CCS6314 Cryptography and Data Security Assignment

Group B

Student Name	Student ID
Wong Zhi Lin	1201203299
Cheah Wei Yan	1201203703
Chay Chun Keat	1211200378
Dylan Cheng Zhi Xin	1211200579

Table of Contents

Abstract

The fundamental role of cryptography in communication protection consists of maintaining data confidentiality and integrity with authentication guarantees. The project demonstrates implementation along with analysis of classical along with modern cryptographic techniques through a Command-Line Interface (CLI) tool. The tool enables users to encrypt and decrypt information with Playfair Cipher as well as Rail Fence Cipher and Product Ciphers (Playfair + Rail Fence) and RSA Key Exchange and AES/Triple DES encryption.

The beginning section of the project utilizes Product Classical Symmetric Ciphers through substitution and transposition approaches to boost encryption security. The combination of Playfair Cipher with its digraph-encrypted 5x5 matrix accompanies the transposition-based Rail Fence Cipher to establish a complex encryption method. An evaluation of encryption and decryption speed takes place at different plaintext length points for these traditional cryptographic methods.

The second method utilizes RSA as an asymmetric cipher before encrypting data using AES or Triple DES in a symmetric key system. The tool provides demonstrations of key creation along with cryptography and decryption methods together with evaluations of systematic variance on ciphertext integrity.

The project demonstrates cryptographic evaluation through computational speed assessment and security assessments of different cryptographic methods. The CLI tool functions as an educational tool because users can perform interactive tests with encryption and decryption functions.

Introduction

Through history cryptography has served as a basic instrument to defend both communication channels and critical information from illicit viewing attempts. The development of encryption started with basic classical ciphers but eventually transformed into sophisticated cryptographic systems of today. Historically classical symmetric ciphers employed substitution and transposition methods in their encryption process of messages. The growth in computing technology rendered classical ciphers susceptible to attack methods. New encryption systems solve these weaknesses through advanced mathematical programs and dual encryption systems which create secure systems without reducing speed performance.

The project develops and scrutinizes traditional and state-of-the-art encryption practices by building a Cryptography Command-Line Interface (CLI) Tool. This tool provides users with an environment for trying different encryption and decryption methods with multiple cryptographic procedures. The program implements encryption with three distinct methods: Playfair Cipher along with Rail Fence Cipher and the Product Cipher that applies to both methods. Secure communication occurs through RSA key exchange which is supported by AES or Triple DES encryption during the symmetric encryption phase of the project.

The main purpose of this investigation involves testing encryption and decryption protocols to measure their computational speed and security reliability levels. This project studies classical ciphers regarding their historic relevance yet explores weaknesses found in those methods and evaluates modern hybrid encryption regarding security capabilities in communication. The research evaluates the impact of data transmission errors on encrypted data to demonstrate the importance of secure communication integrity. The project uses CL-based cryptographic methods to provide users with real-world understanding of encryption and decryption processes in practical settings.

Contribution of each student to the project

Wong Zhi Lin	Coding
Dylan Cheng Zhi Xin	Coding + Slide
Cheah Wei Yan	Report + Slide
Chay Chun Keat	Report

Background Study

The history of protecting communication has depended fundamentally on cryptography as techniques evolved from classical encryption methods into present-day cryptographic standards. Two major classes of encryption methods exist in classical cryptography routines although substitution ciphers and transposition ciphers are their main types. In substitution ciphers each plaintext character gets exchanged for distinct symbols that take their place from a determined key whereas transposition ciphers shuffle message characters to hide structural patterns. These historic cipher methods remain valuable for their functions in past scenarios, yet criminals can easily break them through both frequency analysis methods and forceful attack approaches.

The Playfair Cipher stands as a popular substitution cipher since it hides pairs of letters within a 5×5 key matrix created from a specific secret keyword. The security features of Playfair encryption make frequency analysis complicated due to its digraph processing method that exceeds simple monoalphabetic cipher encryption methods. This cipher faces risk from contemporary computational techniques because it follows a structured encryption procedure. As a transposition cipher the Rail Fence Cipher distributes plaintext through columns formed in a zigzag pattern before reading it row-by-row to generate ciphertext. The disruption of character placement in this cipher makes decryption less complex when analyzed on its own without affecting the distribution of letters between the characters. Product Ciphers which unite Playfair and Rail Fence create stronger protection against cryptanalysis through their implementation of both substitution and transposition cryptographic techniques.

The improvement of technological computing now makes traditional cipher systems insufficient for maintaining secure digital communication. Modern cryptographic technology implemented two encryption approaches, namely symmetric and asymmetric methods to establish stronger security measures. Symmetric encryption incorporates two key encryption methods which include the Advanced Encryption Standard (AES) and Triple Data Encryption Standard (Triple DES). The highly efficient algorithms find broad applications in encrypted data management that demands secure and quick data encryption. The principal challenge using these methods comes from safely sharing the secret key between users attempting communication.

The distribution problem is solved by implementing RSA (Rivest-Shamir-Adleman) asymmetric encryption systems. RSA implements two sets of keys to securely transport the symmetric encryption key needed for message protection. Secure AES or triple DES encryption processes can be implemented to protect data communications after a security check of the secret key has been concluded. The combined method utilizes asymmetric encryption to establish secure key transfer followed by fast symmetric encryption for encrypting extensive datasets.

The integrity of ciphertext transmission depends heavily on how bit errors affect the security measures. Network interruptions as well as storage damage and transmission interferences are factors that cause bit errors to occur. Classical cipher systems protect most of the decrypted message even when minor errors appear. Single-bit transmission errors affect multiple blocks during decryption when modern encryption methods use AES ciphers working under CBC mode. The correct design of secure communication systems requires a full understanding of transmission error effects on data integrity because such systems must handle these errors.

The CLI-based system of this project serves to implement and evaluate cryptographic techniques through interactive user testing of encryption and key exchange processes with decryption functions. The analysis between classical and modern cryptographic methods enables researchers to establish understanding about their operational strengths and weaknesses and their practical uses for secure communication.

Description of the Concepts, Methods, and Algorithms Used

1. Classical Symmetric Ciphers
    a. Playfair Cipher (Substitution Cipher)
        i. The Playfair Cipher is a digraph substitution cipher that encrypts pairs of letters using a 5×5 matrix generated from a secret keyword. The matrix excludes the letter 'J' (often combined with 'I').
        ii. Encryption Process
            1. Create a 5×5 matrix using a keyword (e.g., *MONARCHY*).
            2. Convert plaintext into letter pairs, inserting an extra letter (e.g., 'X') if needed to avoid repeated letters in a pair.
            3. Encrypt the letter pairs using the following rules:
                a. If the pair appears in the same row, replace each letter with the next letter in the row.
                b. If the pair appears in the same column, replace each letter with the letter below it.
                c. If the pair forms a rectangle, swap letters diagonally.
        iii. Decryption Process
            1. Decryption follows the same logic as encryption but moves in the opposite direction within the matrix.
        iv. Security Considerations:
            1. More secure than simple monoalphabetic ciphers but vulnerable to frequency analysis of digraphs.
            2. Easily broken using brute-force attacks due to its limited key space.

b. Rail Fence Cipher (Transposition Cipher)
   i. The Rail Fence Cipher is a simple transposition cipher that scrambles the order of plaintext characters by writing them in a zigzag pattern across multiple rows.
   ii. Encryption Process:
      1. Choose a rail depth (number of rows).
      2. Write the plaintext diagonally down and then up in a zigzag pattern across the rows.
      3. Read the ciphertext row-wise.
   iii. Decryption Process:
      1. Determine the zigzag pattern based on rail depth.
      2. Reconstruct the original message by reading letters back in the same pattern.
   iv. Security Considerations:
      1. Harder to crack than simple ciphers but still vulnerable to anagramming and pattern recognition attacks.
      2. Often used in combination with substitution ciphers for added security.

c. Product Cipher (Playfair + Rail Fence)
   i. A Product Cipher combines substitution and transposition to strengthen encryption. In this project, the Playfair Cipher (substitution) is followed by Rail Fence (transposition).
   ii. Security Benefits:
      1. The substitution layer (Playfair) disguises plaintext letter frequencies.
      2. The transposition layer (Rail Fence) disrupts character order, making frequency analysis more difficult.
      3. More resistant to traditional cryptanalysis than individual ciphers alone.

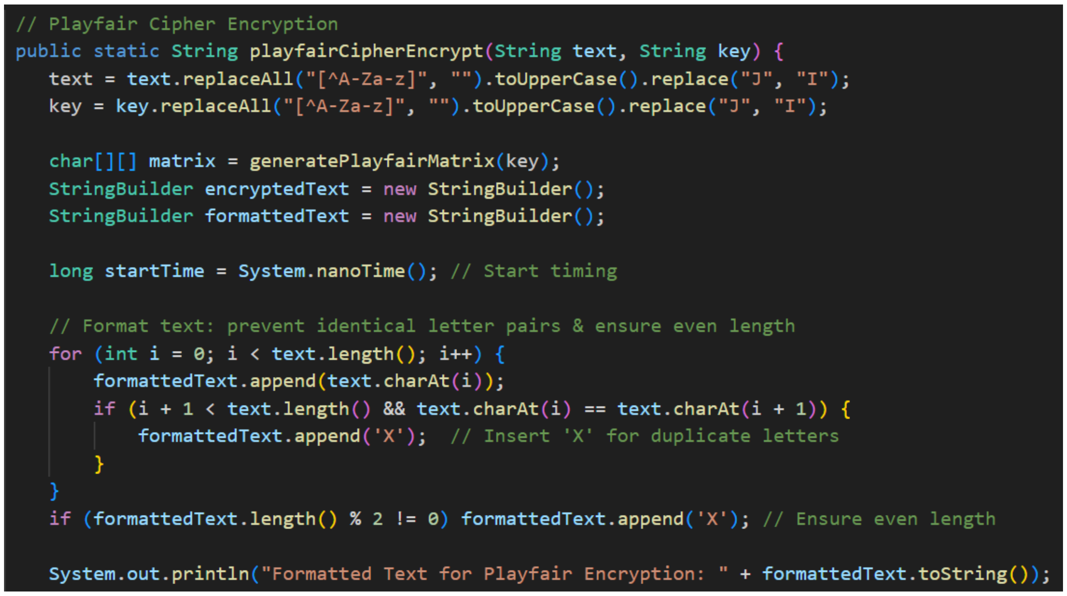
2. Modern Cryptography: Hybrid Encryption Approach
    a. RSA Algorithm (Asymmetric Key Exchange)
        i. RSA (Rivest-Shamir-Adleman) is a widely used public-key cryptosystem that enables secure key exchange.
        ii. Key Generation:
            1. Choose two large prime numbers p and q.
            2. Compute $n = p \times q$ (modulus).
            3. Compute $\varphi(n) = (p-1)(q-1)$ (Euler's totient function).
            4. Select a public key exponent e (commonly 65537).
            5. Compute private key $d = e^{-1} \bmod \varphi(n)$.
        iii. Encryption
            1. Ciphertext C = M^e mod n, where M is the plaintext message.
        iv. Decryption:
            1. Plaintext M = C^d mod n.
        v. Security Considerations:
            1. Based on the difficulty of factoring large numbers, making it computationally infeasible to break with current technology.
            2. Used for secure key exchange, but inefficient for large data encryption.

b. AES (Advanced Encryption Standard) - Symmetric Encryption
    i. AES is a block cipher that encrypts data in fixed-size blocks (128 bits). It is known for its speed and security.
    ii. AES Encryption Process:
        1. Key Expansion – Generates multiple round keys from the original secret key.
        2. Initial Round – Adds the first-round key to the plaintext.
        3. Main Rounds (depends on key size, usually 10 rounds for AES-128):
            a. SubBytes: Replaces each byte using an S-box.
            b. ShiftRows: Shifts rows to the left for diffusion.
            c. ShiftRows: Shifts rows to the left for diffusion.
            d. AddRoundKey: XORs the state with the round key.
        4. Final Round (No MixColumns).
    iii. Security Considerations:
        1. Resistant to brute-force attacks due to large key sizes (128, 192, or 256 bits).
        2. Efficient and widely used in real-world applications like SSL/TLS.

