VIT CHENNAI

Lab Exercise 1

Classical Encryption Techniques

Name: Sparsh Karna

Registration Number: 23BDS1172

Lab Slot: L21 + L22

1 Aim of Exercise

The aim of this laboratory exercise is to implement and understand various classical encryption techniques using Java. The exercise includes implementing both substitution and transposition ciphers in a client-server architecture. Each cipher algorithm is implemented where the server encrypts the plaintext and sends the ciphertext to the client, which then decrypts it using the appropriate key. This helps in understanding the fundamental concepts of cryptography, including encryption, decryption, and key management.

2 Encryption Algorithms

2.1 Caesar Cipher

2.1.1 Algorithm

1. Server reads plaintext message and numeric key from user input
2. For each character in the plaintext:
   - If character is lowercase (a-z), shift it forward by key positions in alphabet
   - If character is uppercase (A-Z), shift it forward by key positions in alphabet
   - If character is non-alphabetic, leave it unchanged
3. Server sends encrypted message to client over socket connection
4. Client receives encrypted message from server
5. Client reads decryption key from user input
6. For each character in ciphertext, client applies reverse shift (subtract key)
7. Client displays decrypted plaintext message

2.1.2 Server Code

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

public class Server {

    static String encrypt(String s, int n) {
        String new_s = "";
        for (char c : s.toCharArray()) {
            if (c >= 'a' && c <= 'z') {
                new_s += (char) ((c - 'a' + n) % 26 + 'a');
            } else if (c >= 'A' && c <= 'Z') {
                new_s += (char) ((c - 'A' + n) % 26 + 'A');
            } else {
                new_s += c;
            }
        }
        return new_s;
    }

    public static void main(String[] args) {
        try {
            Scanner input = new Scanner(System.in);

            System.out.print("Enter message: ");
            String message = input.nextLine();

            System.out.print("Enter key: ");
            int n = input.nextInt();

```

```
            String encrypted = encrypt(message, n);

            ServerSocket serverSocket = new ServerSocket(5000);
            System.out.println("Server started. Waiting for client...");

            Socket socket = serverSocket.accept();
            System.out.println("Client connected.");

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            out.println(encrypted);

            System.out.println("Encrypted message sent: " + encrypted);

            socket.close();
            serverSocket.close();
            input.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2.1.3 Client Code

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

public class Client {

    static String decrypt(String s, int n) {
        String new_s = "";
        for (char c : s.toCharArray()) {
            if (c >= 'a' && c <= 'z') {
                new_s += (char) ((c - 'a' - n + 26) % 26 + 'a');
            } else if (c >= 'A' && c <= 'Z') {
                new_s += (char) ((c - 'A' - n + 26) % 26 + 'A');
            } else {
                new_s += c;
            }
        }
        return new_s;
    }

    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 5000);

            BufferedReader in = new BufferedReader(new
               InputStreamReader(socket.getInputStream()));

            String encrypted = in.readLine();

```

```
29              System.out.println("Encrypted message received: " +
                    encrypted);
30
31              Scanner input = new Scanner(System.in);
32
33              System.out.print("Enter key: ");
34              int n = input.nextInt();
35
36              String decrypted = decrypt(encrypted, n);
37              System.out.println("Decrypted (original) message: " +
                    decrypted);
38
39              socket.close();
40              input.close();
41
42          } catch (Exception e) {
43              e.printStackTrace();
44          }
45      }
46  }
```

2.1.4 Input

