*Green University of Bangladesh*

*Department of Computer Science and Engineering (CSE)*
*Semester: (Spring, Year: 2023), B.Sc. in CSE (Day)*

---

# Secure Text Communication System Using Modern Cryptographic Techniques

---

*Course : Computer and Cyber Security*
*Course Code: CSE 323*
*Section: 221 D1*

Students Details

| Name | ID |
|---|---|
| Meherun Nesa Enta | 221002178 |

*Submission Date: 19.12.24*
*Course Teacher's Name: Md. Jahidul Islam*

[For teachers use only: Don't write anything inside this box]

| Theory Project Status | |
|---|---|
| **Marks:** | **Signature:** |
| **Comments:** | **Date:** |

# Contents

# Chapter 1

# Introduction

## 1.1 Overview

A crucial component of contemporary technology is secure communication, which guarantees the privacy, accuracy, and legitimacy of data transferred between parties. In order to shield text communications from unwanted access, the "Secure Text Communication System Using Modern Cryptographic Techniques" seeks to offer a reliable platform for text message encryption and decryption. Sensitive data is protected both during transmission and storage because to this system's use of well-known cryptographic methods like RSA and AES. It meets the demands of organizations and individuals for safe communication.

## 1.2 Motivations

The need for secure communication networks is highlighted by the rise in cyberthreats and data breaches in the current digital era. Unauthorized access to private data can have serious consequences, such as identity theft, monetary loss, and harm to one's reputation. Inspired by these difficulties, this initiative seeks to:

- Address the escalating data privacy concerns.
- Give users a straightforward but safe communication method.
- Make use of the advantages of contemporary cryptography methods to stop unwanted access.
- Close the gap between sophisticated cryptography and apps that are easy to use.

Because of their distinct advantages—RSA's capacity for safe key exchanges and AES's effectiveness in data encryption—RSA and AES were chosen to be used in the project.

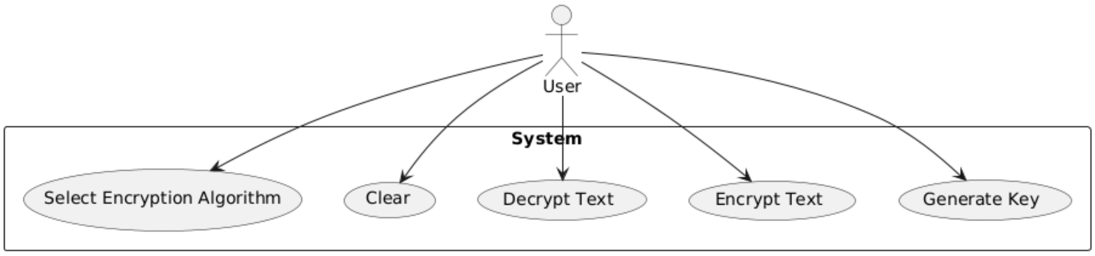


Figure 1.1: Use case diagram of my project

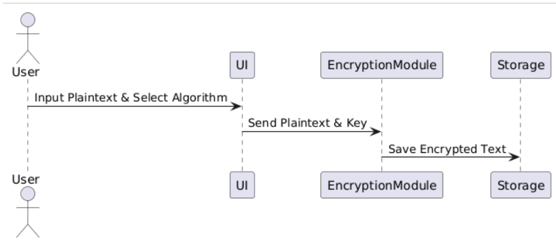


Figure 1.2: Sequence diagram : Encryption Process

## 1.3 Design Goals/Objectives

The following goals are taken into consideration when designing the project:

- Secure Key Management: Make sure that cryptographic keys are generated, stored, and retrieved safely.
- Strong Encryption and Decryption Procedures: To ensure data secrecy, use the RSA and AES algorithms.
- User-Friendly Interface: Make encryption and decryption tasks simple for users to understand.
- Effective Performance: Set up the system to process secure text messages quickly.
- Scalability: Build the system to accommodate future additions of functions or users.
- Compatibility: Guarantee smooth functioning across a range of platforms and gadgets.

## 1.4 Applications

There are numerous uses for the "Secure Text Communication System Using Modern Cryptographic Techniques" across a number of industries, such as:

