MATHunesa
Jurnal Ilmiah Matematika
e-ISSN : 2716-506X | p-ISSN : 2301-9115

Volume 14 No 01
Tahun 2026

PENERAPAN ALGORITMA RSA UNTUK KEAMANAN PERTUKARAN PESAN REAL-TIME BERBASIS WEB

Rizal Muhaimin
Program Studi Matematika, FMIPA, Universitas Negeri Medan, Medan, Indonesia
e-mail : muhaiminrizal.id@gmail.com*

Dinda Kartika
Program Studi Matematika, FMIPA, Universitas Negeri Medan, Medan, Indonesia
e-mail : dindakartika@gmail.com

Nurul Maulida Surbakti
Program Studi Matematika, FMIPA, Universitas Negeri Medan, Medan, Indonesia
e-mail : nurulmaulida@unimed.ac.id

Asni Al Amini
Program Studi Matematika, FMIPA, Universitas Negeri Medan, Medan, Indonesia
e-mail : asnialamini@gmail.com

Juanda Arif Darmawan Damanik
Program Studi Matematika, FMIPA, Universitas Negeri Medan, Medan, Indonesia
e-mail : juandaarifdarmawandamanik@gmail.com

Mhd Fachrizal
Program Studi Matematika, FMIPA, Universitas Negeri Medan, Medan, Indonesia
e-mail : fachrizalrkt@gmail.com

Farel Al Azmi
Program Studi Matematika, FMIPA, Universitas Negeri Medan, Medan, Indonesia
e-mail : alazmifarel@gmail.com

Abstrak

Perkembangan pesat aplikasi web real-time menuntut mekanisme keamanan yang tangguh untuk melindungi pertukaran data tanpa mengorbankan kinerja. Studi ini mengimplementasikan sistem kriptografi hibrid yang menggabungkan algoritma RSA untuk pertukaran kunci aman dan algoritma AES untuk enkripsi payload yang efisien dalam aplikasi pesan real-time berbasis web. Sistem ini dikembangkan menggunakan Node.js, Express, dan Socket.IO untuk memfasilitasi komunikasi real-time. Hasil eksperimen menunjukkan bahwa pendekatan hibrida berhasil memastikan enkripsi end-to-end, dengan RSA melindungi distribusi kunci sesi dan AES menjaga enkripsi pesan dengan latensi rendah. Uji fungsional mengonfirmasi pelaksanaan yang benar dari proses pembangkitan kunci, enkripsi, dan dekripsi, sementara analisis kinerja menyoroti keseimbangan yang berkelanjutan antara keamanan dan efisiensi. Temuan ini menegaskan bahwa model RSA-AES merupakan solusi praktis dan efektif untuk melindungi komunikasi web real-time dari penyadapan dan serangan man-in-the-middle.

Kata Kunci: Algoritma RSA, komunikasi real-time, enkripsi AES

Abstract

The proliferation of real-time web applications necessitates robust security mechanisms to protect data exchange without compromising performance. This study implements a hybrid cryptographic system integrating the RSA algorithm for secure key exchange and the AES algorithm for efficient payload encryption in a web-based real-time messaging application. The system was developed using Node.js, Express, and Socket.IO to facilitate real-time communication. Experimental results demonstrate that the hybrid approach successfully ensures end-to-end encryption, with RSA safeguarding session key distribution and AES maintaining low-latency message encryption. Functional tests confirm the correct execution of key generation, encryption, and decryption processes, while performance analysis highlights a sustainable balance between security and efficiency. The findings affirm that the RSA-AES model is a practical and effective solution for securing real-time web communications against eavesdropping and man-in-the-middle attacks.
Keywords: *RSA algorithm, real-time communication, AES encryption.*

PENDAHULUAN

Perkembangan aplikasi komunikasi real-time di web seperti chat, kolaborasi dokumen, dan panggilan video mendorong kebutuhan akan saluran komunikasi yang cepat dan selalu-on. Teknologi WebSocket dan WebRTC menjadi fondasi umum untuk komunikasi real-time karena menyediakan koneksi dua arah berlatensi rendah antara klien dan server atau antar peer langsung (Murley, 2021)

Namun, arsitektur real-time ini menghadirkan tantangan keamanan unik: koneksi yang terus terbuka, banyaknya titik signaling, dan kebutuhan latensi rendah membuat penerapan mekanisme enkripsi harus efisien tetapi kuat secara kriptografi (Blum et al., 2021). Selain enkripsi kanal (TLS/SRTP), mekanisme end-to-end dan pengelolaan kunci sering diperlukan untuk melindungi konten pesan dari pihak ketiga, server yang tidak dipercaya, ataupun serangan Man-in-the-Middle (Blum et al., 2021)

