KINGDOM OF SAUDI ARABIA
MINISTRY OF HIGHER EDUCATION
TAIBAH UNIVERSITY (039)
COLLEGE OF COMPUTER SCIENCE AND ENGINEERING (CCSE)

# RAIL FENCE CIPHER

## CS433 PROJECT

**Computer Security**

Team Members

| | |
|---|---|
| Lama Alfreah | 4250107 |
| Asma Alsharif | 4253618 |
| Lama Alahmadi | 4251733 |
| Lama Alsharif | 4254112 |

Instructor:
Ohood Alrohili

FALL 2024

# 1. INTRODUCTION

The purpose of this project is to provide a simple user interface where users can upload a file, encrypt & decrypt it using the Rail Fence Cipher encryption algorithm, and save the encrypted file. The project is built using Java and JavaFX, allowing users to easily interact with the encryption& decryption tool through a graphical interface.

The project uses the **Rail Fence Cipher**, which is a form of **transposition** cipher. It rearranges the characters of the plaintext based on a given number of “rails”.

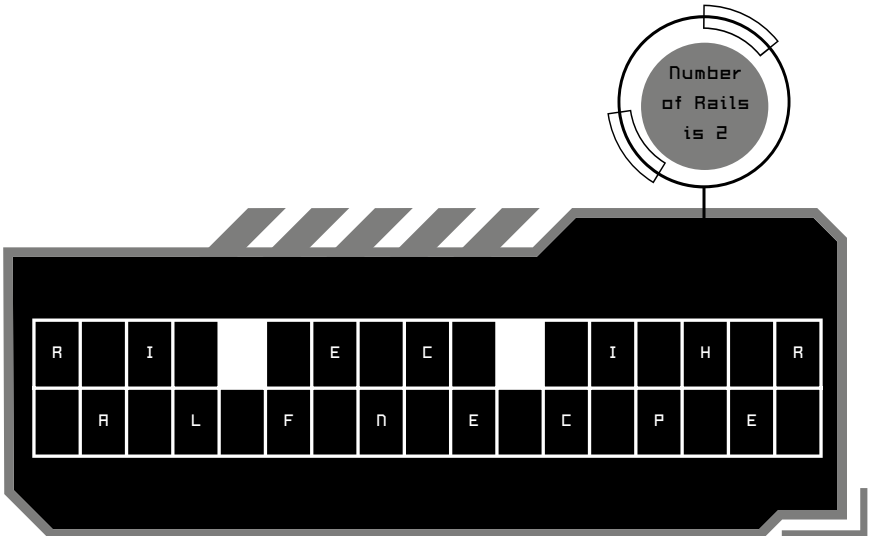
Applying the rail fence technique in a given text results in a zigzag pattern, with each letter in a row written out before moving to the next row.

- To encrypt a message using the rail fence technique, first, we need to write the message in the first row of a table. Furthermore, we have to write the second letter of the message in the second row. We need to continue this process until we’ve written all the letters in the message. Finally, in order to generate the encrypted message, we read the table row-wise.

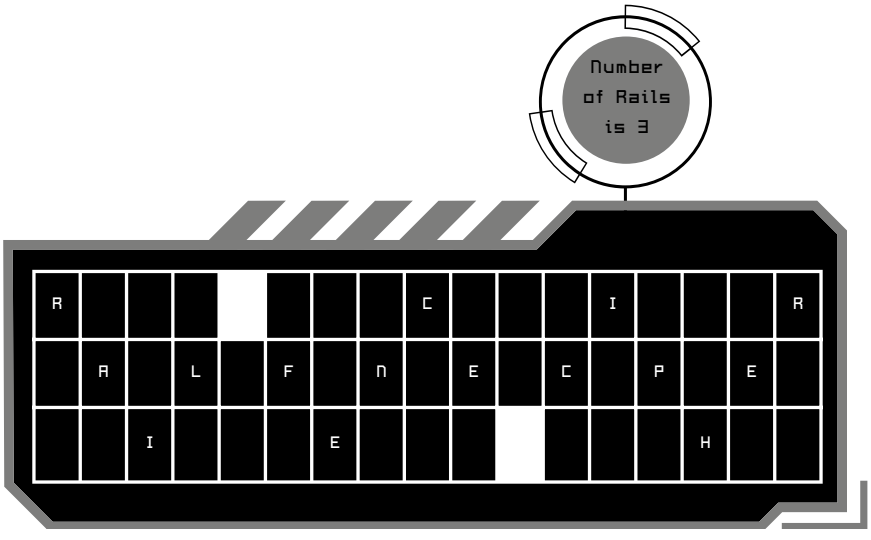
- To decrypt an encrypted message, the first step is to determine the number of rows in the table based on the length of the encrypted message. Furthermore, we need to write the first letter of the encrypted message in the first row, the second letter in the second row, and so on. We need to continue this process until we’ve written all the letters in the message.

# 2. EXAMPLE OF RAIL FENCE CIPHER

Suppose the plaintext is “Rail Fence Cipher ”