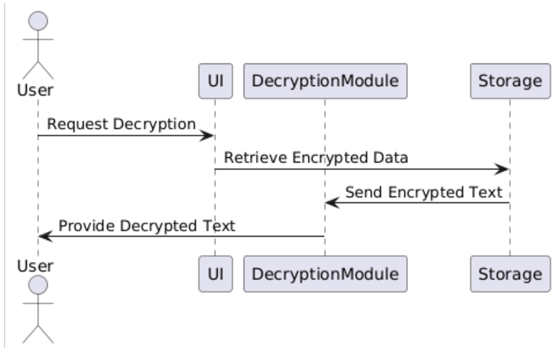


Figure 1.3: Sequence Diagram : Decryption Process

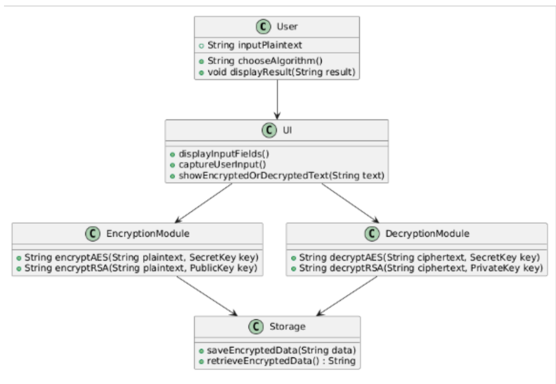


Figure 1.4: Class diagram

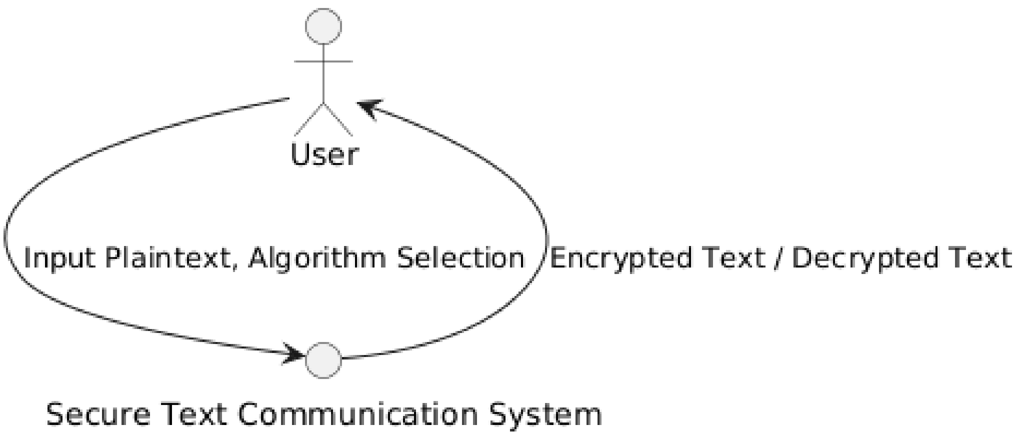


Figure 1.5: Data Flow Diagram level 0

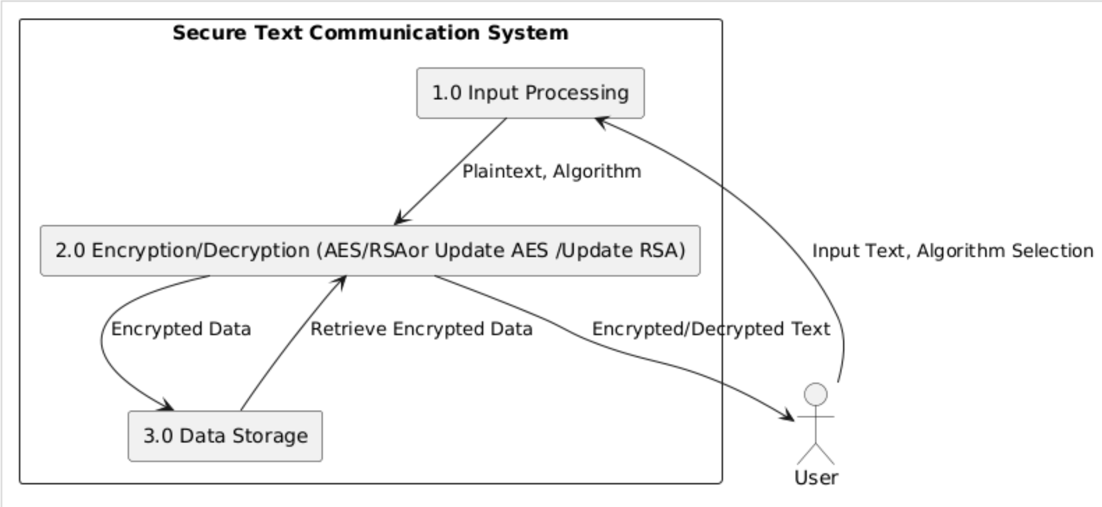


Figure 1.6: Data Flow Diagram Level 1

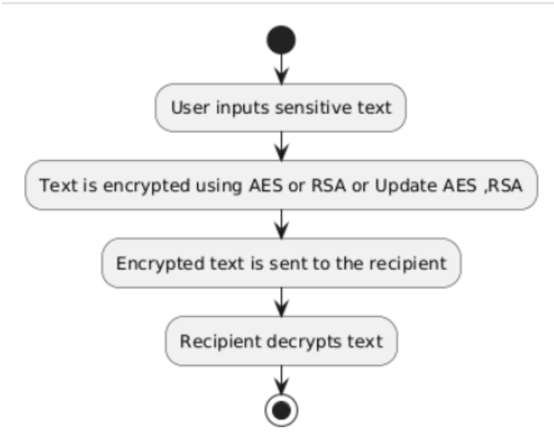


Figure 1.7: Flow Chart for Individual Interaction

### 1.4.1 Individual Interaction

This technology allows users to safely communicate private communications, including financial or personal information.

### 1.4.2 Business Interaction

Companies can use the platform to protect client data, trade secrets, and secure internal communication.

### 1.4.3 Defense and Government

Such systems lower the danger of espionage by enabling governments and defense agencies to securely share classified information.

### 1.4.4 Medical Care

This technology can be used to safely share patient records and other private data in the healthcare industry.

