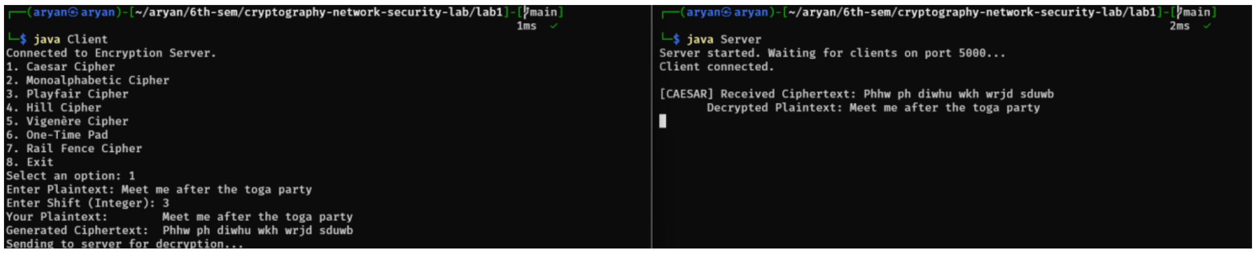
Name: Aryan Sharma
Reg. No.: 23BCE1320
Subject: Cryptography & Network Security
Assignment: Lab 1

Output:

Caesar Cipher: A simple substitution technique where each letter in the plaintext is replaced by a letter a fixed number of positions down the alphabet. For example, with a shift of 3, 'A' would be replaced by 'D', effectively performing a modular arithmetic shift on the character set

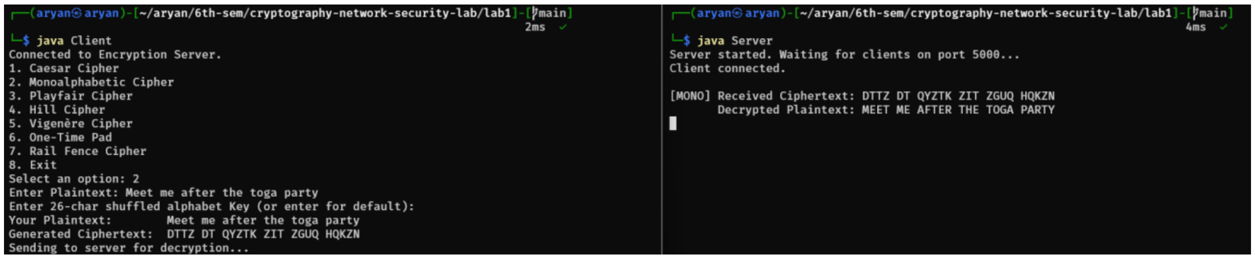


```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                        1ms ✓
└─$ java Client
Connected to Encryption Server.
1. Caesar Cipher
2. Monoalphabetic Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigenère Cipher
6. One-Time Pad
7. Rail Fence Cipher
8. Exit
Select an option: 1
Enter Plaintext: Meet me after the toga party
Enter Shift (Integer): 3
Your Plaintext:        Meet me after the toga party
Generated Ciphertext:  Phhw ph diwhu wkh wrjd sduwb
Sending to server for decryption...
```

```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                        2ms ✓
└─$ java Server
Server started. Waiting for clients on port 5000...
Client connected.

[CAESAR] Received Ciphertext: Phhw ph diwhu wkh wrjd sduwb
       Decrypted Plaintext: Meet me after the toga party
█
```

Monoalphabetic Cipher: A substitution cipher that maps each letter of the plaintext alphabet to a unique letter in a randomly permuted cipher alphabet. Unlike the Caesar cipher, the relationship between plain and cipher letters is arbitrary rather than a uniform shift, which increases the key space.

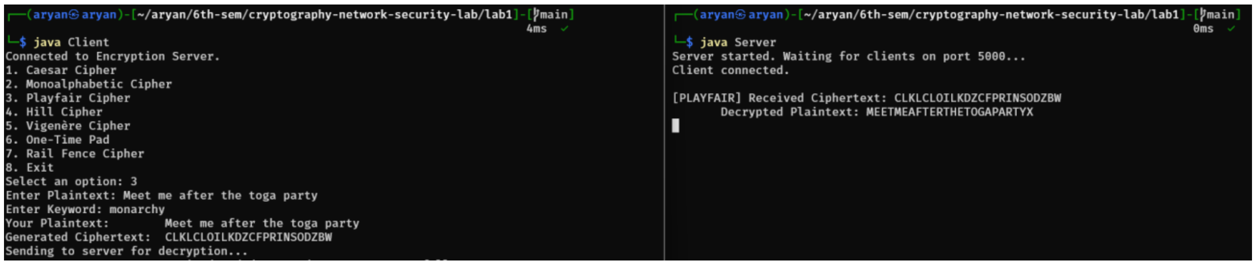


```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                        2ms ✓
└─$ java Client
Connected to Encryption Server.
1. Caesar Cipher
2. Monoalphabetic Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigenère Cipher
6. One-Time Pad
7. Rail Fence Cipher
8. Exit
Select an option: 2
Enter Plaintext: Meet me after the toga party
Enter 26-char shuffled alphabet Key (or enter for default):
Your Plaintext:        Meet me after the toga party
Generated Ciphertext:  DTTZ DT QYZTK ZIT ZGUQ HQKZN
Sending to server for decryption...
```

```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                        4ms ✓
└─$ java Server
Server started. Waiting for clients on port 5000...
Client connected.

[MONO] Received Ciphertext: DTTZ DT QYZTK ZIT ZGUQ HQKZN
       Decrypted Plaintext: MEET ME AFTER THE TOGA PARTY
█
```

