

BCSE309P - Cryptography & Network Security  
Lab

**Lab Assignment 1**

Kedar Shinde 22BCE1765

## 1 Aim

To implement the Caesar cipher, Vigenère cipher, Vernam cipher, Playfair cipher, Hill cipher, and Columnar transposition cipher.

## 2 Steps for Encryption and Decryption

### 2.1 Caesar Cipher

The Caesar cipher, named after Julius Caesar, is a substitution cipher that shifts each letter in the plaintext by a fixed number of positions in the alphabet.

#### 2.1.1 Steps for Encryption

1. For each character in the plaintext message:
  - a) Check if the character is alphabetic using `isalpha()`
  - b) Determine the base value:
    - If lowercase: base = 'a' (97 in ASCII)
    - If uppercase: base = 'A' (65 in ASCII)
  - c) Apply the shift transformation:

$$E(c) = (c - \text{base} + \text{shift}) \bmod 26 + \text{base} \quad (1)$$

- d) Non-alphabetic characters remain unchanged

#### 2.1.2 Steps for Decryption

1. For each character in the ciphertext:
  - a) Check if the character is alphabetic
  - b) Determine the base value (same as encryption)
  - c) Apply the reverse shift:

$$D(c) = (c - \text{base} - \text{shift} + 26) \bmod 26 + \text{base} \quad (2)$$

- d) The +26 ensures positive modulus result

### 2.2 Vigenère Cipher

The Vigenère cipher enhances the Caesar cipher by using a keyword to create multiple shift values, making it more resistant to frequency analysis.

### 2.2.1 Key Generation Process

1. Calculate required key length:

$$\text{required\_length} = \text{message\_length} \quad (3)$$

2. Generate repeating key:

- a) Calculate complete repetitions:  $\text{reps} = \lfloor \frac{\text{message\_length}}{\text{key\_length}} \rfloor$
- b) Calculate remaining characters:  $\text{remainder} = \text{message\_length} \bmod \text{key\_length}$
- c) Repeat key **reps** times
- d) Append first **remainder** characters of key

### 2.2.2 Encryption Steps

1. Generate the extended key to match message length
2. For each character position  $i$ :

- a) Get plaintext character  $p_i$  and key character  $k_i$
- b) If  $p_i$  is alphabetic:
  - Determine character case and base
  - Apply Vigenère transformation:

$$E(p_i) = ((p_i - \text{base} + (k_i - \text{base})) \bmod 26) + \text{base} \quad (4)$$

- c) Preserve non-alphabetic characters

### 2.2.3 Decryption Steps

1. Generate the same extended key
2. For each character position  $i$ :

- a) Get ciphertext character  $c_i$  and key character  $k_i$
- b) If  $c_i$  is alphabetic:
  - Determine character case and base
  - Apply reverse transformation:

$$D(c_i) = ((c_i - \text{base} - (k_i - \text{base}) + 26) \bmod 26) + \text{base} \quad (5)$$

## 2.3 Vernam Cipher

The Vernam cipher, also known as the one-time pad when used with a truly random key, provides perfect secrecy when implemented correctly.

### 2.3.1 Key Requirements

1. Key length must equal or exceed message length
2. Key should ideally be truly random
3. Key must never be reused

### 2.3.2 Encryption Steps

1. Validate key length against message length
2. For each character position  $i$ :
  - a) Perform bitwise XOR operation:

$$\text{temp} = p_i \oplus k_i \quad (6)$$

- b) Ensure printable ASCII result:

$$E(p_i) = \text{temp} \& 0x7F \quad (7)$$

### 2.3.3 Decryption Steps

1. For each character position  $i$ :
  - a) Perform bitwise XOR with same key:

$$\text{temp} = c_i \oplus k_i \quad (8)$$

- b) Ensure printable ASCII:

$$D(c_i) = \text{temp} \& 0x7F \quad (9)$$

## 2.4 Playfair Cipher

The Playfair cipher encrypts pairs of letters using a  $5 \times 5$  matrix constructed from a keyword.

