50,78,200,80]}

3.3.2 Hasil Pengujian



Gambar 3.3.2.1 Tampilan Penerima (Bob) - Invalid Tampered



Gambar 3.3.2.2 Detail Teknis - Status Unverified

3.3.3 Analisis

Hasil pada Gambar 3.3.2.1 menunjukkan bahwa meskipun pesan berhasil disimpan oleh server dan diterima oleh Bob, klien menolaknya dengan status **"Invalid Format"** dan indikator **Merah (Tampered)**. Gambar 3.3.2.2 memperlihatkan status **"Unverified"**. Ini membuktikan bahwa mekanisme **Integrity** berjalan dengan baik. Perubahan satu *byte* saja pada *ciphertext* menyebabkan kegagalan validasi kriptografi sehingga pengguna terlindungi dari pesan palsu atau pesan yang telah dimanipulasi di tengah jalan.

LAMPIRAN

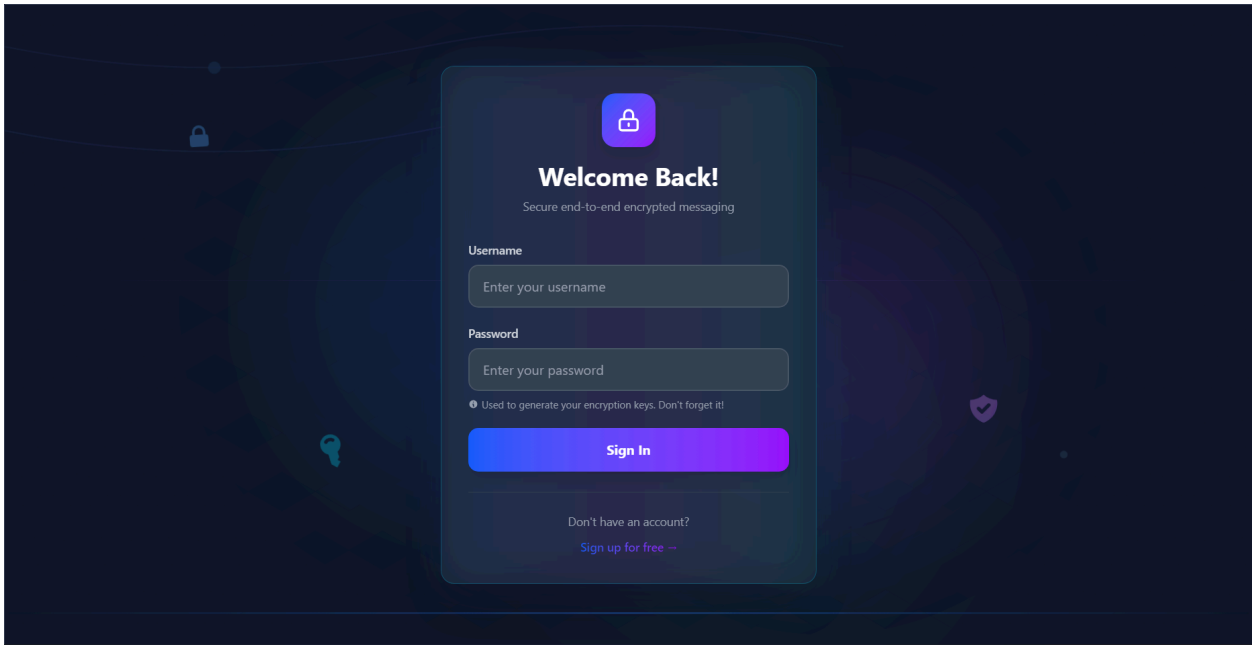
Link repository: https://github.com/sibobbbbbb/end-to-end-encryption-chat-app.git