```
Enter message: HelloWorld
Enter key: 3
```

2.1.5 Output

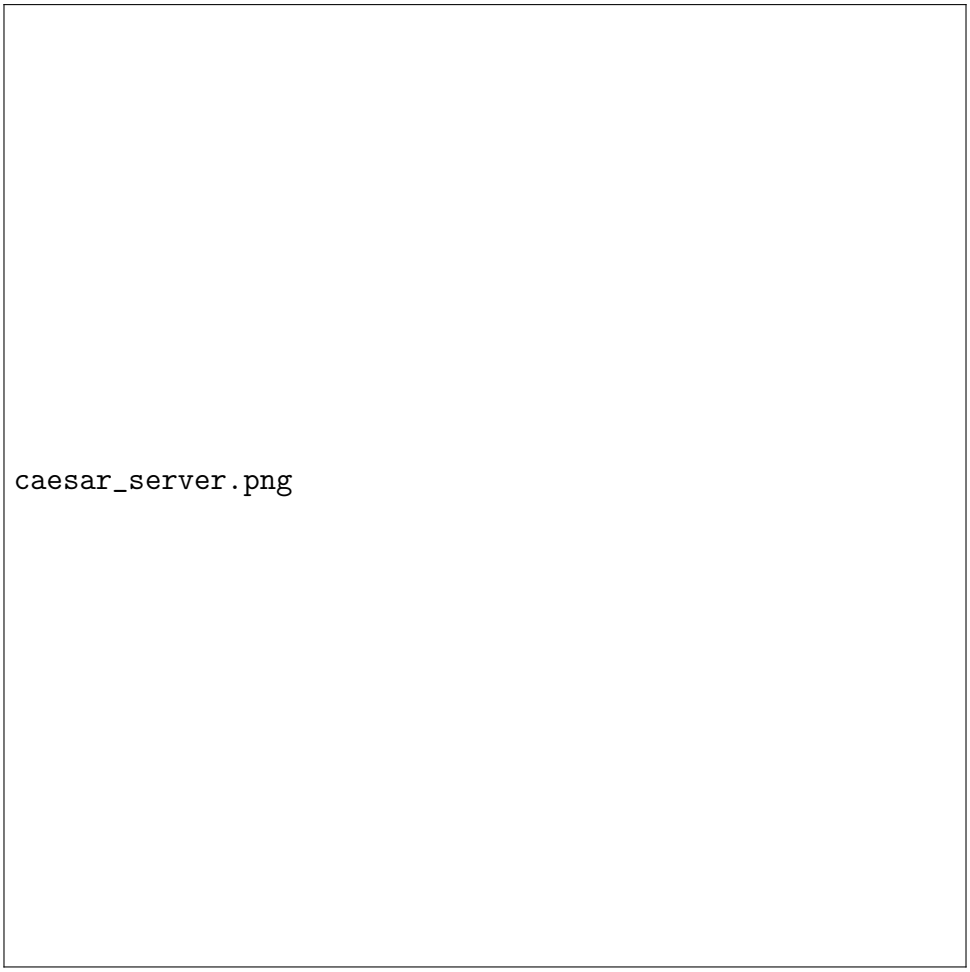

Figure 1: Caesar Cipher - Server Side

2.2 Playfair Cipher

2.2.1 Algorithm

1. Server reads keyword and plaintext from user
2. Create 5x5 key table by placing keyword characters (excluding duplicates, J=I)
3. Fill remaining table cells with unused alphabet letters
4. Prepare plaintext: remove non-alphabetic characters, convert J to I
5. Create digraphs (pairs): if letters are same, insert 'X', pad with 'X' if odd length
6. For each digraph, apply encryption rules:
    - Same row: replace with letters to their right (wrap around)
    - Same column: replace with letters below (wrap around)
    - Rectangle: replace with letters on same row but opposite corners
7. Server sends ciphertext to client
8. Client receives ciphertext and reads key from user

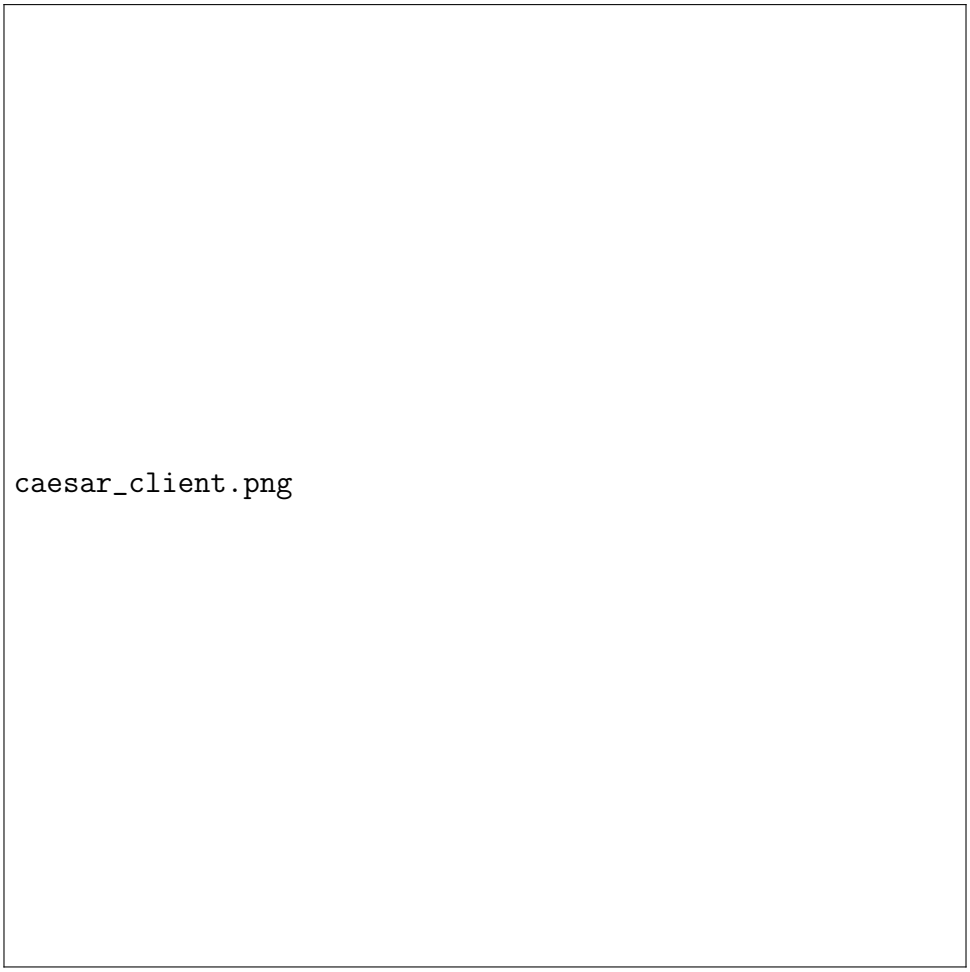

Figure 2: Caesar Cipher - Client Side

9. Client creates same key table using the key
10. For decryption, apply reverse rules on digraphs
11. Client displays decrypted plaintext

2.2.2 Server Code

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

class Server {

    static void makeTable(String k, char[][] kt) {
        int[] hash = new int[26];
        int r = 0;
        int col = 0;
        for (int i = 0; i < k.length(); i++) {
            char c = k.charAt(i);
            if (c < 'A' || c > 'Z')
                continue;
            if (c == 'J')
```

```
                c = 'I';
            int ci = c - 'A';
            if (hash[ci] == 0) {
                kt[r][col] = c;
                hash[ci] = 1;
                if (c == 'I') {
                    hash['J' - 'A'] = 1;
                }
            }
            col++;
            if (col == 5) {
                col = 0;
                r++;
            }
        }
        for (char c = 'A'; c <= 'Z'; c++) {
            if (c == 'J')
                continue;
            int ci = c - 'A';
            if (hash[ci] == 0) {
                kt[r][col] = c;
                hash[ci] = 1;
                col++;
                if (col == 5) {
                    col = 0;
                    r++;
                }
            }
        }
    }

    static int[] findPosition(char ch, char[][] kt) {
        if (ch == 'J')
            ch = 'I';
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (kt[i][j] == ch) {
                    return new int[] { i, j };
                }
            }
        }
        return null;
    }

    static String encode(String p, char[][] kt) {
        p = p.toUpperCase();
        String prepared = "";
        int i = 0;
        while (i < p.length()) {
            char a = p.charAt(i);
            if (a < 'A' || a > 'Z') {
                i++;
                continue;
            }
            if (a == 'J')
                a = 'I';
            prepared += a;
            i++;
```

```
        }
        String digraphs = "";
        i = 0;
        while (i < prepared.length()) {
            char a = prepared.charAt(i);
            digraphs += a;
            if (i == prepared.length() - 1) {
                digraphs += 'X';
                break;
            }
            char b = prepared.charAt(i + 1);
            if (a == b) {
                digraphs += 'X';
            } else {
                digraphs += b;
                i++;
            }
            i++;
        }
        String newCipher = "";
        for (i = 0; i < digraphs.length(); i += 2) {
            char curr = digraphs.charAt(i);
            char nex = digraphs.charAt(i + 1);
            int[] ci = findPosition(curr, kt);
            int[] ni = findPosition(nex, kt);
            if (ci[0] == ni[0]) {
                newCipher += kt[ci[0]][(ci[1] + 1) % 5];
                newCipher += kt[ni[0]][(ni[1] + 1) % 5];
            } else if (ci[1] == ni[1]) {
                newCipher += kt[(ci[0] + 1) % 5][ci[1]];
                newCipher += kt[(ni[0] + 1) % 5][ni[1]];
            } else {
                newCipher += kt[ci[0]][ni[1]];
                newCipher += kt[ni[0]][ci[1]];
            }
        }
        return newCipher;
    }

    public static void main(String args[]) {