### 2.4.1 Key Matrix Generation

1. Process the keyword:
  - a) Remove duplicate characters
  - b) Convert to lowercase
  - c) Replace 'j' with 'i'
2. Create the alphabet string:
  - a) Use "abcdefghijklmnopqrstuvwxyz" (no 'j')

- b) Remove letters already in processed key
- 3. Construct  $5 \times 5$  matrix:
  - a) Fill with processed key first
  - b) Fill remaining positions with remaining alphabet

### 2.4.2 Plaintext Preprocessing

1. Remove non-alphabetic characters
2. Convert to lowercase
3. Replace 'j' with 'i'
4. Split into digraphs (pairs):
  - a) If pair has same letters, insert 'x' between them
  - b) If text length is odd, append 'x'

### 2.4.3 Encryption Rules

1. For each letter pair  $(a, b)$ :
  - a) Find positions  $(row_1, col_1)$  and  $(row_2, col_2)$  in matrix
  - b) Apply transformation rules:
    - Same row:  $E(a) = M[row_1][(col_1+1) \bmod 5]$ ,  $E(b) = M[row_2][(col_2+1) \bmod 5]$
    - Same column:  $E(a) = M[(row_1+1) \bmod 5][col_1]$ ,  $E(b) = M[(row_2+1) \bmod 5][col_2]$
    - Rectangle:  $E(a) = M[row_1][col_2]$ ,  $E(b) = M[row_2][col_1]$

### 2.4.4 Decryption Rules

1. For each letter pair  $(a, b)$ :
  - a) Find positions in matrix
  - b) Apply reverse transformations:
    - Same row:  $D(a) = M[row_1][(col_1-1) \bmod 5]$ ,  $D(b) = M[row_2][(col_2-1) \bmod 5]$
    - Same column:  $D(a) = M[(row_1-1) \bmod 5][col_1]$ ,  $D(b) = M[(row_2-1) \bmod 5][col_2]$
    - Rectangle:  $D(a) = M[row_1][col_2]$ ,  $D(b) = M[row_2][col_1]$

## 2.5 Hill Cipher

The Hill cipher uses linear algebra for encryption, representing text using matrices and performing matrix multiplication.

### 2.5.1 Matrix Operations

1. Calculate matrix determinant:

$$\det(K) = k_{11}k_{22} - k_{12}k_{21} \quad (10)$$

2. Find modular multiplicative inverse:

$$\det(K)^{-1} \bmod 26 \text{ where } \det(K) \times \det(K)^{-1} \equiv 1 \pmod{26} \quad (11)$$

3. Calculate adjugate matrix:

$$\text{adj}(K) = \begin{bmatrix} k_{22} & -k_{12} \\ -k_{21} & k_{11} \end{bmatrix} \quad (12)$$

### 2.5.2 Encryption Steps

1. Preprocess plaintext:
  - a) Remove non-alphabetic characters
  - b) Convert to lowercase
  - c) Pad with 'x' if needed
2. Convert text to numbers (a=0, b=1, etc.)
3. For each block of n letters (n = matrix size):
  - a) Create column vector  $P$
  - b) Compute  $C = KP \bmod 26$
  - c) Convert result back to letters

### 2.5.3 Decryption Steps

1. Calculate inverse key matrix  $K^{-1}$ :

$$K^{-1} = \det(K)^{-1} \times \text{adj}(K) \bmod 26 \quad (13)$$

2. For each block of n letters:
  - a) Create column vector  $C$
  - b) Compute  $P = K^{-1}C \bmod 26$
  - c) Convert result back to letters

## 2.6 Rail Fence Cipher (Columnar Transposition)

The columnar transposition cipher rearranges characters based on the alphabetical ordering of a keyword.

**2.6.1 Encryption Steps**

1. Calculate dimensions:
  - a) Number of columns = key length
  - b) Number of rows =  $\lceil \frac{\text{message\_length}}{\text{key\_length}} \rceil$
2. Create grid:
  - a) Fill grid row by row with plaintext
  - b) Pad incomplete final row with spaces
3. Determine column order:
  - a) Number columns based on keyword letter positions
  - b) Create mapping of column numbers to positions
4. Read off columns:
  - a) Read columns in order determined by key
  - b) Ignore padding characters

**2.6.2 Decryption Steps**

1. Calculate dimensions (same as encryption)
2. Calculate column lengths:
  - a) Full columns:  $\lfloor \frac{\text{message\_length}}{\text{key\_length}} \rfloor$
  - b) Extra characters:  $\text{message\_length} \bmod \text{key\_length}$
3. Reconstruct columns:
  - a) Determine original column order from key
  - b) Place appropriate number of characters in each column
4. Read plaintext:
  - a) Read grid row by row
  - b) Combine characters to form plaintext