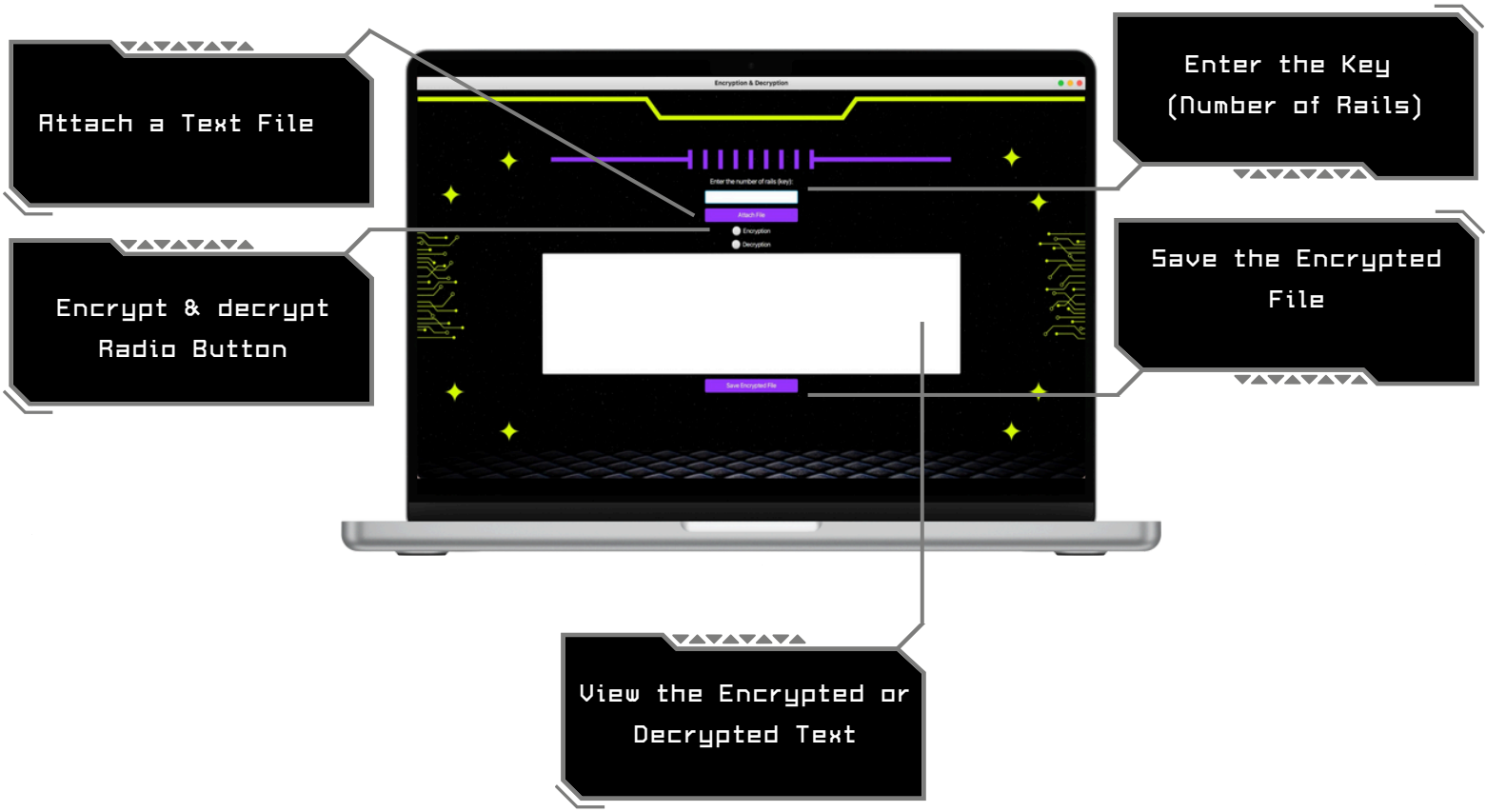



C: RI EC IHRALFNECPE




C: R CIRRLFNECPEIE H

# 3. INSTRUCTIONS FOR USING THE TOOL




- Step 1 : You should attach your file by clicking on ‘ Attach File ‘ button .
- Step 2 : Enter the number of of rails ( key ) in ‘ Enter number of rails ‘ text field , and the number of rails should start from 1 any number out of the range will not be accepted .
- Step 3 : You can choose either encryption or decryption the file .
- Step 4 : The encrypted or decrypted text will be viewed .
- Step 5 : You can save the encrypted or decrypted file by clicking on ‘ save file ‘ button .

ENTER THE KEY (NUMBER OF RAILS):

INPUT THE NUMBER OF RAILS YOU WANT TO USE FOR THE ENCRYPTION OR DECRYPTION. THIS NUMBER DETERMINES THE COMPLEXITY OF THE RAIL FENCE CIPHER.

ENCRYPT & DECRYPT RADIO BUTTON

A RADIO BUTTON THAT THE USER SELECTS IF THEY WANT TO ENCRYPT OR DECRYPT THE INPUT TEXT.

WHEN THIS OPTION IS SELECTED, THE PROGRAM WILL APPLY THE RAIL FENCE CIPHER TO TRANSFORM THE INPUT .

ATTACH A FILE

THIS WILL OPEN A FILE CHOOSER WINDOW WHERE YOU CAN SELECT THE FILE YOU WANT TO ENCRYPT OR DECRYPT. THE FILE CAN BE ANY TEXT-BASED FILE, AND THE CONTENT WILL BE DISPLAYED IN THE OUTPUT AREA AFTER ENCRYPTION OR DECRYPTION.

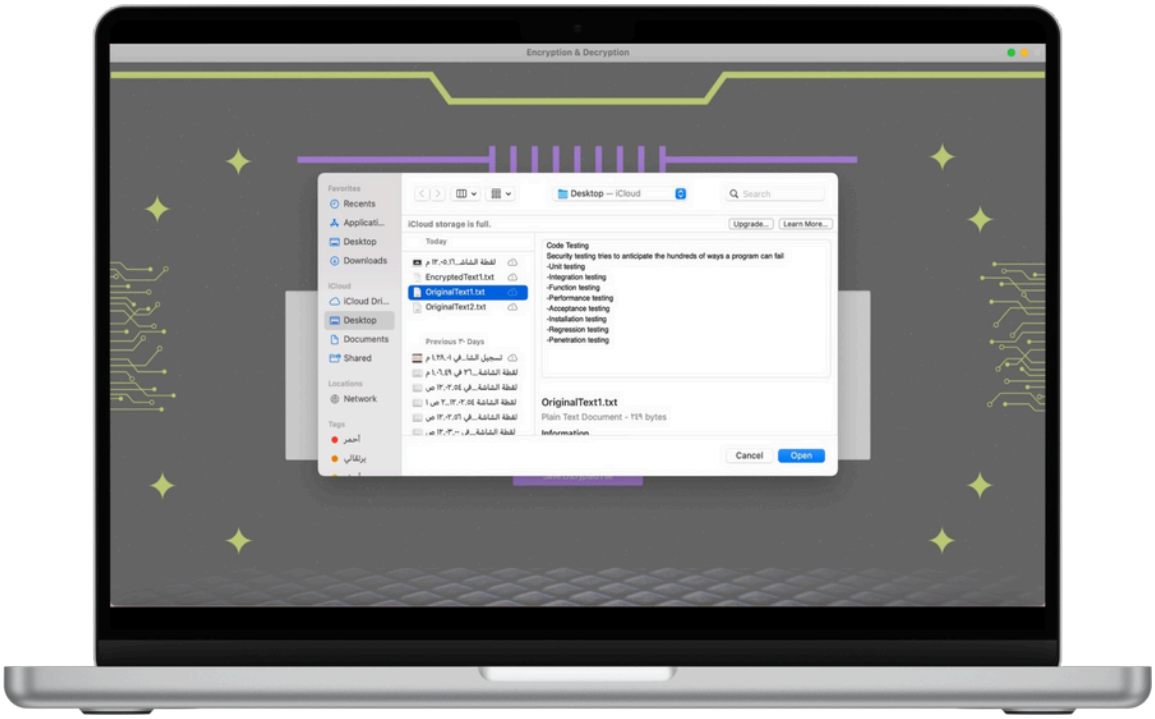
VIEW THE ENCRYPTED TEXT

ONCE THE FILE IS UPLOADED, THE PROGRAM WILL AUTOMATICALLY APPLY THE RAIL FENCE CIPHER BASED ON THE NUMBER OF RAILS PROVIDED. THE ENCRYPTED OR DECRYPTED TEXT WILL APPEAR IN THE OUTPUT AREA.