        try {
            Scanner input = new Scanner(System.in);

            System.out.print("Enter key: ");
            String key = input.nextLine().toUpperCase();

            System.out.print("Enter message: ");
            String plain = input.nextLine();

            char[][] keyTable = new char[5][5];
            makeTable(key, keyTable);
            String cipher = encode(plain, keyTable);

            ServerSocket serverSocket = new ServerSocket(5001);
            System.out.println("Server started. Waiting for client...")
                ;

            Socket socket = serverSocket.accept();
```

```
            System.out.println("Client connected.");

            PrintWriter out = new PrintWriter(socket.getOutputStream(),
                true);

            out.println(cipher);

            System.out.println("Encrypted message sent: " + cipher);

            socket.close();
            serverSocket.close();
            input.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2.2.3 Client Code

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

class Client {

    static void makeTable(String k, char[][] kt) {
        int[] hash = new int[26];
        int r = 0;
        int col = 0;
        for (int i = 0; i < k.length(); i++) {
            char c = k.charAt(i);
            if (c < 'A' || c > 'Z')
                continue;
            if (c == 'J')
                c = 'I';
            int ci = c - 'A';
            if (hash[ci] == 0) {
                kt[r][col] = c;
                hash[ci] = 1;
                if (c == 'I') {
                    hash['J' - 'A'] = 1;
                }
            }
            col++;
            if (col == 5) {
                col = 0;
                r++;
            }
        }
        for (char c = 'A'; c <= 'Z'; c++) {
            if (c == 'J')
                continue;
            int ci = c - 'A';
            if (hash[ci] == 0) {
                kt[r][col] = c;
                hash[ci] = 1;
```

```
                col++;
                if (col == 5) {
                    col = 0;
                    r++;
                }
            }
        }
    }

    static int[] findPosition(char ch, char[][] kt) {
        if (ch == 'J')
            ch = 'I';
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (kt[i][j] == ch) {
                    return new int[] { i, j };
                }
            }
        }
        return null;
    }

    static String decode(String c, char[][] kt) {
        String plain = "";
        for (int i = 0; i < c.length(); i += 2) {
            char curr = c.charAt(i);
            char nex = c.charAt(i + 1);
            int[] ci = findPosition(curr, kt);
            int[] ni = findPosition(nex, kt);
            if (ci[0] == ni[0]) {
                plain += kt[ci[0]][(ci[1] - 1 + 5) % 5];
                plain += kt[ni[0]][(ni[1] - 1 + 5) % 5];
            } else if (ci[1] == ni[1]) {
                plain += kt[(ci[0] - 1 + 5) % 5][ci[1]];
                plain += kt[(ni[0] - 1 + 5) % 5][ni[1]];
            } else {
                plain += kt[ci[0]][ni[1]];
                plain += kt[ni[0]][ci[1]];
            }
        }
        return plain;
    }

    public static void main(String args[]) {
        try {
            Socket socket = new Socket("localhost", 5001);

            BufferedReader in = new BufferedReader(new
                InputStreamReader(socket.getInputStream()));

            String cipher = in.readLine();

            System.out.println("Encrypted message received: " + cipher)
                ;

            Scanner input = new Scanner(System.in);

            System.out.print("Enter key: ");
```

```
94              String key = input.nextLine().toUpperCase();
95
96              char[][] keyTable = new char[5][5];
97              makeTable(key, keyTable);
98              String decrypted = decode(cipher, keyTable);
99              System.out.println("Decrypted (original) message: " +
                    decrypted);
100
101             socket.close();
102             input.close();
103
104         } catch (Exception e) {
105             e.printStackTrace();
106         }
107     }
108 }
```