## 3 Implementation

### 3.1 Caesar Cipher Implementation

```
1 std::string caesar_encrypt(const std::string &message, int
  shift) {
2     std::string encrypted;
3     for (char c : message) {
4         if (std::isalpha(c)) {
5             char base = std::islower(c) ? 'a' : 'A';
6             encrypted += (c - base + shift) % 26 + base;
7         } else {
8             encrypted += c;
9         }
10    }
11    return encrypted;
12 }
13
14 std::string caesar_decrypt(const std::string &message, int
  shift) {
15     std::string decrypted;
16     for (char c : message) {
17         if (std::isalpha(c)) {
18             char base = std::islower(c) ? 'a' : 'A';
19             decrypted += (c - base - shift + 26) % 26 + base
20             ;
21         } else {
22             decrypted += c;
23         }
24     }
25    return decrypted;
26 }
```

### 3.2 Vigenère Cipher Implementation

```
1 std::string generate_repeating_key(const std::string &key,
  int message_length) {
2     std::string actual_key;
3     int k_len = key.length();
4
5     int reps = message_length / k_len;
6     int rem = message_length % k_len;
7
8     for (int i = 0; i < reps; i++) {
9         actual_key += key;
10    }
11    actual_key += key.substr(0, rem);
12    return actual_key;
13 }
```



```
13 }
14
15 std::string vigenere_encrypt(const std::string &message,
16                             const std::string &key) {
17     std::string encrypted;
18     std::string actual_key = generate_repeating_key(key,
19                                                     message.length());
20
21     for (int i = 0; i < message.length(); i++) {
22         char c = message[i];
23         char k = actual_key[i];
24         if (std::isalpha(c)) {
25             char base = std::islower(c) ? 'a' : 'A';
26             encrypted += ((c - base + (k - base)) % 26) +
27                         base;
28         } else {
29             encrypted += c;
30         }
31     }
32     return encrypted;
33 }
34
35 std::string vigenere_decrypt(const std::string &message,
36                             const std::string &key) {
37     std::string decrypted;
38     std::string actual_key = generate_repeating_key(key,
39                                                     message.length());
40
41     for (int i = 0; i < message.length(); i++) {
42         char c = message[i];
43         char k = actual_key[i];
44         if (std::isalpha(c)) {
45             char base = std::islower(c) ? 'a' : 'A';
46             decrypted += ((c - base - (k - base) + 26) % 26) +
47                         base;
48         } else {
49             decrypted += c;
50         }
51     }
52     return decrypted;
53 }
```

### 3.3 Vernam Cipher Implementation

```
1 std::string vernam_encrypt(const std::string &message, const
2                             std::string &key) {
3     if (key.length() < message.length()) {
```

```

3         throw std::invalid_argument("Key length must be at
4             least equal to the message length.");
5     }
6     std::string encrypted;
7     for (size_t i = 0; i < message.length(); i++) {
8         encrypted += (message[i] ^ key[i]) & 0x7F;
9     }
10    return encrypted;
11}
12
13std::string vernam_decrypt(const std::string &encrypted,
14    const std::string &key) {
15    std::string decrypted;
16    for (size_t i = 0; i < encrypted.length(); i++) {
17        decrypted += (encrypted[i] ^ key[i]) & 0x7F;
18    }
19    return decrypted;
20}

```

### 3.4 Playfair Cipher Implementation

```

1 std::string removeDuplicates(const std::string& str) {
2     std::string result;
3     for (char ch : str) {
4         if (result.find(ch) == std::string::npos) {
5             result += ch;
6         }
7     }
8     return result;
9 }
10
11std::vector<std::vector<char>> generateKeyMatrix(const std:::
12    string& key) {
13    std::string processedKey = removeDuplicates(key);
14    processedKey.erase(std::remove(processedKey.begin(),
15        processedKey.end(), 'j'),
16        processedKey.end());
17    std::string alphabet = "abcdefghiklmnopqrstuvwxyz";
18
19    for (char ch : alphabet) {
20        if (processedKey.find(ch) == std::string::npos) {
21            processedKey += ch;
22        }
23    }
24
25    std::vector<std::vector<char>> keyMatrix(5, std::vector<
26        char>(5));

