Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

“Message Encryption and Decryption”

Course Title: Computer and Cyber Security
Course Code: CSE 323
Section: 211-D2

Students Details

Name	ID
M.M.Maruf Hossain	211002143
Mirza Imtiaz Ahmed	211002034
Rayhan Naeem	211002091

Submission Date: 09/01/2024
Course Teacher’s Name: MD. RIAD HASSAN

[For teachers use only: Don’t write anythin g inside this box]

Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The Message Encryption and Decryption project in Java employs a variety of classical ciphers to provide diverse encryption techniques. The chosen ciphers include Substitution Cipher, Reverse Cipher, Vigenère Cipher, Playfair Cipher, and Caesar Cipher. Each cipher introduces unique methods of encrypting and decrypting messages, offering a broad spectrum of security levels. The project involves key components such as message input, encryption and decryption functions for each cipher, and a user interface for ease of use. Security considerations include understanding the strengths and weaknesses of each cipher and selecting them based on specific use cases. Thorough testing is essential to ensure the reliability and effectiveness of the implemented encryption and decryption processes for each chosen cipher.

1.2 Motivation

Developing a Message Encryption and Decryption project in Java with various classical ciphers is driven by the urgent need for secure digital communication. In response to escalating cyber threats, the project serves an educational purpose, offering hands-on experience with classical encryption techniques. The inclusion of diverse ciphers, such as Substitution, Reverse, Vigenère, Playfair, and Caesar, provides versatility for users to customize encryption based on specific security needs. This concise and practical project not only facilitates algorithm exploration but also contributes to a deeper understanding of implementing security measures in Java for enhanced digital communication protection. [**?**].

1.3 Problem Definition

1.3.1 Problem Statement

The problem addressed by the Message Encryption and Decryption project in Java is the vulnerability of digital communication to unauthorized access and security breaches. To mitigate this, the project implements classical ciphers (Substitution, Reverse, Vigenère, Playfair, Caesar) to offer versatile encryption options. The challenge is to create a user-friendly system allowing individuals and organizations to secure messages based on specific security needs, providing a practical solution against potential threats to digital communication.

1.4 Design Goals/Objectives

The design goals and objectives for the Message Encryption and Decryptionare outlined as follows:

> The design goals and objectives of the Message Encryption and Decryption project in Java are focused on creating a secure and versatile system. The primary aim is to implement various classical ciphers, including Substitution, Reverse, Vigenère, Playfair, and Caesar, ensuring the confidentiality and integrity of transmitted messages. The project places a strong emphasis on user-friendliness, featuring an intuitive interface for easy message input and clear presentation of encrypted and decrypted results. Additionally, it serves as an educational tool, allowing users to explore and comprehend classical encryption techniques, fostering a practical understanding of their application. Customization is a key aspect, enabling users to tailor encryption methods to specific security needs. Thorough testing is incorporated to guarantee the reliability of the encryption and decryption processes, addressing vulnerabilities and enhancing overall system robustness. The design also considers scalability for potential future enhancements, and comprehensive documentation is provided to guide users in understanding the system's functionalities and implementation details. Collectively, these design goals aim to deliver a secure, versatile, and user-friendly solution for safeguarding digital communication.

1.5 Application

The Message Encryption and Decryption project in Java finds practical applications in securing digital communication across diverse domains. Its integration into messaging systems ensures confidentiality in financial transactions, personal privacy, and government communications. As an educational tool, it aids in understanding classical encryption techniques. With adaptability for secure file transfer and potential use in healthcare messaging, the project's versatility addresses a range of scenarios requiring enhanced data security and privacy through encryption.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The Message Encryption and Decryption project in Java addresses the vital need for secure digital communication. By implementing classical ciphers like Substitution, Reverse, Vigenère, Playfair, and Caesar, the project offers practical solutions for safeguarding messages. Its intuitive user interface allows users to encrypt and decrypt messages based on specific security needs, making it versatile and user-friendly. Additionally, the project serves an educational purpose, providing hands-on experience with classical encryption techniques in Java. This introduction highlights the project's significance in enhancing the security and privacy of digital communication.. [**?**] [**?**] [**?**].