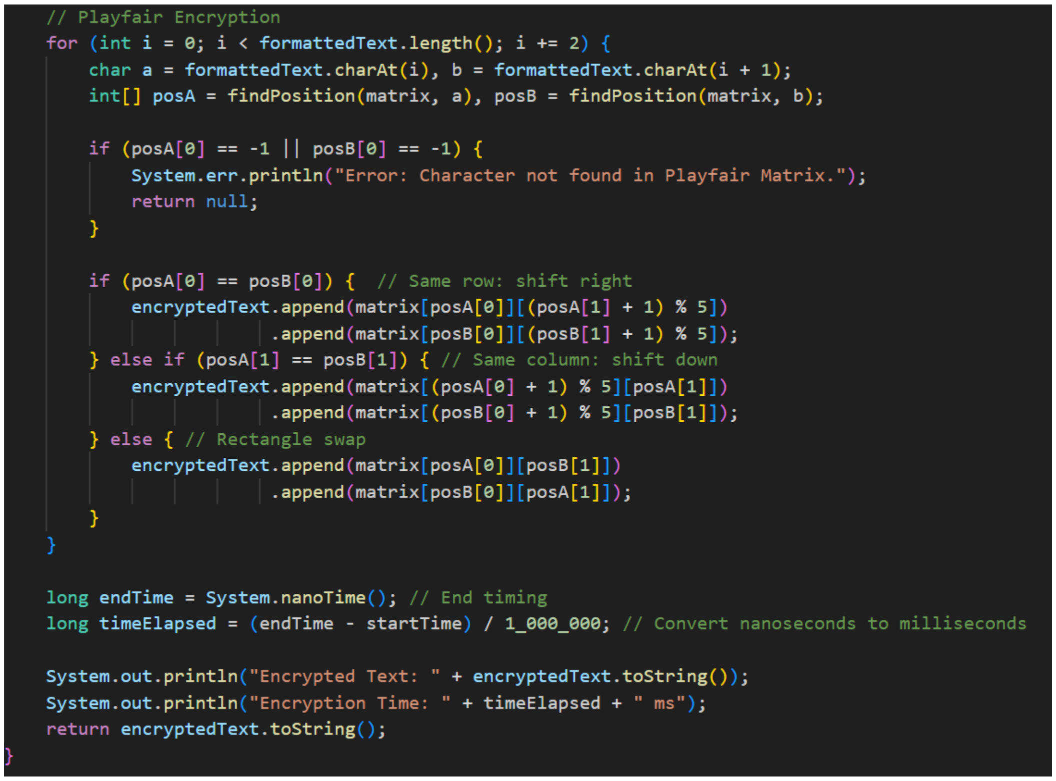
Implementation Details

1. Development Environment and Tools
    a. Programming Language: Java
    b. Java Version: JDK 8 or later
    c. Libraries Used:
        i. java.security.SecureRandom → Generates cryptographically secure random numbers for key and IV generation.
        ii. java.util.Base64 → Encodes and decodes data in Base64 format for secure key transmission.
        iii. java.util.Scanner → Reads user input from the command line for menu selection and message input.
        iv. javax.crypto.SecretKey → Represents a secret key used for AES encryption and decryption.
        v. java.time.Duration → Measures the time elapsed between encryption and decryption operations.
        vi. java.time.Instant → Captures timestamps to calculate execution time.
        vii. java.math.BigInteger → Handles large integer calculations needed for RSA encryption and decryption.
        viii. javax.crypto.spec.SecretKeySpec → Converts a byte array into a SecretKey for AES decryption.
        ix. javax.crypto.Cipher → Performs encryption and decryption using cryptographic algorithms like AES.
        x. javax.crypto.spec.IvParameterSpec → Stores the Initialization Vector (IV) for AES CBC mode encryption.

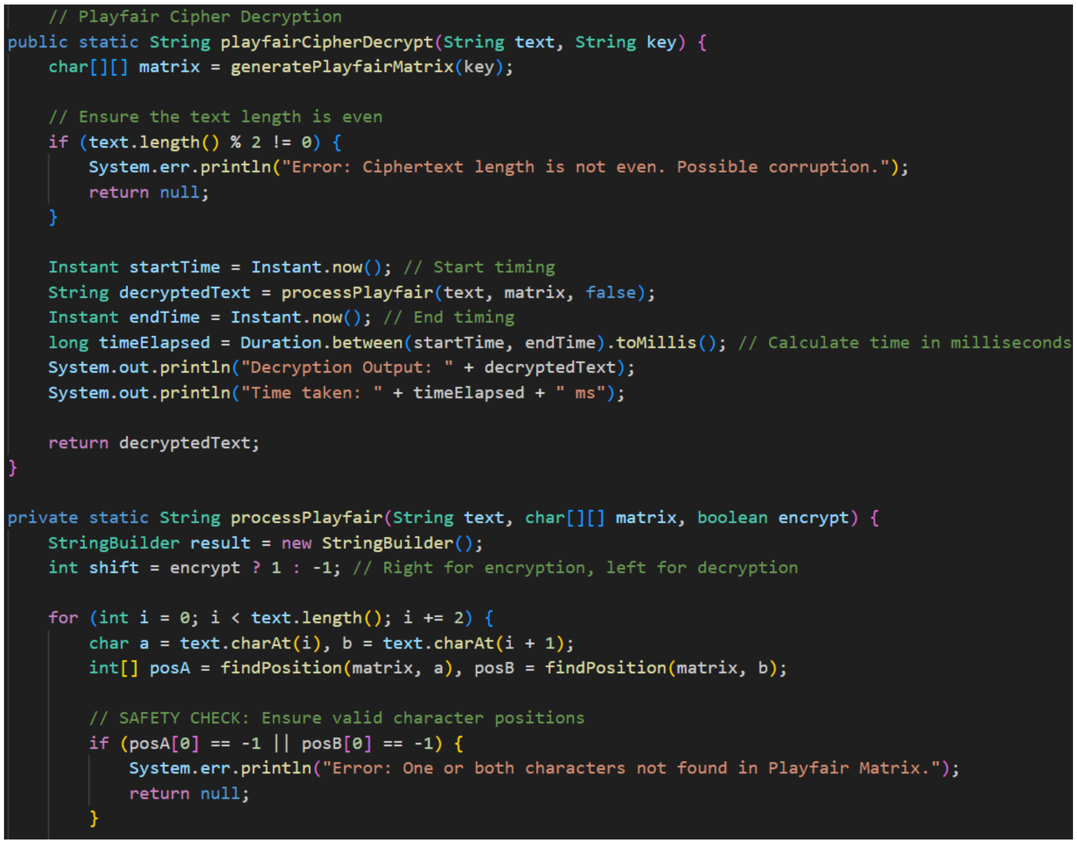
2. Implementation of Classical Symmetric Ciphers
    a. Playfair Cipher Implementation
        i. Encryption Process:
            1. Generate a 5×5 key matrix from the user-provided key.
            2. Format the plaintext:
                a. Convert text to uppercase and replace 'J' with 'I'.
                b. Insert 'X' between repeating letters in a digraph.
                c. Ensure the text has an even number of characters.
            3. Apply Playfair encryption rules:
                a. If both letters in a digraph are in the same row, replace them with the letters to their right.
                b. If both are in the same column, replace them with the letters below.
                c. Otherwise, swap the letters diagonally in the 5×5 matrix.
            4. Measure encryption time using System.nanoTime().
        ii. Decryption Process:
            1. The decryption logic follows the same steps as encryption but shifts letters in the opposite direction.