2.2.4 Input

```
Enter key: MONARCHY
Enter message: balloon
```

2.2.5 Output

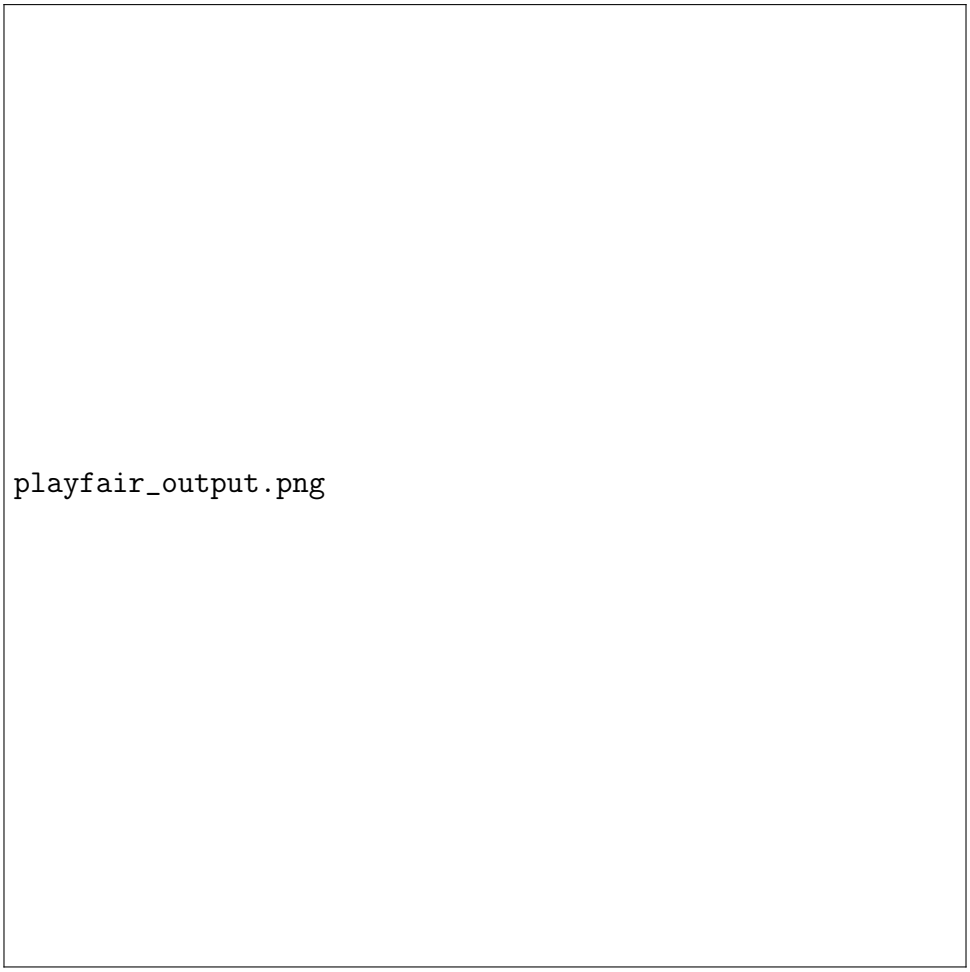

Figure 3: Playfair Cipher Output

2.3 Railfence Cipher

2.3.1 Algorithm

1. Server reads plaintext and numeric key (number of rails) from user
2. Create 2D rail array with dimensions: key x plaintext length
3. Fill rail array in zigzag pattern:
    - Start at top rail (row 0), move diagonally down
    - When bottom rail reached, change direction and move up
    - When top rail reached, change direction and move down
    - Continue until all characters placed
4. Read rail array row-wise to create ciphertext
5. Server sends ciphertext to client
6. Client receives ciphertext and reads key from user
7. Client marks zigzag pattern in empty rail array with '*'

8. Fill marked positions with ciphertext characters row-wise
9. Read array in zigzag pattern to decrypt plaintext
10. Client displays decrypted message

2.3.2 Server Code

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

class Server {

    static String encrypt(String p, int k) {
        String c = "";
        char[][] rail = new char[k][p.length()];
        boolean dir_down = true;
        int row = 0;
        for (int i = 0; i < p.length(); i++) {
            rail[row][i] = p.charAt(i);
            if (dir_down) {
                if (row == k - 1) {
                    dir_down = false;
                    row--;
                } else {
                    row++;
                }
            } else {
                if (row == 0) {
                    dir_down = true;
                    row++;
                } else {
                    row--;
                }
            }
        }
        for (int i = 0; i < k; i++) {
            for (int j = 0; j < p.length(); j++) {
                if (rail[i][j] != 0) {
                    c += rail[i][j];
                }
            }
        }
        return c;
    }

    public static void main(String args[]) {
        try {
            Scanner input = new Scanner(System.in);

            System.out.print("Enter message: ");
            String plainText = input.nextLine();

            System.out.print("Enter key: ");
            int key = input.nextInt();

```

```
            String cipher = encrypt(plainText, key);

            ServerSocket serverSocket = new ServerSocket(5000);
            System.out.println("Server started. Waiting for client...");

            Socket socket = serverSocket.accept();
            System.out.println("Client connected.");

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            out.println(cipher);

            System.out.println("Encrypted message sent: " + cipher);

            socket.close();
            serverSocket.close();
            input.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2.3.3 Client Code

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

class Client {

    static String decrypt(String c, int k) {
        String p = "";
        char[][] rail = new char[k][c.length()];
        boolean dir_down = true;
        int row = 0;
        for (int i = 0; i < c.length(); i++) {
            rail[row][i] = '*';
            if (dir_down) {
                if (row == k - 1) {
                    dir_down = false;
                    row--;
                } else {
                    row++;
                }
            } else {
                if (row == 0) {
                    dir_down = true;
                    row++;
                } else {
                    row--;
                }
            }
        }
        int index = 0;
```

```
        for (int i = 0; i < k; i++) {
            for (int j = 0; j < c.length(); j++) {
                if (rail[i][j] == '*') {
                    rail[i][j] = c.charAt(index++);
                }
            }
        }
        dir_down = true;
        row = 0;
        for (int i = 0; i < c.length(); i++) {
            p += rail[row][i];
            if (dir_down) {
                if (row == k - 1) {
                    dir_down = false;
                    row--;
                } else {
                    row++;
                }
            } else {
                if (row == 0) {
                    dir_down = true;
                    row++;
                } else {
                    row--;
                }
            }
        }
        return p;
    }

    public static void main(String args[]) {
        try {
            Socket socket = new Socket("localhost", 5000);

            BufferedReader in = new BufferedReader(new
                InputStreamReader(socket.getInputStream()));

            String encrypted = in.readLine();

            System.out.println("Encrypted message received: " +
                encrypted);

            Scanner input = new Scanner(System.in);

            System.out.print("Enter key: ");
            int n = input.nextInt();

            String decrypted = decrypt(encrypted, n);
            System.out.println("Decrypted (original) message: " +
                decrypted);

            socket.close();
            input.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
}
```

2.3.4 Input