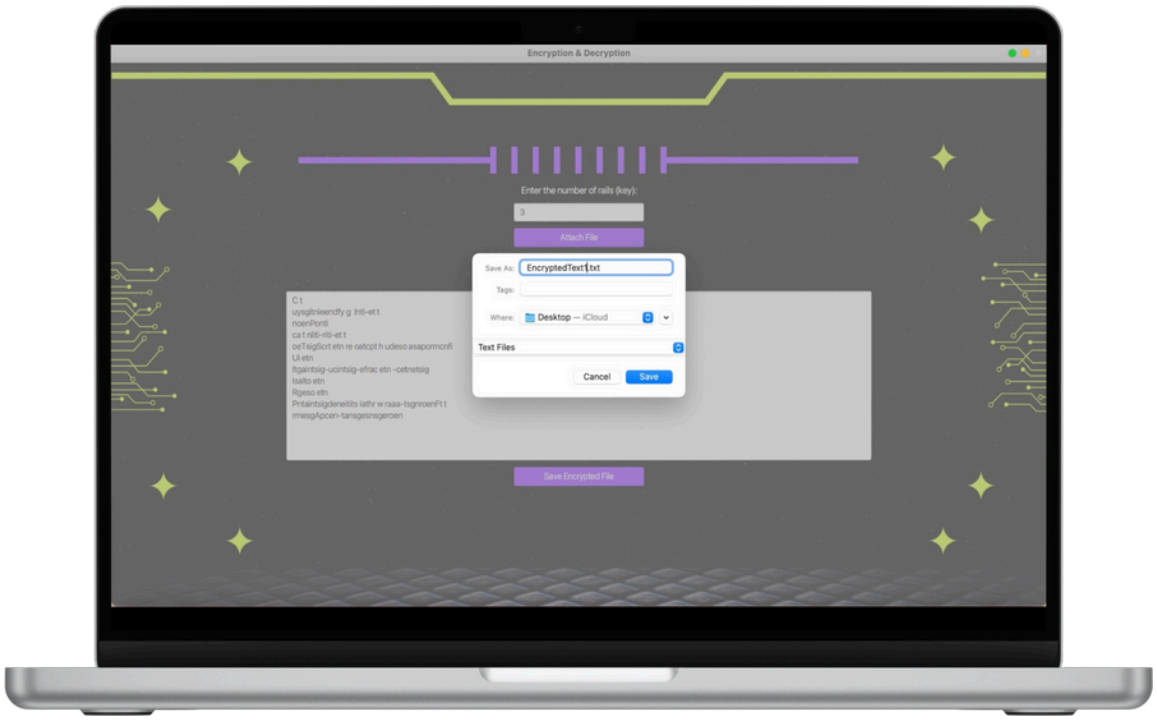
SAVE THE FILE

IF YOU WANT TO SAVE THE ENCRYPTED OR DECRYPTED TEXT TO A FILE, CLICK THE "SAVE FILE" BUTTON. YOU CAN CHOOSE THE LOCATION AND NAME FOR THE ENCRYPTED FILE, AND IT WILL BE SAVED AS A `.TXT` FILE.