Playfair Cipher: A digraph substitution cipher that encrypts pairs of letters (digrams) rather than single characters. It utilizes a 5x5 matrix constructed from a keyword, applying specific geometric rules—such as being in the same row, column, or forming a rectangle—to determine the substitution.

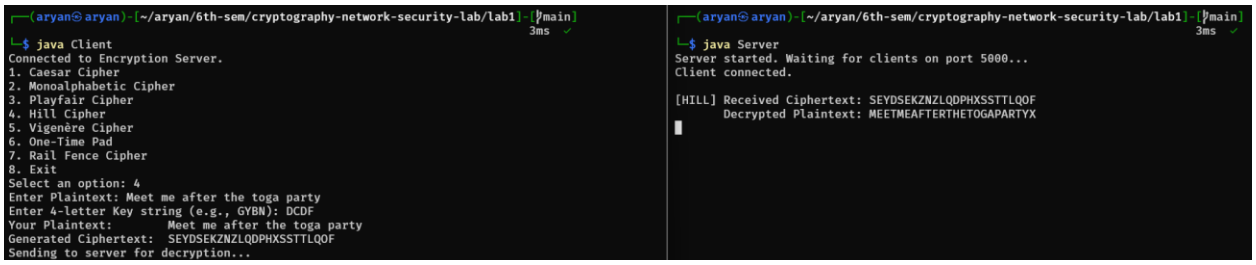


```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                        4ms ✓
└─$ java Client
Connected to Encryption Server.
1. Caesar Cipher
2. Monoalphabetic Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigenère Cipher
6. One-Time Pad
7. Rail Fence Cipher
8. Exit
Select an option: 3
Enter Plaintext: Meet me after the toga party
Enter Keyword: monarchy
Your Plaintext:        Meet me after the toga party
Generated Ciphertext:  CLKLCLOILKDZCFPRINSODZBW
Sending to server for decryption...
```

```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                        0ms ✓
└─$ java Server
Server started. Waiting for clients on port 5000...
Client connected.

[PLAYFAIR] Received Ciphertext: CLKLCLOILKDZCFPRINSODZBW
       Decrypted Plaintext: MEETMEAFTERTHETOGAPARTYX
█
```

Hill Cipher: A polygraphic substitution cipher based on linear algebra. It divides the plaintext into blocks of vectors and multiplies them by an invertible square matrix (the key) modulo 26 to produce the ciphertext, allowing it to mask letter frequencies effectively

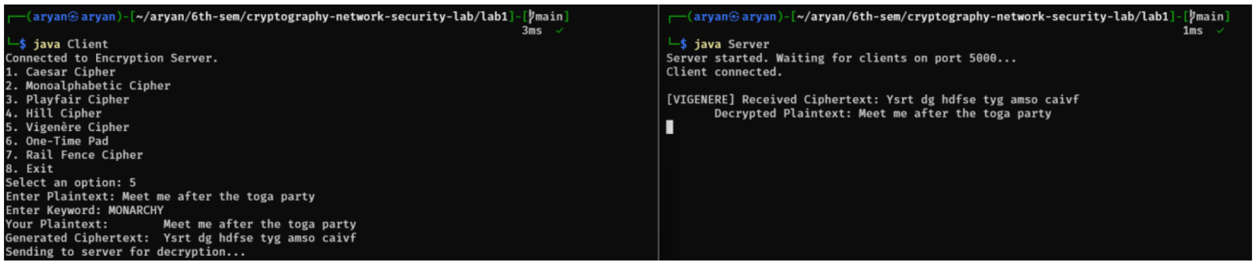


```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                          3ms ✓
└─$ java Client
Connected to Encryption Server.
1. Caesar Cipher
2. Monoalphabetic Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigenère Cipher
6. One-Time Pad
7. Rail Fence Cipher
8. Exit
Select an option: 4
Enter Plaintext: Meet me after the toga party
Enter 4-letter Key string (e.g., GYBN): DCDF
Your Plaintext:        Meet me after the toga party
Generated Ciphertext:  SEYDSEKZNZLQDPHXSSTTLQOF
Sending to server for decryption...
```

```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                          3ms ✓
└─$ java Server
Server started. Waiting for clients on port 5000...
Client connected.

[HILL] Received Ciphertext: SEYDSEKZNZLQDPHXSSTTLQOF
       Decrypted Plaintext: MEETMEAFTERTHETOGAPARTYX
```