```
Enter message: GeeksforGeeks
Enter key: 3
```

2.3.5 Output

railfence_server.png

Figure 4: Railfence Cipher - Server Side

railfence_client.png

Figure 5: Railfence Cipher - Client Side

2.4 Vernam Cipher

2.4.1 Algorithm

1. Server reads plaintext and key from user (both same length, uppercase)
2. For each character position:
   - Convert plaintext character to number (A=0, B=1, ..., Z=25)
   - Convert key character to number similarly
   - XOR the two numbers
   - Convert result modulo 26 back to character
3. Server sends ciphertext to client
4. Client receives ciphertext and reads key from user
5. For decryption, apply same XOR operation (XOR is self-inverse)
6. Client displays decrypted plaintext

2.4.2 Server Code

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

class Server {
    static String encode(String p, String k) {
        StringBuilder c = new StringBuilder();
        for (int i = 0; i < p.length(); i++) {
            char pc = (char) (((p.charAt(i) - 'A') ^ (k.charAt(i) - 'A'
                )) % 26 + 'A');
            c.append(pc);
        }
        return c.toString();
    }

    public static void main(String args[]) throws Exception {
        ServerSocket ss = new ServerSocket(1234);
        Socket s = ss.accept();
        Scanner sc = new Scanner(System.in);

        String p = sc.nextLine().toUpperCase();

        String k = sc.nextLine().toUpperCase();

        String ciphertext = encode(p, k);

        DataOutputStream out = new DataOutputStream(s.getOutputStream()
            );
        out.writeBytes(ciphertext + "\n");

        s.close();
        ss.close();
        sc.close();
    }
}
```

2.4.3 Client Code

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

class Client {
    static String decode(String c, String k) {
        StringBuilder p = new StringBuilder();
        for (int i = 0; i < c.length(); i++) {
            char pc = (char) (((c.charAt(i) - 'A') ^ (k.charAt(i) - 'A'
                )) % 26 + 'A');
            p.append(pc);
        }
        return p.toString();
    }

    public static void main(String args[]) throws Exception {
```

```
16          Socket s = new Socket("localhost", 1234);
17
18          BufferedReader in = new BufferedReader(new InputStreamReader(s.
                getInputStream()));
19          Scanner sc = new Scanner(System.in);
20
21          String ciphertext = in.readLine();
22
23          String key = sc.nextLine().toUpperCase();
24
25          String plaintext = decode(ciphertext, key);
26
27          System.out.println("Ciphertext: " + ciphertext);
28          System.out.println("Plaintext: " + plaintext);
29
30          s.close();
31          sc.close();
32      }
33  }
```

2.4.4 Input

