INFORMATION SECURITY – SEMESTER PROJECT

Project Title: ***Secure End-to-End Encrypted Messaging & File-Sharing System***

Course: Information Security – BSSE (7th Semester)

Group Size: 3 Students (Cross Section Group is Allowed)

1. Project Overview

The purpose of this semester project is to design and develop a **secure communication system** that provides **end-to-end encryption (E2EE)** for text messaging and file sharing. The system must ensure that:

- Messages and files **never exist in plaintext outside the sender or receiver device**
- The server is unable to decrypt or view any user content
- Students implement **hybrid cryptography** combining asymmetric encryption (RSA/ECC) with symmetric encryption (AES-GCM)
- A secure **key exchange protocol** is designed, implemented, tested, and documented
- Students must simulate and demonstrate **attacks** (MITM, replay) and show how their protocol prevents or mitigates them
- Students must conduct **security analysis**, **logging**, and **threat modeling**

This is a *development + security + cryptography + attack/defense* project and represents a complete applied Information Security system.

2. Project Requirements

Your system must include the following **mandatory components**.

2.1 Functional Requirements

1. User Authentication (Basic)

- Create user accounts (username + password or OAuth)
- Store passwords **securely** (salted + hashed using bcrypt/argon2)
- Optional (bonus): Implement two-factor authentication

2. Key Generation & Secure Key Storage

Each user must generate on registration:

- **Asymmetric key pair**
  - RSA-2048/3072 OR
  - ECC (P-256 or P-384)

Private keys must NEVER be stored on the server.
They must be stored only on the client device using:

- Web Crypto + IndexedDB **OR**
- Secure local encrypted storage (e.g., CryptoJS + localStorage encryption)

Students must justify and demonstrate secure storage.

3. Secure Key Exchange Protocol

You must design and implement a **custom key exchange protocol** (your own variant, not a copy from textbooks).
Protocol must:

- Use Diffie-Hellman (DH) OR Elliptic Curve Diffie-Hellman (ECDH)
- Combine with a **digital signature** mechanism
- Ensure authenticity to prevent MITM attacks
- Derive a session key using HKDF or SHA-256
- Implement a final **"Key Confirmation"** message

Students MUST draw and explain the message flow in their report.

4. End-to-End Message Encryption

Every message must use:

- **AES-256-GCM**
- A fresh random IV per message
- An authentication tag (MAC) to protect integrity

Server may only store:

- Ciphertext
- IV
- Metadata (sender/receiver IDs, timestamp)

No plaintext message content may be stored anywhere on the backend.

5. End-to-End Encrypted File Sharing

Files must:

- Be encrypted **client-side** (before uploading)
- Be split into chunks (recommended, not mandatory)
- Have each chunk encrypted with AES-256-GCM
- Be stored on the server only in encrypted form

Receivers must be able to download and decrypt files locally.

6. Replay Attack Protection

Implement **ALL** of the following:

- Nonces
- Timestamps
- Message sequence numbers OR counters
- Verification logic to reject replayed messages

Attack demonstration must be included in your report.

7. MITM Attack Demonstration

Your group must:

- Create an “attacker script” OR use BurpSuite
- Show how MITM successfully breaks DH **without** signatures
- Show how digital signatures **prevent MITM** in your final system

Screenshots, logs, and explanations must be provided.

8. Logging & Security Auditing

System must maintain logs for:

- Authentication attempts
- Key exchange attempts
- Failed message decryptions
- Detected replay attacks
- Invalid signatures
- Server-side metadata access

Logs must be shown in the report.

9. Threat Modeling

Using **STRIDE**, perform threat modeling for your system:

- Identify threats
- Identify vulnerable components
- Propose countermeasures
- Map threats to your implemented defenses

This section must be detailed and personalized for your design.

10. System Architecture & Documentation

Your submission must include:

- **High-level architecture diagram**
- Client-side flow diagrams
- Key exchange protocol diagrams
- Encryption/decryption workflows
- Schema design
- Deployment description (local or cloud)

3. Technical Requirements

3.1 Allowed Technologies *(mandatory unless permission granted)*

Frontend:
✔ React.js
✔ Web Crypto API (SubtleCrypto) for cryptographic operations
✔ IndexedDB or encrypted localStorage for key storage
✔ Axios/WebSocket as needed

Backend:
✔ Node.js + Express
✔ MongoDB for metadata
✔ Socket.io (optional for real-time chat)

Tools for attacks & analysis:
✔ Wireshark
✔ BurpSuite Community Edition
✔ OpenSSL CLI

3.2 Forbidden Technologies (Not Allowed)

- Firebase or third-party authentication
- Third-party E2EE libraries (e.g., Signal, Libsodium, OpenPGP.js)
- Pre-built cryptography wrappers (CryptoJS for RSA/ECC, NodeForge, etc.)
- Using ChatGPT to generate full code modules
- Copying existing encrypted messaging apps
- Using pre-built themes or templates for the cryptographic core

Only the following crypto sources are allowed:

- Browser's **Web Crypto API** (mandatory for client-side)
- Node's **crypto** module for backend digital signatures (optional)
- Raw JavaScript implementations written by the group

4. Constraints & Limitations

4.1 Development Constraints

- All encryption must occur **client-side.**
- Private keys must never leave the client.
- No plaintext must be logged, stored, or transmitted.
- Students must implement at least 70% of the cryptographic logic themselves.
- All communication must use HTTPS.

4.2 Security Constraints

- AES-GCM only (no CBC, no ECB).
- RSA key size must be ≥2048 bits.
- ECC must use NIST curves only (P-256 or P-384).
- IVs must be unpredictable and non-repeating.
- All signature verification must include timestamp checks.

4.3 Evaluation Constraints

To ensure originality:

- Each group must implement a **unique variant** of the key exchange protocol.

- Each group must implement **different message structures**.
- Each group must provide **real packet captures** as evidence.
- Code similarity above **35%** with another group will be flagged.

5. Deliverables

1. Full Project Report (PDF)

Must include:

- Introduction
- Problem statement
- Threat model (STRIDE)
- Cryptographic design
- Key exchange protocol diagrams
- Encryption/decryption workflows
- Attack demonstrations (MITM & replay)
- Logs and evidence
- Architecture diagrams
- Evaluation and conclusion

2. Working Application

- Functional E2EE messaging
- Encrypted file sharing
- Replay/disconnect handling
- Error handling
- Decryption logic on client only

3. Video Demonstration (10–15 min)

Must include:

- Protocol explanation
- Working demo of encrypted chat
- Upload/download of encrypted files
- MITM attack demo
- Replay attack demo
- Discussion of limitations and improvements

4. GitHub Repository

Must contain:

- Source code (client + server)
- Code must be maintained using Git as Private Repository
- Equal amount of code contribution must be shown by each team member
- README.md with setup instructions
- Documentation
- Screenshots of Wireshark/BurpSuite tests
- No build artifacts or compiled code

6. Evaluation Criteria *(See Rubrics for Details)*

Component	Marks
Functional correctness	20

Cryptographic design & correctness	20
Key exchange protocol	15
Attack demonstration (MITM, replay)	15
Threat modeling & documentation	10
Logging & auditing	5
UI/UX and stability	5
Code quality & originality	10
Total	**100**

- Using external E2EE libraries is prohibited.
- All groups must demonstrate **real system-specific evidence**, not AI-generated examples.
- All packet capture and attack logs must be from **your own system**.
- Any similarity above threshold will result in disciplinary action.

You may use LLMs **only for conceptual help or debugging**, not for generating complete modules, protocols, or code blocks.