Vignere Cipher: A polyalphabetic substitution system that employs a series of interwoven Caesar ciphers based on the letters of a keyword. By changing the shift value for each character according to the keyword, it disrupts the frequency patterns found in simple monoalphabetic substitutions.

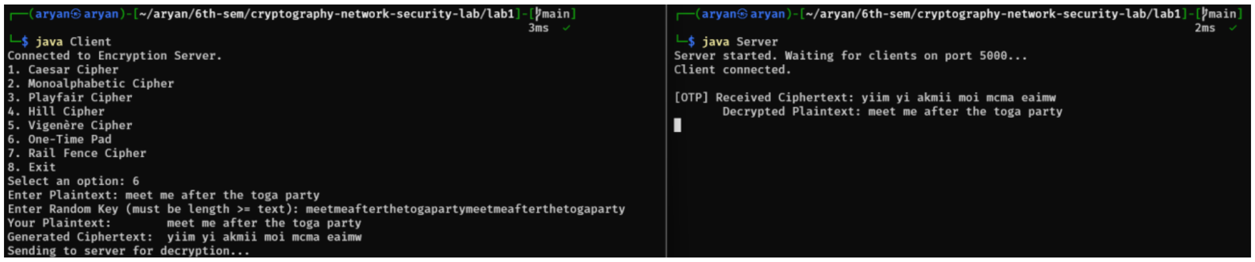


```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                          3ms ✓
└─$ java Client
Connected to Encryption Server.
1. Caesar Cipher
2. Monoalphabetic Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigenère Cipher
6. One-Time Pad
7. Rail Fence Cipher
8. Exit
Select an option: 5
Enter Plaintext: Meet me after the toga party
Enter Keyword: MONARCHY
Your Plaintext:        Meet me after the toga party
Generated Ciphertext:  Ysrt dg hdfse tyg amso caivf
Sending to server for decryption...
```

```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                          1ms ✓
└─$ java Server
Server started. Waiting for clients on port 5000...
Client connected.

[VIGENERE] Received Ciphertext: Ysrt dg hdfse tyg amso caivf
       Decrypted Plaintext: Meet me after the toga party
```

One-Time-Pad: An unbreakable encryption technique that pairs the plaintext with a truly random secret key of the same length. Because the key is random and used only once, the resulting ciphertext bears no statistical relationship to the plaintext, making it theoretically impossible to crack without the key.

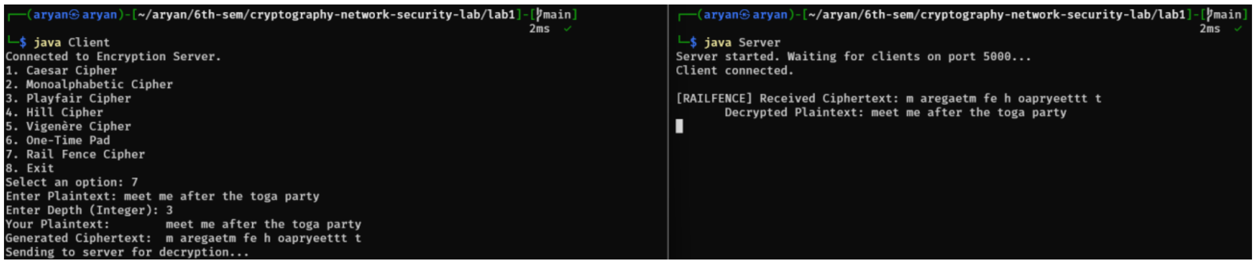


```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                          3ms ✓
└─$ java Client
Connected to Encryption Server.
1. Caesar Cipher
2. Monoalphabetic Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigenère Cipher
6. One-Time Pad
7. Rail Fence Cipher
8. Exit
Select an option: 6
Enter Plaintext: meet me after the toga party
Enter Random Key (must be length >= text): meetmeafterthetogapartymeetmeafterthetogaparty
Your Plaintext:        meet me after the toga party
Generated Ciphertext:  yiim yi akmii moi mcma eaimw
Sending to server for decryption...
```

```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                          2ms ✓
└─$ java Server
Server started. Waiting for clients on port 5000...
Client connected.

[OTP] Received Ciphertext: yiim yi akmii moi mcma eaimw
       Decrypted Plaintext: meet me after the toga party
```