### 1.4.5 Online shopping

This technology can be used by e-commerce platforms to safeguard consumer information, including transaction histories and payment information.

# Chapter 2

# Design/Development/Implementation of the Project

## 2.1 Introduction

In order to guarantee the "Secure Text Communication System Using Modern Cryptographic Techniques"' dependability and effectiveness, it must be carefully planned, implemented, and tested. This project offers a safe and intuitive communication platform by incorporating contemporary encryption algorithms. Strong security and optimal performance are achieved by combining the RSA and AES algorithms. The Advanced Encryption Standard (AES) has been updated in this implementation to use AES-GCM (Galois/Counter Mode).The RSA implementation has been updated to strengthen encryption and decryption mechanisms and improve overall key management.

## 2.2 Project Details

Below is a full description of the system's design and development process, with pertinent subsections highlighting important elements.

### 2.2.1 System Architecture

The project architecture incorporates multiple layers, each responsible for specific functionalities, ensuring modularity and ease of maintenance. Below is the block diagram illustrating the system's architecture:

### 2.2.2 The Process of Encryption

The AES or RSA algorithm is used in the encryption process to secure the plaintext. Prior to being stored or sent, the data is encrypted using the chosen algorithm.

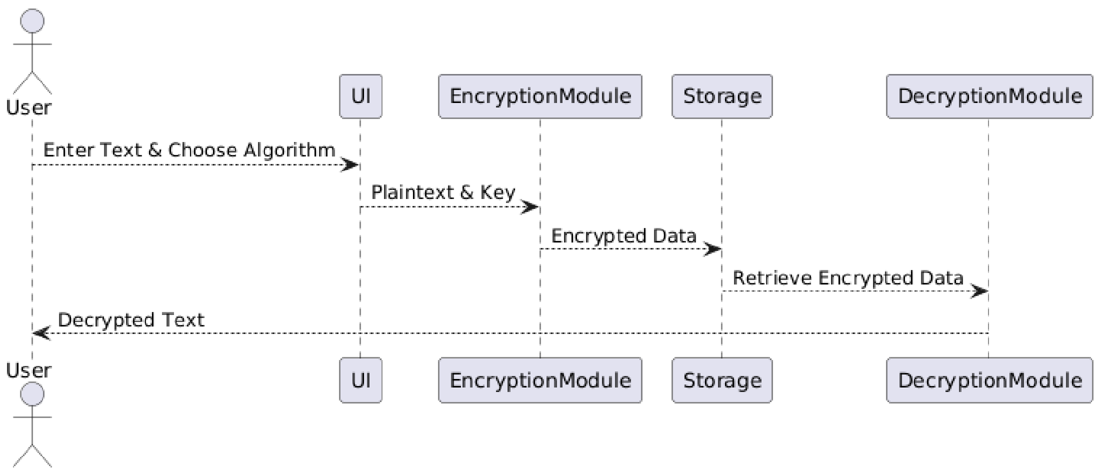


Figure 2.1: System Architecture

### 2.2.3 The Procedure for Decryption

Using the proper keys, the decryption procedure extracts the original plaintext from the ciphertext. In order to ensure secure data recovery, RSA decryption uses the privatekey.

## 2.3 Implementation

Coding the encryption and decryption features, including user interfaces, and conducting security and performance tests are all part of the implementation process.

#### The workflow

- The following is a description of the system's workflow:
- Input handling: To encrypt data, users supply plaintext.
- Key Generation: AES creates a symmetric key, while RSA creates public and private keys.
- Encryption: The chosen algorithm is used to encrypt the text.
- Decryption: The appropriate keys are used to decrypt the encrypted text.

#### Tools and libraries

- Java: The implementation language for core programming.
- Java Cryptography Architecture (JCA): For implementations of RSA and AES.
- JavaFX: Used to construct the user interface.

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.mycompany.securetext;
import javax.crypto.SecretKey;
import javax.swing.*;
import java.awt.*;
import java.security.*;
import java.util.Base64;

public class SecureCommunicationApp {

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new CipherApp().createAndShowGUI());
    }
}

class CipherApp {
    private JTextField txtPlaintext, txtCiphertext, txtKey, txtPublicKey, txtPrivateKey;
    private JComboBox<String> encryptionMethod;
    private SecretKey currentAESKey;
    private PublicKey currentRSAPublicKey;
    private PrivateKey currentRSAPrivateKey;

    public void createAndShowGUI() {
        JFrame frame = new JFrame(title: "Secure Text Communication System");
```

Figure 2.2: SecureCommunicationApp

**Implementation details (with screenshots and programming codes)**

## 2.3.1 Code

### SecureCommunicationApp

First main class - SecureCommunicationApp

### AES and AES update class

Here is the figure 2.12 , figure 2.13 ,figure 2.13

### RSA and RSA update class

Here is the figure 2.14 , figure 2.15 , figure 2.16.

## 2.3.2 Output

```java
        frame.setDefaultCloseOperation(operation: JFrame.EXIT_ON_CLOSE);
        frame.setSize(width: 600, height: 500); // Adjusted window size to fit new fields

        // Main panel with modern color background
        JPanel panel = new JPanel(new GridBagLayout());
        panel.setBackground(new Color(r: 240, g: 248, b: 255)); // Light pastel blue background