```
HELLO
XMCKL
```

2.4.5 Output

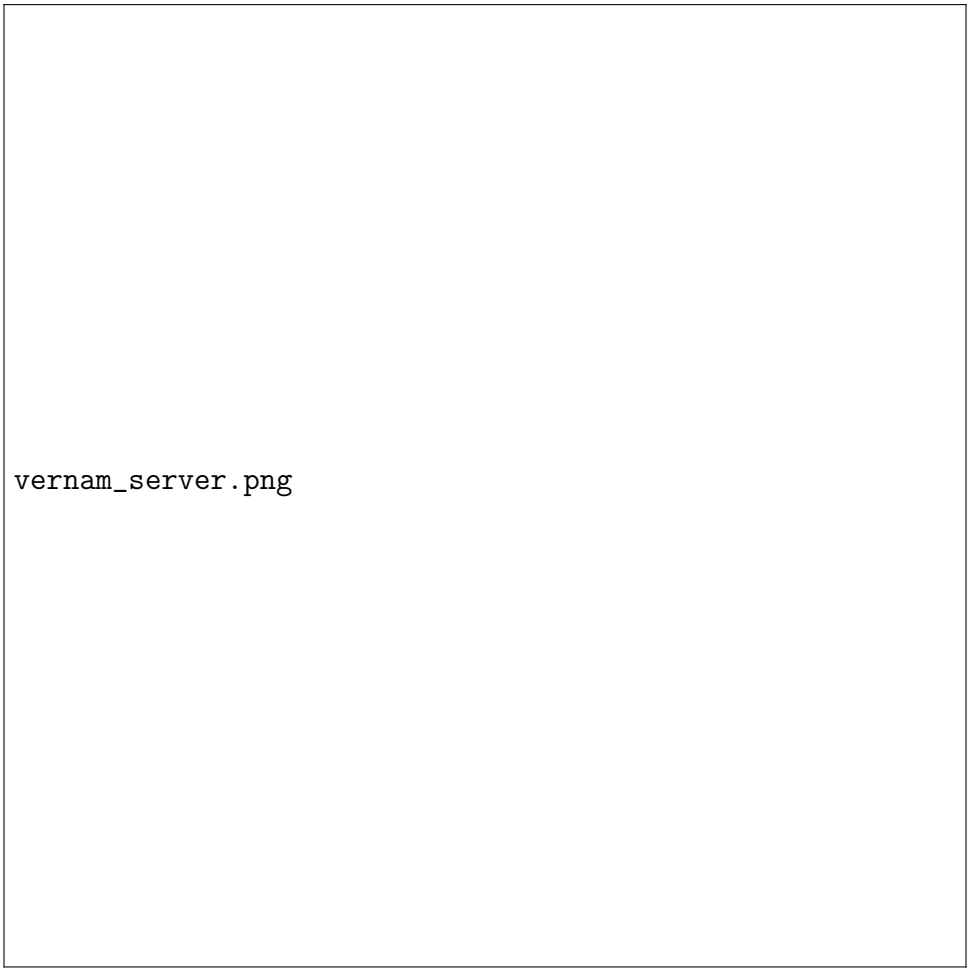

Figure 6: Vernam Cipher - Server Side

2.5 Vigenere Cipher

2.5.1 Algorithm

1. Server reads plaintext and keyword from user (uppercase letters only)
2. For each character in plaintext at position i:
   - Get corresponding key character at position (i mod key_length)
   - Convert both plaintext and key characters to numbers (A=0 to Z=25)
   - Add the two numbers
   - Take modulo 26 and convert back to character
3. Server sends ciphertext to client
4. Client receives ciphertext and reads key from user
5. For decryption, subtract key character values instead of adding
6. Apply modulo 26 and convert back to characters
7. Client displays decrypted plaintext

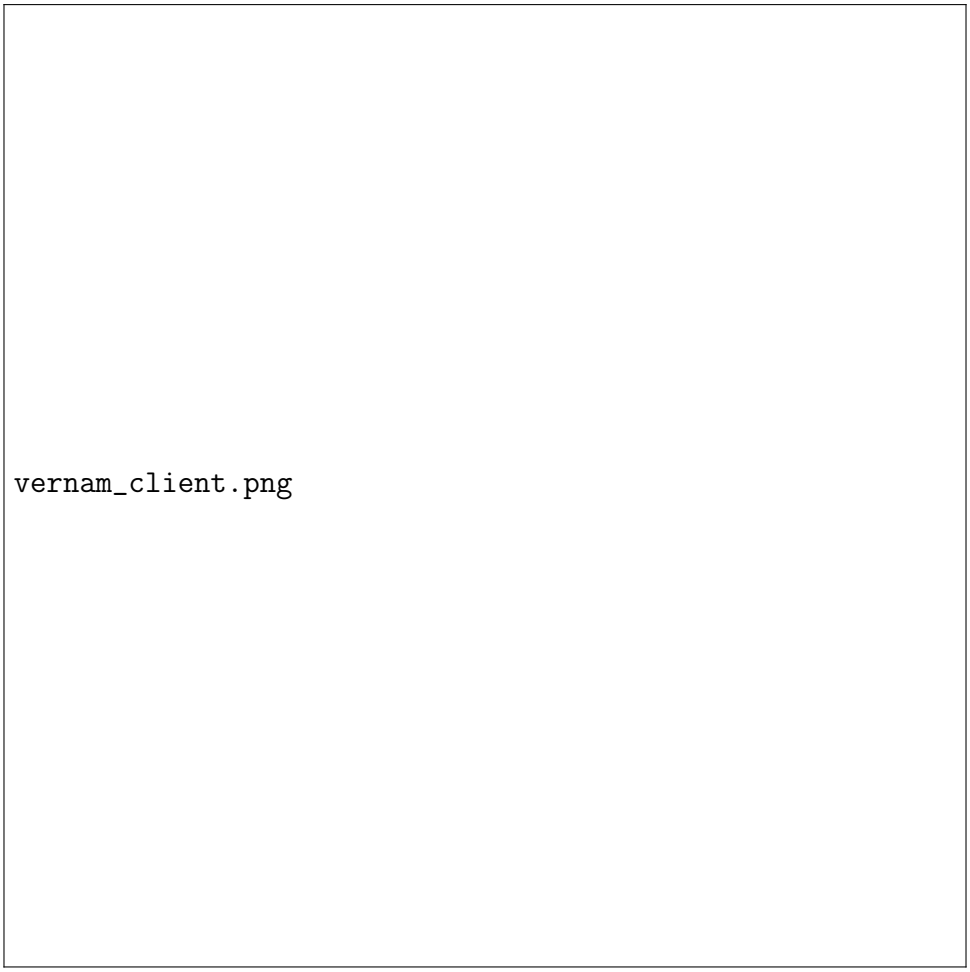

Figure 7: Vernam Cipher - Client Side

2.5.2 Server Code

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

class Server {

    static String encode(String p, String k) {
        String c = "";
        for (int i = 0; i < p.length(); i++) {
            char pi = p.charAt(i);
            char ki = k.charAt(i % k.length());
            c += (char)(((pi - 'A' + ki - 'A') % 26) + 'A');
        }
        return c;
    }

    public static void main(String args[]) throws Exception {
        ServerSocket ss = new ServerSocket(9999);
        System.out.println("Server started. Waiting for client...");
        Socket s = ss.accept();
        System.out.println("Client connected.");

```

```
        Scanner input = new Scanner(System.in);

        System.out.print("Enter plaintext (uppercase letters only): ");
        String plainText = input.nextLine().toUpperCase();

        System.out.print("Enter key (uppercase letters only): ");
        String key = input.nextLine().toUpperCase();

        String cipher = encode(plainText, key);

        PrintWriter out = new PrintWriter(s.getOutputStream(), true);
        out.println(cipher);

        System.out.println("Ciphertext sent to client: " + cipher);

        out.close();
        s.close();
        ss.close();
        input.close();
    }
}
```

2.5.3 Client Code

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

class Client {
    static String decode(String c, String k) {
        String p = "";
        for (int i = 0; i < c.length(); i++) {
            char ci = c.charAt(i);
            char ki = k.charAt(i % k.length());
            p += (char)(((ci - 'A' - (ki - 'A') + 26) % 26) + 'A');
        }
        return p;
    }

    public static void main(String args[]) throws Exception {
        Socket s = new Socket("localhost", 9999);
        System.out.println("Connected to server.");

        BufferedReader in = new BufferedReader(new InputStreamReader(s.
            getInputStream()));
        String cipher = in.readLine();

        System.out.println("Received ciphertext: " + cipher);

        Scanner input = new Scanner(System.in);
        System.out.print("Enter the key to decipher: ");
        String key = input.nextLine().toUpperCase();

        String plainText = decode(cipher, key);
        System.out.println("Deciphered plaintext: " + plainText);

        in.close();
```

```
33          s.close();
34          input.close();
35      }
36  }
```

2.5.4 Input