Rail Fence Cipher: A transposition cipher that rearranges the position of characters rather than substituting them. The message is written in a zig-zag pattern across multiple "rails" (rows) and then read off row-by-row to create the scrambled sequence.



```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                          2ms ✓
└─$ java Client
Connected to Encryption Server.
1. Caesar Cipher
2. Monoalphabetic Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigenère Cipher
6. One-Time Pad
7. Rail Fence Cipher
8. Exit
Select an option: 7
Enter Plaintext: meet me after the toga party
Enter Depth (Integer): 3
Your Plaintext:        meet me after the toga party
Generated Ciphertext:  m aregaetm fe h oapryeettt t
Sending to server for decryption...
```

```
┌──(aryan㉿aryan)-[~/aryan/6th-sem/cryptography-network-security-lab/lab1]-[main]
                                                                          2ms ✓
└─$ java Server
Server started. Waiting for clients on port 5000...
Client connected.

[RAILFENCE] Received Ciphertext: m aregaetm fe h oapryeettt t
       Decrypted Plaintext: meet me after the toga party
```

Codes:

Server.java

```java
import java.io.*;
import java.net.*;

public class Server {
    private static final int PORT = 5000;

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server started. Waiting for clients on port " + PORT + "...");

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Client connected.");
                new ClientHandler(socket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class ClientHandler extends Thread {
    private Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (
            DataInputStream input = new DataInputStream(socket.getInputStream());
            DataOutputStream output = new DataOutputStream(socket.getOutputStream());
        ) {
            while (true) {
                String request = input.readUTF();
                if (request.equals("EXIT")) {
                    System.out.println("Client sent EXIT command.");
                    break;
                }

                String[] parts = request.split("::");
                if (parts.length < 3) {
                    output.writeUTF("Error: Invalid format");
                    continue;
                }

                String type = parts[0];
                String key = parts[1];
                String ciphertext = parts[2];
                String decryptedText = "";
                System.out.println("\n[" + type + "] Received Ciphertext: " + ciphertext);
                switch (type) {
                    case "CAESAR": decryptedText = CryptoUtils.decryptCaesar(ciphertext, key); break;
                    case "MONO": decryptedText = CryptoUtils.decryptMono(ciphertext, key); break;
                    case "PLAYFAIR": decryptedText = CryptoUtils.decryptPlayfair(ciphertext, key);
break;
                    case "HILL": decryptedText = CryptoUtils.decryptHill(ciphertext, key); break;
                    case "VIGENERE": decryptedText = CryptoUtils.decryptVigenere(ciphertext, key);
break;
                    case "OTP": decryptedText = CryptoUtils.decryptVigenere(ciphertext, key); break;
                    case "RAILFENCE": decryptedText = CryptoUtils.decryptRailFence(ciphertext, key);
break;
                    default: decryptedText = "Unknown Cipher Type";
                }

                System.out.println("       Decrypted Plaintext: " + decryptedText);

                output.writeUTF("Server received and decrypted message successfully.");
            }
        } catch (EOFException e) {
            System.out.println("Client disconnected.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Client {
    private static final String SERVER_ADDRESS = "localhost";
    private static final int PORT = 5000;

    public static void main(String[] args) {
        try (Socket socket = new Socket(SERVER_ADDRESS, PORT);
             DataInputStream input = new DataInputStream(socket.getInputStream());
             DataOutputStream output = new DataOutputStream(socket.getOutputStream());
             Scanner scanner = new Scanner(System.in)) {

            System.out.println("Connected to Encryption Server.");

            while (true) {
                System.out.println("1. Caesar Cipher");
                System.out.println("2. Monoalphabetic Cipher");
                System.out.println("3. Playfair Cipher");
                System.out.println("4. Hill Cipher");
                System.out.println("5. Vigenère Cipher");
                System.out.println("6. One-Time Pad");
                System.out.println("7. Rail Fence Cipher");
                System.out.println("8. Exit");
                System.out.print("Select an option: ");

                int choice = scanner.nextInt();
                scanner.nextLine();

                if (choice == 8) {
                    output.writeUTF("EXIT");
                    break;
                }

                System.out.print("Enter Plaintext: ");
                String plaintext = scanner.nextLine();
                String key = "";
                String type = "";
                String ciphertext = "";

                switch (choice) {
                    case 1:
                        type = "CAESAR";
                        System.out.print("Enter Shift (Integer): ");
