# INFORMATION SECURITY – SEMESTER PROJECT

# 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** |

# 7. Plagiarism & LLM Misuse Policy

- Using ChatGPT, GitHub Copilot, or similar tools for **full code generation** is prohibited.
- 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.

# Rubrics

# Evaluation Criteria: Detailed Rubric

## Functional Correctness (20 Marks)

| Criteria | 0-7 Marks (Poor/Incomplete) | 8-14 Marks (Fair/Meets Minimum) | 15-20 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| **User Authentication & Key Generation** | Authentication is insecure (e.g., plaintext password storage) or key generation fails/is non-functional. Private key storage is insecure (e.g., on server or unencrypted client storage). | Secure password hashing (bcrypt/argon2) implemented. Keys (Asymmetric & Symmetric) are generated, but key storage justification is weak or non-standard client storage is used. | Strong, standard-compliant authentication (salted/hashed). Secure client-side private key storage demonstrated and well-justified (Web Crypto + IndexedDB or strong local encryption). |
| **Messaging & File Sharing Functionality** | Basic messaging or file sharing fails frequently. Only basic E2EE for messages works, file sharing is non-functional or encrypts server-side. | Both E2EE messaging (AES-GCM) and encrypted file sharing (client-side encryption) are functional but may have minor bugs or poor performance. | Seamless, robust, and stable E2EE messaging and client-side encrypted file sharing, including chunking (if implemented). Excellent error handling. |

## Cryptographic Design & Correctness (20 Marks)

| Criteria | 0-7 Marks (Poor/Incomplete) | 8-14 Marks (Fair/Meets Minimum) | 15-20 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| **Hybrid Crypto Implementation** | Uses incorrect/forbidden crypto primitives (e.g., AES-CBC, weak RSA). IV reuse or weak symmetric key management. | Correctly uses AES-256-GCM, RSA/ECC of required strength. Uses fresh random IVs. Demonstrates basic hybrid flow. | Flawless implementation of the hybrid scheme. IV generation is provably random and correct. **Authenticated encryption** is correctly implemented and verified. |

| Criteria | 0-7 Marks (Poor/Incomplete) | 8-14 Marks (Fair/Meets Minimum) | 15-20 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| | | | Demonstrates a clear understanding of crypto parameters and constraints. |
| **Adherence to Constraints** | Violates multiple constraints (e.g., uses forbidden library, stores plaintext on server, wrong key sizes). | Meets mandatory constraints (AES-GCM only, correct key sizes). All encryption is strictly client-side. No forbidden technologies are used. | Meets all mandatory and security constraints perfectly. Logic is implemented with minimum reliance on high-level library functions, demonstrating deep self-implementation (70% or more). |

## Key Exchange Protocol (15 Marks)

| Criteria | 0-5 Marks (Poor/Incomplete) | 6-10 Marks (Fair/Meets Minimum) | 11-15 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| **Protocol Design & Implementation** | Protocol is a direct copy or fails to implement DH/ECDH or digital signatures. Key Confirmation is missing or non-functional. | Implements DH/ECDH and digital signatures. Protocol is functional but lacks originality or is vulnerable to attack (e.g., weak signature handling). Session key derived using a basic method (e.g., simple concatenation). | **Unique, well-designed variant** of the key exchange protocol. Flawless combination of DH/ECDH with robust digital signatures. Uses HKDF or similar standard-compliant key derivation function. Key Confirmation message is correctly implemented and verified. |
| **Documentation & Diagrams** | No protocol diagrams or explanation provided. Message flow is confusing or incomplete. | Basic message flow diagram provided. Report contains a textual explanation of the steps. | Clear, professional-quality protocol diagrams (e.g., sequence diagrams) in the report. Detailed, step-by-step explanation of the cryptographic logic and message structures. |