# HOW TO ATTACH THE FILE

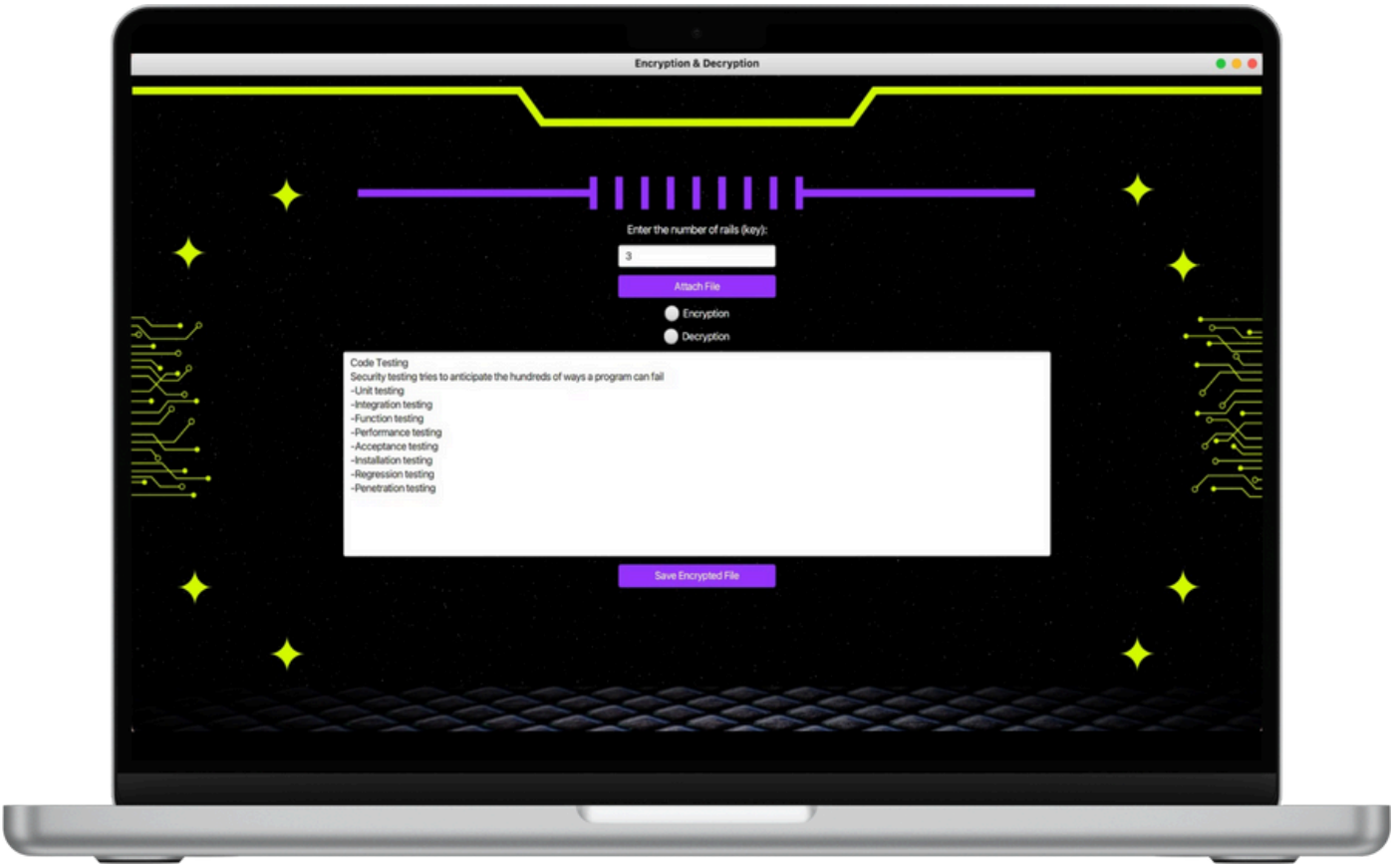



# HOW TO SAVE THE FILE



# 4. EXAMPLES OF ENCRYPTING TEXT

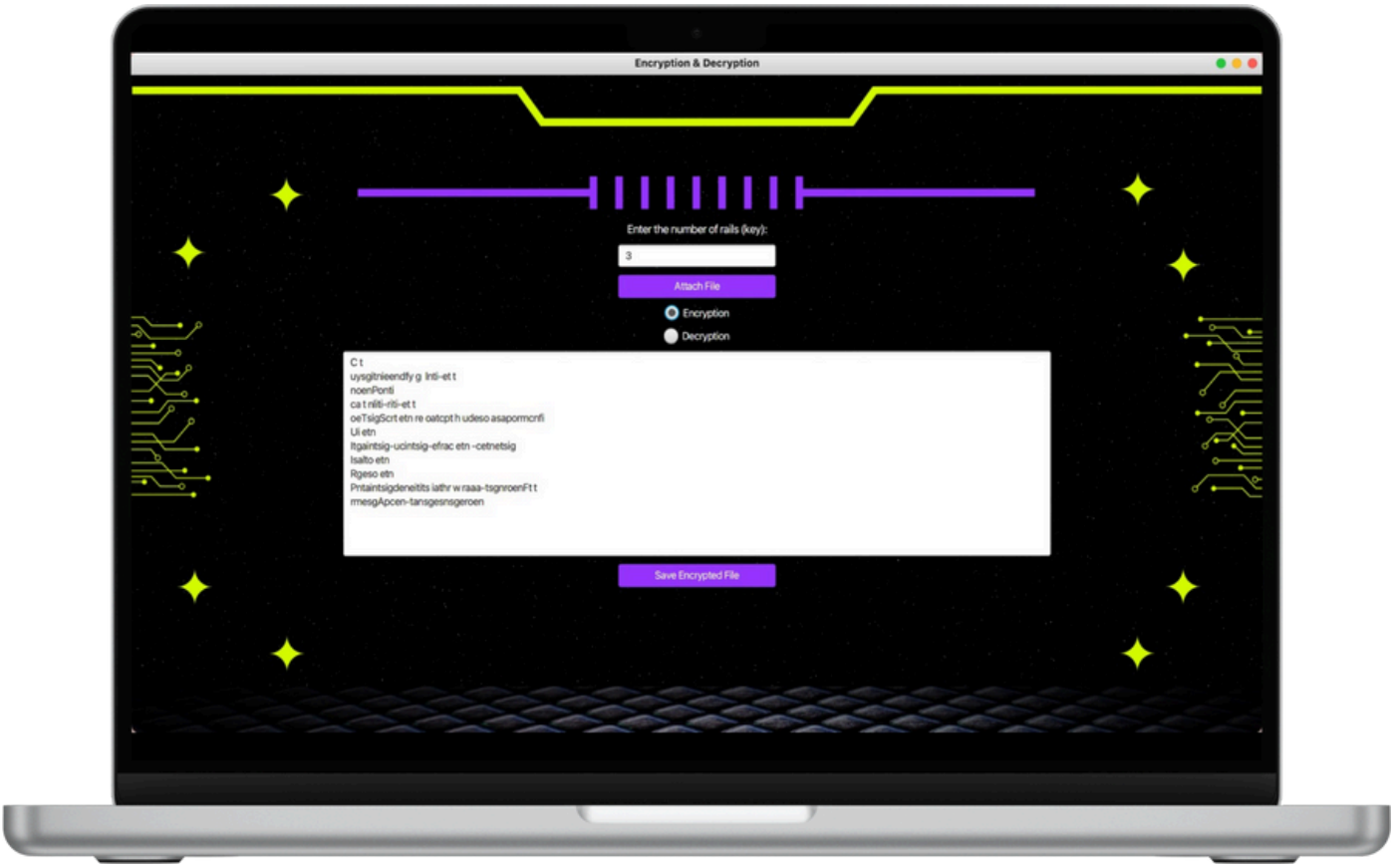
## TEXT BEFORE ENCRYPTION




OriginalText1.txt

Code Testing
Security testing tries to anticipate the hundreds of ways a program can fail
-Unit testing
-Integration testing
-Function testing
-Performance testing
-Acceptance testing
-Installation testing
-Regression testing
-Penetration testing

# TEXT AFTER ENCRYPTION




EncryptedText1.txt

C t
uysgitnieendfy g Inti-et t
noenPonti
ca t nliti-riti-et t
oeTsigScrt etn re oatcpt h udeso asapormcnfi
Ui etn
Itgaintsig-ucintsig-efrac etn -cetnetsig
Isalto etn
Rgeso etn
Pntaintsigdeneitits iathr w raaa-tsgnroenFt t
rmesgApcen-tansgesnsgeroen

# 5. EXAMPLES OF DECRYPTING TEXT

## TEXT BEFORE DECRYPTION


Note:

During testing, decryption worked successfully for normal text input but did not function correctly for file inputs. After making the output text area editable by changing **outputArea.setEditable(false)** to true, I was able to verify that the decrypted text matched the original text when dealing with direct text input.

However, when attempting to apply the decryption process on file attachments, the expected output was not achieved. Further investigation is required to address this issue for file handling, while the text-based functionality operates as intended.

# TEXT AFTER DECRYPTION

# 6. IMPLEMENTATION

## INTERFACE COMPONENTS CODE

```
package encryption.decryption;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.stage.FileChooser;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.text.Font;
import javafx.geometry.Pos;
import javafx.geometry.Insets;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.IOException;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.image.Image;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundImage;
import javafx.scene.layout.BackgroundPosition;
import javafx.scene.layout.BackgroundRepeat;
import javafx.scene.layout.BackgroundSize;
import javafx.scene.paint.Color;
public class EncryptionDecryption extends Application {
    private TextArea outputArea;
    private TextField keyInput;

    public static void main(String[] args) {
        launch(args);// Launches the application
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Encryption & Decryption");


        // Creating a label to prompt the user for the key
        Label keyLabel = new Label("Enter the number of rails (key):");
        keyLabel.setFont(new Font(14));
        keyLabel.setTextFill(Color.WHITE);
        keyLabel.setStyle("-fx-background-color: transparent;");

        // Input field for the user to enter the number of rails
        keyInput = new TextField();
        keyInput.setPromptText("Enter number of rails");
        keyInput.setMaxWidth(200);

        // Button to attach a file
        Button fileButton = new Button("Attach File");
        fileButton.setOnAction(e -> attachFile(primaryStage));
        fileButton.setMaxWidth(200);
        fileButton.setStyle("-fx-background-color: #9537FF");
        fileButton.setTextFill(Color.WHITE);

        // Radio button for encryption & Decryption
        ToggleGroup toggleGroup = new ToggleGroup();
        RadioButton encryptionButton = new RadioButton("Encryption");
        encryptionButton.setToggleGroup(toggleGroup);
        encryptionButton.setTextFill(Color.WHITE);

        RadioButton decryptionButton = new RadioButton("Decryption");
        decryptionButton.setToggleGroup(toggleGroup);
        decryptionButton.setTextFill(Color.WHITE);

        // TextArea to display the output (encrypted text)
        outputArea = new TextArea();
        outputArea.setWrapText(true);
        outputArea.setEditable(false);
```

```
        outputArea.setMaxWidth(900);
        outputArea.setMinWidth(900);
        outputArea.setPrefHeight(250);

        Button saveButton = new Button("Save Encrypted File");
        saveButton.setOnAction(e -> saveEncryptedFile(primaryStage));// Calls the saveEncryptedFile method when clicked
        saveButton.setMaxWidth(200);
        saveButton.setStyle("-fx-background-color: #9537FF");
        saveButton.setTextFill(Color.WHITE);



        // VBox layout to arrange components vertically
        VBox layout = new VBox(10);
        layout.setAlignment(Pos.CENTER);
        layout.setPadding(new Insets(20));
        layout.getChildren().addAll(keyLabel, keyInput, fileButton, encryptionButton, decryptionButton, outputArea, saveButton);

        toggleGroup.selectedToggleProperty().addListener((observable, oldValue, newValue) -> {
            if (newValue != null) {
                try {
                    int key = Integer.parseInt(keyInput.getText());
                    if (key < 1) {
                        Alert alert = new Alert(AlertType.WARNING);
                        alert.setTitle("Warning");
                        alert.setHeaderText(null);
                        alert.setContentText("Please enter key greater than 1 :)");
                        alert.showAndWait();
                        return;
                    }
                    if (key == 1 ||key >= outputArea.getText().length()) {
                        Alert alert = new Alert(AlertType.WARNING);
                        alert.setTitle("Warning");
                        alert.setHeaderText(null);
                        alert.setContentText("The output text is identical to the input text\n "
                                + "please enter a small key to protect your data :)");
                        alert.showAndWait();
                        return;
                    }

                    if (encryptionButton.isSelected()) {
                        outputArea.setText(encryptText(outputArea.getText(), key));
                    } else if (decryptionButton.isSelected()) {
                        outputArea.setText(decryptText(outputArea.getText(), key));
                    }

                } catch (NumberFormatException e) {
                        Alert alert = new Alert(AlertType.WARNING);
                        alert.setTitle("Warning");
                        alert.setHeaderText(null);
                        alert.setContentText("Please enter a valid key :)");
                        alert.showAndWait();
                }
            }
        });

        // Adding a background image to the layout
        Image bgImage = new Image("RailFenceCipherBackground.png");
        BackgroundImage backgroundImage = new BackgroundImage(bgImage, BackgroundRepeat.NO_REPEAT,
                                    BackgroundRepeat.NO_REPEAT, BackgroundPosition.CENTER,
                                    new BackgroundSize(100, 100, true, true, false, true));
        layout.setBackground(new Background(backgroundImage));

        // Set up the scene and window size
        Scene scene = new Scene(layout, 1500, 1000);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

# METHOD TO ATTACH A FILE AND ENCRYPT ITS CONTENTS