                        key = scanner.nextLine();
                        ciphertext = CryptoUtils.encryptCaesar(plaintext, key);
                        break;
                    case 2:
                        type = "MONO";
                        System.out.print("Enter 26-char shuffled alphabet Key (or enter for default):
");

                        String k = scanner.nextLine();
                        key = k.length() == 26 ? k : "QWERTYUIOPASDFGHJKLZXCVBNM";
                        ciphertext = CryptoUtils.encryptMono(plaintext, key);
                        break;
                    case 3:
                        type = "PLAYFAIR";
                        System.out.print("Enter Keyword: ");
                        key = scanner.nextLine();
                        ciphertext = CryptoUtils.encryptPlayfair(plaintext, key);
                        break;
                    case 4:
                        type = "HILL";
                        System.out.print("Enter 4-letter Key string (e.g., GYBN): ");
                        key = scanner.nextLine();
                        ciphertext = CryptoUtils.encryptHill(plaintext, key);
                        break;
                    case 5:
                        type = "VIGENERE";
                        System.out.print("Enter Keyword: ");
                        key = scanner.nextLine();
                        ciphertext = CryptoUtils.encryptVigenere(plaintext, key);
                        break;
                    case 6:
                        type = "OTP";
                        System.out.print("Enter Random Key (must be length >= text): ");
                        key = scanner.nextLine();
                        if (key.length() < plaintext.length()) {
                            System.out.println("Error: Key too short.");
                            continue;
                        }
                        key = key.substring(0, plaintext.length());
                        ciphertext = CryptoUtils.encryptVigenere(plaintext, key);
                        break;
                    case 7:
                        type = "RAILFENCE";
                        System.out.print("Enter Depth (Integer): ");
                        key = scanner.nextLine();
                        ciphertext = CryptoUtils.encryptRailFence(plaintext, key);
                        break;
                    default:
                        System.out.println("Invalid option.");
                        continue;
                }

                System.out.println("Your Plaintext:        " + plaintext);
                System.out.println("Generated Ciphertext:  " + ciphertext);
                System.out.println("Sending to server for decryption...");
```

```java
                output.writeUTF(type + "::" + key + "::" + ciphertext);

                String response = input.readUTF();
                System.out.println("Server Status: " + response);
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
import java.io.*;
import java.net.*;

public class Server {
    private static final int PORT = 5000;

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server started. Waiting for clients on port " + PORT + "...");

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Client connected.");
                new ClientHandler(socket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class ClientHandler extends Thread {
    private Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (
            DataInputStream input = new DataInputStream(socket.getInputStream());
            DataOutputStream output = new DataOutputStream(socket.getOutputStream());
        ) {
            while (true) {
                String request = input.readUTF();
                if (request.equals("EXIT")) {
                    System.out.println("Client sent EXIT command.");
                    break;
                }

                String[] parts = request.split("::");
                if (parts.length < 3) {
                    output.writeUTF("Error: Invalid format");
                    continue;
                }

                String type = parts[0];
                String key = parts[1];
                String ciphertext = parts[2];
                String decryptedText = "";

                System.out.println("\n[" + type + "] Received Ciphertext: " + ciphertext);

                switch (type) {
                    case "CAESAR": decryptedText = CryptoUtils.decryptCaesar(ciphertext, key); break;
                    case "MONO": decryptedText = CryptoUtils.decryptMono(ciphertext, key); break;
                    case "PLAYFAIR": decryptedText = CryptoUtils.decryptPlayfair(ciphertext, key);
break;
                    case "HILL": decryptedText = CryptoUtils.decryptHill(ciphertext, key); break;
                    case "VIGENERE": decryptedText = CryptoUtils.decryptVigenere(ciphertext, key);
break;
                    case "OTP": decryptedText = CryptoUtils.decryptVigenere(ciphertext, key); break;
                    case "RAILFENCE": decryptedText = CryptoUtils.decryptRailFence(ciphertext, key);
break;
                    default: decryptedText = "Unknown Cipher Type";
                }

                System.out.println("       Decrypted Plaintext: " + decryptedText);