```
Enter plaintext (uppercase letters only): ATTACKATDAWN
Enter key (uppercase letters only): LEMON
```

2.5.5 Output

Figure 8: Vigenere Cipher - Server Side

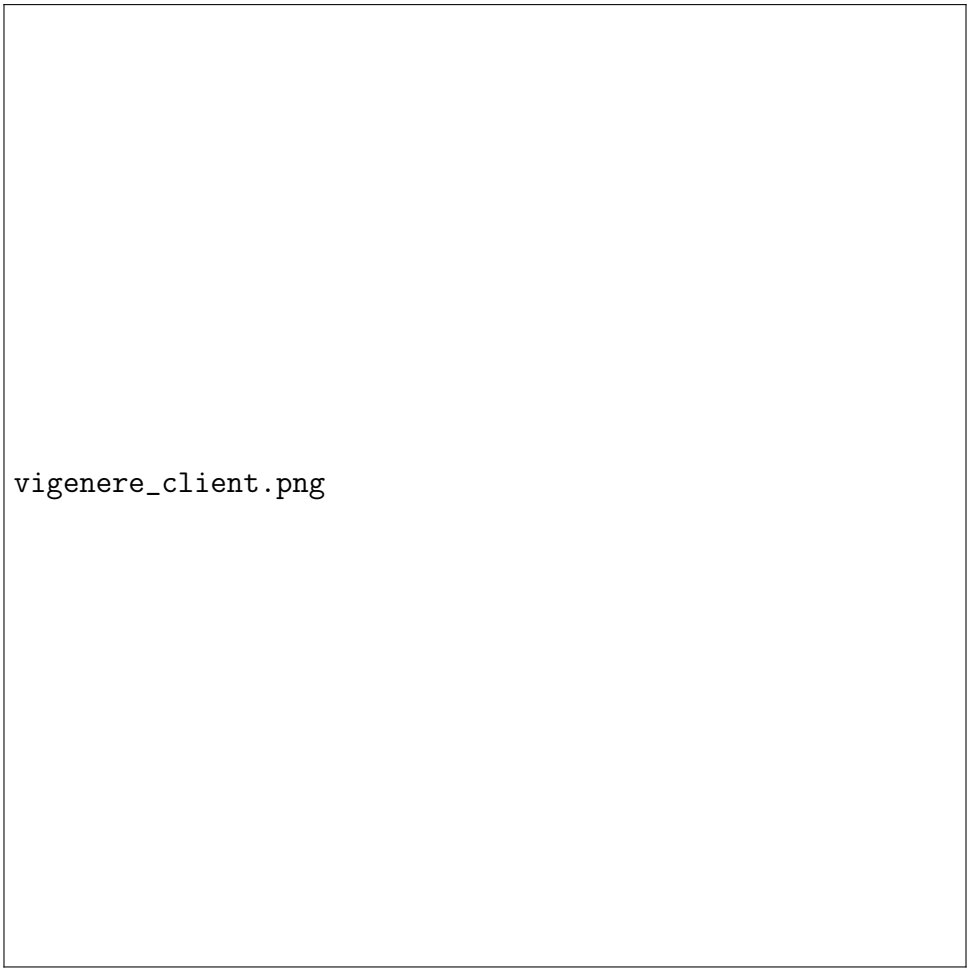

Figure 9: Vigenere Cipher - Client Side

2.6 Row-Column Transposition Cipher

2.6.1 Algorithm

1. Server reads plaintext and keyword from user
2. Calculate table dimensions: columns = key length, rows = ceiling(text length / columns)
3. Create 2D table and fill it row-wise with plaintext characters
4. Pad remaining cells with underscore character if needed
5. Sort table columns alphabetically based on keyword characters
6. Read sorted table column-wise to create ciphertext
7. Server sends ciphertext to client
8. Client receives ciphertext and reads key from user
9. Client creates table and fills it column-wise with ciphertext
10. Decrypt by finding original column positions using key

11. Read table row-wise to get plaintext
12. Remove padding characters and display plaintext

2.6.2 Server Code

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

class Server {

    static void swapCol(char[][] rt, int i, int j) {
        for (int k = 0; k < rt.length; k++) {
            char temp = rt[k][i];
            rt[k][i] = rt[k][j];
            rt[k][j] = temp;
        }
    }

    static void sortTable(char[][] rt, String k) {
        char[] keyArr = k.toCharArray();
        for (int i = 0; i < k.length(); i++) {
            for (int j = 0; j < k.length() - i - 1; j++) {
                if (keyArr[j] > keyArr[j + 1]) {
                    swapCol(rt, j, j + 1);
                    char temp = keyArr[j];
                    keyArr[j] = keyArr[j + 1];
                    keyArr[j + 1] = temp;
                }
            }
        }
    }

    static String encode(char[][] rt, String k) {
        String cipher = "";
        sortTable(rt, k);
        for (int i = 0; i < k.length(); i++) {
            for (int j = 0; j < rt.length; j++) {
                cipher += rt[j][i];
            }
        }
        return cipher;
    }

    static void makeTable(String p, String k, char[][] rt) {
        int idx = 0;
        for (int r = 0; r < rt.length; r++) {
            for (int c = 0; c < rt[0].length; c++) {
                if (idx < p.length()) {
                    rt[r][c] = p.charAt(idx++);
                } else {
                    rt[r][c] = '_';
                }
            }
        }
    }
```

```
52
53      public static void main(String args[]) {
54          try {
55              Scanner input = new Scanner(System.in);
56
57              System.out.print("Enter message: ");
58              String plainText = input.nextLine();
59
60              System.out.print("Enter key: ");
61              String key = input.nextLine();
62
63              int cols = key.length();
64              int rows = (int) Math.ceil((double) plainText.length() /
                    cols);
65              char[][] ranktable = new char[rows][cols];
66              makeTable(plainText, key, ranktable);
67              String cipher = encode(ranktable, key);
68
69              ServerSocket serverSocket = new ServerSocket(5000);
70              System.out.println("Server started. Waiting for client...")
                    ;
71
72              Socket socket = serverSocket.accept();
73              System.out.println("Client connected.");
74
75              PrintWriter out = new PrintWriter(socket.getOutputStream(),
                     true);
76
77              out.println(cipher);
78
79              System.out.println("Encrypted message sent: " + cipher);
80
81              socket.close();
82              serverSocket.close();
83              input.close();
84          } catch (Exception e) {
85              e.printStackTrace();
86          }
87      }
88  }
```