```
    // Method to attach a file and encrypt its contents
    private void attachFile(Stage stage) {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Choose File");
        File file = fileChooser.showOpenDialog(stage);

        if (file != null) {
            try {
                BufferedReader reader = new BufferedReader(new FileReader(file));
                StringBuilder content = new StringBuilder();
                String line;

                while ((line = reader.readLine()) != null) {
                    content.append(line).append("\n");
                }

                reader.close();
                outputArea.setText(content.toString());
            } catch (IOException | NumberFormatException e) {
                // Display an error message in case of any issues
                outputArea.setText("Error: " + e.getMessage());
            }
        }
        else {
        // Show alert if no file is selected
        Alert alert = new Alert(AlertType.WARNING);
        alert.setTitle("Warning");
        alert.setHeaderText(null);
        alert.setContentText("Please attach the file :)");
        alert.showAndWait();
    }
    }
```

# METHOD TO ENCRYPT TEXT

```
// Method to encrypt text
private String encryptText(String OriginalText, int key) {
    String textWithSpacesReplaced = OriginalText.replace(" ", "*");
    char[][] encryptionMatrix = new char[key][OriginalText.length()];
    boolean downwardTrend= false;
    int layer = 0;

    for (int matrixColumn = 0; matrixColumn < OriginalText.length(); matrixColumn++) {
        encryptionMatrix[layer][matrixColumn] = textWithSpacesReplaced.charAt(matrixColumn);

        if (layer ==0 ) {
            downwardTrend = true;
        }
        else if(layer== key - 1){
        downwardTrend = false;
        }
        if (downwardTrend) {
            layer++;
        } else {
            layer--;
        }
    }

    StringBuilder textAfterEncryption = new StringBuilder();
    for (int i = 0; i < key; i++) {
        for (int j = 0; j < OriginalText.length(); j++) {
            if (encryptionMatrix[i][j] != '\0') {
                textAfterEncryption.append(encryptionMatrix[i][j]);
            }
        }
    }

    return textAfterEncryption.toString().replace("*"," ");
}
```

# METHOD TO DECRYPT TEXT

```java
// Method to decrypt text
private String decryptText(String encryptedText, int key) {
    char[][] decryptionMatrix = new char[key][encryptedText.length()];
    boolean downwardTrend= false;
    int index = 0;
    int layer=0;

        for (int matrixColumn = 0; matrixColumn < encryptedText.length(); matrixColumn++) {
        decryptionMatrix[layer][matrixColumn] = '~';

        if (layer ==0 ) {
            downwardTrend = true;
        }
        else if(layer== key - 1){
        downwardTrend = false;
        }
        if (downwardTrend) {
            layer++;
        } else {
            layer--;
        }}
    for (int i = 0; i < key; i++) {
        for (int j = 0; j < encryptedText.length(); j++) {
            if (decryptionMatrix[i][j] == '~' && index < encryptedText.length()) {
                decryptionMatrix[i][j] = encryptedText.charAt(index++);
            }}}

    StringBuilder decryptedText = new StringBuilder();
    int row = 0;
    for (int matrixColumn = 0; matrixColumn < encryptedText.length(); matrixColumn++) {
        if (decryptionMatrix[row][matrixColumn] != '\0') {
            decryptedText.append(decryptionMatrix[row][matrixColumn]);
        }
        if (row ==0 ) {
            downwardTrend = true;
        }
        else if(row== key - 1){
        downwardTrend = false;
        }
        if (downwardTrend) {
            row++;
        } else {
            row--;
        } }
    return decryptedText.toString();
}
```

# METHOD FOR SAVING FILE

```java
    // Method for saving encrypted file
    private void saveEncryptedFile(Stage stage) {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save Encrypted File");
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("Text Files", "*.txt"));
        File file = fileChooser.showSaveDialog(stage);