                output.writeUTF("Server received and decrypted message successfully.");
            }
        } catch (EOFException e) {
            System.out.println("Client disconnected.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
import java.io.*;
import java.net.*;

public class Server {
    private static final int PORT = 5000;

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server started. Waiting for clients on port " + PORT + "...");

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Client connected.");
                new ClientHandler(socket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class ClientHandler extends Thread {
    private Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (
            DataInputStream input = new DataInputStream(socket.getInputStream());
            DataOutputStream output = new DataOutputStream(socket.getOutputStream());
        ) {
            while (true) {
                String request = input.readUTF();
                if (request.equals("EXIT")) {
                    System.out.println("Client sent EXIT command.");
                    break;
                }

                String[] parts = request.split("::");
                if (parts.length < 3) {
                    output.writeUTF("Error: Invalid format");
                    continue;
                }
                String type = parts[0];
                String key = parts[1];
                String ciphertext = parts[2];
                String decryptedText = "";

                System.out.println("\n[" + type + "] Received Ciphertext: " + ciphertext);

                switch (type) {
                    case "CAESAR": decryptedText = CryptoUtils.decryptCaesar(ciphertext, key); break;
                    case "MONO": decryptedText = CryptoUtils.decryptMono(ciphertext, key); break;
                    case "PLAYFAIR": decryptedText = CryptoUtils.decryptPlayfair(ciphertext, key);
break;
                    case "HILL": decryptedText = CryptoUtils.decryptHill(ciphertext, key); break;
                    case "VIGENERE": decryptedText = CryptoUtils.decryptVigenere(ciphertext, key);
break;
                    case "OTP": decryptedText = CryptoUtils.decryptVigenere(ciphertext, key); break;
                    case "RAILFENCE": decryptedText = CryptoUtils.decryptRailFence(ciphertext, key);
break;
                    default: decryptedText = "Unknown Cipher Type";
                }

                System.out.println("       Decrypted Plaintext: " + decryptedText);

                output.writeUTF("Server received and decrypted message successfully.");
            }
        } catch (EOFException e) {
            System.out.println("Client disconnected.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
import java.io.*;
import java.net.*;

public class Server {
    private static final int PORT = 5000;

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server started. Waiting for clients on port " + PORT + "...");

            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Client connected.");
                new ClientHandler(socket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class ClientHandler extends Thread {
    private Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try (
            DataInputStream input = new DataInputStream(socket.getInputStream());
            DataOutputStream output = new DataOutputStream(socket.getOutputStream());
        ) {
            while (true) {
                String request = input.readUTF();
                if (request.equals("EXIT")) {
                    System.out.println("Client sent EXIT command.");
                    break;
                }

                String[] parts = request.split("::");
                if (parts.length < 3) {
                    output.writeUTF("Error: Invalid format");
                    continue;
                }

                String type = parts[0];
                String key = parts[1];
                String ciphertext = parts[2];
                String decryptedText = "";

                System.out.println("\n[" + type + "] Received Ciphertext: " + ciphertext);

                switch (type) {
                    case "CAESAR": decryptedText = CryptoUtils.decryptCaesar(ciphertext, key); break;
                    case "MONO": decryptedText = CryptoUtils.decryptMono(ciphertext, key); break;
                    case "PLAYFAIR": decryptedText = CryptoUtils.decryptPlayfair(ciphertext, key);
break;
                    case "HILL": decryptedText = CryptoUtils.decryptHill(ciphertext, key); break;
                    case "VIGENERE": decryptedText = CryptoUtils.decryptVigenere(ciphertext, key);
break;
                    case "OTP": decryptedText = CryptoUtils.decryptVigenere(ciphertext, key); break;
                    case "RAILFENCE": decryptedText = CryptoUtils.decryptRailFence(ciphertext, key);
break;
                    default: decryptedText = "Unknown Cipher Type";
                }

                System.out.println("       Decrypted Plaintext: " + decryptedText);

                output.writeUTF("Server received and decrypted message successfully.");
            }
        } catch (EOFException e) {
            System.out.println("Client disconnected.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

CryptoUtils.java