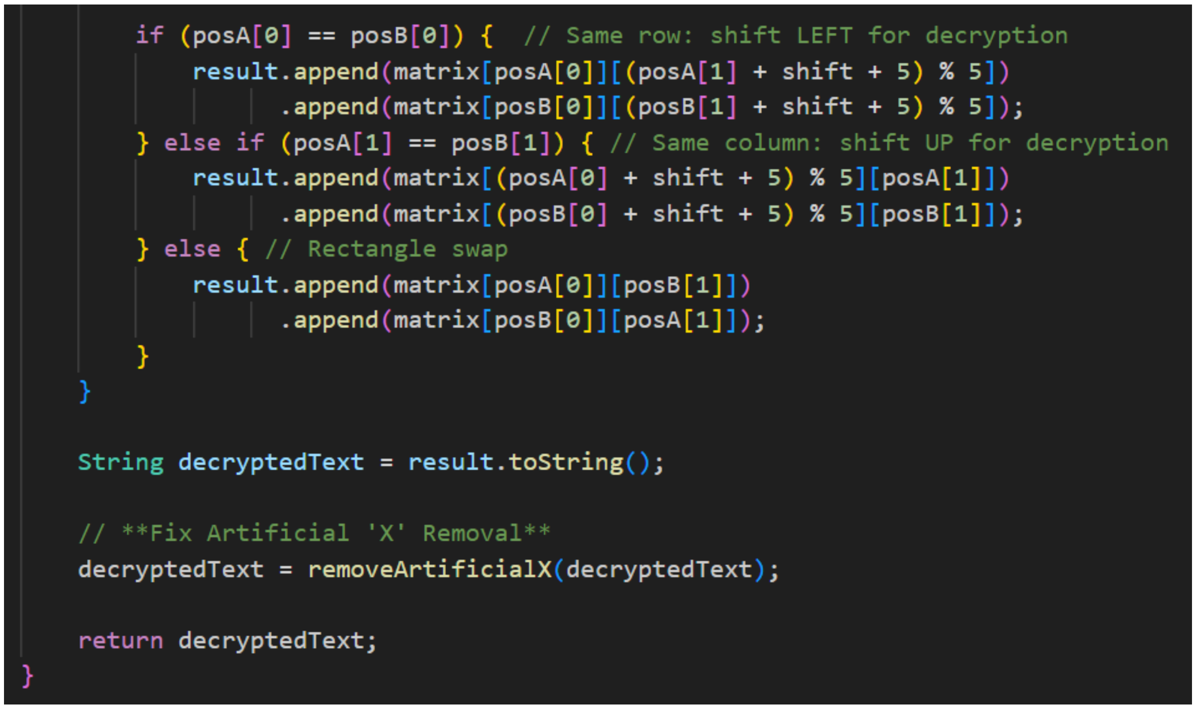
```java
// Playfair Cipher Encryption
public static String playfairCipherEncrypt(String text, String key) {
    text = text.replaceAll("[^A-Za-z]", "").toUpperCase().replace("J", "I");
    key = key.replaceAll("[^A-Za-z]", "").toUpperCase().replace("J", "I");

    char[][] matrix = generatePlayfairMatrix(key);
    StringBuilder encryptedText = new StringBuilder();
    StringBuilder formattedText = new StringBuilder();

    long startTime = System.nanoTime(); // Start timing

    // Format text: prevent identical letter pairs & ensure even length
    for (int i = 0; i < text.length(); i++) {
        formattedText.append(text.charAt(i));
        if (i + 1 < text.length() && text.charAt(i) == text.charAt(i + 1)) {
            formattedText.append('X');  // Insert 'X' for duplicate letters
        }
    }
    if (formattedText.length() % 2 != 0) formattedText.append('X'); // Ensure even length

    System.out.println("Formatted Text for Playfair Encryption: " + formattedText.toString());
```



```
    // Playfair Encryption
    for (int i = 0; i < formattedText.length(); i += 2) {
        char a = formattedText.charAt(i), b = formattedText.charAt(i + 1);
        int[] posA = findPosition(matrix, a), posB = findPosition(matrix, b);

        if (posA[0] == -1 || posB[0] == -1) {
            System.err.println("Error: Character not found in Playfair Matrix.");
            return null;
        }

        if (posA[0] == posB[0]) {  // Same row: shift right
            encryptedText.append(matrix[posA[0]][(posA[1] + 1) % 5])
                         .append(matrix[posB[0]][(posB[1] + 1) % 5]);
        } else if (posA[1] == posB[1]) { // Same column: shift down
            encryptedText.append(matrix[(posA[0] + 1) % 5][posA[1]])
                         .append(matrix[(posB[0] + 1) % 5][posB[1]]);
        } else { // Rectangle swap
            encryptedText.append(matrix[posA[0]][posB[1]])
                         .append(matrix[posB[0]][posA[1]]);
        }
    }

    long endTime = System.nanoTime(); // End timing
    long timeElapsed = (endTime - startTime) / 1_000_000; // Convert nanoseconds to milliseconds

    System.out.println("Encrypted Text: " + encryptedText.toString());
    System.out.println("Encryption Time: " + timeElapsed + " ms");
    return encryptedText.toString();
}
```



```
    // Playfair Cipher Decryption
public static String playfairCipherDecrypt(String text, String key) {
    char[][] matrix = generatePlayfairMatrix(key);

    // Ensure the text length is even
    if (text.length() % 2 != 0) {
        System.err.println("Error: Ciphertext length is not even. Possible corruption.");
        return null;
    }

    Instant startTime = Instant.now(); // Start timing
    String decryptedText = processPlayfair(text, matrix, false);
    Instant endTime = Instant.now(); // End timing
    long timeElapsed = Duration.between(startTime, endTime).toMillis(); // Calculate time in milliseconds
    System.out.println("Decryption Output: " + decryptedText);
    System.out.println("Time taken: " + timeElapsed + " ms");

    return decryptedText;
}

private static String processPlayfair(String text, char[][] matrix, boolean encrypt) {
    StringBuilder result = new StringBuilder();
    int shift = encrypt ? 1 : -1; // Right for encryption, left for decryption

    for (int i = 0; i < text.length(); i += 2) {
        char a = text.charAt(i), b = text.charAt(i + 1);
        int[] posA = findPosition(matrix, a), posB = findPosition(matrix, b);

        // SAFETY CHECK: Ensure valid character positions
        if (posA[0] == -1 || posB[0] == -1) {
            System.err.println("Error: One or both characters not found in Playfair Matrix.");
            return null;
        }
```



```
        if (posA[0] == posB[0]) {  // Same row: shift LEFT for decryption
            result.append(matrix[posA[0]][(posA[1] + shift + 5) % 5])
                  .append(matrix[posB[0]][(posB[1] + shift + 5) % 5]);
        } else if (posA[1] == posB[1]) { // Same column: shift UP for decryption
            result.append(matrix[(posA[0] + shift + 5) % 5][posA[1]])
                  .append(matrix[(posB[0] + shift + 5) % 5][posB[1]]);
        } else { // Rectangle swap
            result.append(matrix[posA[0]][posB[1]])
                  .append(matrix[posB[0]][posA[1]]);
        }
    }

    String decryptedText = result.toString();

    // **Fix Artificial 'X' Removal**
    decryptedText = removeArtificialX(decryptedText);

    return decryptedText;
}
```

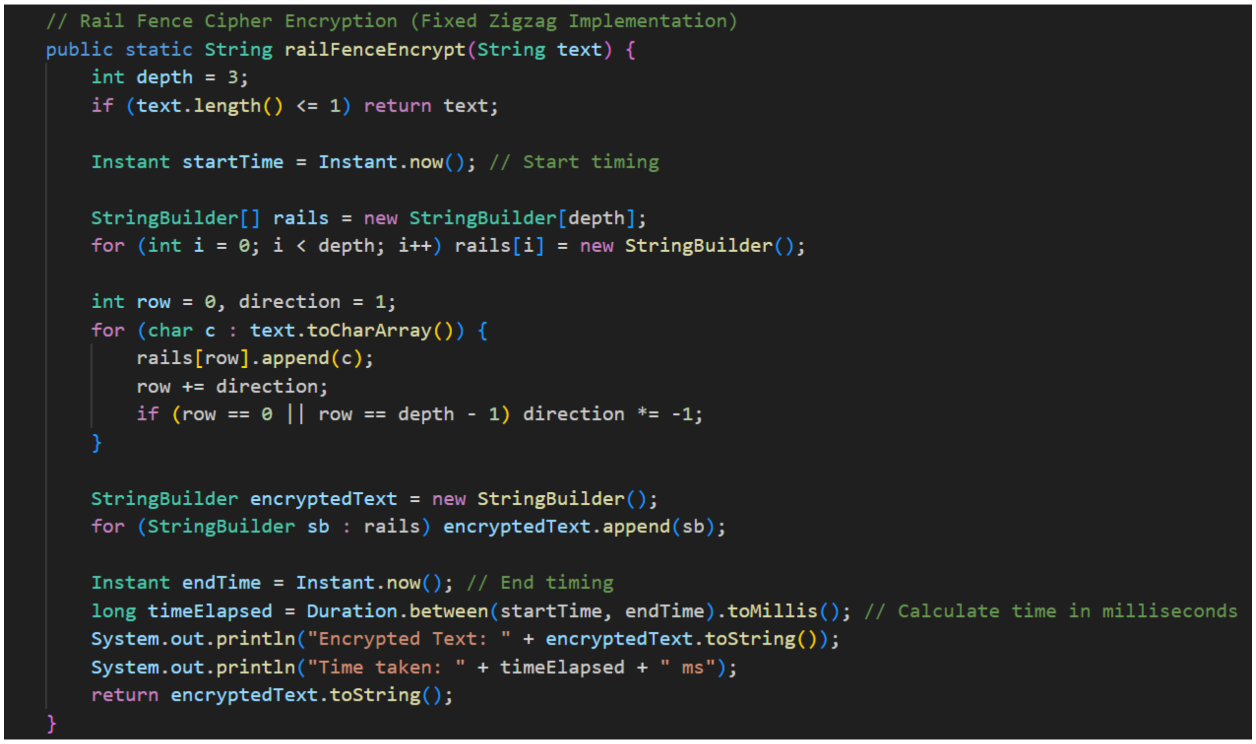
b. Rail Fence Cipher Implementation

   i. Encryption Process:

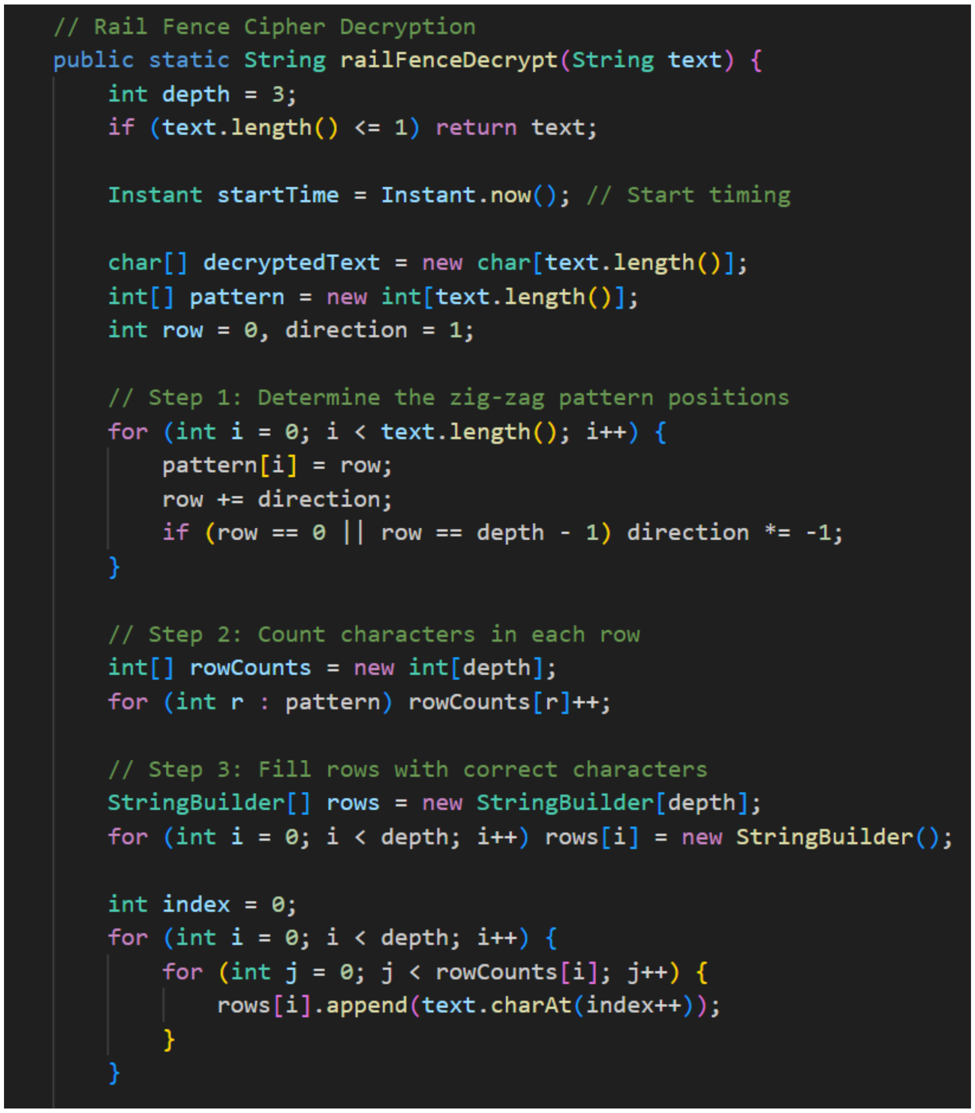
      1. Write the plaintext in a zigzag pattern across a given depth (number of rows).
      2. Read characters row-wise to produce the ciphertext.

   ii. Decryption Process:

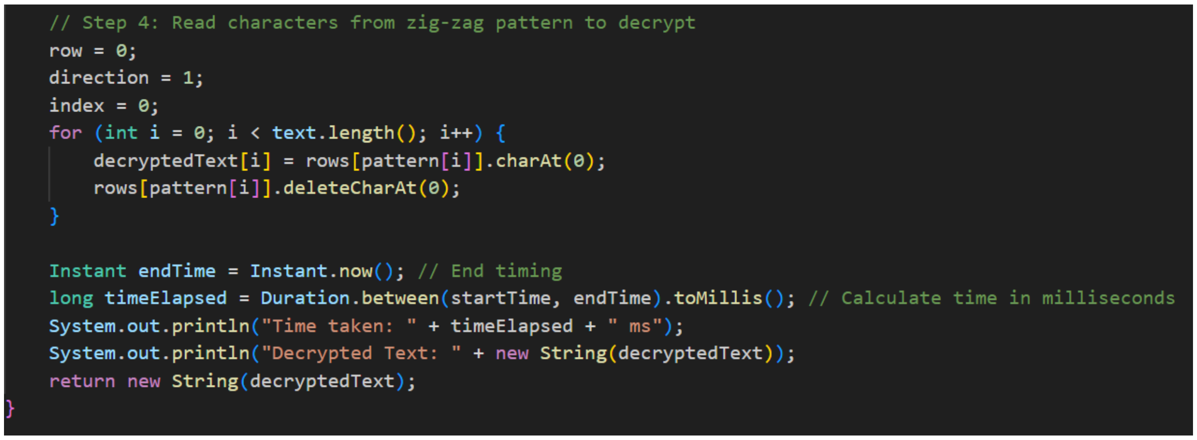
      1. Reconstruct the zigzag pattern based on the rail depth.
      2. Place the encrypted characters into their original positions.
      3. Read the message in the original sequence.



```java
// Rail Fence Cipher Encryption (Fixed Zigzag Implementation)
public static String railFenceEncrypt(String text) {
    int depth = 3;
    if (text.length() <= 1) return text;

    Instant startTime = Instant.now(); // Start timing

    StringBuilder[] rails = new StringBuilder[depth];
    for (int i = 0; i < depth; i++) rails[i] = new StringBuilder();

    int row = 0, direction = 1;
    for (char c : text.toCharArray()) {
        rails[row].append(c);
        row += direction;
        if (row == 0 || row == depth - 1) direction *= -1;
    }

    StringBuilder encryptedText = new StringBuilder();
    for (StringBuilder sb : rails) encryptedText.append(sb);

    Instant endTime = Instant.now(); // End timing
    long timeElapsed = Duration.between(startTime, endTime).toMillis(); // Calculate time in milliseconds
    System.out.println("Encrypted Text: " + encryptedText.toString());
    System.out.println("Time taken: " + timeElapsed + " ms");
    return encryptedText.toString();
}
```

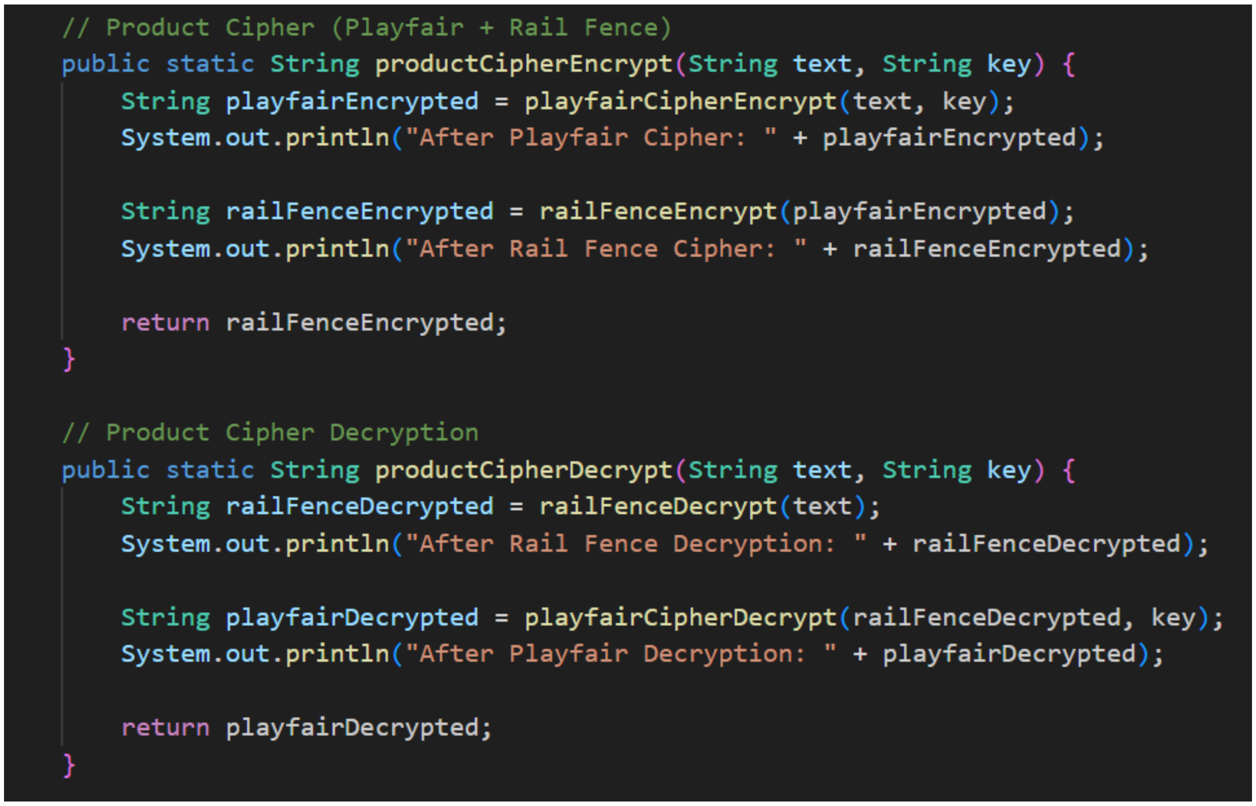


```java
// Rail Fence Cipher Decryption
public static String railFenceDecrypt(String text) {
    int depth = 3;
    if (text.length() <= 1) return text;

    Instant startTime = Instant.now(); // Start timing

    char[] decryptedText = new char[text.length()];
    int[] pattern = new int[text.length()];
    int row = 0, direction = 1;

    // Step 1: Determine the zig-zag pattern positions
    for (int i = 0; i < text.length(); i++) {
        pattern[i] = row;
        row += direction;
        if (row == 0 || row == depth - 1) direction *= -1;
    }

    // Step 2: Count characters in each row
    int[] rowCounts = new int[depth];
    for (int r : pattern) rowCounts[r]++;

    // Step 3: Fill rows with correct characters
    StringBuilder[] rows = new StringBuilder[depth];
    for (int i = 0; i < depth; i++) rows[i] = new StringBuilder();

    int index = 0;
    for (int i = 0; i < depth; i++) {
        for (int j = 0; j < rowCounts[i]; j++) {
            rows[i].append(text.charAt(index++));
        }
    }
```



```java
    // Step 4: Read characters from zig-zag pattern to decrypt
    row = 0;
    direction = 1;
    index = 0;
    for (int i = 0; i < text.length(); i++) {
        decryptedText[i] = rows[pattern[i]].charAt(0);
        rows[pattern[i]].deleteCharAt(0);
    }

    Instant endTime = Instant.now(); // End timing
    long timeElapsed = Duration.between(startTime, endTime).toMillis(); // Calculate time in milliseconds
    System.out.println("Time taken: " + timeElapsed + " ms");
    System.out.println("Decrypted Text: " + new String(decryptedText));
    return new String(decryptedText);
}
```

c. Product Cipher (Playfair + Rail Fence) Implementation
   i. Encryption Process:
      1. Apply Playfair Cipher to plaintext.
      2. Pass the Playfair-encrypted text into Rail Fence Cipher for further scrambling.
   ii. Decryption Process:
      1. Reverse the Rail Fence encryption.
      2. Reverse the Playfair Cipher to obtain the original plaintext.

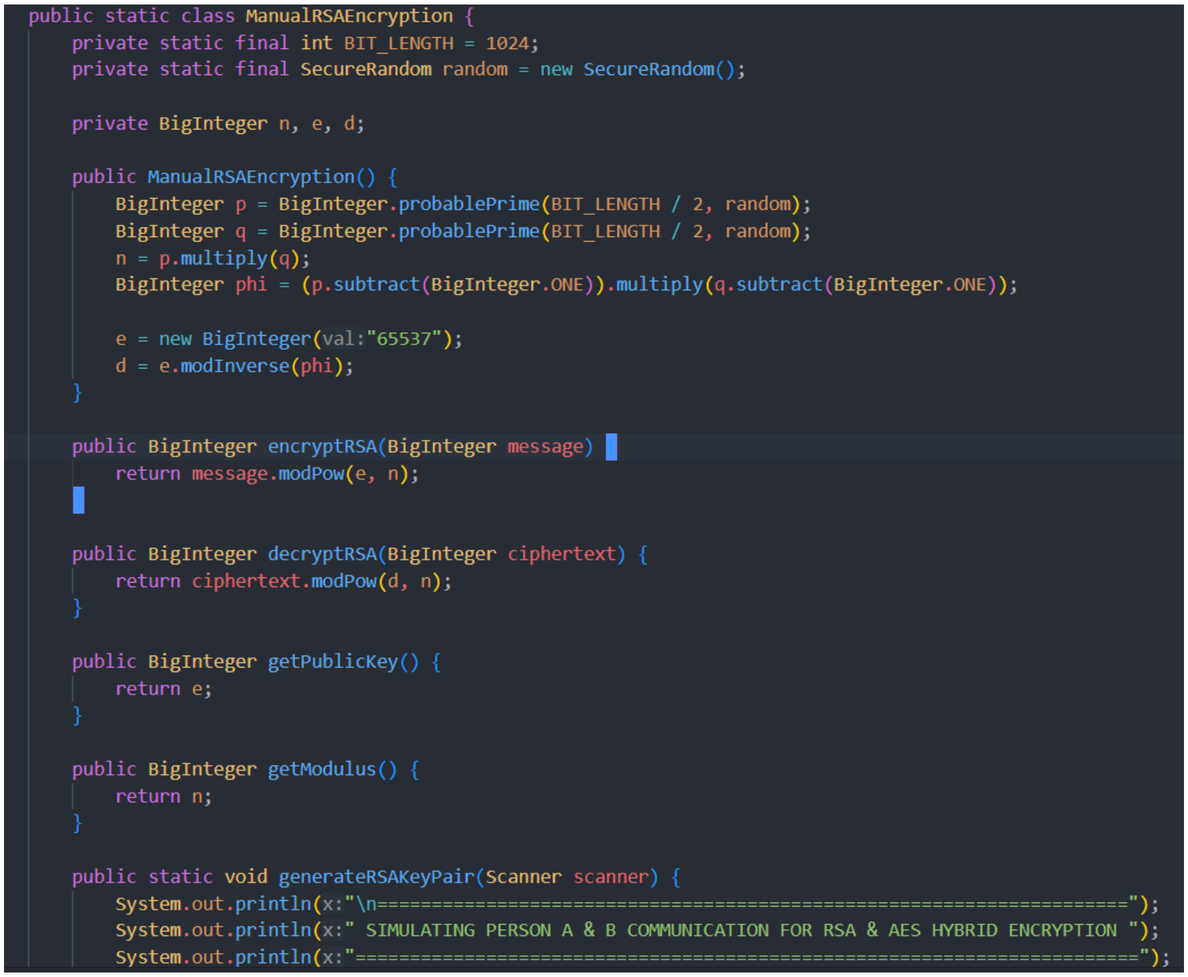


```java
// Product Cipher (Playfair + Rail Fence)
public static String productCipherEncrypt(String text, String key) {
    String playfairEncrypted = playfairCipherEncrypt(text, key);
    System.out.println("After Playfair Cipher: " + playfairEncrypted);

    String railFenceEncrypted = railFenceEncrypt(playfairEncrypted);
    System.out.println("After Rail Fence Cipher: " + railFenceEncrypted);

    return railFenceEncrypted;
}

// Product Cipher Decryption
public static String productCipherDecrypt(String text, String key) {
    String railFenceDecrypted = railFenceDecrypt(text);
    System.out.println("After Rail Fence Decryption: " + railFenceDecrypted);

    String playfairDecrypted = playfairCipherDecrypt(railFenceDecrypted, key);
    System.out.println("After Playfair Decryption: " + playfairDecrypted);

    return playfairDecrypted;
}
```