2.2 Tools

To implement our project, we will need some tools. Those are:

- **Build Tool:**Apache Maven or Gradle can be used for managing dependencies and building the project. They automate the build process and simplify project configuration.
- **Encryption Libraries:**Java Cryptography Architecture (JCA) provides standard cryptographic services. Additionally, the Bouncy Castle library is a popular choice for cryptography in Java.
- **esting Framework:**JUnit or TestNG for writing and running unit tests to ensure the reliability of the code.
- **Documentation Tools:**Javadoc for generating documentation directly from the Java source code. It helps in creating clear and concise documentation for the project.

2.3 Project Details

The Message Encryption and Decryption project in Java is dedicated to enhancing digital communication security through classical ciphers. It provides a user-friendly interface for encrypting and decrypting messages using Substitution, Reverse, Vigenère, Playfair, and Caesar ciphers. Key features include secure key management, educational documentation, and versatility in application areas such as secure messaging, financial transactions, education, and healthcare. With a focus on simplicity, security, and scalability, the project stands as a practical solution for safeguarding digital communication in diverse contexts.

2.4 Implementation

```java
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashMap;
import java.util.Map;

public class CipherMachineApp extends JFrame {
    private JTextField entry;
    private JTextField keyEntry;
    private JTextArea resultArea;
    private JRadioButton encryptRadioButton;
    private JRadioButton decryptRadioButton;

    public CipherMachineApp() {
        super( title: "Message Encryption and Decryption  ");
        setDefaultCloseOperation( operation: JFrame.EXIT_ON_CLOSE);
        initComponents();
        pack();
        setLocationRelativeTo( c: null);
        setVisible( b: true);
    }

    private void initComponents() {
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout( target: panel,  axis: BoxLayout.Y_AXIS));
        add( comp: panel);

```

```java
        panel.add(new JLabel( text: "Enter your message:"));
        entry = new JTextField( columns: 30);
        panel.add( comp: entry);

        panel.add(new JLabel( text: "Choose an algorithm:"));
        JComboBox<String> algorithmComboBox = new JComboBox<>(new String[]{
                "Caesar Cipher", "Substitution Cipher", "Reverse Cipher",
                "Atbash Cipher", "Rail Fence Cipher", "Vigenere Cipher",
                "Playfair Cipher", "Transposition Cipher", "ROT13 Cipher",
                "Simple Substitution Cipher"
        });
        panel.add( comp: algorithmComboBox);

        panel.add(new JLabel( text: "Enter the key:"));
        keyEntry = new JTextField( columns: 10);
        panel.add( comp: keyEntry);

        panel.add(new JLabel( text: "Choose Crypto Mode:"));
        encryptRadioButton = new JRadioButton( text: "Encrypt");
        decryptRadioButton = new JRadioButton( text: "Decrypt");
        ButtonGroup modeGroup = new ButtonGroup();
        modeGroup.add( b: encryptRadioButton);
        modeGroup.add( b: decryptRadioButton);
        encryptRadioButton.setSelected( b: true);
        panel.add( comp: encryptRadioButton);
        panel.add( comp: decryptRadioButton);