## Attack Demonstration (MITM, Replay) (15 Marks)

| Criteria | 0-5 Marks (Poor/Incomplete) | 6-10 Marks (Fair/Meets Minimum) | 11-15 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| **MITM Attack/Defense** | Fails to simulate a MITM attack or defense against it is non-existent. No evidence of attack simulation (Wireshark/BurpSuite). | Successfully demonstrates how MITM breaks plain DH/ECDH. Successfully demonstrates the defense (digital signatures) prevents the attack, but evidence (logs/screenshots) is minimal. | Clearly demonstrates the MITM attack breaking the unsecured protocol variant. Provides detailed log evidence (packet captures, BurpSuite logs) and a clear explanation of how digital signatures and key confirmation effectively mitigate the attack. |
| **Replay Attack/Defense** | Fails to implement replay protection or fails to simulate the attack. | Replay protection (Nonces, Timestamps, Sequence Numbers) is implemented, and a basic attack is demonstrated, showing the defense works. | Robust implementation of **all** mandatory replay protection mechanisms (Nonces, Timestamps, Sequence Numbers). Clear, documented demonstration of a successful replay attempt and the system's logic for rejecting the replayed message, with logs as evidence. |

## Threat Modeling & Documentation (10 Marks)

| Criteria | 0-3 Marks (Poor/Incomplete) | 4-6 Marks (Fair/Meets Minimum) | 7-10 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| **Threat Modeling (STRIDE)** | STRIDE analysis is generic, incomplete, or copied. Vulnerabilities/countermeasures are not mapped to the implemented system. | Identifies major threats using STRIDE. Proposes relevant countermeasures. Basic mapping of threats to implemented defenses is present. | Comprehensive, system-specific STRIDE analysis. Identifies subtle threats related to E2EE and key management. Detailed, effective countermeasures are proposed and clearly mapped to the implemented defenses and architecture. |

| Criteria | 0-3 Marks (Poor/Incomplete) | 4-6 Marks (Fair/Meets Minimum) | 7-10 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| **System Architecture & Documentation** | Missing or unclear architecture diagrams, workflow diagrams, or schema. Report structure is poor. | All mandatory documentation components are present (architecture, flows, schema) but may lack detail or clarity. | High-quality, clear, and professional-grade documentation. All required diagrams (architecture, client flow, crypto flow, protocol) are detailed, accurate, and easy to follow. Report is well-structured and comprehensive. |

## Logging & Auditing (5 Marks)

| Criteria | 0-1 Marks (Poor/Incomplete) | 2-3 Marks (Fair/Meets Minimum) | 4-5 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| **Logging Implementation** | No security-relevant logging is implemented, or logs are incomplete/insecure. | Implements logging for most mandatory events (Authentication, Key Exchange, Decryption Failures, Replay Detections). Logs are shown in the report. | Comprehensive logging for **all** mandatory events and other critical actions (e.g., key revocation attempts). Logs are structured, include necessary metadata (timestamps, user IDs), and are clearly presented and analyzed in the report. |

## UI/UX and Stability (5 Marks)

| Criteria | 0-1 Marks (Poor/Incomplete) | 2-3 Marks (Fair/Meets Minimum) | 4-5 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| **Application Quality** | Application is highly unstable, crashes frequently, and the user interface is minimal or unusable. | Application is stable and functional. UI/UX is basic but allows for all mandatory features to be tested easily. | Application is highly stable and robust. UI/UX is clean, intuitive, and enhances the demonstration. Error messages and user |

| Criteria | 0-1 Marks (Poor/Incomplete) | 2-3 Marks (Fair/Meets Minimum) | 4-5 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| | | | feedback (e.g., successful key exchange) are clear and helpful. |

## Code Quality & Originality (10 Marks)

| Criteria | 0-3 Marks (Poor/Incomplete) | 4-6 Marks (Fair/Meets Minimum) | 7-10 Marks (Excellent/Exceeds Expectations) |
|---|---|---|---|
| **Code Structure & Maintainability** | Code is messy, poorly commented, and violates basic coding standards. Significant code similarity detected. | Code is generally readable and commented, following basic structure. Minimal reliance on forbidden or external code. | Clean, modular, and well-structured code (client and server). Comprehensive, accurate comments explaining the cryptographic logic. Repository is professionally maintained. Demonstrates high originality in design and implementation of the protocol logic. |
| **Video Demonstration** | Video is missing, too short, or lacks mandatory content (protocol explanation, attack demos). | Video covers all mandatory components but may be poorly organized or rushed. Demos are functional but lack clear explanation. | Professional, well-edited video (10–15 min). Excellent, clear explanation of the protocol and cryptographic core. Flawless demonstration of all features and attacks/defenses. |