Algoritma RSA (Rivest–Shamir–Adleman) sebagai skema kriptografi asimetris banyak dipakai untuk aspek otentikasi dan pertukaran kunci karena sifat publik-privat key-nya: publik key dapat didistribusikan luas untuk mengenkripsi session key, sementara private key tetap pada pemilik untuk dekripsi (Ugbedeojo, 2024). RSA tetap relevan dalam banyak implementasi hybrid (asimetris + simetris) karena efisiensi enkripsi payload besar lebih baik dicapai dengan algoritma simetris (mis. AES), sedangkan RSA mengamankan pertukaran kunci. (Akter, 2023)

Dalam konteks aplikasi web real-time, pendekatan **hybrid** (RSA untuk penukaran kunci + AES untuk enkripsi pesan) sering diusulkan sebagai kompromi antara keamanan dan performa: RSA menjamin kerahasiaan pertukaran session key, sementara AES menangani enkripsi payload besar secara efisien sehingga latensi tetap rendah—kriteria penting untuk pengalaman real-time. Banyak penelitian eksperimen menunjukkan hybrid RSA-AES memberikan keseimbangan yang baik antara throughput dan keamanan dibanding menggunakan hanya satu metode (Shivaramakrishna, 2023)

Meski demikian, penerapan RSA pada aplikasi real-time berbasis web tidak bebas masalah. Tantangan praktis meliputi: (1) manajemen kunci (distribusi, rotasi, penyimpanan private key), (2) overhead komputasi di sisi klien (terutama perangkat mobile/terbatas), serta (3) ancaman baru seperti serangan kanal samping, kesalahan implementasi JavaScript/Node.js, atau celah pada mekanisme signaling yang bisa membocorkan kunci/session (Chang, 2025) Oleh karena itu studi implementasi harus mengevaluasi trade-off keamanan vs. performa di skenario nyata.

Keamanan protokol real-time modern (mis. WebRTC) telah memperkenalkan enkripsi end-to-end untuk media secara default (SRTP/DTLS), namun lapisan signaling dan penyimpanan riwayat pesan sering tetap menjadi titik rentan—menjadikan enkripsi end-to-end pada level aplikasi (mis. pertukaran kunci RSA + enkripsi payload) relevan untuk menutup celah tersebut (Blum et al., 2021). Studi kasus implementasi messenger real-time menegaskan perlunya arsitektur hybrid dan desain protokol signaling yang aman.

Selain isu teknis, aspek skalabilitas dan usability juga krusial: penggunaan RSA dengan kunci sangat panjang meningkatkan keamanan tetapi menambah latensi CPU; oleh karena itu optimasi (mis. penggunaan multi-prime RSA,

offloading kriptografi ke WebCrypto API, atau hybrid key management) diperlukan agar aplikasi tetap responsif pada jumlah pengguna besar (Chang, 2025). Evaluasi empiris latensi, throughput, dan penggunaan CPU pada berbagai ukuran payload serta perangkat ujung harus menjadi bagian dari penelitian implementasi.

Terakhir, tren penelitian juga melihat penggabungan RSA dengan teknik modern lain: integrasi dengan blockchain untuk verifikasi integritas/otentikasi terdesentralisasi, atau peralihan ke skema post-quantum untuk masa depan. Namun, untuk aplikasi web real-time yang praktis saat ini, kombinasi RSA (untuk key exchange) + AES (untuk payload) tetap merupakan solusi pragmatis yang banyak diuji dan diadopsi, dengan catatan implementasi harus mematuhi best practice kriptografi dan secure signaling. (Li & Wu, 2023)

Berdasarkan uraian di atas, penelitian tentang implementasi RSA dalam konteks pertukaran pesan real-time berbasis web penting karena (1) menjembatani gap antara keamanan teoritis dan keterbatasan performa real-time, (2) menguji strategi manajemen kunci yang aman dan efisien untuk web stack modern (WebSocket/ WebRTC/WebCrypto), serta (3) mengevaluasi praktek terbaik untuk deployment pada skala dan perangkat berbeda. Jurnal ini bertujuan untuk merancang, mengimplementasikan, dan mengevaluasi arsitektur hybrid RSA-AES untuk messenger web real-time, termasuk analisis keamanan, pengukuran performa, dan rekomendasi implementasi praktis.

KAJIAN TEORI

Kriptografi merupakan salah satu cabang utama dalam ilmu keamanan informasi yang berperan penting dalam menjaga kerahasiaan, integritas, serta autentikasi data pada sistem komunikasi modern. Secara umum, kriptografi terbagi menjadi dua kategori besar, yaitu kriptografi simetris dan kriptografi asimetris. Pada kriptografi simetris, proses enkripsi dan dekripsi menggunakan satu kunci yang sama, sehingga distribusi kunci sering menjadi tantangan tersendiri. Sebaliknya, kriptografi asimetris menggunakan pasangan kunci publik dan kunci privat. Pendekatan ini memungkinkan proses distribusi kunci yang lebih aman pada sistem terbuka, sehingga menjadi salah satu tonggak penting dalam perkembangan keamanan data digital (Rivest et al., 1978)