        // Font settings for labels and fields
        Font labelFont = new Font(name: "Segoe UI", style: Font.BOLD, size: 14);
        Font inputFont = new Font(name: "Segoe UI", style: Font.PLAIN, size: 14);

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(top:10, left: 10, bottom: 10, right: 10);
        gbc.fill = GridBagConstraints.HORIZONTAL;

        JLabel lblMethod = new JLabel(text: "Select Encryption Method:");
        lblMethod.setFont(font: labelFont);
        gbc.gridx = 0;
        gbc.gridy = 0;
        panel.add(comp: lblMethod, constraints:gbc);

        encryptionMethod = new JComboBox<>(new String[]{"AES", "RSA", "Update AES", "Update RSA"});
        encryptionMethod.setFont(font: inputFont);
        gbc.gridx = 1;
        panel.add(comp: encryptionMethod, constraints:gbc);
        JLabel lblPlaintext = new JLabel(text: "Plaintext:");
```

Figure 2.3: SecureCommunicationApp

```java
        lblPlaintext.setFont(font: labelFont);
        gbc.gridx = 0;
        gbc.gridy = 1;
        panel.add(comp: lblPlaintext, constraints:gbc);

        txtPlaintext = new JTextField();
        txtPlaintext.setFont(f: inputFont);
        txtPlaintext.setBackground(new Color(r: 255, g: 255, b: 255))
        gbc.gridx = 1;
        panel.add(comp: txtPlaintext, constraints:gbc);

        JLabel lblKey = new JLabel(text: "Encryption Key:");
        lblKey.setFont(font: labelFont);
        gbc.gridx = 0;
        gbc.gridy = 2;
        panel.add(comp: lblKey, constraints:gbc);

        txtKey = new JTextField();
        txtKey.setFont(f: inputFont);
        txtKey.setBackground(new Color(r: 245, g: 245, b: 245)); // I
        txtKey.setEditable(b: false);
        gbc.gridx = 1;
        panel.add(comp: txtKey, constraints:gbc);

        JLabel lblPublicKey = new JLabel(text: "Public Key:");
        lblPublicKey.setFont(font: labelFont);
```

Figure 2.4: SecureCommunicationApp

```java
lblPublicKey.setFont(font: labelFont);
gbc.gridx = 0;
gbc.gridy = 3;
panel.add(comp: lblPublicKey, constraints:gbc);

txtPublicKey = new JTextField();
txtPublicKey.setFont(f: inputFont);
txtPublicKey.setBackground(new Color(r: 245, g: 245, b: 245));
txtPublicKey.setEditable(b: false);
gbc.gridx = 1;
panel.add(comp: txtPublicKey, constraints:gbc);

JLabel lblPrivateKey = new JLabel(text: "Private Key:");
lblPrivateKey.setFont(font: labelFont);
gbc.gridx = 0;
gbc.gridy = 4;
panel.add(comp: lblPrivateKey, constraints:gbc);

txtPrivateKey = new JTextField();
txtPrivateKey.setFont(f: inputFont);
txtPrivateKey.setBackground(new Color(r: 245, g: 245, b: 245));
txtPrivateKey.setEditable(b: false);
gbc.gridx = 1;
panel.add(comp: txtPrivateKey, constraints:gbc);

JLabel lblCiphertext = new JLabel(text: "Ciphertext:");
```

Figure 2.5: SecureCommunicationApp

```java
        lblCiphertext.setFont(font: labelFont);
        gbc.gridx = 0;
        gbc.gridy = 5;
        panel.add(comp: lblCiphertext, constraints:gbc);

        txtCiphertext = new JTextField();
        txtCiphertext.setFont(f: inputFont);
        txtCiphertext.setBackground(new Color(r: 255, g: 255, b: 255));
        txtCiphertext.setEditable(b: false);
        gbc.gridx = 1;
        panel.add(comp: txtCiphertext, constraints:gbc);

        // Button styles
        JButton btnGenerateKey = new JButton(text: "Generate Key");
        styleButton(button: btnGenerateKey);
        JButton btnEncrypt = new JButton(text: "Encrypt");
        styleButton(button: btnEncrypt);
        JButton btnDecrypt = new JButton(text: "Decrypt");
        styleButton(button: btnDecrypt);
        JButton btnClear = new JButton(text: "Clear");
        styleButton(button: btnClear);

        // Add buttons with grid constraints
        gbc.gridx = 0;
        gbc.gridy = 6;
        panel.add(comp: btnGenerateKey, constraints:gbc);
```

Figure 2.6: SecureCommunicationApp

```java
        gbc.gridx = 1;
        panel.add(comp: btnEncrypt, constraints:gbc);
        gbc.gridx = 0;
        gbc.gridy = 7;
        panel.add(comp: btnDecrypt, constraints:gbc);
        gbc.gridx = 1;
        panel.add(comp: btnClear, constraints:gbc);