```

```
24     int index = 0;
25     for (int i = 0; i < 5; ++i) {
26         for (int j = 0; j < 5; ++j) {
27             keyMatrix[i][j] = processedKey[index++];
28         }
29     }
30     return keyMatrix;
31 }
32
33 std::string preprocessPlaintext(std::string plaintext) {
34     plaintext.erase(std::remove_if(plaintext.begin(),
35                                     plaintext.end(),
36                                     [](char ch) { return !std::isalpha(ch);
37                                     }), plaintext.end());
38     std::transform(plaintext.begin(), plaintext.end(),
39                   plaintext.begin(), ::tolower);
40     std::replace(plaintext.begin(), plaintext.end(), 'j', 'i');
41
42     std::string processedText;
43     for (size_t i = 0; i < plaintext.length(); ++i) {
44         processedText += plaintext[i];
45         if (i + 1 < plaintext.length() && plaintext[i] ==
46             plaintext[i + 1]) {
47             processedText += 'x';
48         }
49     }
50     if (processedText.length() % 2 != 0) {
51         processedText += 'x';
52     }
53     return processedText;
54 }
55
56 std::string playfair_encrypt(const std::string& plaintext,
57                             const std::string& key) {
58     std::vector<std::vector<char>> keyMatrix =
59         generateKeyMatrix(key);
60     std::string processedText = preprocessPlaintext(
61         plaintext);
62
63     std::string ciphertext;
64     for (size_t i = 0; i < processedText.length(); i += 2) {
65         ciphertext += encryptPair(keyMatrix, processedText[i],
66                                   processedText[i + 1]);
67     }
68     return ciphertext;
69 }
```

```

64 std::string playfair_decrypt(const std::string& ciphertext,
    const std::string& key) {
65     std::vector<std::vector<char>> keyMatrix =
        generateKeyMatrix(key);
66
67     std::string plaintext;
68     for (size_t i = 0; i < ciphertext.length(); i += 2) {
69         plaintext += decryptPair(keyMatrix, ciphertext[i],
            ciphertext[i + 1]);
70     }
71     return plaintext;
72 }

```

### 3.5 Hill Cipher Implementation

```

1  int determinant(const std::vector<std::vector<int>>& matrix,
    int n) {
2      if (n == 1) return matrix[0][0];
3
4      int det = 0;
5      std::vector<std::vector<int>> submatrix(n - 1, std::
        vector<int>(n - 1));
6      for (int x = 0; x < n; x++) {
7          int subi = 0;
8          for (int i = 1; i < n; i++) {
9              int subj = 0;
10             for (int j = 0; j < n; j++) {
11                 if (j == x) continue;
12                 submatrix[subi][subj] = matrix[i][j];
13                 subj++;
14             }
15             subi++;
16         }
17         det += (x % 2 == 0 ? 1 : -1) * matrix[0][x] *
            determinant(submatrix, n - 1);
18     }
19     return det;
20 }
21
22 std::vector<std::vector<int>> matrixInverse(const std::
    vector<std::vector<int>>& matrix, int n) {
23     int det = determinant(matrix, n);
24     int detModInverse = modularInverse(det, 26);
25
26     std::vector<std::vector<int>> adjoint(n, std::vector<int>
        >(n));
27     if (n == 1) {
28         adjoint[0][0] = 1;

```

```

29         return adjoint;
30     }
31
32     // Calculate adjoint matrix
33     std::vector<std::vector<int>> temp(n - 1, std::vector<
34         int>(n - 1));
35     for (int i = 0; i < n; i++) {
36         for (int j = 0; j < n; j++) {
37             int subi = 0;
38             for (int x = 0; x < n; x++) {
39                 if (x == i) continue;
40                 int subj = 0;
41                 for (int y = 0; y < n; y++) {
42                     if (y == j) continue;
43                     temp[subi][subj] = matrix[x][y];
44                     subj++;
45                 }
46                 subi++;
47             }
48             adjoint[j][i] = (determinant(temp, n - 1) * ((i
49                 + j) % 2 == 0 ? 1 : -1));
50         }
51     }
52
53     // Multiply adjoint by modular multiplicative inverse of
54     // determinant
55     for (int i = 0; i < n; i++) {
56         for (int j = 0; j < n; j++) {
57             adjoint[i][j] = mod(adjoint[i][j] *
58                 detModInverse, 26);
59         }
60     }
61     return adjoint;
62 }
63
64 std::string hill_encrypt(const std::string& plaintext,
65     const std::vector<std::vector<int>>&
66     key) {
67     int n = key.size();
68     std::string processedText = plaintext;
69     processedText.erase(
70         std::remove_if(processedText.begin(), processedText.
71             end(),
72                 [](char c) { return !std::isalpha(c);
73                     }),
74         processedText.end()
75     );
76     std::transform(processedText.begin(), processedText.end
77         (),
78         processedText.begin(), ::tolower);