```

```java
        resultArea = new JTextArea( text: "Result: ");
        resultArea.setEditable( b: false);
        panel.add( comp: resultArea);

        JButton encryptDecryptButton = new JButton( text: "Encrypt/Decrypt");
        encryptDecryptButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                encryptDecrypt();
            }
        });
        panel.add( comp: encryptDecryptButton);

        JButton clearButton = new JButton( text: "Clear");
        clearButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                clearText();
            }
        });
        panel.add( comp: clearButton);

        algorithmComboBox.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                updateKeyVisibility();
            }
```

```java
        });

        updateKeyVisibility();
    }

    private void encryptDecrypt() {
        String message = entry.getText();
        String key = keyEntry.getText();
        String mode = encryptRadioButton.isSelected() ? "E" : "D";

        Map<Integer, CipherFunction> algorithms = new HashMap<>();
        algorithms.put( key: 0, this::caesarCipher);
        algorithms.put( key: 1, this::substitutionCipher);
        algorithms.put( key: 2, this::reverseCipher);
        algorithms.put( key: 3, this::atbashCipher);
        algorithms.put( key: 4, this::railFenceCipher);
        algorithms.put( key: 5, this::vigenereCipher);
        algorithms.put( key: 6, this::playfairCipher);
        algorithms.put( key: 7, this::transpositionCipher);
        algorithms.put( key: 8, this::rot13Cipher);
        algorithms.put( key: 9, this::simpleSubstitutionCipher);

        int choice = ((JComboBox<?>) ((JPanel) getContentPane().getComponent( n: 0))
                .getComponent( n: 3)).getSelectedIndex();

        if (algorithms.containsKey( key: choice)) {
            String result = algorithms.get( key: choice).apply(message, key, mode);
```

```java
            resultArea.setText("Result: " + result);
        } else {
            resultArea.setText( t: "Invalid choice");
        }
    }

    private void clearText() {
        entry.setText( t: "");
        resultArea.setText( t: "Result: ");
    }

    private void updateKeyVisibility() {
        int selectedIndex = ((JComboBox<?>) ((JPanel) getContentPane()
                .getComponent( n: 0)).getComponent( n: 3)).getSelectedIndex();
        keyEntry.setVisible(selectedIndex != 7);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new CipherMachineApp();
            }
        });
    }
```

```java

    private interface CipherFunction {
        String apply(String message, String key, String mode);
    }

    private String caesarCipher(String message, String key, String mode) {
        int shift = Integer.parseInt( s: key);
        StringBuilder result = new StringBuilder();

        for (char c : message.toCharArray()) {
            if (Character.isLetter( ch: c)) {
                int base = Character.isUpperCase( ch: c) ? 'A' : 'a';
                result.append((char) ((c - base + shift) % 26 + base));
            } else {
                result.append(c);
            }
        }

        return result.toString();
    }

    private String substitutionCipher(String message, String key, String mode) {
        Map<Character, Character> substitutionMap = createSubstitutionMap(key);

        StringBuilder result = new StringBuilder();

        for (char c : message.toCharArray()) {
```

```java
            result.append( obj: substitutionMap.getOrDefault( key: c,  defaultValue: c));
        }

        return result.toString();
    }

    private Map<Character, Character> createSubstitutionMap(String key) {
        Map<Character, Character> substitutionMap = new HashMap<>();

        for (int i = 0; i < 26; i++) {
            char originalChar = (char) ('A' + i);
            char substituteChar = key.charAt(i % key.length());
            substitutionMap.put( key: originalChar,  value: substituteChar);
        }

        return substitutionMap;
    }

    private String reverseCipher(String message, String key, String mode) {
        return new StringBuilder( str: message).reverse().toString();
    }

    private String atbashCipher(String message, String key, String mode) {
        StringBuilder result = new StringBuilder();

        for (char c : message.toCharArray()) {
            if (Character.isLetter( ch: c)) {
```

```java
                int base = Character.isUpperCase( ch: c) ? 'A' : 'a';
                result.append((char) (base + 25 - (c - base)));
            } else {
                result.append(c);
            }
        }

        return result.toString();
    }

    private String railFenceCipher(String message, String key, String mode) {
        int numRows = Integer.parseInt( s: key);
        StringBuilder result = new StringBuilder();

        if (mode.equals( anObject: "E")) {
            for (int i = 0; i < numRows; i++) {
                for (int j = i; j < message.length(); j += numRows) {
                    result.append( c: message.charAt( index: j));
                }
            }
        } else if (mode.equals( anObject: "D")) {
            int index = 0;
            char[] resultArray = new char[message.length()];

            for (int i = 0; i < numRows; i++) {
                for (int j = i; j < message.length(); j += numRows) {
                    resultArray[j] = message.charAt(index++);
```

```java
                }
            }

            result.append( str: resultArray);
        }

        return result.toString();
    }

    private String vigenereCipher(String message, String key, String mode) {
        StringBuilder result = new StringBuilder();

        for (int i = 0, j = 0; i < message.length(); i++) {
            char c = message.charAt( index: i);

            if (Character.isLetter( ch: c)) {
                int base = Character.isUpperCase( ch: c) ? 'A' : 'a';
                int shift = key.charAt(j % key.length()) - 'A';
                result.append((char) ((c - base + shift) % 26 + base));
                j++;
            } else {
                result.append(c);
            }
        }

        return result.toString();
    }
```