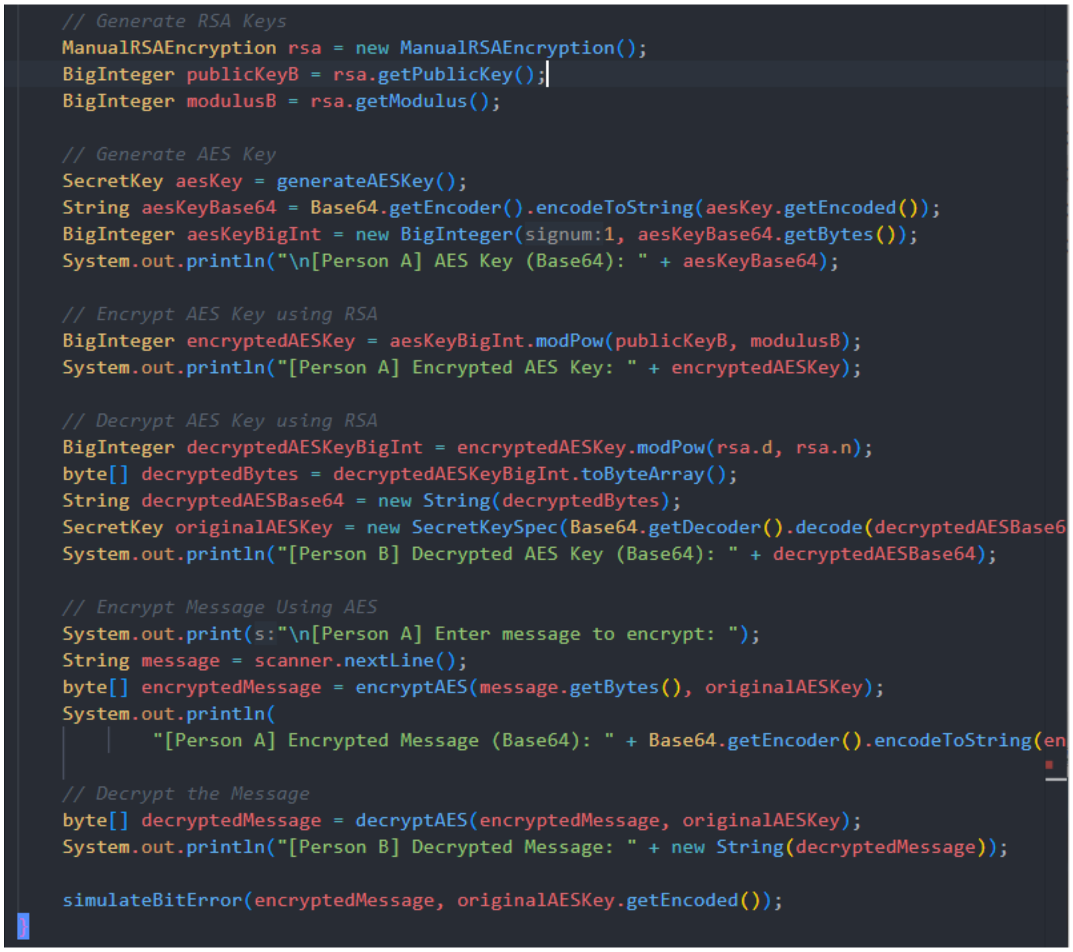
d. Implementation of Modern Cryptography (RSA Key Exchange Implementation

i. Process:

1. Person A generates an AES key for encryption.
2. Person A encrypts the AES key using RSA (Public Key of Person B).
3. Person B decrypts the AES key using RSA (Private Key).
4. Person A encrypts the message using AES and sends it to Person B.
5. Person B decrypts the message using the decrypted AES key.
6. A bit-flip error is simulated to analyze the impact on AES decryption.



```java
public static class ManualRSAEncryption {
    private static final int BIT_LENGTH = 1024;
    private static final SecureRandom random = new SecureRandom();

    private BigInteger n, e, d;

    public ManualRSAEncryption() {
        BigInteger p = BigInteger.probablePrime(BIT_LENGTH / 2, random);
        BigInteger q = BigInteger.probablePrime(BIT_LENGTH / 2, random);
        n = p.multiply(q);
        BigInteger phi = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));

        e = new BigInteger(val:"65537");
        d = e.modInverse(phi);
    }

    public BigInteger encryptRSA(BigInteger message) 
        return message.modPow(e, n);
    

    public BigInteger decryptRSA(BigInteger ciphertext) {
        return ciphertext.modPow(d, n);
    }

    public BigInteger getPublicKey() {
        return e;
    }

    public BigInteger getModulus() {
        return n;
    }

    public static void generateRSAKeyPair(Scanner scanner) {
        System.out.println(x:"\n=================================================================");
        System.out.println(x:" SIMULATING PERSON A & B COMMUNICATION FOR RSA & AES HYBRID ENCRYPTION ");
        System.out.println(x:"=================================================================");
```

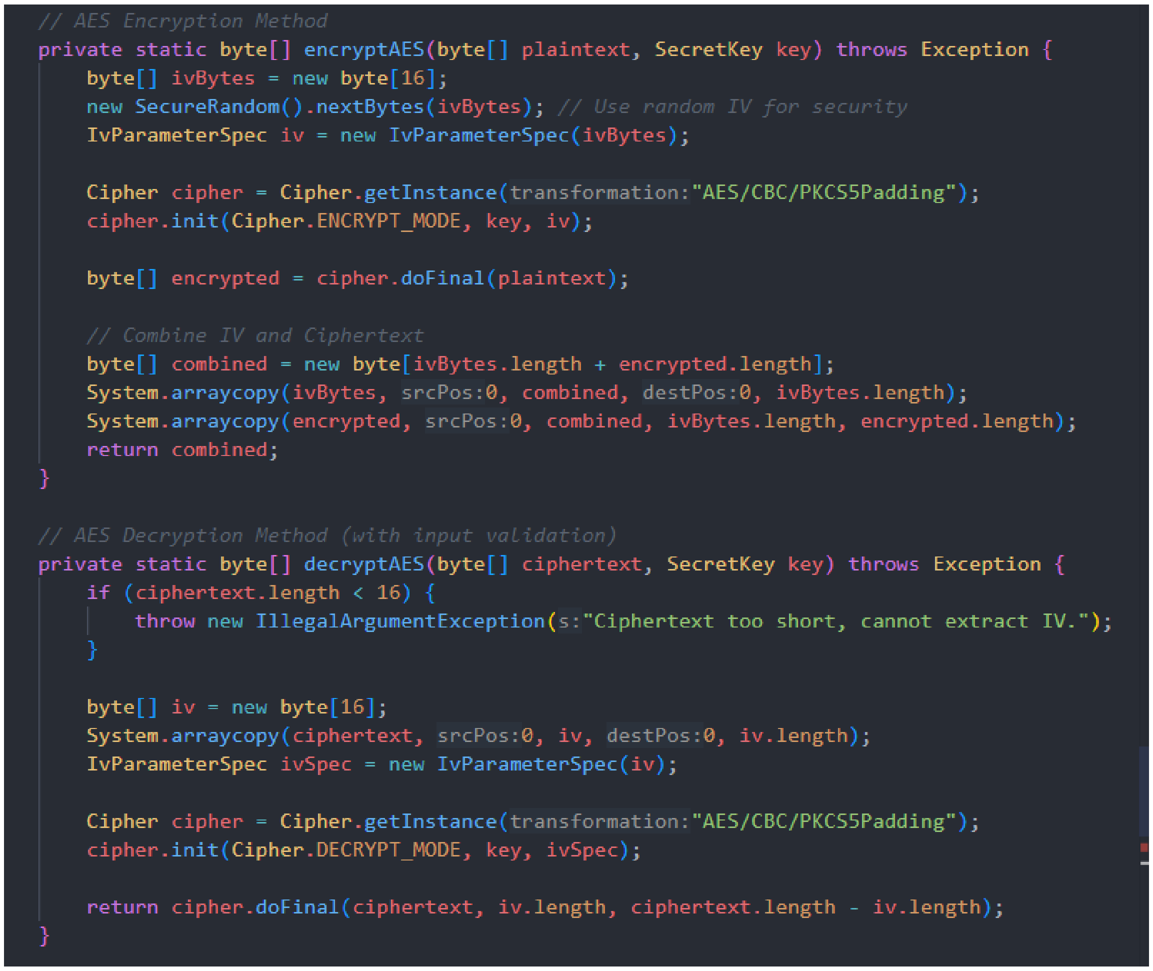


```java
    // Generate RSA Keys
    ManualRSAEncryption rsa = new ManualRSAEncryption();
    BigInteger publicKeyB = rsa.getPublicKey();
    BigInteger modulusB = rsa.getModulus();

    // Generate AES Key
    SecretKey aesKey = generateAESKey();
    String aesKeyBase64 = Base64.getEncoder().encodeToString(aesKey.getEncoded());
    BigInteger aesKeyBigInt = new BigInteger(signum:1, aesKeyBase64.getBytes());
    System.out.println("\n[Person A] AES Key (Base64): " + aesKeyBase64);

    // Encrypt AES Key using RSA
    BigInteger encryptedAESKey = aesKeyBigInt.modPow(publicKeyB, modulusB);
    System.out.println("[Person A] Encrypted AES Key: " + encryptedAESKey);

    // Decrypt AES Key using RSA
    BigInteger decryptedAESKeyBigInt = encryptedAESKey.modPow(rsa.d, rsa.n);
    byte[] decryptedBytes = decryptedAESKeyBigInt.toByteArray();
    String decryptedAESBase64 = new String(decryptedBytes);
    SecretKey originalAESKey = new SecretKeySpec(Base64.getDecoder().decode(decryptedAESBase6
    System.out.println("[Person B] Decrypted AES Key (Base64): " + decryptedAESBase64);

    // Encrypt Message Using AES
    System.out.print(s:"\n[Person A] Enter message to encrypt: ");
    String message = scanner.nextLine();
    byte[] encryptedMessage = encryptAES(message.getBytes(), originalAESKey);
    System.out.println(
            "[Person A] Encrypted Message (Base64): " + Base64.getEncoder().encodeToString(en

    // Decrypt the Message
    byte[] decryptedMessage = decryptAES(encryptedMessage, originalAESKey);
    System.out.println("[Person B] Decrypted Message: " + new String(decryptedMessage));

    simulateBitError(encryptedMessage, originalAESKey.getEncoded());
}
```

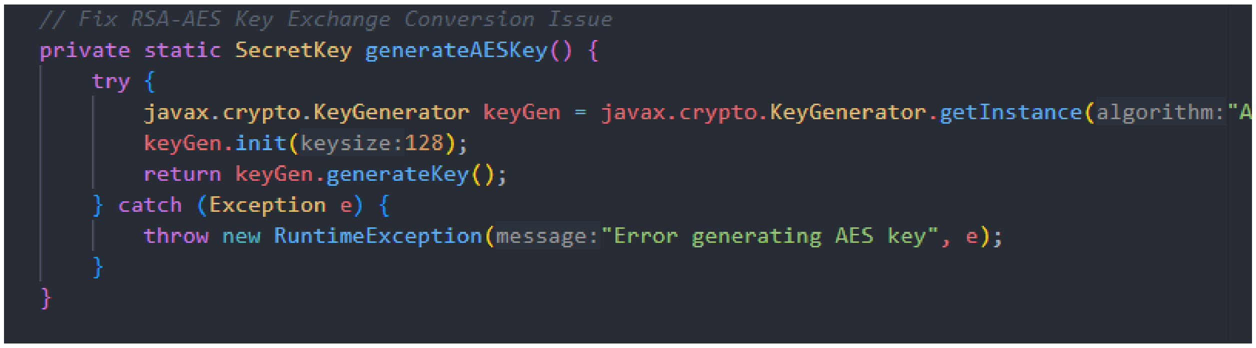
e. AES Encryption (CBC Mode) Implementation

   i. Process:

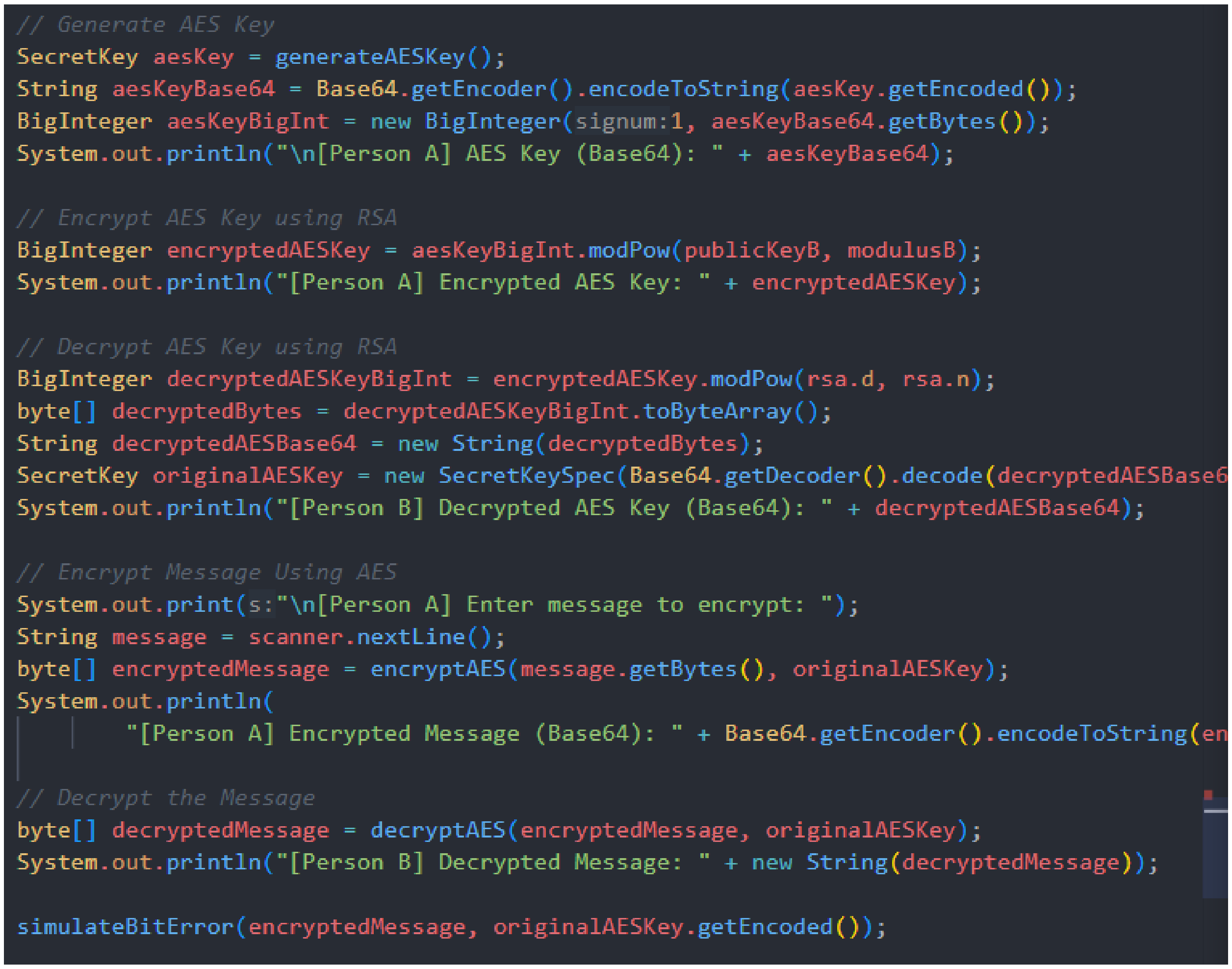
      1. Person A encrypts a plaintext message using AES.
      2. Person B decrypts the received ciphertext using the shared AES key.
      3. A simulated bit error is introduced in the ciphertext to analyze its effect on decryption.



```java
// AES Encryption Method
private static byte[] encryptAES(byte[] plaintext, SecretKey key) throws Exception {
    byte[] ivBytes = new byte[16];
    new SecureRandom().nextBytes(ivBytes); // Use random IV for security
    IvParameterSpec iv = new IvParameterSpec(ivBytes);

    Cipher cipher = Cipher.getInstance(transformation:"AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, key, iv);

    byte[] encrypted = cipher.doFinal(plaintext);

    // Combine IV and Ciphertext
    byte[] combined = new byte[ivBytes.length + encrypted.length];
    System.arraycopy(ivBytes, srcPos:0, combined, destPos:0, ivBytes.length);
    System.arraycopy(encrypted, srcPos:0, combined, ivBytes.length, encrypted.length);
    return combined;
}

// AES Decryption Method (with input validation)
private static byte[] decryptAES(byte[] ciphertext, SecretKey key) throws Exception {
    if (ciphertext.length < 16) {
        throw new IllegalArgumentException(s:"Ciphertext too short, cannot extract IV.");
    }

    byte[] iv = new byte[16];
    System.arraycopy(ciphertext, srcPos:0, iv, destPos:0, iv.length);
    IvParameterSpec ivSpec = new IvParameterSpec(iv);

    Cipher cipher = Cipher.getInstance(transformation:"AES/CBC/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);

    return cipher.doFinal(ciphertext, iv.length, ciphertext.length - iv.length);
}
```



```
// Fix RSA-AES Key Exchange Conversion Issue
private static SecretKey generateAESKey() {
    try {
        javax.crypto.KeyGenerator keyGen = javax.crypto.KeyGenerator.getInstance(algorithm:"A
        keyGen.init(keysize:128);
        return keyGen.generateKey();
    } catch (Exception e) {
        throw new RuntimeException(message:"Error generating AES key", e);
    }
}
```



```
// Generate AES Key
SecretKey aesKey = generateAESKey();
String aesKeyBase64 = Base64.getEncoder().encodeToString(aesKey.getEncoded());
BigInteger aesKeyBigInt = new BigInteger(signum:1, aesKeyBase64.getBytes());
System.out.println("\n[Person A] AES Key (Base64): " + aesKeyBase64);

// Encrypt AES Key using RSA
BigInteger encryptedAESKey = aesKeyBigInt.modPow(publicKeyB, modulusB);
System.out.println("[Person A] Encrypted AES Key: " + encryptedAESKey);

// Decrypt AES Key using RSA
BigInteger decryptedAESKeyBigInt = encryptedAESKey.modPow(rsa.d, rsa.n);
byte[] decryptedBytes = decryptedAESKeyBigInt.toByteArray();
String decryptedAESBase64 = new String(decryptedBytes);
SecretKey originalAESKey = new SecretKeySpec(Base64.getDecoder().decode(decryptedAESBase6
System.out.println("[Person B] Decrypted AES Key (Base64): " + decryptedAESBase64);

// Encrypt Message Using AES
System.out.print(s:"\n[Person A] Enter message to encrypt: ");
String message = scanner.nextLine();
byte[] encryptedMessage = encryptAES(message.getBytes(), originalAESKey);
System.out.println(
        "[Person A] Encrypted Message (Base64): " + Base64.getEncoder().encodeToString(en

// Decrypt the Message
byte[] decryptedMessage = decryptAES(encryptedMessage, originalAESKey);
System.out.println("[Person B] Decrypted Message: " + new String(decryptedMessage));

simulateBitError(encryptedMessage, originalAESKey.getEncoded());
```

Comparison and Discussion of the Result

1. Comparison of Classical Cipher
    a. Playfair Cipher
        i. The Playfair Cipher encrypts text by replacing digraphs (pairs of letters) using a 5×5 matrix.
        ii. As observed, the formatted text is modified before encryption (e.g., inserting "X" between duplicate letters and at the end).
        iii. The encryption and decryption processes are fast, taking around 1 ms, indicating that substitution-based encryption is computationally lightweight.
        iv. The output ciphertext maintains letter frequency patterns, making it susceptible to frequency analysis attacks.
        v. Discussion:
            1. While the Playfair Cipher provides better security than monoalphabetic substitution ciphers, it is still vulnerable to pattern recognition. The presence of digraphs instead of single-character substitutions adds some security, but the method lacks diffusion, making it relatively weak against modern cryptanalysis. ￼

b. Rail Fence Cipher (Depth=3)
    i. The Rail Fence Cipher encrypts by arranging plaintext in a zigzag pattern across multiple rows.
    ii. The ciphertext appears scrambled due to transposition, but the frequency of letters remains unchanged.
    iii. The encryption and decryption times were 0 ms, showing that transposition-based encryption is computationally efficient.
    iv. The cipher can be broken easily using rail depth brute-force attacks.
    v. Discussion:
        1. The Rail Fence Cipher is not secure for practical encryption because it only rearranges the plaintext without altering individual characters. This makes it vulnerable to anagramming and permutation-based attacks. However, it can be useful when combined with other encryption methods, such as substitution ciphers.

c. Product Cipher (Playfair + Rail Fence)
    i. This method first encrypts plaintext using the Playfair Cipher, then applies the Rail Fence Cipher to the Playfair output.
    ii. The result is a ciphertext that is both substituted and transposed, making it harder to analyze through simple frequency attacks.
    iii. Encryption and decryption times remain low (1 ms + 0 ms), meaning the additional layer of security does not significantly affect performance.
    iv. Discussion
        1. The Product Cipher provides better security than either Playfair or Rail Fence alone, as it benefits from both confusion (substitution) and diffusion (transposition). However, modern cryptanalysis techniques can still break it with statistical analysis and brute-force decryption. While this

method enhances classical encryption, it remains weaker than modern cryptographic standards.

2. Comparison of Modern Encryption Methods

d. RSA Key Exchange
    i. The RSA algorithm is used for secure key exchange rather than direct encryption of messages.
    ii. The generated RSA key pair consists of public and private keys, ensuring asymmetric encryption security.
    iii. The encryption and decryption of the AES key via RSA were successful, proving that RSA is effective for secure key distribution.
    iv. Discussion:
        1. RSA is computationally intensive compared to classical ciphers but provides strong security due to its reliance on prime factorization complexity. However, RSA is not suitable for encrypting large messages due to its high processing overhead, making it best suited for key exchange in hybrid encryption systems.

e. AES Encryption (CBC Mode)
    i. AES encryption produces complex ciphertext, ensuring high security.
    ii. The encryption and decryption times are low, making AES efficient for practical applications.
    iii. An intentional bit error was introduced into the ciphertext to analyze error propagation in AES-CBC mode.
    iv. The decryption process resulted in corrupted plaintext, proving that error propagation is significant in block ciphers.
    v. DIscussion:
        1. AES in CBC mode provides strong security, but it is sensitive to transmission errors. A single bit-flip affects multiple blocks due to chaining, making it crucial to use error detection mechanisms such as message authentication codes (MACs) or digital signatures in real-world applications.