        // Add action listeners to buttons
        btnGenerateKey.addActionListener(e -> generateKeyAction());
        btnEncrypt.addActionListener(e -> encryptAction());
        btnDecrypt.addActionListener(e -> decryptAction());
        btnClear.addActionListener(e -> clearFields());

        frame.add(comp: panel);
        frame.setVisible(b: true);
    }

    // Button style helper
    private void styleButton(JButton button) {
        button.setFont(new Font(name: "Segoe UI", style: Font.PLAIN, size: 14));
        button.setBackground(new Color(r: 70, g: 130, b: 180)); // Steel blue background
        button.setForeground(fg: Color.WHITE); // White text color
        button.setFocusPainted(b: false); // Remove button focus border
    }

    private void generateKeyAction() {
```

Figure 2.7: SecureCommunicationApp

```java
        try {
            String encryptionMethodSelected = (String) encryptionMethod.getSelectedItem();
            if ("AES".equals(anObject: encryptionMethodSelected)) {
                currentAESKey = AESUtils.generateAESKey();
                txtKey.setText(t: Base64.getEncoder().encodeToString(src:currentAESKey.getEncoded()));
                txtPublicKey.setText(t: "");  // Clear Public Key
                txtPrivateKey.setText(t: "");  // Clear Private Key
            } else if ("RSA".equals(anObject: encryptionMethodSelected)) {
                KeyPair rsaKeys = RSAUtils.generateRSAKeys();
                currentRSAPublicKey = rsaKeys.getPublic();
                currentRSAPrivateKey = rsaKeys.getPrivate();
                txtKey.setText(t: "");  // Clear Encryption Key
                txtPublicKey.setText("Public Key: " + Base64.getEncoder().encodeToString(src:currentRSAPublicKey.getEncoded()));
                txtPrivateKey.setText("Private Key: " + Base64.getEncoder().encodeToString(src:currentRSAPrivateKey.getEncoded()));
            } else if ("Update AES".equals(anObject: encryptionMethodSelected)) {
                currentAESKey = AESUtils.generateAESKey();
                txtKey.setText(t: Base64.getEncoder().encodeToString(src:currentAESKey.getEncoded()));
                txtPublicKey.setText(t: "");  // Clear Public Key
                txtPrivateKey.setText(t: "");  // Clear Private Key
            } else if ("Update RSA".equals(anObject: encryptionMethodSelected)) {
                KeyPair rsaKeys = RSAUtils.generateRSAKeys();
                currentRSAPublicKey = rsaKeys.getPublic();
                currentRSAPrivateKey = rsaKeys.getPrivate();
                txtKey.setText(t: "");  // Clear Encryption Key
                txtPublicKey.setText("Public Key: " + Base64.getEncoder().encodeToString(src:currentRSAPublicKey.getEncoded()));
                txtPrivateKey.setText("Private Key: " + Base64.getEncoder().encodeToString(src:currentRSAPrivateKey.getEncoded()));
            }
```

Figure 2.8: SecureCommunicationApp

```java
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(parentComponent:null, "Key generation failed: " + e.getMessage());
        }
    }

    private void encryptAction() {
        try {
            String plaintext = txtPlaintext.getText();
            String encryptionMethodSelected = (String) encryptionMethod.getSelectedItem();
            if (plaintext.isEmpty()) {
                JOptionPane.showMessageDialog(parentComponent:null, message:"Please enter plaintext.");
                return;
            }

            if ("AES".equals(anObject: encryptionMethodSelected)) {
                String ciphertext = AESUtils.encryptAES(plaintext, secretKey: currentAESKey);
                txtCiphertext.setText(t: ciphertext);
            } else if ("RSA".equals(anObject: encryptionMethodSelected)) {
                String ciphertext = RSAUtils.encryptRSA(plaintext, publicKey: currentRSAPublicKey);
                txtCiphertext.setText(t: ciphertext);
            } else if ("Update AES".equals(anObject: encryptionMethodSelected)) {
                String ciphertext = AESUtils.encryptAESGCM(plaintext, secretKey: currentAESKey);
                txtCiphertext.setText(t: ciphertext);
            } else if ("Update RSA".equals(anObject: encryptionMethodSelected)) {
                String ciphertext = RSAUtils.encryptRSAWithUpdatedKey(plaintext, publicKey: currentRSAPublicKey);
                txtCiphertext.setText(t: ciphertext);
```

Figure 2.9: SecureCommunicationApp

```java
                txtCiphertext.setText(t: ciphertext);
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(parentComponent:null, "Encryption failed: " + e.getMessage());
        }
    }

    private void decryptAction() {
        try {
            String ciphertext = txtCiphertext.getText();
            String encryptionMethodSelected = (String) encryptionMethod.getSelectedItem();

            if (ciphertext.isEmpty()) {
                JOptionPane.showMessageDialog(parentComponent:null, message:"Please enter ciphertext.");
                return;
            }

            if ("AES".equals(anObject: encryptionMethodSelected)) {
                String decryptedText = AESUtils.decryptAES(ciphertext, secretKey: currentAESKey);
                txtPlaintext.setText(t: decryptedText);
            } else if ("RSA".equals(anObject: encryptionMethodSelected)) {
                if (currentRSAPrivateKey == null) {
                    JOptionPane.showMessageDialog(parentComponent:null, message:"Private Key is missing for RSA decryption.");
                    return;
                }
                String decryptedText = RSAUtils.decryptRSA(ciphertext, privateKey: currentRSAPrivateKey);
```

Figure 2.10: SecureCommunicationApp