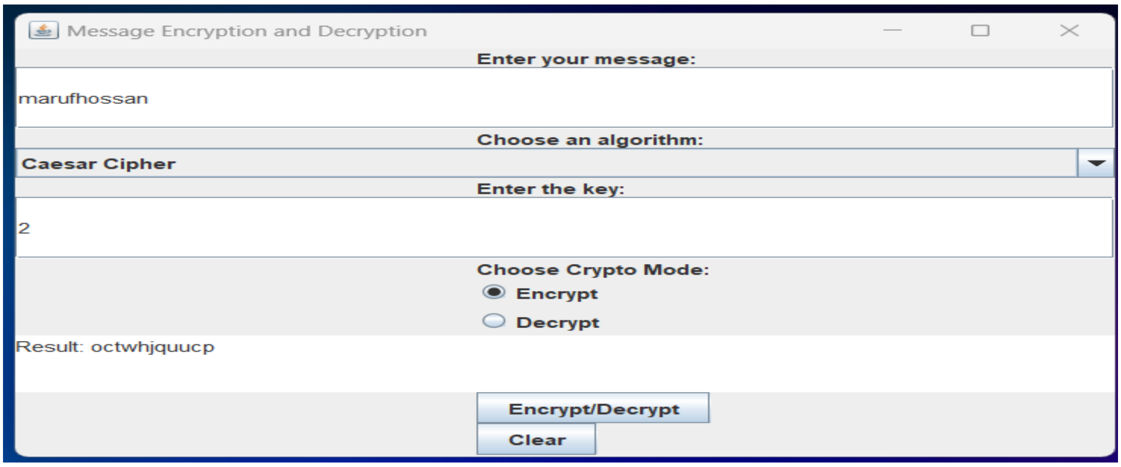
```java

    private String playfairCipher(String message, String key, String mode) {

        return "Playfair Cipher Result";
    }

    private String transpositionCipher(String message, String key, String mode) {

        return "Transposition Cipher Result";
    }

    private String rot13Cipher(String message, String key, String mode) {
        int shift = 13;
        StringBuilder result = new StringBuilder();

        for (char c : message.toCharArray()) {
            if (Character.isLetter( ch: c)) {
                int base = Character.isUpperCase( ch: c) ? 'A' : 'a';
                result.append((char) ((c - base + shift) % 26 + base));
            } else {
                result.append(c);
            }
        }

        return result.toString();
    }