```
import java.util.*;

public class CryptoUtils {

    public static String encryptCaesar(String text, String keyStr) {
        int shift = Integer.parseInt(keyStr);
        StringBuilder result = new StringBuilder();
        for (char character : text.toCharArray()) {
            if (Character.isLetter(character)) {
                char base = Character.isUpperCase(character) ? 'A' : 'a';
                result.append((char) (((character - base + shift) % 26) + base));
            } else {
                result.append(character);
            }
        }
        return result.toString();
    }

    public static String decryptCaesar(String text, String keyStr) {
        int shift = Integer.parseInt(keyStr);
        StringBuilder result = new StringBuilder();
```

```java
        for (char character : text.toCharArray()) {
            if (Character.isLetter(character)) {
                char base = Character.isUpperCase(character) ? 'A' : 'a';
                result.append((char) (((character - base - shift + 26) % 26) + base));
            } else {
                result.append(character);
            }
        }
        return result.toString();
    }

    public static String encryptMono(String text, String key) {
        String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        key = key.toUpperCase();
        StringBuilder result = new StringBuilder();
        for (char c : text.toUpperCase().toCharArray()) {
            if (Character.isLetter(c)) {
                int index = alphabet.indexOf(c);
                result.append(key.charAt(index));
            } else {
                result.append(c);
            }
        }
        return result.toString();
    }

    public static String decryptMono(String text, String key) {
        String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        key = key.toUpperCase();
        StringBuilder result = new StringBuilder();
        for (char c : text.toUpperCase().toCharArray()) {
            if (Character.isLetter(c)) {
                int index = key.indexOf(c);
                result.append(alphabet.charAt(index));
            } else {
                result.append(c);
            }
        }
        return result.toString();
    }

    public static String encryptPlayfair(String text, String key) {
        char[][] matrix = generatePlayfairMatrix(key);
        text = text.toUpperCase().replaceAll("[^A-Z]", "").replace("J", "I");
        StringBuilder sb = new StringBuilder(text);
        for (int i = 0; i < sb.length() - 1; i += 2) {
            if (sb.charAt(i) == sb.charAt(i + 1)) sb.insert(i + 1, 'X');
        }
        if (sb.length() % 2 != 0) sb.append('X');

        StringBuilder result = new StringBuilder();
        for (int i = 0; i < sb.length(); i += 2) {
            char a = sb.charAt(i);
            char b = sb.charAt(i + 1);
            int[] posA = findPos(matrix, a);
            int[] posB = findPos(matrix, b);

            if (posA[0] == posB[0]) { // Same Row
                result.append(matrix[posA[0]][(posA[1] + 1) % 5]);
                result.append(matrix[posB[0]][(posB[1] + 1) % 5]);
            } else if (posA[1] == posB[1]) { // Same Col
                result.append(matrix[(posA[0] + 1) % 5][posA[1]]);
                result.append(matrix[(posB[0] + 1) % 5][posB[1]]);
            } else { // Rectangle
                result.append(matrix[posA[0]][posB[1]]);
                result.append(matrix[posB[0]][posA[1]]);
            }
        }
        return result.toString();
    }

    public static String decryptPlayfair(String text, String key) {
        char[][] matrix = generatePlayfairMatrix(key);
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < text.length(); i += 2) {
            char a = text.charAt(i);
            char b = text.charAt(i + 1);
            int[] posA = findPos(matrix, a);
            int[] posB = findPos(matrix, b);

            if (posA[0] == posB[0]) {
                result.append(matrix[posA[0]][(posA[1] + 4) % 5]);
                result.append(matrix[posB[0]][(posB[1] + 4) % 5]);
            } else if (posA[1] == posB[1]) {
                result.append(matrix[(posA[0] + 4) % 5][posA[1]]);
                result.append(matrix[(posB[0] + 4) % 5][posB[1]]);
            } else {
                result.append(matrix[posA[0]][posB[1]]);
                result.append(matrix[posB[0]][posA[1]]);
            }
```

```java
        }
        return result.toString();
    }

    public static String encryptHill(String text, String key) {
        text = text.toUpperCase().replaceAll("[^A-Z]", "");
        if(text.length() % 2 != 0) text += "X";

        int[][] keyMatrix = new int[2][2];
        int k = 0;
        for (int i = 0; i < 2; i++)
            for (int j = 0; j < 2; j++)
                keyMatrix[i][j] = (key.charAt(k++) - 'A');

        StringBuilder result = new StringBuilder();
        for (int i = 0; i < text.length(); i += 2) {
            int r1 = text.charAt(i) - 'A';
            int r2 = text.charAt(i+1) - 'A';
            int c1 = (keyMatrix[0][0]*r1 + keyMatrix[0][1]*r2) % 26;
            int c2 = (keyMatrix[1][0]*r1 + keyMatrix[1][1]*r2) % 26;
            result.append((char)(c1 + 'A'));
            result.append((char)(c2 + 'A'));
        }
        return result.toString();
    }

    public static String decryptHill(String text, String key) {
        text = text.toUpperCase().replaceAll("[^A-Z]", "");
        int[][] keyMatrix = new int[2][2];
        int k = 0;
        for (int i = 0; i < 2; i++)
            for (int j = 0; j < 2; j++)
                keyMatrix[i][j] = (key.charAt(k++) - 'A');

        int det = (keyMatrix[0][0] * keyMatrix[1][1] - keyMatrix[0][1] * keyMatrix[1][0]) % 26;
        if (det < 0) det += 26;

        int detInv = -1;
        for (int i = 0; i < 26; i++) {
            if ((det * i) % 26 == 1) {
                detInv = i;
                break;
            }
        }
        if (detInv == -1) return "Error: Key not invertible";
        int[][] invMatrix = new int[2][2];
        invMatrix[0][0] = (keyMatrix[1][1] * detInv) % 26;
        invMatrix[1][1] = (keyMatrix[0][0] * detInv) % 26;
        invMatrix[0][1] = (-keyMatrix[0][1] * detInv) % 26;
        invMatrix[1][0] = (-keyMatrix[1][0] * detInv) % 26;

        for(int i=0; i<2; i++)
            for(int j=0; j<2; j++)
                if(invMatrix[i][j] < 0) invMatrix[i][j] += 26;