```java
                txtPlaintext.setText(t: decryptedText);
            } else if ("Update AES".equals(anObject: encryptionMethodSelected)) {
                String decryptedText = AESUtils.decryptAESGCM(ciphertext, secretKey: currentAESKey);
                txtPlaintext.setText(t: decryptedText);
            } else if ("Update RSA".equals(anObject: encryptionMethodSelected)) {
                if (currentRSAPrivateKey == null) {
                    JOptionPane.showMessageDialog(parentComponent:null, message:"Private Key is missing for RSA decryption.");
                    return;
                }
                String decryptedText = RSAUtils.decryptRSAWithUpdatedKey(ciphertext, privateKey: currentRSAPrivateKey);
                txtPlaintext.setText(t: decryptedText);
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(parentComponent:null, "Decryption failed: " + e.getMessage());
        }
    }

    private void clearFields() {
        txtPlaintext.setText(t: "");
        txtCiphertext.setText(t: "");
        txtKey.setText(t: "");
        txtPublicKey.setText(t: "");
        txtPrivateKey.setText(t: "");
    }
}
```

Figure 2.11: SecureCommunicationApp

```java
package com.mycompany.securetext;

/**
 *
 * @author Mehrun Nesa Enta
 */


import javax.crypto.*;
import javax.crypto.spec.GCMParameterSpec;
import java.security.*;
import java.util.Base64;

public class AESUtils {

    // Generate AES key
    public static SecretKey generateAESKey() throws NoSuchAlgorithmException {
        KeyGenerator keyGenerator = KeyGenerator.getInstance(algorithm: "AES");
        keyGenerator.init(keysize:256);  // 256-bit AES key
        return keyGenerator.generateKey();
    }

    // AES Encryption (CBC mode)
    public static String encryptAES(String plaintext, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance(transformation: "AES");
        cipher.init(opmode: Cipher.ENCRYPT_MODE, key:secretKey);
        byte[] encryptedBytes = cipher.doFinal(input: plaintext.getBytes());
        return Base64.getEncoder().encodeToString(src:encryptedBytes);
    }
```

Figure 2.12: AES and AES update

```java
    public static String decryptAES(String ciphertext, SecretKey secretKey) throws Exception {
        byte[] cipherBytes = Base64.getDecoder().decode(src:ciphertext);
        Cipher cipher = Cipher.getInstance(transformation: "AES");
        cipher.init(opmode: Cipher.DECRYPT_MODE, key:secretKey);
        byte[] decryptedBytes = cipher.doFinal(input: cipherBytes);
        return new String(bytes: decryptedBytes);
    }

    // AES Encryption (GCM mode)
    public static String encryptAESGCM(String plaintext, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance(transformation: "AES/GCM/NoPadding");
        byte[] iv = new byte[12];  // 12 bytes for GCM IV
        SecureRandom random = new SecureRandom();
        random.nextBytes(bytes: iv);
        GCMParameterSpec spec = new GCMParameterSpec(tLen: 128, src:iv);
        cipher.init(opmode: Cipher.ENCRYPT_MODE, key:secretKey, params: spec);
        byte[] encryptedBytes = cipher.doFinal(input: plaintext.getBytes());
        byte[] ivAndEncrypted = new byte[iv.length + encryptedBytes.length];
        System.arraycopy(src:iv, srcPos: 0, dest: ivAndEncrypted, destPos:0, length: iv.length);
        System.arraycopy(src:encryptedBytes, srcPos: 0, dest: ivAndEncrypted, destPos:iv.length, length: encryptedBytes.length);
        return Base64.getEncoder().encodeToString(src:ivAndEncrypted);
    }

    // AES Decryption (GCM mode)
    public static String decryptAESGCM(String ciphertext, SecretKey secretKey) throws Exception {
        byte[] ivAndCiphertext = Base64.getDecoder().decode(src:ciphertext);
        byte[] iv = new byte[12];  // 12 bytes for GCM IV
        System.arraycopy(src:ivAndCiphertext, srcPos: 0, dest: iv, destPos:0, length: iv.length);
        byte[] cipherBytes = new byte[ivAndCiphertext.length - iv.length];
```

Figure 2.13: AES and AES update

```java
        System.arraycopy(src:encryptedBytes, srcPos: 0, dest: ivAndEncrypted, destPos:iv.length, length: encryptedBytes.length);
        return Base64.getEncoder().encodeToString(src:ivAndEncrypted);
    }

    // AES Decryption (GCM mode)
    public static String decryptAESGCM(String ciphertext, SecretKey secretKey) throws Exception {
        byte[] ivAndCiphertext = Base64.getDecoder().decode(src:ciphertext);
        byte[] iv = new byte[12];  // 12 bytes for GCM IV
        System.arraycopy(src:ivAndCiphertext, srcPos: 0, dest: iv, destPos:0, length: iv.length);
        byte[] cipherBytes = new byte[ivAndCiphertext.length - iv.length];
        System.arraycopy(src:ivAndCiphertext, srcPos: iv.length, dest: cipherBytes, destPos:0, length: cipherBytes.length);