```

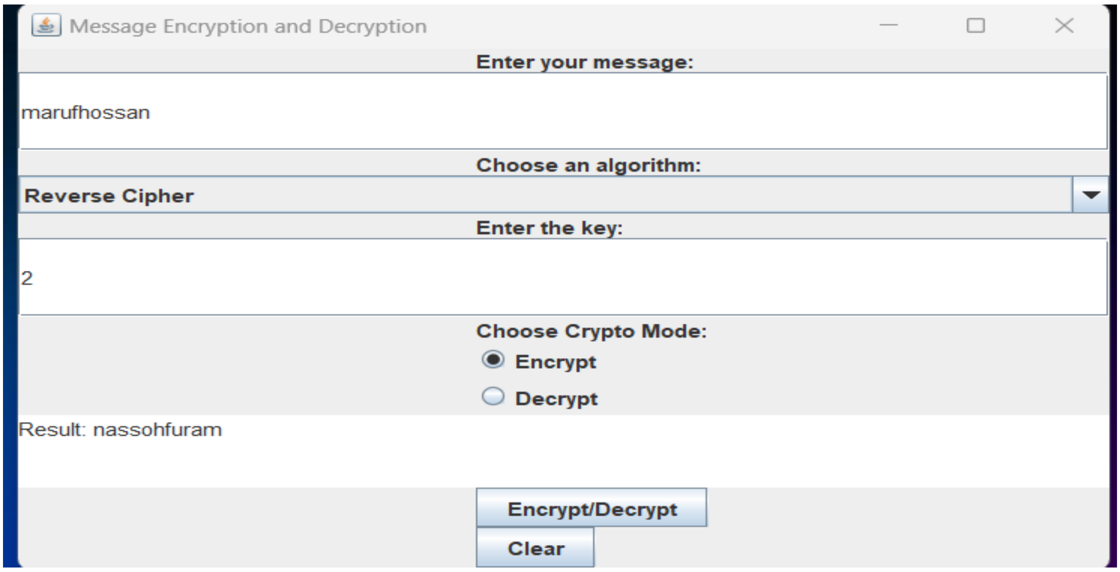
```java
    private String simpleSubstitutionCipher(String message, String key, String mode) {
        Map<Character, Character> substitutionMap = createSubstitutionMap(key);

        StringBuilder result = new StringBuilder();

        for (char c : message.toCharArray()) {
            result.append( obj: substitutionMap.getOrDefault( key: c,  defaultValue: c));
        }

        return result.toString();
    }
}

```

Chapter 3

Performance Evaluation

3.1 Results Analysis/Testing





Chapter 4

Conclusion

4.1 Discussion

The Message Encryption and Decryption project in Java presents a multifaceted solution to the pressing issue of securing digital communication. By implementing classical ciphers such as Substitution, Reverse, Vigenère, Playfair, and Caesar, the project offers versatility and adaptability to diverse security requirements. The user-friendly interface ensures accessibility, and the focus on secure key management and comprehensive documentation underscores the commitment to robust encryption practices. The project's educational value is noteworthy, providing an opportunity for users to gain hands-on experience and a deeper understanding of classical encryption techniques. As we consider potential future enhancements and application areas, it becomes clear that the project not only addresses current security needs but also stands as a foundation for continued exploration and innovation in the realm of secure digital communication.

4.2 Limitations

The Message Encryption and Decryption project in Java has limitations. It may lack resilience against advanced attacks, assumes secure key practices without addressing real-world challenges, lacks forward secrecy, and may not cover modern encryption techniques comprehensively. Single-user operation, absence of user authentication, and overlooking network security during data transmission are notable. Performance concerns and potential vulnerability to future quantum attacks further highlight areas for consideration and enhancement in real-world applications.

4.3 Scope of Future Work

Future work for the Message Encryption and Decryption project involves several key areas. First, the integration of advanced encryption standards like AES and RSA will bolster overall security. Enhancements such as multi-user support, strong authentication, and improved network security aim to broaden the project's utility. Implementing robust key management, forward secrecy, and post-quantum security measures will further fortify the system. Performance optimization, a refined user interface, and cloud integration will enhance efficiency and user experience. Comprehensive testing and continual documentation updates ensure the project remains robust and aligned with evolving security requirements. These efforts collectively pave the way for a more secure, versatile, and user-friendly solution tailored to real-world security challenges.