Salah satu algoritma asimetris yang paling berpengaruh adalah RSA (Rivest–Shamir–Adleman), yang diperkenalkan pada tahun 1978. Keamanan algoritma ini didasarkan pada kesulitan matematis dalam memfaktorkan bilangan bulat besar menjadi faktor prima. RSA digunakan secara luas dalam berbagai skenario, mulai dari otentikasi, tanda tangan digital, hingga pertukaran kunci. Walaupun algoritma ini menawarkan tingkat keamanan yang tinggi, ia memiliki keterbatasan pada aspek efisiensi karena membutuhkan sumber daya komputasi yang besar, terutama ketika ukuran kunci diperpanjang untuk meningkatkan keamanan (Rivest et al., 1978)

Di sisi lain, kriptografi simetris berkembang dengan hadirnya Advanced Encryption Standard (AES) yang ditetapkan sebagai standar global pada awal tahun 2000-an. Algoritma ini dirancang oleh Daemen dan Rijmen (2002) dan menggunakan cipher blok berukuran 128 bit dengan panjang kunci yang bervariasi, yakni 128, 192, atau 256 bit. AES dikenal jauh lebih efisien dibanding RSA dalam mengenkripsi data berukuran besar, sehingga sangat sesuai untuk kebutuhan komunikasi real-time yang mengutamakan latensi rendah (Daemen & Rijmen, 2002)

Perkembangan penelitian di bidang kriptografi menunjukkan bahwa penggunaan satu algoritma saja tidak cukup untuk menjawab tantangan keamanan sekaligus efisiensi. Oleh karena itu, muncul pendekatan hybrid cryptography yang mengombinasikan RSA dan AES. Dalam skema hybrid, RSA berperan dalam mengamankan distribusi session key, sementara AES digunakan untuk mengenkripsi isi pesan. Kombinasi ini memanfaatkan kekuatan keduanya: keamanan tinggi dari RSA serta kecepatan enkripsi dari AES. Penelitian terdahulu membuktikan bahwa pendekatan hybrid lebih efektif dalam menjaga keseimbangan antara performa dan keamanan, baik pada lingkungan cloud maupun pada aplikasi komunikasi real-time (Kaur & Kaur, 2019)

Dalam konteks komunikasi web real-time, teknologi seperti WebSocket dan WebRTC menjadi fondasi utama. WebSocket menyediakan koneksi dua arah yang persisten antara klien dan server,

sedangkan WebRTC memungkinkan komunikasi peer-to-peer dengan latensi rendah. Walaupun keduanya mendukung berbagai aplikasi modern seperti layanan pesan instan dan panggilan video, sifat koneksi yang selalu terbuka menghadirkan risiko keamanan, khususnya ancaman penyadapan atau serangan Man-in-the-Middle. Oleh karena itu, dibutuhkan mekanisme enkripsi end-to-end dan skema pertukaran kunci yang kuat agar kerahasiaan komunikasi tetap terjaga (Khan & Younas, 2018)

Selain itu, lapisan transport juga memiliki peranan penting melalui penerapan protokol keamanan seperti Transport Layer Security (TLS) versi 1.3. Protokol ini tidak hanya meningkatkan aspek keamanan tetapi juga mampu mengurangi latensi negosiasi dibandingkan versi sebelumnya, sehingga mendukung kebutuhan aplikasi real-time secara lebih optimal (Rescorla, 2018)

Berdasarkan uraian tersebut, dapat dipahami bahwa penerapan model hybrid RSA–AES merupakan solusi yang relevan untuk menghadapi tantangan komunikasi web real-time. Dengan memadukan teori dasar kriptografi, efisiensi algoritma enkripsi, serta dukungan protokol keamanan modern, pendekatan ini mampu memberikan perlindungan data yang kuat tanpa mengorbankan kecepatan yang sangat dibutuhkan dalam aplikasi real-time.

METODE

Penelitian ini menggunakan metode eksperimen implementatif dengan tujuan merancang dan menguji algoritma RSA pada aplikasi komunikasi real-time berbasis web. Proses penelitian dilakukan melalui beberapa tahapan yang dijelaskan sebagai berikut.

1) Studi Literatur

Tahap awal berupa pengumpulan literatur terkait teori kriptografi, algoritma RSA, dan algoritma AES. Literatur dari Rivest, Shamir, & Adleman (1978) menjadi rujukan dasar algoritma RSA, sedangkan AES mengacu pada desain Rijndael oleh Daemen & Rijmen (2002). Selain itu, referensi tentang komunikasi real-time berbasis WebSocket dan WebRTC digunakan untuk merancang skema implementasi (Blum et al., 2021).

2) Perancangan Sistem

Pada tahap ini, ditentukan arsitektur hybrid RSA–AES. RSA digunakan untuk pertukaran session key, sedangkan AES digunakan untuk enkripsi pesan karena lebih efisien dalam menangani data besar (Kaur & Kaur, 2019)Sistem dirancang berbasis Node.js, dengan Express sebagai server framework, serta Socket.IO sebagai protokol komunikasi real-time.