```

```

71     while (processedText.size() % n != 0) {
72         processedText += 'x';
73     }
74
75     std::string ciphertext;
76     for (size_t i = 0; i < processedText.size(); i += n) {
77         for (int row = 0; row < n; ++row) {
78             int sum = 0;
79             for (int col = 0; col < n; ++col) {
80                 sum += key[row][col] * (processedText[i +
81                     col] - 'a');
82             }
83             ciphertext += (sum % 26) + 'a';
84         }
85     }
86     return ciphertext;
87 }
88
89 std::string hill_decrypt(const std::string& ciphertext,
90                         const std::vector<std::vector<int>>&
91                             key) {
92     int n = key.size();
93     std::vector<std::vector<int>> inverseKey = matrixInverse
94         (key, n);
95
96     std::string plaintext;
97     for (size_t i = 0; i < ciphertext.size(); i += n) {
98         for (int row = 0; row < n; ++row) {
99             int sum = 0;
100             for (int col = 0; col < n; ++col) {
101                 sum += inverseKey[row][col] * (ciphertext[i
102                     + col] - 'a');
103             }
104             plaintext += (mod(sum, 26)) + 'a';
105         }
106     }
107     return plaintext;
108 }

```

### 3.6 Rail Fence Cipher (Columnar Transposition) Implementation

```

1 std::string columnar_encrypt(const std::string& plaintext,
2                             const std::string& key) {
3     int numCols = key.size();
4     int numRows = (plaintext.size() + numCols - 1) / numCols
5         ;

```

```
4
5     std::vector<std::string> grid(numRows, std::string(
6         numCols, ' '));
7     for (size_t i = 0; i < plaintext.size(); ++i) {
8         grid[i / numCols][i % numCols] = plaintext[i];
9     }
10
11     std::vector<int> columnOrder(numCols);
12     for (size_t i = 0; i < key.size(); ++i) {
13         columnOrder[i] = i;
14     }
15
16     std::sort(columnOrder.begin(), columnOrder.end(),
17         [&key](int a, int b) { return key[a] < key[b];
18         });
19
20     std::string ciphertext;
21     for (int col : columnOrder) {
22         for (int row = 0; row < numRows; ++row) {
23             if (grid[row][col] != ' ') {
24                 ciphertext += grid[row][col];
25             }
26         }
27     }
28     return ciphertext;
29 }
30
31 std::string columnar_decrypt(const std::string& ciphertext,
32     const std::string& key) {
33     int numCols = key.size();
34     int numRows = (ciphertext.size() + numCols - 1) /
35         numCols;
36
37     std::vector<int> columnLengths(numCols, numRows);
38     int extraChars = ciphertext.size() % numCols;
39     for (int i = 0; i < numCols; ++i) {
40         if (i >= extraChars) {
41             columnLengths[i]--;
42         }
43     }
44
45     std::vector<int> columnOrder(numCols);
46     for (size_t i = 0; i < key.size(); ++i) {
47         columnOrder[i] = i;
48     }
49
50     std::sort(columnOrder.begin(), columnOrder.end(),
51         [&key](int a, int b) { return key[a] < key[b];
52         });
```

```
49     std::vector<std::string> grid(numCols);
50     int index = 0;
51     for (int col : columnOrder) {
52         grid[col] = ciphertext.substr(index, columnLengths[
53             col]);
54         index += columnLengths[col];
55     }
56
57     std::string plaintext;
58     for (int row = 0; row < numRows; ++row) {
59         for (int col = 0; col < numCols; ++col) {
60             if (row < grid[col].size()) {
61                 plaintext += grid[col][row];
62             }
63         }
64     }
65
66     return plaintext;
67 }
```