        Cipher cipher = Cipher.getInstance(transformation: "AES/GCM/NoPadding");
        GCMParameterSpec spec = new GCMParameterSpec(tLen: 128, src:iv);
        cipher.init(opmode: Cipher.DECRYPT_MODE, key:secretKey, params: spec);
        byte[] decryptedBytes = cipher.doFinal(input: cipherBytes);
        return new String(bytes: decryptedBytes);
    }
}
```

Figure 2.14: AES and AES update

```java
package com.mycompany.securetext;

import java.security.*;
import java.util.Base64;
import javax.crypto.Cipher;

public class RSAUtils {

    // Generate RSA key pair
    public static KeyPair generateRSAKeys() throws Exception {
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(algorithm: "RSA");
        keyPairGenerator.initialize(keysize:2048);
        return keyPairGenerator.generateKeyPair();
    }

    // RSA Encryption with Public Key
    public static String encryptRSA(String plaintext, PublicKey publicKey) throws Exception {
        Cipher cipher = Cipher.getInstance(transformation: "RSA");
        cipher.init(opmode: Cipher.ENCRYPT_MODE, key:publicKey);
        byte[] encryptedBytes = cipher.doFinal(input: plaintext.getBytes());
        return Base64.getEncoder().encodeToString(src:encryptedBytes);
    }

    // RSA Decryption with Private Key
    public static String decryptRSA(String ciphertext, PrivateKey privateKey) throws Exception {
        byte[] cipherBytes = Base64.getDecoder().decode(src:ciphertext);
        Cipher cipher = Cipher.getInstance(transformation: "RSA");
```

Figure 2.15: RSA and RSA update

```java
        Cipher cipher = Cipher.getInstance(transformation: "RSA");
        cipher.init(opmode: Cipher.DECRYPT_MODE, key:privateKey);
        byte[] decryptedBytes = cipher.doFinal(input: cipherBytes);
        return new String(bytes: decryptedBytes);
    }

    // RSA Encryption with Updated Key (using Public Key for Encryption)
    public static String encryptRSAWithUpdatedKey(String plaintext, PublicKey publicKey) throws Exception {
        Cipher cipher = Cipher.getInstance(transformation: "RSA");
        cipher.init(opmode: Cipher.ENCRYPT_MODE, key:publicKey);
        byte[] encryptedBytes = cipher.doFinal(input: plaintext.getBytes());
        return Base64.getEncoder().encodeToString(src:encryptedBytes);
    }