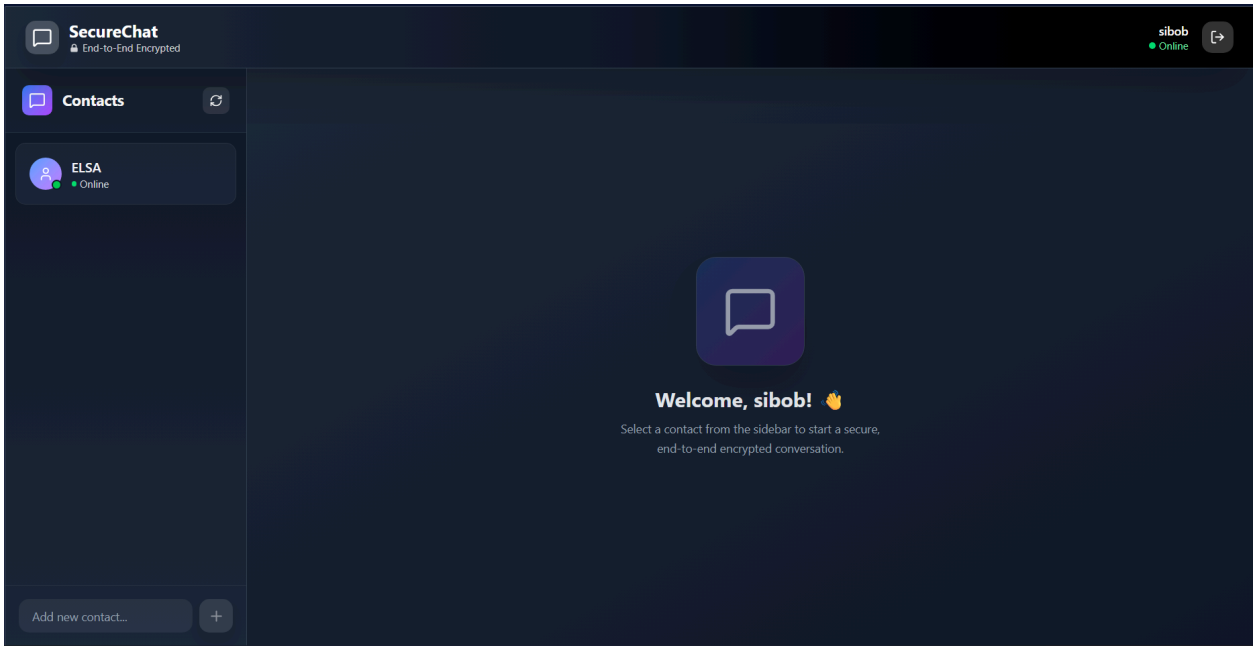
Link video demo: https://youtu.be/5MHcmMYmPEI

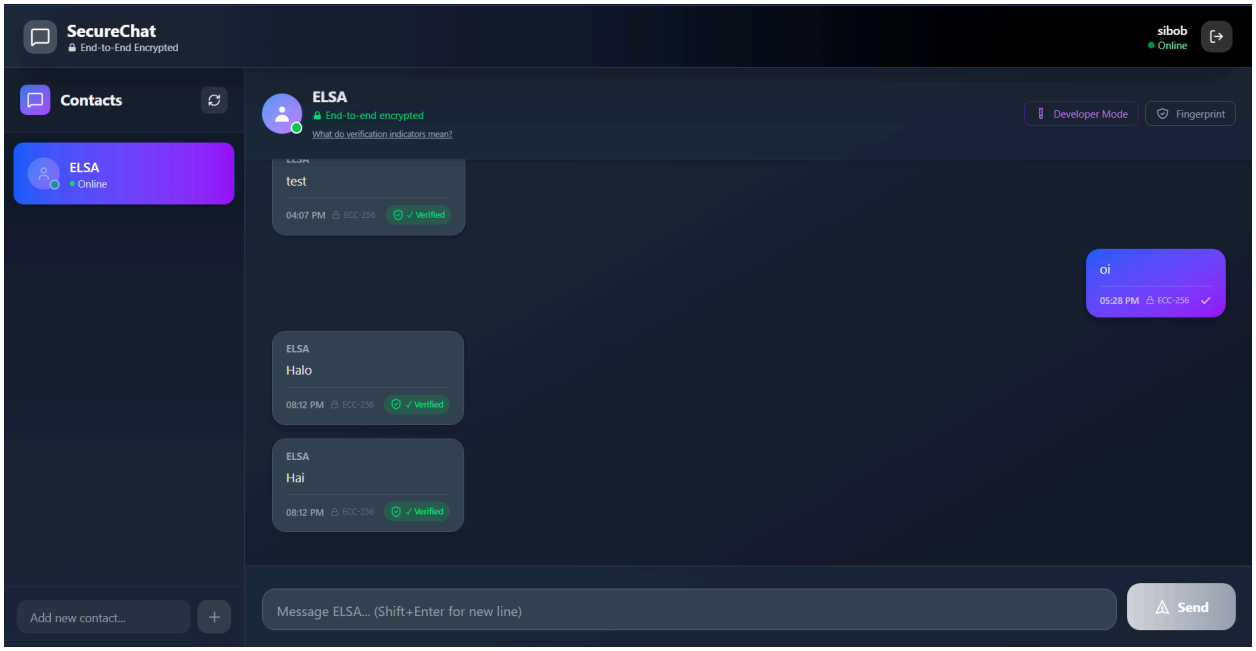
Pembagian tugas:

NIM	KERJA
13522142	server-side, laporan
13522146	client-side, laporan

Tangkapan layar aplikasi:




SecureChat
End-to-End Encrypted
sibob
Online
Contacts
ELSA
Online
Welcome, sibob! 👋
Select a contact from the sidebar to start a secure, end-to-end encrypted conversation.
Add new contact...


SecureChat
End-to-End Encrypted
sibob
Online
Contacts
ELSA
Online
ELSA
End-to-end encrypted
What do verification indicators mean?
Developer Mode
Fingerprint
test
04:07 PM ECC-256 Verified
oi
05:28 PM ECC-256
ELSA
Halo
08:12 PM ECC-256 Verified
ELSA
Hai
08:12 PM ECC-256 Verified
Add new contact...
Message ELSA... (Shift+Enter for new line)
Send

ASU

DAFTAR PUSTAKA

- Munir, Rinaldi. (2025). 24 - Elliptic Curve Cryptography (ECC) (Bagian 1). Materi Kuliah IF4020 Kriptografi, Institut Teknologi Bandung. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/24-ECC-Bagian1-2025.pdf
- Munir, Rinaldi. (2025). 25 - Elliptic Curve Cryptography (ECC) (Bagian 2). Materi Kuliah IF4020 Kriptografi, Institut Teknologi Bandung. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/25-ECC-Bagian2-2025.pdf
- Munir, Rinaldi. (2025). 26 - Fungsi Hash. Materi Kuliah IF4020 Kriptografi, Institut Teknologi Bandung. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/26-Fungsi-hash-2025.pdf
- Munir, Rinaldi. (2025). 27 - Beberapa Fungsi Hash. Materi Kuliah IF4020 Kriptografi, Institut Teknologi Bandung. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/27-Beberapa-fungsi-hash-2025.pdf
- Munir, Rinaldi. (2024). 31 - Digital Signature Algorithm (DSA) dan Elliptic Curve DSA (ECDSA). Materi Kuliah IF4020 Kriptografi, Institut Teknologi Bandung. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/31-DSA-dan-ECDSA-2025.pdf
- Hono Contributors. (2025). Hono Documentation - Getting Started. Dokumentasi Resmi Hono. Diakses dari https://hono.dev/docs
- Oven. (2025). Bun Documentation - Introduction. Dokumentasi Resmi Bun. Diakses dari https://bun.sh/docs
- Meta Open Source. (2025). React - The library for web and native user interfaces. Dokumentasi Resmi React. Diakses dari https://react.dev/reference/react
- Vite Contributors. (2025). Vite - Next Generation Frontend Tooling. Dokumentasi Resmi Vite. Diakses dari https://vite.dev/guide/
- Tailwind Labs. (2025). Tailwind CSS Documentation - Installation. Dokumentasi Resmi Tailwind CSS. Diakses dari https://tailwindcss.com/docs/installation
- Drizzle Team. (2025). Drizzle ORM - Get Started with PostgreSQL. Dokumentasi Resmi Drizzle ORM. Diakses dari https://orm.drizzle.team/docs/get-started-postgresql