### 3.7 Server Code

```
1     // Server code
2     #include
3     #include
4     #include
5     #include
6     #include
7     #include
8     #include "encrypt.h"
9     #define PORT 8080
10    #define BUFFER_SIZE 256
11    int main() {
12        int encryption = 0;
13        std::cout << "Select Algorithm:\n";
14        std::cout << "1) Caesar\n";
15        std::cout << "2) Vigenere\n";
16        std::cout << "3) Vernam\n";
17        std::cout << "4) Playfair\n";
18        std::cout << "5) Hill\n";
19        std::cout << "6) Rail Fence\n";
20        std::cin >> encryption;
21        int server_fd, client_fd;
22        struct sockaddr_in server_addr, client_addr;
23        socklen_t client_len;
24        char buffer[BUFFER_SIZE];
25        // Create TCP socket
```



```
26     server_fd = socket(AF_INET, SOCK_STREAM, 0);
27     if (server_fd == -1) {
28         perror("Socket creation failed");
29         return 1;
30     }
31     const int enable = 1;
32     if(setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &
33         enable, sizeof(int)) < 0)
34         perror("setsockopt(SO_REUSEADDR) failed");
35     // Set up server address
36     memset(&server_addr, 0, sizeof(server_addr));
37     server_addr.sin_family = AF_INET;
38     server_addr.sin_addr.s_addr = INADDR_ANY;
39     server_addr.sin_port = htons(PORT);
40     // Bind the socket
41     if (bind(server_fd, (struct sockaddr*)&server_addr,
42         sizeof(server_addr)) == -1) {
43         perror("Bind failed");
44         close(server_fd);
45         return 1;
46     }
47     // Listen for connections
48     if (listen(server_fd, 5) == -1) {
49         perror("Listen failed");
50         close(server_fd);
51         return 1;
52     }
53     std::cout << "Server listening on port " << PORT <<
54         std::endl;
55     // Accept a client connection
56     client_len = sizeof(client_addr);
57     client_fd = accept(server_fd, (struct sockaddr*)&
58         client_addr, &client_len);
59     if (client_fd == -1) {
60         perror("Accept failed");
61         close(server_fd);
62         return 1;
63     }
64     std::cout << "Client connected." << std::endl;
65     // Communicate with client
66     while (true) {
67         memset(buffer, 0, BUFFER_SIZE);
68         ssize_t bytes_read = read(client_fd, buffer,
69             BUFFER_SIZE - 1);
70         if (bytes_read <= 0) {
71             if (bytes_read == 0)
72                 std::cout << "Client
73                     disconnected." << std::
74                     endl;
75             else
```

```

69         perror("Read error");
70         break;
71     }
72     std::cout << "Received:\t" << buffer << std
        ::endl;
73     std::cout << "Decrypted:\t" << decrypt(
        buffer, encryption) << std::endl;
74     std::string message = "";
75     std::cout << "Enter Message:\t" ;
76     std::cin >> message;
77     std::string encrypted_message = encrypt(
        message, encryption);
78     std::cout << "Encrypted message:\t" <<
        encrypted_message << std::endl;
79     if (write(client_fd, encrypted_message.c_str
        (), encrypted_message.length()) == -1) {
80         perror("Write error");
81         break;
82     }
83 }
84 close(client_fd);
85 close(server_fd);
86 return 0;
87 }

```

### 3.8 Client Code

```

1
2 // Client code
3 #include
4 #include
5 #include
6 #include
7 #include
8 #include "encrypt.h"
9 #define PORT 8080
10 #define BUFFER_SIZE 256
11 int main() {
12     int encryption = 0;
13     std::cout << "Select Algorithm:\n";
14     std::cout << "1) Caesar\n";
15     std::cout << "2) Vigenere\n";
16     std::cout << "3) Vernam\n";
17     std::cout << "4) Playfair\n";
18     std::cout << "5) Hill\n";
19     std::cout << "6) Rail Fence\n";
20     std::cin >> encryption;
21     int client_fd;