        if (file != null) {
            try {
                FileWriter writer = new FileWriter(file);
                writer.write(outputArea.getText());
                writer.close();
            } catch (IOException e) {
                // Display an error message in case of any issues
                outputArea.setText("Error saving file: " + e.getMessage());
            }
        }
    }
}
```

# 7. COUNTERMEASURES IMPLEMENTED IN THE CODE

### Modular code

We implemented this countermeasure by dividing our code into small simple modules , each module has a single purpose and performing a specific task.

### Encapsulation

It means grouping related data and behavior together inside a single unit , this unit hides away the details of how things work inside it. encapsulation prevents outside code from directly meddling with an object's inside. for example ,in our code the details of encryption and decryption methods are hidden when they are invoked in the main method .

### Information hiding

It means keeping the inner workings of class or method hidden from the outside world . hiding the details of how the methods are implemented but we provide a clear set of instructions to make it easier to understand and use them without any problems .

Mutual Suspicion

Is a security principle based on cross-verification between different system components to ensure data integrity and accuracy. Rather than relying solely on the output of one component, each part checks the validity of inputs and outputs of the other. This approach helps detect errors or potential tampering, enhancing the system's reliability and trustworthiness.

First Check: Validating key Value for Rail Fence Cipher

we first validate the key value for the Rail Fence Cipher. If the user inputs a key less than or equal to 1 or Key equal or Longer than the Text , an error message is displayed, preventing any encryption from occurring. If the key is valid, the text is encrypted and then decrypted

And through it we will check

- If the length of the decrypted text matches the original text.
- If the decrypted text is identical to the original text.

Second Check: Validating Key Type for Rail Fence Cipher

Before proceeding with the encryption or decryption in the Rail Fence Cipher, we ensure that the key entered by the user is a valid numeric value. If the user inputs any non-numeric character (such as symbols, letters, or special characters), an error message is displayed, preventing any operation from being performed. If the key is valid, the program continues with the encryption and decryption processes.

Confinement

It refers to limiting access to data or resources so that they can be better controlled, errors reduced, programming quality improved, and code organization organized. It also deals with hardware, and our program only deals with hardware in a simple way (saving the files to be encrypted).

Genetic diversity

It means introducing variety to avoid vulnerabilities. This could involve using more resources or libraries or different configurations, encryption methods, or even varying code structures within the same framework. The goal is to reduce the chance that a single exploit could compromise an entire system by making each part or instance unique in some way. While using diverse libraries could contribute to genetic diversity.

In our project, we developed all functionalities and did not utilize any external libraries or resources

# 8. TESTING

## UNIT TESTING

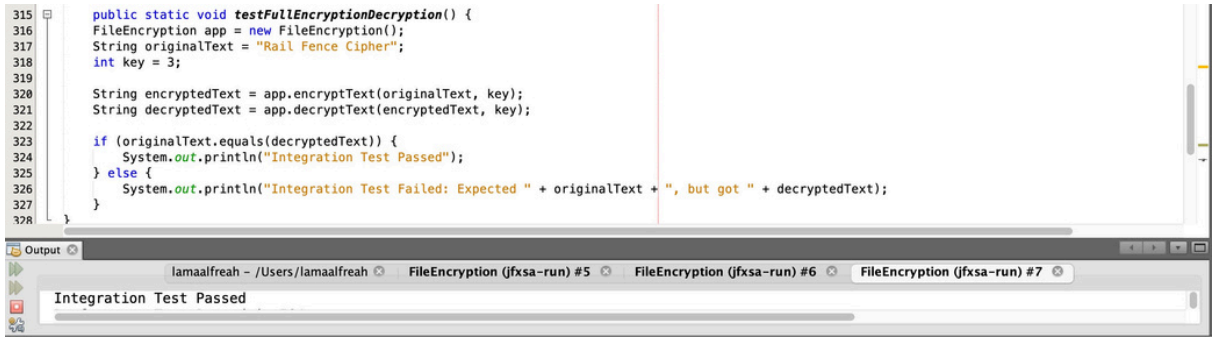
Tests individual functions or components in isolation to ensure each part works correctly with specific inputs, helping identify bugs in specific parts of the code.

```
    public static void testEncryptText() {
        FileEncryption app = new FileEncryption();
        String result = app.encryptText("Rail Fence Cipher", 3);
        String expected = "R ciralFneCpeie h";
        if (result.equals(expected)) {
            System.out.println("Encrypt Text Test Passed");
        } else {
            System.out.println("Encrypt Text Test Failed: Expected " + expected + ", but got " + result);
        }
    }

    public static void testDecryptText() {
        FileEncryption app = new FileEncryption();
        String result = app.decryptText("R ciralFneCpeie h", 3);
        String expected = "Rail Fence Cipher";
        if (result.equals(expected)) {
            System.out.println("Decrypt Text Test Passed");
        } else {
            System.out.println("Decrypt Text Test Failed: Expected " + expected + ", but got " + result);
        }
    }
}
```

```
Output
lamaalfreah - /Users/lamaalfreah | FileEncryption (jfxsa-run) #5
Encrypt Text Test Passed
Decrypt Text Test Passed
```

## INTEGRATION TESTING

Ensures that different functions or modules work well together as a sequence of operations, verifying interactions between various parts of the system.



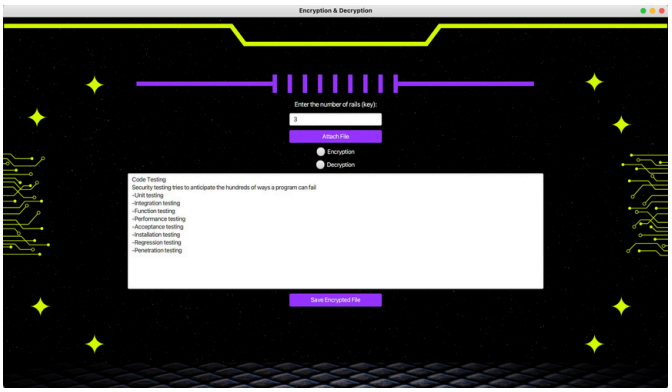
```
    public static void testFullEncryptionDecryption() {
        FileEncryption app = new FileEncryption();
        String originalText = "Rail Fence Cipher";
        int key = 3;

        String encryptedText = app.encryptText(originalText, key);
        String decryptedText = app.decryptText(encryptedText, key);