2.6.3 Client Code

```
1   import java.util.*;
2   import java.net.*;
3   import java.io.*;
4
5   class Client {
6
7       static String decrypt(char[][] rt, String k) {
8           String plain = "";
9           int cols = k.length();
10          int rows = rt.length;
11          char[] sortedKey = k.toCharArray();
12          Arrays.sort(sortedKey);
13          for (char ch : sortedKey) {
14              int colIndex = k.indexOf(ch);
```

```
            for (int r = 0; r < rows; r++) {
                plain += rt[r][colIndex];
            }
            k = k.substring(0, colIndex) + '_' + k.substring(colIndex +
                1);
        }
        return plain.replace("_", "");
    }

    public static void main(String args[]){
        Scanner input = new Scanner(System.in);
        try {
            Socket socket = new Socket("localhost", 5000);

            BufferedReader in = new BufferedReader(new
               InputStreamReader(socket.getInputStream()));

            String cipher = in.readLine();

            System.out.println("Ciphertext received: " + cipher);

            System.out.print("Enter key: ");
            String key = input.nextLine();

            int cols = key.length();
            int rows = (int) Math.ceil((double) cipher.length() / cols)
               ;
            char[][] ranktable = new char[rows][cols];

            int idx = 0;
            for (int c = 0; c < cols; c++) {
                for (int r = 0; r < rows; r++) {
                    if (idx < cipher.length()) {
                        ranktable[r][c] = cipher.charAt(idx++);
                    } else {
                        ranktable[r][c] = '_';
                    }
                }
            }

            String plain = decrypt(ranktable, key);
            System.out.println("Decrypted (original) message: " + plain
               );

            socket.close();
            input.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2.6.4 Input

```
Enter message: attackpostponeduntiltwoam
Enter key: 4312567
```

2.6.5 Output

Figure 10: Row-Column Transposition Cipher - Server Side

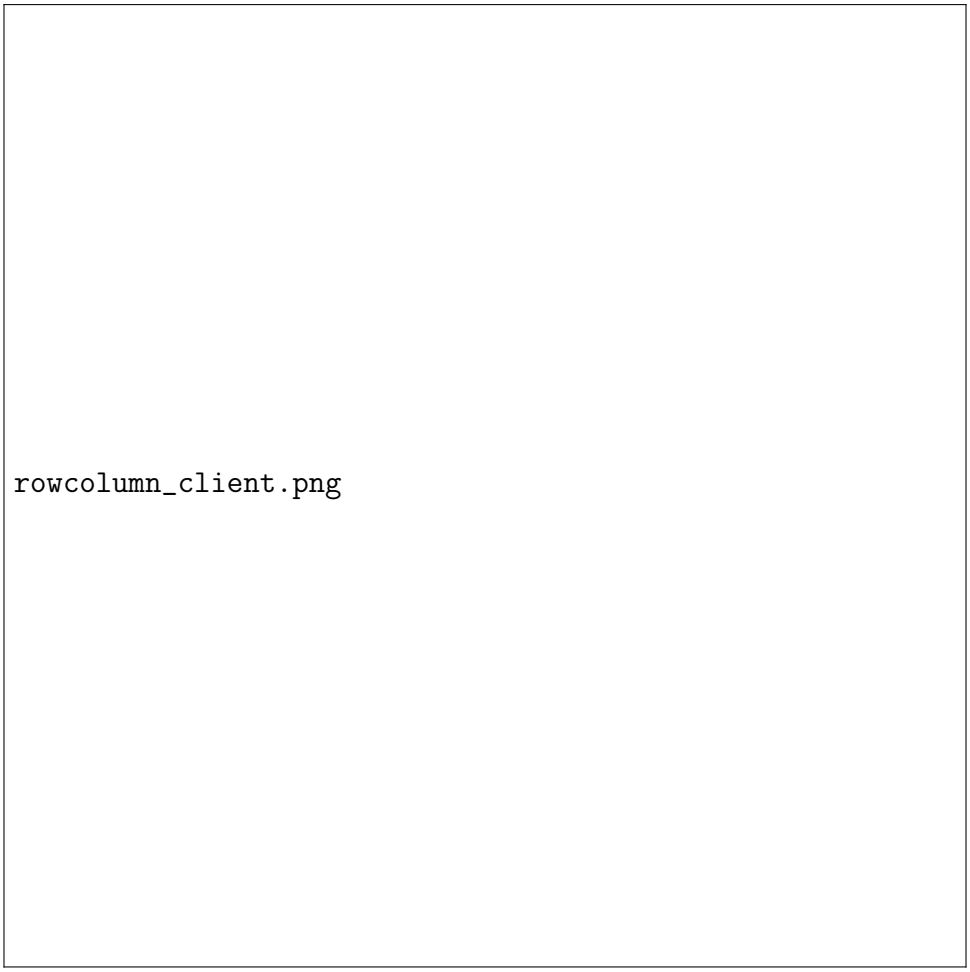

Figure 11: Row-Column Transposition Cipher - Client Side

3 Conclusion

This laboratory exercise successfully demonstrated the implementation of various classical encryption techniques using Java. The implementation covered both substitution ciphers (Caesar, Playfair, Vernam, and Vigenere) and transposition ciphers (Railfence and Row-Column).

Each cipher was implemented using a client-server architecture where the server encrypts the plaintext using the specified algorithm and key, then transmits the ciphertext to the client. The client receives the encrypted message, takes the decryption key from the user, and displays the decrypted plaintext.

Key learnings from this exercise include:

- Understanding the fundamental differences between substitution and transposition ciphers
- Implementation of encryption and decryption algorithms in Java
- Network programming using sockets for client-server communication
- The importance of key management in cryptographic systems
- Practical limitations of classical ciphers and their vulnerability to cryptanalysis

While these classical ciphers are not secure by modern standards, they provide an excellent foundation for understanding the principles of cryptography. Modern encryption systems build upon these concepts with much more sophisticated mathematical operations and longer key spaces to provide adequate security for contemporary applications.