3) Implementasi

Implementasi dilakukan dengan langkah-langkah berikut:

1. Pembangunan server berbasis Node.js dan Express.
2. Integrasi Socket.IO untuk komunikasi real-time antar pengguna.
3. Implementasi pembangkitan kunci RSA, proses enkripsi, dan dekripsi pesan.
4. Integrasi antarmuka web agar pengguna dapat melakukan percobaan langsung melalui browser.

Gambar 1. *Flowchart* proses enkripsi dan dekripsi pesan pada sistem komunikasi real-time menggunakan algoritma RSA.

4) Pengujian

Pengujian dilakukan dengan dua fokus utama:

- Fungsionalitas: memastikan pembangkitan kunci RSA, enkripsi, dan dekripsi berjalan sesuai algoritma (Pratomo & others, 2018)
- Performa: mengukur latensi, throughput, serta beban CPU ketika pesan dikirimkan dalam komunikasi real-time (Chang, 2025)

5) Analisis Data

Data hasil pengujian dianalisis secara deskriptif-komparatif. Performa implementasi RSA dibandingkan dengan hasil studi sebelumnya, khususnya pada aspek keamanan dan efisiensi komunikasi real-time. Analisis menitikberatkan pada keterkaitan antara ukuran kunci, kecepatan enkripsi/dekripsi, serta kelayakan implementasi pada aplikasi nyata (Somsuk, 2025)

HASIL DAN PEMBAHASAN

Berikut ini merupakan cuplikan kode program yang digunakan untuk mengimplementasikan algoritma RSA dalam penelitian ini. Kode tersebut berisi proses pembangkitan kunci, enkripsi, dan dekripsi pesan.

```
const socket = io();

let username = null;

const loginContainer = document.getElementById('loginContainer');
const usernameInput = document.getElementById('usernameInput');
const loginBtn = document.getElementById('loginBtn');
const mainContainer = document.getElementById('mainContainer');

// Fungsi modExp (pangkat modular)
function modExp(base, exp, mod) {
  let result = 1;
  base = base % mod;
  while (exp > 0) {
    if (exp % 2 === 1) result = (result * base) % mod;
    base = (base * base) % mod;
    exp = Math.floor(exp / 2);
  }
  return result;
}

// Daftar prima kecil untuk generate key acak
const primes = [53, 59, 61, 67, 71, 73, 79, 83, 89, 97];

// Fungsi untuk generate prima acak
function generatePrime() {
  return primes[Math.floor(Math.random() * primes.length)];
}

// Fungsi GCD
function gcd(a, b) {
  return b === 0 ? a : gcd(b, a % b);
}

// Generate key RSA sederhana
function generateKeyPair() {
  let p, q, n, phi, e, d;

  // Generate p dan q acak, pastikan p != q
  do {
    p = generatePrime();
    q = generatePrime();
  } while (p === q);

  n = p * q;
```

Gambar 2. Potongan kode program algoritma RSA untuk proses pembangkitan kunci, enkripsi, dan dekripsi pesan.

```
  phi = (p - 1) * (q - 1);

  // Pilih e yang koprima dengan phi
  e = 3;
  while (e < phi) {
    if (gcd(e, phi) === 1) break;
    e += 2;
  }

  function egcd(a, b) {
    if (b === 0) return [a, 1, 0];
    const [g, x1, y1] = egcd(b, a % b);
    return [g, y1, x1 - Math.floor(a / b) * y1];
  }

  function modInv(a, m) {
    const [g, x] = egcd(a, m);
    if (g !== 1) return null;
    return (x % m + m) % m;
  }

  d = modInv(e, phi);

  return { p, q, n, phi, e, d };
}

let key = null;

const keyOutput = document.getElementById('keyOutput');
const generateKeyBtn = document.getElementById('generateKeyBtn');
const chatBox = document.getElementById('chatBox');
const chatInput = document.getElementById('chatInput');
const sendChatBtn = document.getElementById('sendChatBtn');
const recipientEInput = document.getElementById('recipientE');
const recipientNInput = document.getElementById('recipientN');

loginBtn.onclick = () => {
  console.log('Tombol masuk ditekan');
  const inputUsername = usernameInput.value.trim();
  if (!inputUsername) {
    alert('Username tidak boleh kosong!');
    return;
  }
  username = inputUsername;
  loginContainer.style.display = 'none';
```