```

```
22     struct sockaddr_in server_addr;
23     char buffer[BUFFER_SIZE];
24     // Create TCP socket
25     client_fd = socket(AF_INET, SOCK_STREAM, 0);
26     if (client_fd == -1) {
27         perror("Socket creation failed");
28         return 1;
29     }
30     // Set up server address
31     memset(&server_addr, 0, sizeof(server_addr));
32     server_addr.sin_family = AF_INET;
33     server_addr.sin_addr.s_addr = INADDR_ANY;
34     server_addr.sin_port = htons(PORT);
35     // Connect to server
36     if (connect(client_fd, (struct sockaddr*)&
37         server_addr, sizeof(server_addr)) == -1) {
38         perror("Connect failed");
39         close(client_fd);
40         return 1;
41     }
42     std::cout << "Connected to server." << std::endl;
43     // Communicate with server
44     while (true) {
45         std::cout << "Enter message: ";
46         std::cin >> buffer;
47         std::string encrypted_message = encrypt(
48             buffer, encryption);
49         strcpy(buffer, encrypted_message.c_str());
50         if (std::strcmp(buffer, "exit") == 0) {
51             break;
52         }
53         // Send message to server
54         if (write(client_fd, buffer, strlen(buffer))
55             == -1) {
56             perror("Write error");
57             break;
58         }
59         // Read response from server
60         memset(buffer, 0, BUFFER_SIZE);
61         ssize_t bytes_read = read(client_fd, buffer,
62             BUFFER_SIZE - 1);
63         if (bytes_read <= 0) {
64             if (bytes_read == 0)
65                 std::cout << "Server
66                     disconnected." << std::
67                     endl;
68             else
69                 perror("Read error");
70             break;
71         }
72     }
```

```
66         std::cout << "Server response: " << buffer
67             << std::endl;
68         std::cout << "Decrypted response: " <<
69             decrypt(buffer, encryption) << std::endl;
70     }
71     close(client_fd);
72     return 0;
73 }
```

## 4 Outputs

### 4.1 Caesar Cipher

```
→ build ./server
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
1
Server listening on port 8080
Client connected.
Received:      KH00R
Decrypted:     HELLO
Enter Message: TEST
Encrypted message:  WHVW
█
```

```
→ build ./client
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
1
Connected to server.
Enter message: HELLO
Server response: WHVW
Decrypted response: TEST
Enter message: █
```

## 4.2 Vigenère Cipher

```
→ build ./server
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
2
Server listening on port 8080
Client connected.
Received:      RIOLF
Decrypted:     HELLO
Enter Message: TEST
Encrypted message:  DIVT
█
```

```
→ build ./client
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
2
Connected to server.
Enter message: HELLO
Server response: DIVT
Decrypted response: TEST
Enter message: █
```

### 4.3 Vernam Cipher

```
→ build ./server
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
3
Server listening on port 8080
Client connected.
Received:

Decrypted:      ABCD
Enter Message: DCBA
Encrypted message:
█
```

```
→ build ./client
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
3
Connected to server.
Enter message: ABCD
Server response:
Decrypted response: DCB
Enter message: █
```

#### 4.4 Playfair Cipher

```
→ build ./server
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
4
Server listening on port 8080
Client connected.
Received:      lckyog
Decrypted:      m i n u a
b c d e f
g h k l o
p q r s t
v w x y z
helxlo
Enter Message: yellow
m i n u a
b c d e f
g h k l o
p q r s t
v w x y z
Encrypted message:      ulkyogxy
█

→ build ./client
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
4
Connected to server.
Enter message: hello
m i n u a
b c d e f
g h k l o
p q r s t
v w x y z
Server response: ulkyogxy
Decrypted response: m i n u a
b c d e f
g h k l o
p q r s t
v w x y z
yelxlowx
Enter message: █
```



## 4.5 Hill Cipher

```
→ build /home/ked1108/Documents/Sem 6/BCSE309/Lab/Lab1//
→ build ./server
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
5
Server listening on port 8080
Client connected.
Received:      axddtc
Decrypted:     hellox
Enter Message: TEST
Encrypted message:  yjpq
█
```

```
→ build ./client
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
5
Connected to server.
Enter message: HELLO
Server response: yjpq
Decrypted response: test
Enter message: █
```

#### 4.6 Rail Fence Cipher (Columnar Transposition Cipher)

```
→ build ./server
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
6
Server listening on port 8080
Client connected.
Received:      EVACDESERODEWIR
Decrypted:     WEAREDISCOVERED
Enter Message: SEEKSHELTER
Encrypted message:  SRETELKEHSE
█
```

```
→ build ./client
Select Algorithm:
1) Caesar
2) Vigenere
3) Vernam
4) Playfair
5) Hill
6) Rail Fence
6
Connected to server.
Enter message: WEAREDISCOVERED
Server response: SRETELKEHSE
Decrypted response: SEEKSHELTER
Enter message: █
```