2. Key Observations and Comparisons

Feature	Playfair Cipher	Rail Fence Cipher	Product Cipher (Playfair + Rail Fence)	RSA (Key Exchange)	AES (CBC Mode)
Type	Substitution	Transposition	Hybrid	Asymmetric	Symmetric
Encryption Speed	Fast	Very Fast	Fast	Slow	Fast
Decryption Speed	Fast	Very Fast	Fast	Slow	Fast
Security Level	Low	Low	Moderate	High	Very High
Vulnerability	Frequency Analysis	Depth Brute Force	Statistical Analysis	Quantum Computing Attacks	Key Management & Bit Errors
Usability	Simple	Simple	Moderate	Secure Key Exchange	Widely Used in Modern Security

3. Discussion
    a. Security vs. Performance Trade-Off
        i. The classical ciphers (Playfair and Rail Fence) offer fast encryption and decryption times but low security, making them unsuitable for modern secure communication. While the Product Cipher improves security, it still lacks the robustness of modern cryptographic techniques.
        ii. On the other hand, RSA and AES provide strong security but at a computational cost. RSA is best suited for secure key exchange, while AES is ideal for encrypting large data efficiently. The error analysis in AES-CBC mode shows that minor bit errors can cause significant corruption, which must be managed in practical applications.
    b. Best Use Cases for Each Cipher
        i. Playfair Cipher: Best used for simple, low-security applications (historical context, puzzles).
        ii. Rail Fence Cipher: Suitable for encoding messages with light security needs.
        iii. Product Cipher: Offers better security than single classical ciphers but is still not secure for modern applications.
        iv. RSA: Primarily for secure key exchange in hybrid encryption schemes.
        v. AES: The most practical choice for secure data encryption in real-world applications.

Conclusion

This project executes and studies traditional and contemporary cryptographic approaches while showcasing their practical features as well as their vulnerabilities and applications. When Playfair joins the Rail Fence inside a product ciphers the methods of substitution and transposition work together to strengthen encryption security while remaining susceptible to analysis attempts. Security through encryption is achieved effectively by combining RSA key exchange with AES data encryption to demonstrate current powerful cryptographic methods. The assessment between classical and modern encryption methods shows that though classic ciphers function quickly their vulnerable security standards make them unfit for current application needs.

The project unites cryptographic concepts with operational encryption implementations. Through this investigation, we learn about the necessity of secure communication while gaining knowledge that helps improve encryption methods to deal with emerging security threats. This study demonstrates how the research compares encryption speeds to determine security weaknesses which demonstrates why selecting appropriate crypto methods requires specific use purposes. Although classical ciphers have historical importance the modern secure standards of RSA and AES remain essential for digital communication protection in contemporary technology environments.

Reference

Sugirtham, N., Jenny, R. S., Thiyaneswaran, B., Kumarganesh, S., Venkatesan, C., Sagayam, K. M., Dang, L., Dinh, L., & Dang, H. (2024). Modified Playfair for Text File Encryption and Meticulous Decryption with Arbitrary Fillers by Septenary Quadrate Pattern. *the International Journal of Networked and Distributed Computing*, *12*(1), 108–118. https://doi.org/10.1007/s44227-023-00019-4

Singh, S. (2020). *A novel technique for enhancement of the security of Playfair Cipher*. International Journal of Computer Engineering in Research Trends. https://www.ijcert.org/index.php/ijcert/article/view/831

Nahar, Khairun & Chakraborty, Partha. (2020). Improved Approach of Rail Fence for Enhancing Security. International Journal of Innovative Technology and Exploring Engineering. 9. 583-585. 10.35940/ijitee.I7637.079920.

Muazu, I. M. (2020). Formulation of an improved hybrid cipher system. *www.academia.edu*. https://www.academia.edu/101660502/Formulation_of_an_Improved_Hybrid_Cipher_System

Appendix

1. Link for GitHub Source Code
   https://github.com/WONG-ZHI-LIN/Cryptography_Asg/tree/1be78e4724c8b2acfbac95d40858016478700970
2. User Manual for Cryptography CLI tool

Introduction

The CryptographyCLI program is a command-line tool that provides various cryptographic functions, including:

- Playfair Cipher (Classical encryption technique)
- Rail Fence Cipher (Transposition-based encryption)
- Product Cipher (Combination of Playfair and Rail Fence)

- Hybrid RSA-AES Encryption (Secure communication using asymmetric and symmetric encryption)

This manual explains how to run the program, input commands, and understand the expected outputs.

Prerequisites

Required Setup

Before running the program, ensure that:

- You have **Java installed** (JDK 8 or later).
- The CryptographyCLI.java file is in your working directory.

Compiling and Running the Program

Step 1: Open a Terminal

If you're using:

- **Windows** → Open **Command Prompt (cmd)**
- **Linux/Mac** → Open **Terminal**

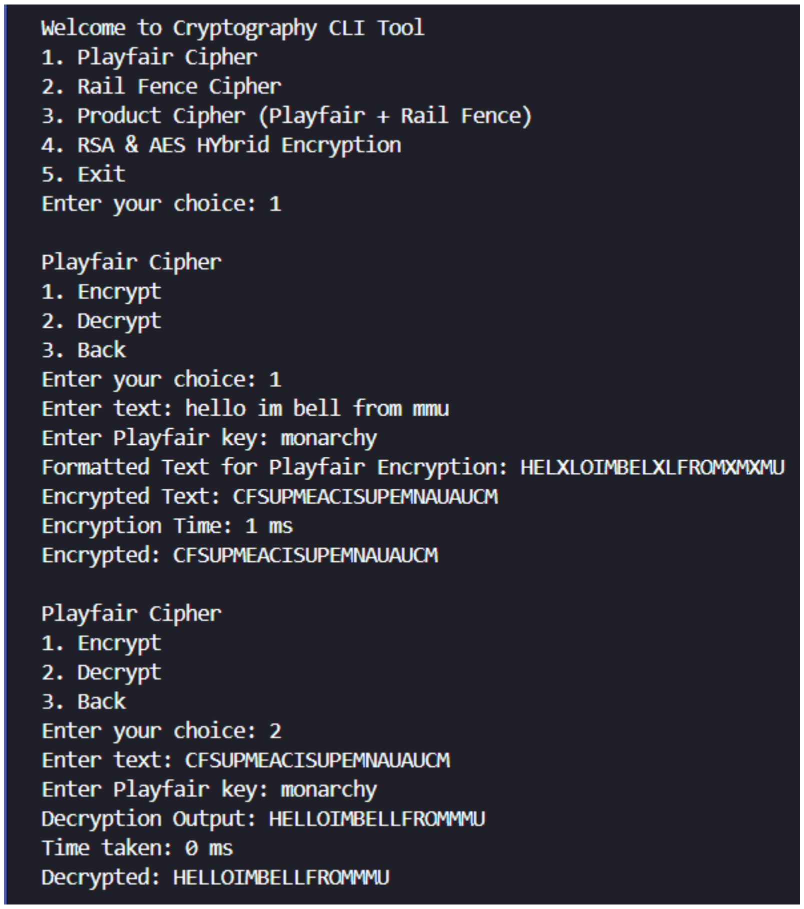
Step 2: Navigate to the Program's Directory