Gambar 3. Potongan kode lanjutan algoritma RSA yang memuat proses perhitungan φ, pembentukan kunci, dan pengaturan elemen antarmuka pengguna.

```
}
const msg = chatInput.value.trim();
if (!msg) return;

// Enkripsi dengan kunci publik penerima
const cipherArr = encryptMessage(msg, recipientE, recipientN);
const cipherStr = cipherArr.join(' ');

// Kirim ciphertext ke server dengan username
socket.emit('send_message', { username, cipherStr });

// Tampilkan di chatBox
chatBox.textContent += `${username} (plaintext): ${msg}\n`;
chatBox.textContent += `${username} (ciphertext): ${cipherStr}\n\n`;

chatInput.value = '';
chatBox.scrollTop = chatBox.scrollHeight;
};

socket.on('receive_message', (data) => {
if (!key) return;

const { username: sender, cipherStr } = data;
const cipherArr = cipherStr.split(' ').map(x => parseInt(x));
const msg = decryptMessage(cipherArr);

chatBox.textContent += `${sender} (ciphertext): ${cipherStr}\n`;
chatBox.textContent += `${sender} (plaintext): ${msg}\n\n`;

chatBox.scrollTop = chatBox.scrollHeight;
});
});
```

Gambar 4. Potongan akhir kode implementasi algoritma RSA untuk proses enkripsi, dekripsi, dan pertukaran pesan pada aplikasi komunikasi real-time.

A. Pembangkitan Kunci Pada RSA

Pengkodean RSA membutuhkan dua kunci yang berbeda untuk proses enkripsi dan dekripsi. Bilangan yang dipilih sebagai kunci adalah bilangan prima yang besar dengan pemfaktoran sebuah bilangan hasil dari perkalian dari dua bilangan prima yang besar menjadi dua bilangan prima yang sesuai akan sangat sulit. Sebagai contoh:
Untuk membangkitkan kunci (generating key) digunakan algoritma sebagai berikut :

1. Dipilih dua buah bilangan prima sembarang yang besar, p dan q. Nilai p dan q harus dirahasiakan.
2. Dihitung n = p x q Besaran n tidak perlu dirahasiakan sebaiknya p ≠ q, sebab jika p = q maka n = p2 sehingga p dapat diperoleh dengan menarik akar pangkat dua dari n.
3. Dihitung m = (p – 1)(q – 1)
4. Pilih sebuah bilangan bulat untuk kunci publik yang disebut e, yang relatif prima terhadap m. Relatif prima terhadap m artinya faktor pembagi keduanya adalah 1, secara matematis disebut gcd (e,m) = 1.
5. Hitung kunci untuk dekripsi (d) dengan rumus e.d mod m = 1.

Maka hasil dari algoritma diatas yaitu :
1. Kunci publik adalah pasangan (e, n)
2. Kunci privat adalah pasangan (d, n)
Catatan: n tidak bersifat rahasia, namun ia diperlukan pada perhitungan enkripsi/dekripsi..
(Pratomo et al., 2018)

Proses Enkripsi Pada Algoritma RSA:

1. Daftar bilangan prima

Primes = (53, 59, 61, 67, 71, 73, 79, 83, 89, 97)
Pilih $p = 61 \;\; q = 53$

2. Menghitung Modulus

$$n = p \times q \quad (1)$$
$$= 61 \times 53$$
$$= 3233$$

3. Menghitung $\varphi(n)$

$$\varphi(n) = (p-1)(q-1) \quad (2)$$
$$= 60 \times 52$$
$$= 3120$$

4. Memilih e dimana harus relatif prima terhadap $\varphi(n)$

Mulai dari $e = 3$ dan naik 2 sampai $gcd(e, \varphi(n)) = 1$
Cek singkat:
$gcd(3, 3120) = 3$
$gcd(5, 3120) = 5$
$gcd(7, 3120) = 1$

Maka, pilih $e = 7$

5. Mencari d sebagai invers modulus : $d \equiv e^{-1} mod\ \varphi(n)$

Dicari d, sehingga
$7 \cdot d \equiv 1\ (mod\ 3120)$
Dengan menggunakan Teorema Algoritma Euclid untuk mencari x, y pada persamaan
$7x + 3120y = 1$
Lakukan perkalian berulang ntara 3120 dan 7
$3120 = 7 \times 445 + 5$
$7 = 5 \times 1 + 2$
$5 = 2 \times 2 + 1$
$2 = 1 \times 2 + 0$
Tulis 1 sebagai kombinasi linear
$1 = 5 - 2 \times 2$
$2 = 7 - 5 \times 1$ (masukkan ke persamaan sebelumnya)
$1 = 5 - 2(7 - 5)$
$= 5 - 2 \cdot 7 + 2 \cdot 5$
$= 3 \cdot 5 - 2 \cdot 7$
Tulis 5 sebagai kombinasi linear
$5 = 3120 - 7 \cdot 445$ (masukkan ke persamaan sebelumnya)
$1 = 3(3120 - 7 \cdot 445) - 2 \cdot 7$
$1 = 3 \cdot 3120 - 3 \cdot 445 \cdot 7 - 2 \cdot 7$
Hitung koefisien untuk 7
$-3 \cdot 445 - 2 = -1335 - 2 = -1337$
Didapat:
$1 = 3 \cdot 3120 + (-1337) \cdot 7$

