The architecture of our secure communication system is designed around client-server interaction using RSA and AES encryption algorithms to ensure confidentiality, integrity, and authenticity of transmitted data. The system comprises a server-side application (Server.java) responsible for managing client connections and a client-side application (Client.java) facilitating user interactions.

The server application initializes a fixed thread pool to handle client connections efficiently. Upon startup, it loads its private RSA key from a PEM file for decrypting AES keys sent by clients. Each client connection is managed by a ClientHandler thread, which handles encryption/decryption and message/file processing. The server maintains an in-memory collection (ConcurrentHashMap) of connected clients and their handlers for concurrency management.

The client application presents a graphical user interface (GUI) using AWT and Swing components, allowing users to send text messages or files securely. Upon launching, the client prompts the user for a username and establishes a socket connection with the server. It loads the server's public RSA key from a PEM file for encrypting AES keys. Each client session generates a random AES key for message/file encryption, ensuring session-specific security.

During the key generation phase, the client application generates a new AES key (256-bit) using Java's KeyGenerator class for each session, ensuring session-specific security. On the server side, the architecture relies on clients to send encrypted AES keys, which are then decrypted by the server using its private RSA key. This key exchange process involves the client encrypting the AES key with the server's public RSA key and securely transmitting it over the socket, while the server decrypts the received AES key using its private RSA key for subsequent encryption and decryption operations. Regarding key storage, the client stores AES keys in memory during the session and generates new keys for each new session to maintain security. Similarly, the server securely stores its private RSA key in memory and manages client-specific AES keys throughout the sessions to ensure confidentiality and integrity of communications. While the server's private RSA key is stored in memory during runtime, the client's AES keys are also stored in memory but are not persisted to disk between sessions.

In terms of security implementation details, our system prioritizes several key aspects. Firstly, confidentiality is ensured through AES encryption, which safeguards the transmission of messages and files between clients and the server, preventing unauthorized access. Secondly, to maintain data integrity and prevent tampering, SHA-256 hashes are computed and appended to data, allowing for rigorous integrity verification. This process guarantees that transmitted data remains unchanged and trustworthy throughout its journey. Lastly, for authentication purposes, RSA encryption plays a crucial role in securing the key exchange process, thereby verifying the identity of both clients and the server. This authentication mechanism adds an extra layer of security, ensuring that communications are established only between trusted entities, thereby minimizing the risk of unauthorized access or data breaches.

To address potential problems and enhance the robustness of our secure communication system, we implement various mitigations. First, to mitigate security risks such as cryptographic vulnerabilities or attacks, we prioritize regular updates, conduct thorough code reviews, and strictly adhere to best practices in encryption and key management. Secondly, to tackle concurrency issues arising from concurrent access to shared resources, we ensure proper synchronization mechanisms and utilize thread-safe data structures like ConcurrentHashMap to maintain data integrity and consistency. Additionally, we tackle file handling challenges associated with large transfers and network bottlenecks by implementing optimization strategies, chunking files for efficient transmission, and integrating progress indicators to enhance scalability and user experience. Lastly, for user experience enhancements, we focus on GUI improvements that provide clear user feedback, including progress bars, status indicators, and robust error handling mechanisms, resulting in a smoother and more intuitive user interaction throughout the application.