Use the cd command to go to the folder containing CryptographyCLI.java

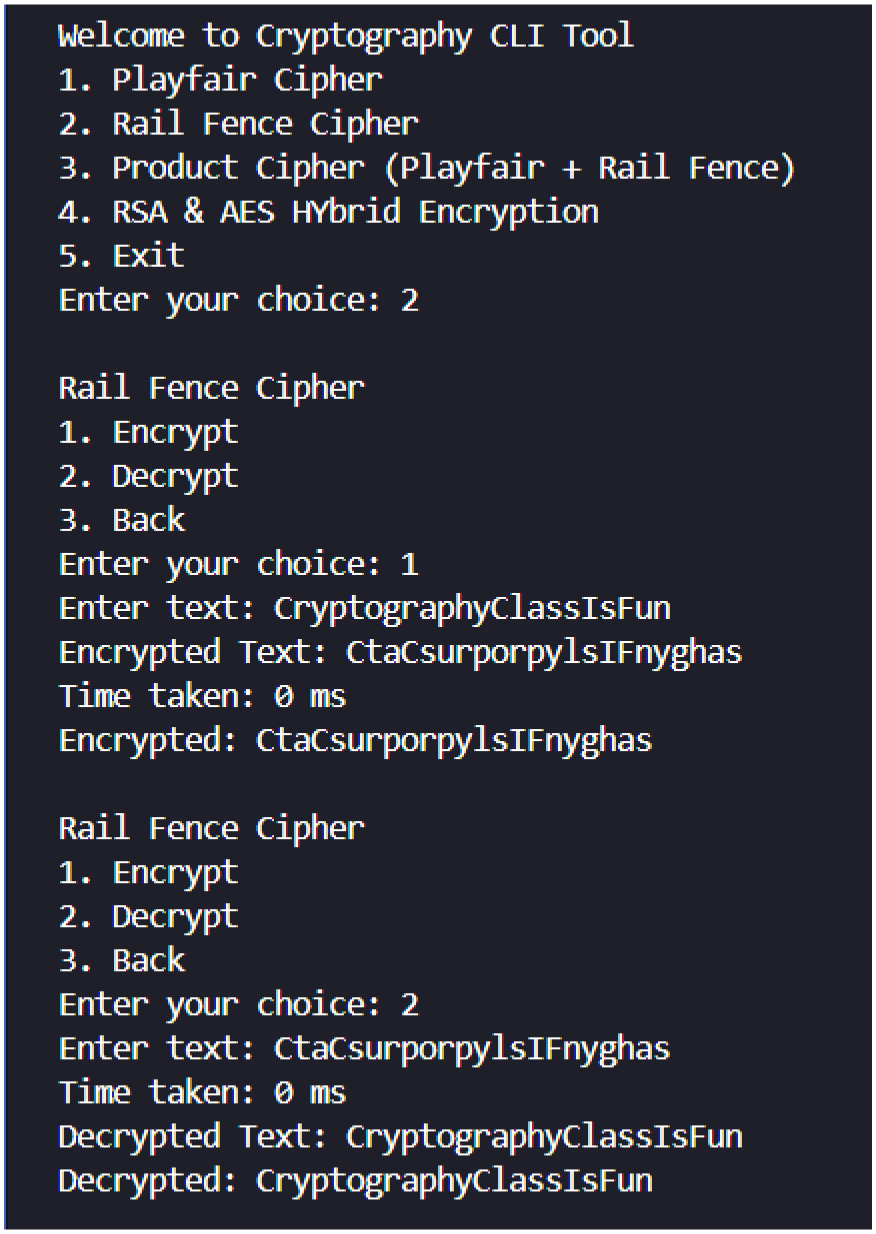
Step 3: Run and execute your program

How To Use The Program

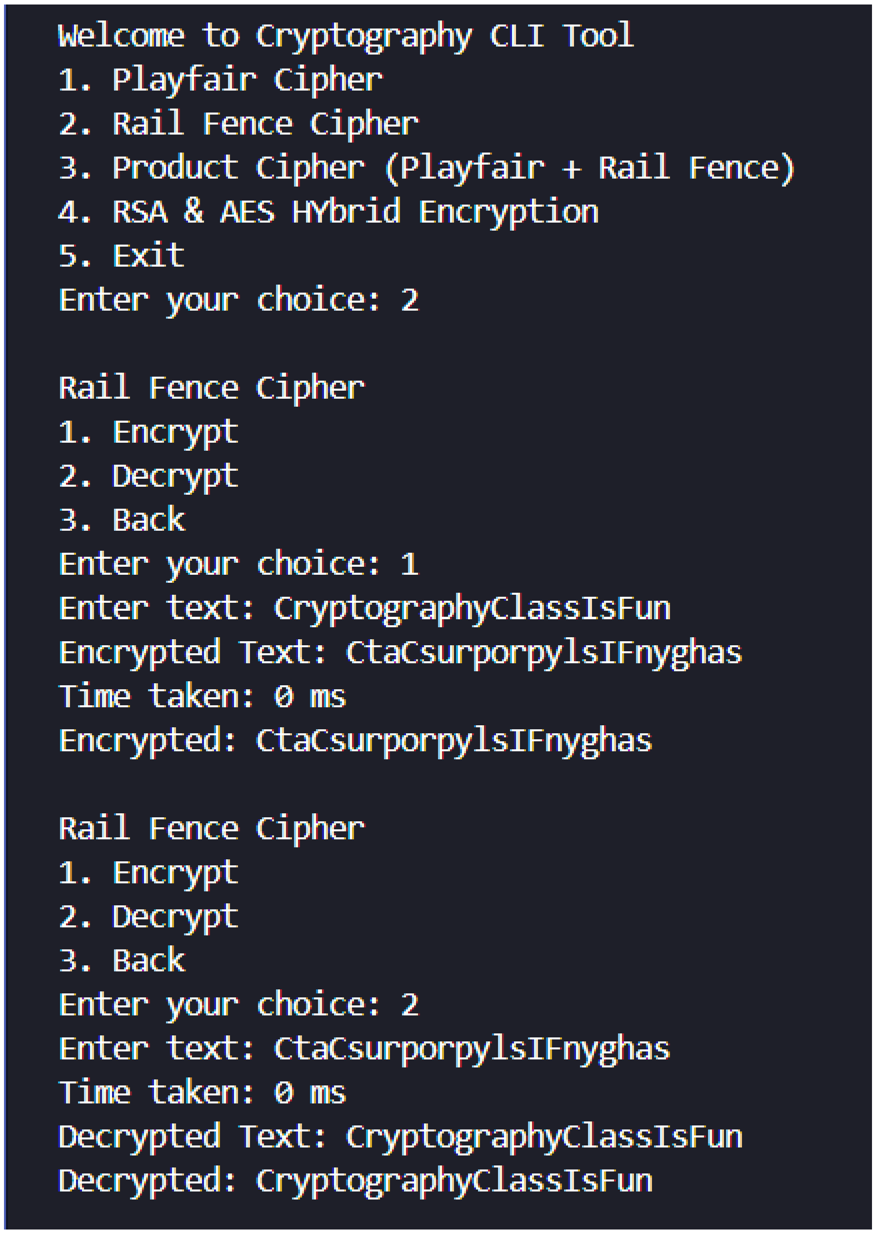
Once entering the program, you will see the main menu. User are allowed to choose for option they want.



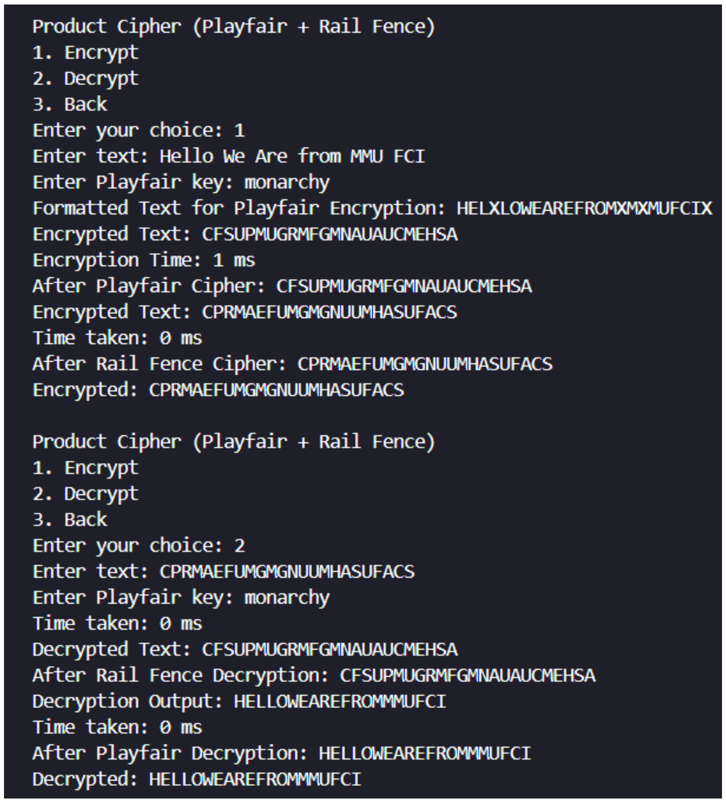
```
Welcome to Cryptography CLI Tool
1. Playfair Cipher
2. Rail Fence Cipher
3. Product Cipher (Playfair + Rail Fence)
4. RSA & AES HYbrid Encryption
5. Exit
Enter your choice: 1

Playfair Cipher
1. Encrypt
2. Decrypt
3. Back
Enter your choice: 1
Enter text: hello im bell from mmu
Enter Playfair key: monarchy
Formatted Text for Playfair Encryption: HELXLOIMBELXLFROMXMXMU
Encrypted Text: CFSUPMEACISUPEMNAUAUCM
Encryption Time: 1 ms
Encrypted: CFSUPMEACISUPEMNAUAUCM

Playfair Cipher
1. Encrypt
2. Decrypt
3. Back
Enter your choice: 2
Enter text: CFSUPMEACISUPEMNAUAUCM
Enter Playfair key: monarchy
Decryption Output: HELLOIMBELLFROMMMU
Time taken: 0 ms
Decrypted: HELLOIMBELLFROMMMU
```



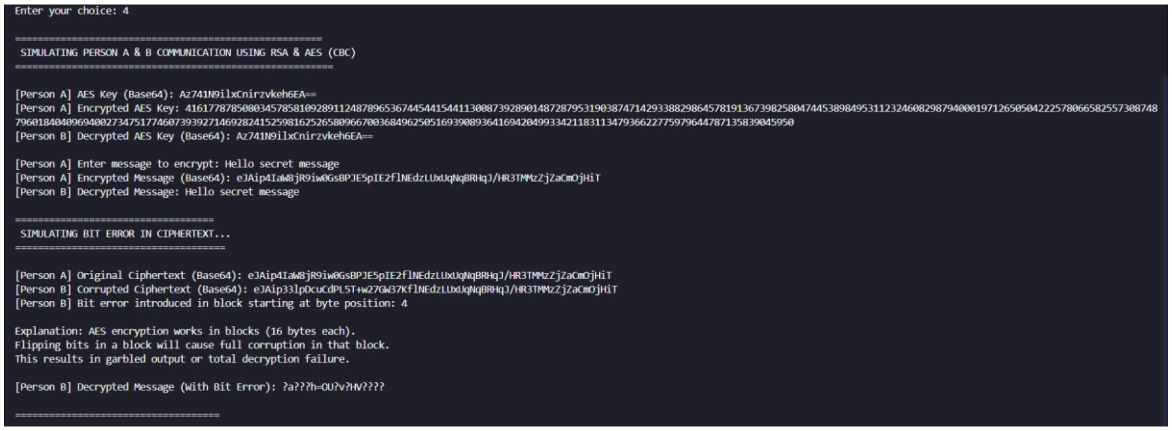
```
Welcome to Cryptography CLI Tool
1. Playfair Cipher
2. Rail Fence Cipher
3. Product Cipher (Playfair + Rail Fence)
4. RSA & AES HYbrid Encryption
5. Exit
Enter your choice: 2

Rail Fence Cipher
1. Encrypt
2. Decrypt
3. Back
Enter your choice: 1
Enter text: CryptographyClassIsFun
Encrypted Text: CtaCsurporpylsIFnyghas
Time taken: 0 ms
Encrypted: CtaCsurporpylsIFnyghas

Rail Fence Cipher
1. Encrypt
2. Decrypt
3. Back
Enter your choice: 2
Enter text: CtaCsurporpylsIFnyghas
Time taken: 0 ms
Decrypted Text: CryptographyClassIsFun
Decrypted: CryptographyClassIsFun
```



```
Welcome to Cryptography CLI Tool
1. Playfair Cipher
2. Rail Fence Cipher
3. Product Cipher (Playfair + Rail Fence)
4. RSA & AES HYbrid Encryption
5. Exit
Enter your choice: 2

Rail Fence Cipher
1. Encrypt
2. Decrypt
3. Back
Enter your choice: 1
Enter text: CryptographyClassIsFun
Encrypted Text: CtaCsurporpylsIFnyghas
Time taken: 0 ms
Encrypted: CtaCsurporpylsIFnyghas

Rail Fence Cipher
1. Encrypt
2. Decrypt
3. Back
Enter your choice: 2
Enter text: CtaCsurporpylsIFnyghas
Time taken: 0 ms
Decrypted Text: CryptographyClassIsFun
Decrypted: CryptographyClassIsFun
```



```
Product Cipher (Playfair + Rail Fence)
1. Encrypt
2. Decrypt
3. Back
Enter your choice: 1
Enter text: Hello We Are from MMU FCI
Enter Playfair key: monarchy
Formatted Text for Playfair Encryption: HELXLOWEAREFROMXMXMUFCIX
Encrypted Text: CFSUPMUGRMFGMNAUAUCMEHSA
Encryption Time: 1 ms
After Playfair Cipher: CFSUPMUGRMFGMNAUAUCMEHSA
Encrypted Text: CPRMAEFUMGMGNUUMHASUFACS
Time taken: 0 ms
After Rail Fence Cipher: CPRMAEFUMGMGNUUMHASUFACS
Encrypted: CPRMAEFUMGMGNUUMHASUFACS

Product Cipher (Playfair + Rail Fence)
1. Encrypt
2. Decrypt
3. Back
Enter your choice: 2
Enter text: CPRMAEFUMGMGNUUMHASUFACS
Enter Playfair key: monarchy
Time taken: 0 ms
Decrypted Text: CFSUPMUGRMFGMNAUAUCMEHSA
After Rail Fence Decryption: CFSUPMUGRMFGMNAUAUCMEHSA
Decryption Output: HELLOWEAREFROMMMUFCI
Time taken: 0 ms
After Playfair Decryption: HELLOWEAREFROMMMUFCI
Decrypted: HELLOWEAREFROMMMUFCI
```



```
Enter your choice: 4

=======================================================
 SIMULATING PERSON A & B COMMUNICATION USING RSA & AES (CBC)
=========================================================

[Person A] AES Key (Base64): Az741N9ilxCnirzvkeh6EA==
[Person A] Encrypted AES Key: 41617787850803457858109289112487896536744544154411300873928901487287953190387471429338829864578191367398258047445389849531123246082987940001971265050422257806658255730 8748
79601840409694002734751774607393927146928241525981625265809667003684962505169390893641694204993342118311347936622775979644787135839045950
[Person B] Decrypted AES Key (Base64): Az741N9ilxCnirzvkeh6EA==

[Person A] Enter message to encrypt: Hello secret message
[Person A] Encrypted Message (Base64): eJAip4IaW8jR9iw0GsBPJE5pIE2flNEdzLUxUqNqBRHqJ/HR3TMMzZjZaCmOjHiT
[Person B] Decrypted Message: Hello secret message

===================================
 SIMULATING BIT ERROR IN CIPHERTEXT...
=====================================

[Person A] Original Ciphertext (Base64): eJAip4IaW8jR9iw0GsBPJE5pIE2flNEdzLUxUqNqBRHqJ/HR3TMMzZjZaCmOjHiT
[Person B] Corrupted Ciphertext (Base64): eJAip33lpDcuCdPL5T+w27GW37KflNEdzLUxUqNqBRHqJ/HR3TMMzZjZaCmOjHiT
[Person B] Bit error introduced in block starting at byte position: 4

Explanation: AES encryption works in blocks (16 bytes each).
Flipping bits in a block will cause full corruption in that block.
This results in garbled output or total decryption failure.

[Person B] Decrypted Message (With Bit Error): ?a???h=OU?v?HV????

====================================
```