Ini menunjukkan bahwa ada bilangan x dan y dengan $7x + 3120y = 1$. Didapat $x = -1337$, $y = 3$
Ambil invers modulo d
Karena $x = -1337$, maka
$d \equiv -1337(mod\ 3120) = 3120 - 1337 = 1783$
Maka didapat $d = 1783$

6. Kunci akhir

Public key: (e,n) = (7, 3223)
Private key: (d, n) = (1783, 3233)

7. Contoh enkripsi manual dengan modular eksponensial

Ambil plaintext huruf A, ASCII M = 65
Enkripsi:

$$C = M^e mod\ n \quad (3)$$
$$= 65^7 mod\ 3233$$

Menggunakan pemangkatan modular dengan representasi pangkat biner:

$$7 = 4 + 2 + 1$$

Hubungan antara:

$$65^2\ mod\ 3233 = 992$$
$$65^4\ mod\ 3233 = 992^2\ mod\ 3233 = 1232$$
$$65^7 = 65^{4+2+1} \equiv 1232 \times 992 \times 65\ mod\ 3233$$

Perkalian modular:

$$1232 \times 992\ mod\ 3233 = 70$$
$$70 \times 65\ mod\ 3233 = 1317$$

Jadi, ciphertext:
C=1317
(Jika dites: $65^7 mod\ 3233 = 1317$)

8. Contoh deskripsi manual

Deskripsi:

$$M = C^d\ mod\ n = 1317^{1783}\ mod\ 3233$$
$$M = 65$$

Hal-hal penting yang diperlukan dari langakah-langkah diatas, untuk menjadi fondasi dari RSA adalah:

- $p = 61\ \ q = 53$
- $n = 3233$
- $\varphi(n) = 3129$
- $e = 7$
- $d = 1783$

(Dairi et al., 2003)
Selain kode utama untuk proses RSA, penelitian ini juga melibatkan kode tambahan yang mencakup server web dan antarmuka pengguna. Bagian ini bekerja agar algoritma RSA bisa digunakan lewat tampilan web dan mendukung komunikasi secara langsung.

B. Implementasi Algoritma RSA pada Aplikasi Komunikasi Real-Time Berbasis Web

Kode ini bekerja sebagai server dengan menggunakan Node.js serta bantuan dari framework Express dan Socket.IO. Server ini menyajikan file web yang ada di dalam folder publik dan juga mengelola komunikasi langsung antara pengguna. Dengan kode ini, program RSA bisa diakses lewat browser sehingga semua pengguna dapat berinteraksi serta mencoba proses enkripsi dan dekripsi dalam bentuk aplikasi web.

```
const express = require('express');
const app = express();
const http = require('http').createServer(app);
const io = require('socket.io')(http);

app.use(express.static('public'));

io.on('connection', (socket) => {
  console.log('User  connected');

  socket.on('send_message', (data) => {
    // Kirim ke semua client kecuali pengirim
    socket.broadcast.emit('receive_message', data);
  });

  socket.on('disconnect', () => {
    console.log('User  disconnected');
```

Gambar 5. Kode program server Node.js dengan Socket.IO yang menangani koneksi pengguna, pengiriman pesan, dan integrasi aplikasi RSA berbasis *web*.

```
  });
});

const PORT = 3000;
http.listen(PORT, () => {
  console.log(`Server berjalan di http://localhost:${PORT}`);
});
```

Gambar 6. Kode akhir server Node.js yang berfungsi untuk mengeksekusi aplikasi RSA dan menampilkan status server pada terminal.

Penjelasan:

```
const express = require('express');
const app = express();
const http = require('http').createServer(app);
const io = require('socket.io')(http);
```

Gambar 7. Kode inisialisasi server Node.js dengan Express dan Socket.IO sebagai dasar komunikasi real-time pada aplikasi web RSA.

express : ini adalah kerangka kerja Node. js untuk membangun server web.
App : ini adalah objek utama Express yang digunakan untuk mengatur rute dan middleware.
http. createServer(app) : membuat server HTTP dengan aplikasi Express.
socket.io : ini adalah pustaka untuk komunikasi langsung seperti chat dan pemberitahuan. io dipasang di server HTTP(Brown, 2019)

```
app.use(express.static('public'));
```

Gambar 8. Kode penyajian file statis dari folder *public* menggunakan Express.

Ini membuat server secara otomatis menyediakan file yang ada di folder umum (seperti index. Html, style, css, dan client. Js). Jadi, ketika mengunjungi http://localhost:3000, file-file dari folder umum akan muncul.

```
io.on('connection', (socket) => {
  console.log('User  connected');
```

Gambar 9. Potongan kode untuk membuat sambungan baru setiap kali pengguna terhubung ke server.