        if (originalText.equals(decryptedText)) {
            System.out.println("Integration Test Passed");
        } else {
            System.out.println("Integration Test Failed: Expected " + originalText + ", but got " + decryptedText);
        }
    }
```

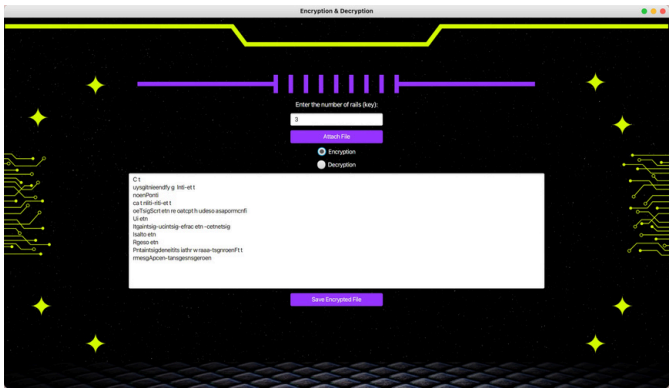
```
Output
lamaalfreah - /Users/lamaalfreah | FileEncryption (jfxsa-run) #5 | FileEncryption (jfxsa-run) #6 | FileEncryption (jfxsa-run) #7
Integration Test Passed
```

# FUNCTION TESTING

Functional testing ensures that the application work as intended, We entered valid and invalid inputs to test how the code would behave.

When we entered the inputs correctly, the code results were correct in both encryption and decryption cases as a text , when we entered the text as a file the encryption result is correct .



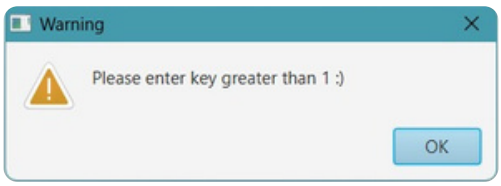




Note:

During testing, decryption worked successfully for normal text input but did not function correctly for file inputs. After making the output text area editable by changing **outputArea.setEditable(false)** to true, I was able to verify that the decrypted text matched the original text when dealing with direct text input.

However, when attempting to apply the decryption process on file attachments, the expected output was not achieved. Further investigation is required to address this issue for file handling, while the text-based functionality operates as intended.

when we entered invalid inputs, a window appeared showing the problem and how to fix it.


## PERFORMANCE TESTING

Measures the speed and responsiveness of the code when handling large amounts of data, identifying any performance issues.

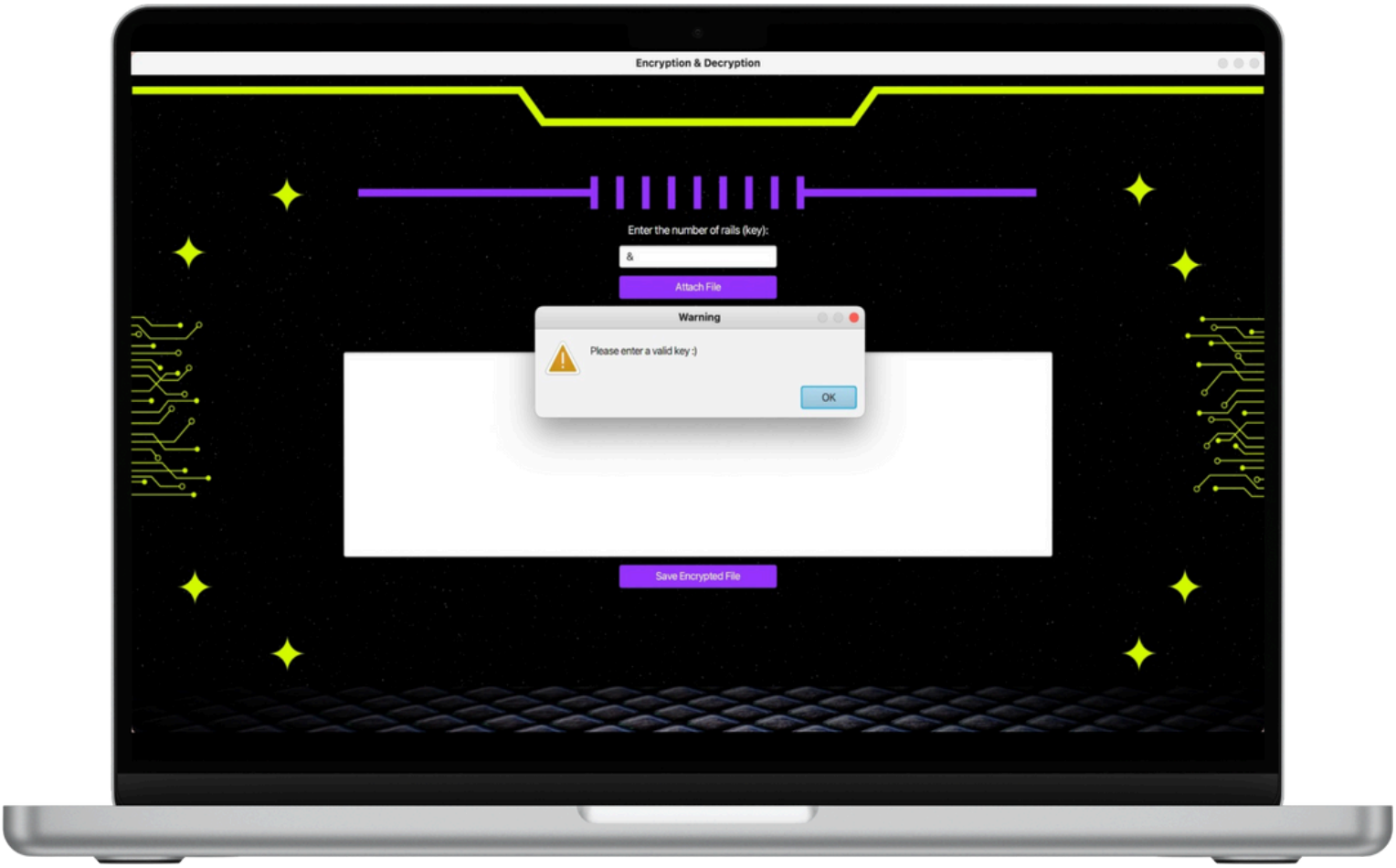
```
public static void testPerformance() {
FileEncryption app = new FileEncryption();
StringBuilder longText = new StringBuilder();

// long text to test
for (int i = 0; i < 100000; i++) {
    longText.append("Lama ");
}
int key = 5;

long startTime = System.currentTimeMillis();
String encryptedText = app.encryptText(longText.toString(), key);
String decryptedText = app.decryptText(encryptedText, key);
long endTime = System.currentTimeMillis();

if (longText.toString().equals(decryptedText)) {
    System.out.println("Performance Test Passed in " + (endTime - startTime) + "ms");
} else {
    System.out.println("Performance Test Failed");
}
}
}
```