    // RSA Decryption with Updated Key (using Private Key for Decryption)
    public static String decryptRSAWithUpdatedKey(String ciphertext, PrivateKey privateKey) throws Exception {
        byte[] cipherBytes = Base64.getDecoder().decode(src:ciphertext);
        Cipher cipher = Cipher.getInstance(transformation: "RSA");
        cipher.init(opmode: Cipher.DECRYPT_MODE, key:privateKey);
        byte[] decryptedBytes = cipher.doFinal(input: cipherBytes);
        return new String(bytes: decryptedBytes);
    }
}
```

Figure 2.16: RSA and RSA update

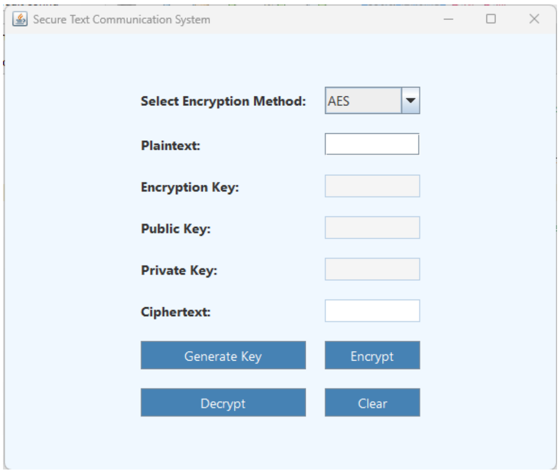


Figure 2.17: Output Show Interface

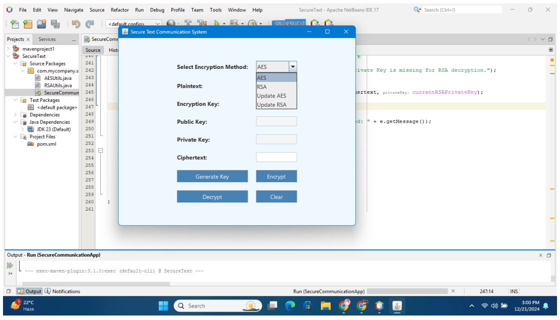

Figure 2.18: Which algorithm needs

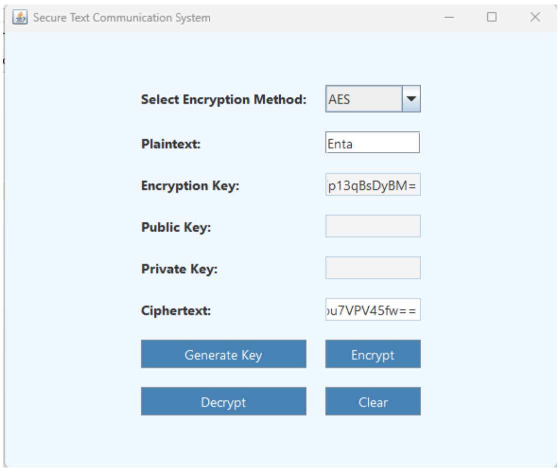


Figure 2.19: AES

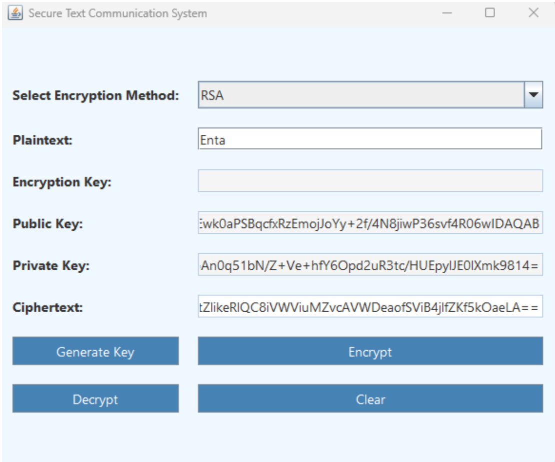


Figure 2.20: RSA

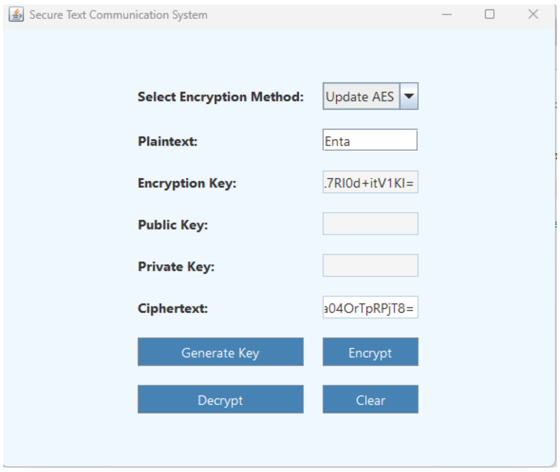


Figure 2.21: Update AES

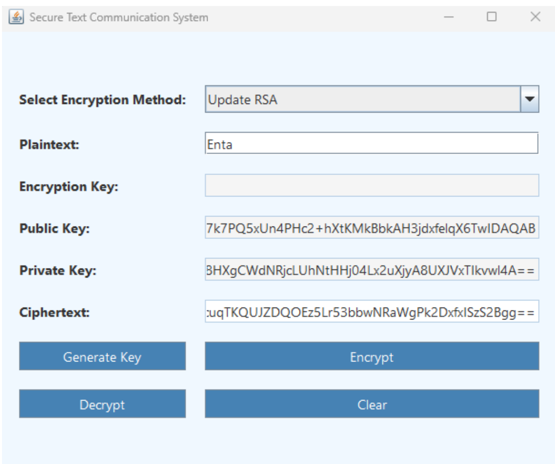


Figure 2.22: Update RSA

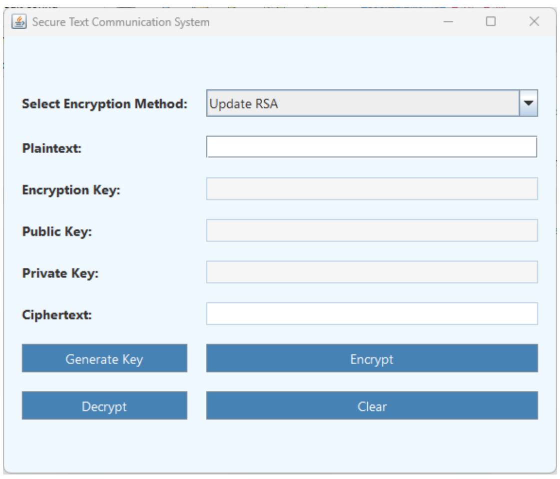


Figure 2.23: Clear Button

# Chapter 3

# Conclusion

## 3.1 Discussion

This chapter provides a comprehensive overview of the design, development, and implementation of the"Secure Text Communication System Using Modern Cryptographic Techniques." It describes the design of the system in depth, including the AES and RSA algorithms used for modular encryption and decryption. The Java-based encryption and decryption code, the incorporation of an intuitive user interface, and security testing to guarantee resilience are highlighted in the implementation section. Using cutting-edge cryptographic algorithms, the system enables safe and effective communication. Detailed sequence diagrams and application screenshots demonstrate its operation. The outcomes confirm the system's dependability and compliance with security guidelines.

# References


1. M. A. Khan and H. A. Hashmi, "Implementation of secure communication using RSA and AES encryption techniques," International Journal of Computer Applications, vol. 183, no. 33, pp. 10–15, 2021.

2. W. Stallings, Cryptography and Network Security: Principles and Practice, 8th ed. Upper Saddle River, NJ, USA: Pearson, 2020.

3. P. Rogaway and T. Shrimpton, "Cryptographic authentication of replays," in Proceedings of the 2006 ACM Conference on Computer and Communications Security (CCS '06), Alexandria, VA, USA, Oct. 2006, pp. 111–120.

4. N. Ferguson and B. Schneier, Practical Cryptography, 1st ed. Indianapolis, IN, USA: Wiley, 2003.