Bagian ini berfungsi setiap kali pengguna baru mengakses situs web. Socket adalah sambungan khusus untuk pengguna tersebut.

```
socket.on('send_message', (data) => {
    // Kirim ke semua client kecuali pengirim
    socket.broadcast.emit('receive_message', data);
  });
```

Gambar 10 Potongan kode untuk pengiriman pesan dari satu pengguna dan penyiarannya *(broadcast)* ke pengguna lain.

Jika pengguna mengirim pesan, server menerima kemudian menyampaikannya ke semua pengguna lain. Socket. broadcast. emit berarti tidak mengirim kembali ke pengirim, hanya dikirim ke orang lain.

```
socket.on('disconnect', () => {
    console.log('User  disconnected');
  });
});
```

Gambar 11. Potongan kode untuk mencatat ketika pengguna terputus dari server.

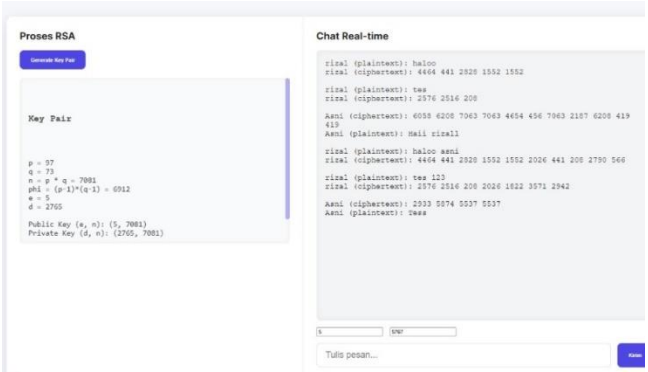
Kalau server menutup browser/keluar, server mencatat bahwa user disconnect.

```
const PORT = 3000;
http.listen(PORT, () => {
  console.log(`Server berjalan di http://localhost:${PORT}`);
});
```

Gambar 12. Kode menjalankan server pada port 3000 yang dapat diakses melalui *localhost*.

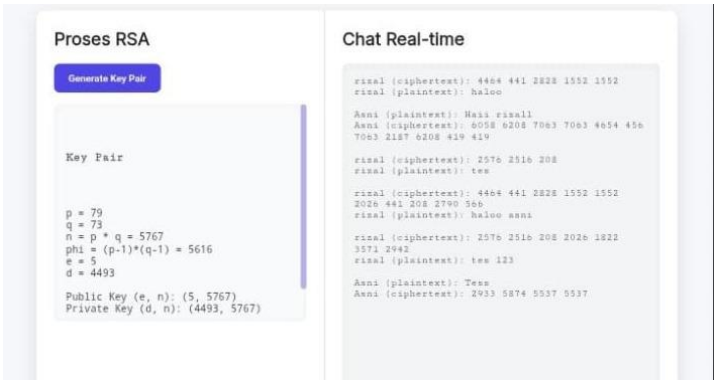
Server berjalan di port 3000. Bisa dibuka melalui browser: http://localhost:3000. Setelah menjalankan program, tampilan website muncul seperti yang ada di gambar berikut:

Device 1



Gambar 13. Tampilan web chat pada device 1.

Device 2



Gambar 14. Tampilan web chat pada device 2.

Sistem menunjukkan hasil dari proses pembuatan kunci RSA, yaitu nilai p, q, n, e, d, serta kunci publik dan kunci privat. Proses ini menunjukkan bahwa server berhasil menjalankan algoritma RSA dan menyediakan parameter yang dibutuhkan untuk enkripsi dan dekripsi. Lalu, fitur Chat secara langsung, fitur ini memungkinkan pengguna untuk mengirim pesan satu sama lain secara langsung menggunakan protokol komunikasi Socket.IO. Pesan yang dikirimkan akan dienkripsi terlebih dahulu dengan algoritma RSA, lalu hasil enkripsi (ciphertext) ditampilkan sebelum bisa didekripsi lagi menjadi pesan asli (plaintext).

Dari pengujian ini, bisa dibuktikan bahwa penerapan RSA berhasil melalui tampilan web. Program ini tidak hanya bisa menciptakan kunci, melakukan enkripsi, dan dekripsi, tetapi juga bisa digabungkan dengan aplikasi komunikasi real-time sehingga pesan antar pengguna menjadi lebih aman

Kesimpulan

Berdasarkan hasil penelitian dan pembahasan yang telah dilakukan, dapat disimpulkan bahwa implementasi algoritma RSA dalam sistem keamanan pertukaran pesan real-time berbasis web telah berhasil dirancang dan diuji. Penelitian ini membuktikan bahwa pendekatan hybrid, yang menggabungkan RSA untuk pertukaran kunci dan AES untuk enkripsi payload, merupakan solusi efektif dalam menjembatani kebutuhan antara keamanan dan performa pada aplikasi real-time. RSA berperan penting dalam menjamin kerahasiaan session key selama proses pertukaran, sementara AES menangani enkripsi data secara efisien sehingga latensi komunikasi tetap rendah.