        StringBuilder result = new StringBuilder();
        for (int i = 0; i < text.length(); i += 2) {
            int r1 = text.charAt(i) - 'A';
            int r2 = text.charAt(i+1) - 'A';
            int c1 = (invMatrix[0][0]*r1 + invMatrix[0][1]*r2) % 26;
            int c2 = (invMatrix[1][0]*r1 + invMatrix[1][1]*r2) % 26;
            result.append((char)(c1 + 'A'));
            result.append((char)(c2 + 'A'));
        }
        return result.toString();
    }

    public static String encryptVigenere(String text, String key) {
        StringBuilder result = new StringBuilder();
        key = key.toUpperCase();
        int j = 0;
        for (int i = 0; i < text.length(); i++) {
            char c = text.charAt(i);
            if (Character.isLetter(c)) {
                boolean isUpper = Character.isUpperCase(c);
                char p = (char) ((Character.toUpperCase(c) + key.charAt(j) - 2 * 'A') % 26 + 'A');
                result.append(isUpper ? p : Character.toLowerCase(p));
                j = (j + 1) % key.length();
            } else {
                result.append(c);
            }
        }
        return result.toString();
    }

    public static String decryptVigenere(String text, String key) {
        StringBuilder result = new StringBuilder();
        key = key.toUpperCase();
        int j = 0;
```

```
        for (int i = 0; i < text.length(); i++) {
            char c = text.charAt(i);
            if (Character.isLetter(c)) {
                boolean isUpper = Character.isUpperCase(c);
                char p = (char) ((Character.toUpperCase(c) - key.charAt(j) + 26) % 26 + 'A');
                result.append(isUpper ? p : Character.toLowerCase(p));
                j = (j + 1) % key.length();
            } else {
                result.append(c);
            }
        }
        return result.toString();
    }

    public static String encryptRailFence(String text, String keyStr) {
        int rails = Integer.parseInt(keyStr);
        char[][] fence = new char[rails][text.length()];
        for(int i=0; i<rails; i++) Arrays.fill(fence[i], '\n');

        boolean down = false;
        int row = 0, col = 0;

        for(int i=0; i<text.length(); i++){
            if(row == 0 || row == rails-1) down = !down;
            fence[row][col++] = text.charAt(i);
            row += down ? 1 : -1;
        }

        StringBuilder result = new StringBuilder();
        for(int i=0; i<rails; i++){
            for(int j=0; j<text.length(); j++){
                if(fence[i][j] != '\n') result.append(fence[i][j]);
            }
        }
        return result.toString();
    }

    public static String decryptRailFence(String text, String keyStr) {
        int rails = Integer.parseInt(keyStr);
        char[][] fence = new char[rails][text.length()];
        for(int i=0; i<rails; i++) Arrays.fill(fence[i], '\n');

        boolean down = false;
        int row = 0, col = 0;
```

```java
        for(int i=0; i<text.length(); i++){
            if(row == 0 || row == rails-1) down = !down;
            fence[row][col++] = '*';
            row += down ? 1 : -1;
        }

        int index = 0;
        for(int i=0; i<rails; i++){
            for(int j=0; j<text.length(); j++){
                if(fence[i][j] == '*' && index < text.length()){
                    fence[i][j] = text.charAt(index++);
                }
            }
        }

        StringBuilder result = new StringBuilder();
        row = 0; col = 0; down = false;
        for(int i=0; i<text.length(); i++){
            if(row == 0 || row == rails-1) down = !down;
            result.append(fence[row][col++]);
            row += down ? 1 : -1;
        }
        return result.toString();
    }

    private static char[][] generatePlayfairMatrix(String key) {
        char[][] matrix = new char[5][5];
        String keyString = key.toUpperCase().replaceAll("[^A-Z]", "").replace("J", "I") +
"ABCDEFGHIKLMNOPQRSTUVWXYZ";
        Set<Character> seen = new HashSet<>();
        int row = 0, col = 0;
        for (char c : keyString.toCharArray()) {
            if (!seen.contains(c)) {
                seen.add(c);
                matrix[row][col++] = c;
                if (col == 5) { col = 0; row++; }
            }
        }
        return matrix;
    }

    private static int[] findPos(char[][] matrix, char c) {
        if(c == 'J') c = 'I';
        for(int i=0; i<5; i++)
            for(int j=0; j<5; j++)
```

```
                if(matrix[i][j] == c) return new int[]{i,j};
        return null;
    }
}
```