```
Output
lamaalfreah - /Users/lamaalfreah | FileEncryption (jfxsa-run) #5 | FileEncryption (jfxsa-run) #6 | FileEncryption (jfxsa-run) #7
Performance Test Passed in 584ms
```

# 9. TROUBLESHOOTING

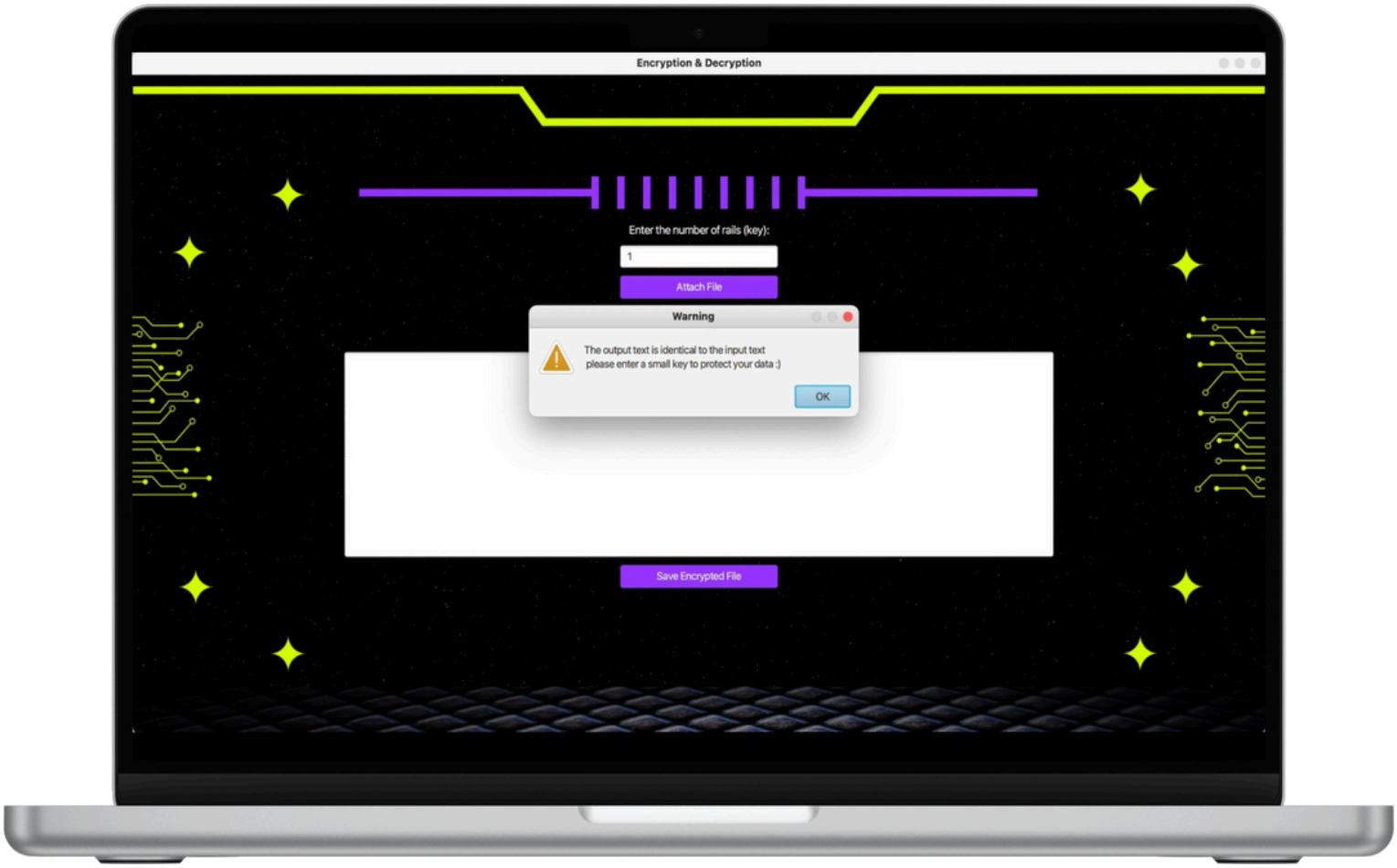
## INVALID KEY ENTRY (NON-NUMERIC)

- Issue: The user enters a key that is not a number (e.g., letters or symbols).
- Solution: Display a message: "Please enter a valid key :)" Ensure that the key field accepts only numeric input.



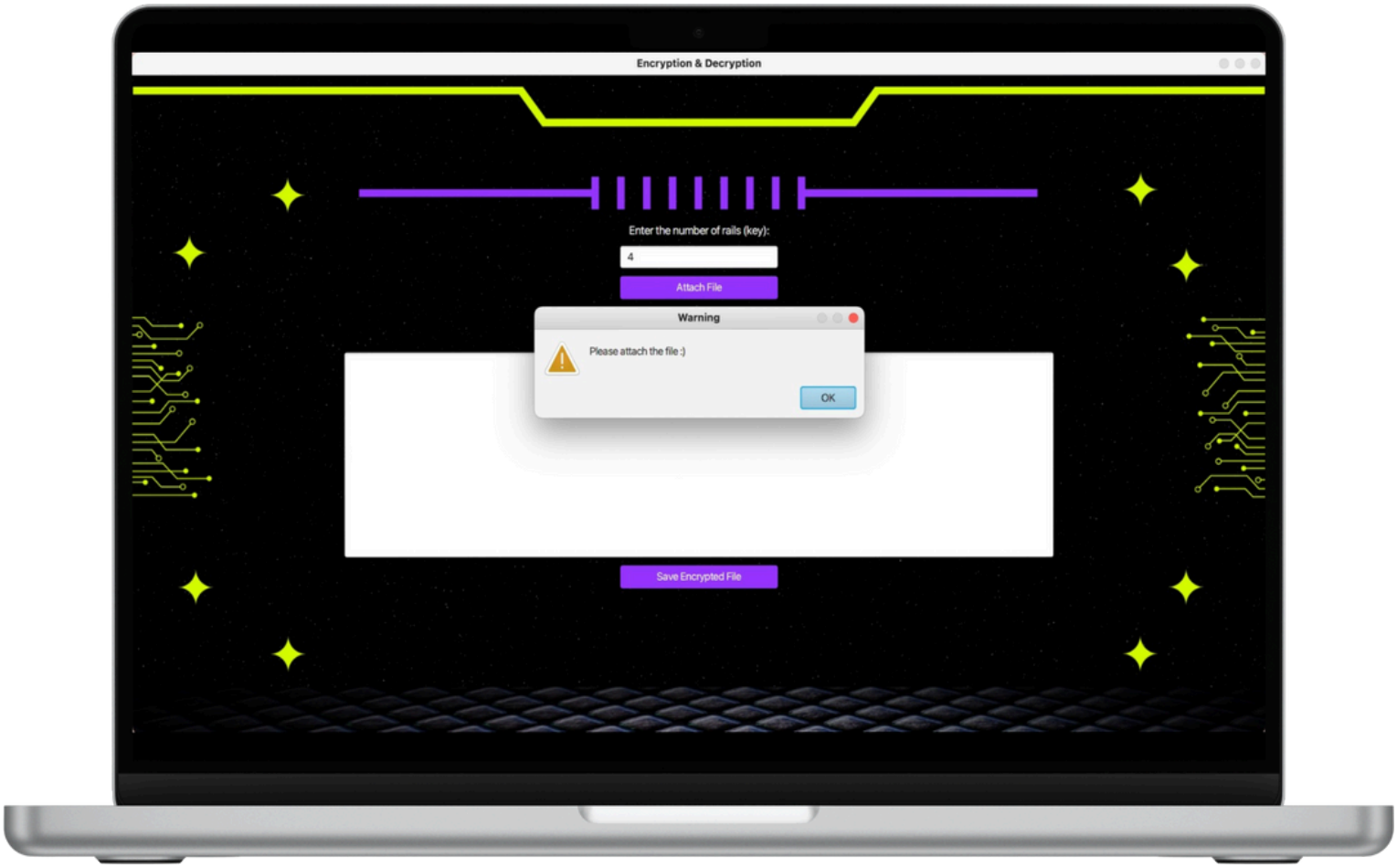
## KEY VALUE OF 1 OR KEY EQUAL OR LONGER THAN THE TEXT

- Issue: The user enters a key value of 1 or a key that is the same length as or longer than the input text, which results in the encrypted text being identical to the original.
- Solution: Display a warning message: "The output text is identical to the input text please enter a small key to protect your data :)"



## NO FILE ATTACHED

- Issue: The user tries to encrypt or decrypt without attaching a file.
- Solution: Display a message: "Please attach the file" Make it clear that the operation cannot continue without a file input.



# 10. REFERENCES LIST


https://m.youtube.com/watch?v=uTf1mJvbPmA

https://m.youtube.com/watch?v=knE4G8DGLoY

https://m.youtube.com/watch?v=27EgALog5Xk

https://m.youtube.com/watch?v=hEwn_2SLENs

https://www.geeksforgeeks.org/rail-fence-cipher-encryption-decryption/

https://youtu.be/reKq19WH4q8?si=_aw0nAEtvx7Ca22m

Larksuite.com

https://youtu.be/h1NtW0wYqGU?si=ph7HKRUfx90nTKdd

https://youtu.be/27EgALog5Xk?si=TyPLGujUGjWoUFZl

https://www.youtube.com/watch?v=EAR9GgQMy5s