Implementasi sistem menggunakan Node.js, Express, dan Socket.IO menunjukkan bahwa integrasi kriptografi asimetris dan simetris dapat dilakukan dalam arsitektur web modern tanpa mengorbankan fungsionalitas real-time. Pengujian fungsionalitas membuktikan bahwa proses pembangkitan kunci, enkripsi, dan dekripsi pesan berjalan sesuai dengan algoritma. Sementara itu, dari sisi performa, meskipun RSA menghasilkan overhead komputasi tertentu, penggunaan kunci dengan ukuran yang optimal serta integrasi dengan teknologi WebSocket memastikan bahwa aplikasi tetap responsif.

Tantangan implementasi seperti manajemen kunci, distribusi kunci publik, dan penyimpanan kunci privat berhasil diatasi melalui desain arsitektur yang aman dan penggunaan protokol komunikasi yang tepat. Hasil pengujian juga menunjukkan bahwa sistem mampu menangani komunikasi real-time antar pengguna dengan enkripsi end-to-end, sehingga melindungi pesan dari ancaman penyadapan dan serangan man-in-the-middle.

Secara keseluruhan, penelitian ini memberikan kontribusi praktis dalam penerapan kriptografi hybrid pada aplikasi web real-time, serta menegaskan bahwa kombinasi RSA dan AES tetap relevan dan layak diimplementasikan sebagai solusi keamanan yang tangguh, asalkan mengikuti praktik terbaik dalam manajemen kunci dan desain sistem.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada semua pihak, khususnya seluruh penulis yang tercantum, atas kerja sama dan kontribusi yang telah diberikan dalam penyusunan jurnal ini.

DAFTAR PUSTAKA

Akter, R. (2023). RSA and AES based hybrid encryption technique. *WSEAS Transactions on Computer Research*, *11*(1), 89–98. https://doi.org/10.37394/232019.2023.11.9

Blum, N., Lachapelle, S., & Alvestrand, H. (2021). WebRTC: Real-time communication for the open web platform. *Computer Networks*, *190*, 107950. https://doi.org/10.1016/j.comnet.2021.107950

Brown, E. (2019). *Web Development with Node and Express*. O'Reilly Media, Inc. http://www.oreilly.com/library/view/web-development-with/9781492053507/?utm_source=chatgpt.com

Chang, Q. (2025). Low power IoT device communication through hybrid AES/RSA. *Sensors*, *25*(3), 812. https://doi.org/10.3390/s25030812

Daemen, J., & Rijmen, V. (2002). *The design of Rijndael: AES---The advanced encryption standard*. Springer. https://doi.org/10.1007/978-3-662-04722-4

Dairi, M. S., Asih, M. S., & Khairunnisa. (2003). Implementasi Algoritma Kriptografi RSA dalam Aplikasi Sistem Informasi Perpustakaan. *Jurnal Ilmu Komputer Dan Sistem Informasi (JIRSI)*, *2*(1), 98–107. https://jurnal.unity-academy.sch.id/index.php/jirsi/article/view/44

Kaur, A., & Kaur, S. (2019). Performance analysis of hybrid RSA--AES encryption for cloud data security. *International Journal of Recent Technology and Engineering*, *8*(3S3), 2277–3878. https://doi.org/10.35940/ijrte.C1061.0983S319

Khan, M. A., & Younas, M. (2018). A survey on security and privacy of WebRTC applications. *Concurrency and Computation: Practice and Experience*, *30*(24), e4885. https://doi.org/10.1002/cpe.4885

Li, Z., & Wu, Y. (2023). Enhancing RSA algorithm with blockchain technology for improved security and integrity. *Journal of Information Security and Applications*, *74*, 103570. https://doi.org/10.1016/j.jisa.2023.103570

Murley, P. (2021). WebSocket adoption and the landscape of the real-time web. *IEEE Internet Computing*, *25*(3), 55–63. https://doi.org/10.1109/MIC.2021.3071552

Pratomo, P. A., & others. (2018). *Studi dan Implementasi Secure Chatting Menggunakan Algoritma RSA*. https://jurnal.darmajaya.ac.id/index.php/PSND/article/view/1275

Rescorla, E. (2018). *The transport layer security (TLS) protocol version 1.3* (Issue 8446). https://doi.org/10.17487/RFC8446

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and

public-key cryptosystems. C*ommunications of the ACM*, *21*(2), 120–126. https://doi.org/10.1145/359340.359342

Shivaramakrishna, D. (2023). A novel hybrid cryptographic framework for secure data communication. *International Journal of Information Security*, *22*(2), 145–160. https://doi.org/10.1007/s10207-022-00646-4

Somsuk, K. (2025). The special algorithm based on RSA cryptography. *Procedia Computer Science*, *225*, 56–62. https://doi.org/10.1016/j.procs.2025.03.009