

Reg no: 22BCE3799
Apurba Koirala
Cryptography and Network Security Lab Assessment I

a. Caesar Cipher

Code:

```
#include <iostream>
#include <cmath>
#include <algorithm>

using namespace std;

string encrypt(string text, int s){
    string result = "";
    for (int i = 0; i < text.length(); i++){
        if (isupper(text[i])){
            result += char(int(text[i] + s - 65) % 26 + 65);
        }
        else {
            result += char(int(text[i] + s - 97) % 26 + 97);
        }
    }

    return result;
}

string decrypt(string text, int s){
    string result = "";
    for (int i = 0; i < text.length(); i++){
        if (isupper(text[i])){
            result += char(int(text[i] - s - 65 + 26) % 26 + 65);
        }
        else {
            result += char(int(text[i] - s - 97 + 26) % 26 + 97);
        }
    }

    return result;
}

int main()
{
    string text;
    int s;
```

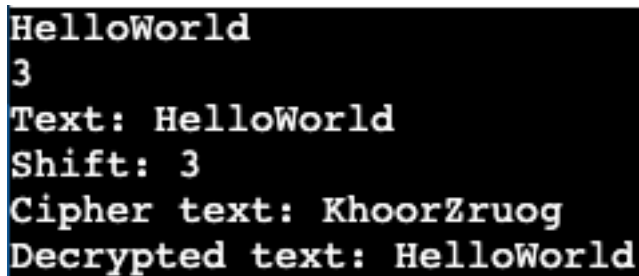
```

cin>>text;
cin>>s;
cout<<"Text: "<<text<<"\n";
cout<<"Shift: "<<s<<"\n";
string encrypted_text = encrypt(text, s);
cout<<"Cipher text: "<<encrypted_text<<"\n";
cout<<"Decrypted text: "<<decrypt(encrypted_text, s);

return 0;
}

```

Output:



```

HelloWorld
3
Text: HelloWorld
Shift: 3
Cipher text: KoorZruog
Decrypted text: HelloWorld

```

b. Playfair Cipher

Code:

```

#include <iostream>
#include <cctype>
#include <string>

using namespace std;

void generateKeyMatrix(const string &key, char keyMatrix[5][5]) {
    bool seen[26] = {false};
    int row = 0, col = 0;

    for (char ch : key) {
        if (ch == 'j') ch = 'i';
        if (!seen[ch - 'a']) {
            keyMatrix[row][col++] = ch;
            seen[ch - 'a'] = true;
            if (col == 5) { row++; col = 0; }
        }
    }

    for (char ch = 'a'; ch <= 'z'; ch++) {
        if (ch == 'j') continue;
        if (!seen[ch - 'a']) {
            keyMatrix[row][col++] = ch;
            seen[ch - 'a'] = true;
            if (col == 5) { row++; col = 0; }
        }
    }
}

```

```

    }
}

```

```

void toLowerCase(string &text) {
    for (char &ch : text) {
        if (isupper(ch)) ch = tolower(ch);
    }
}

```

```

void prepareText(string &text) {
    string prepared = "";
    for (size_t i = 0; i < text.length(); i++) {
        if (text[i] == 'j') text[i] = 'i';
        prepared += text[i];
        if (i + 1 < text.length() && text[i] == text[i + 1]) {
            prepared += 'x';
        }
    }
    if (prepared.length() % 2 != 0) {
        prepared += 'x';
    }
    text = prepared;
}

```

```

void findPosition(char keyMatrix[5][5], char ch, int &row, int &col) {
    if (ch == 'j') ch = 'i';
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (keyMatrix[i][j] == ch) {
                row = i; col = j;
                return;
            }
        }
    }
}

```

```

void encryptText(string &text, char keyMatrix[5][5]) {
    for (size_t i = 0; i < text.length(); i += 2) {
        int r1, c1, r2, c2;
        findPosition(keyMatrix, text[i], r1, c1);
        findPosition(keyMatrix, text[i + 1], r2, c2);

        if (r1 == r2) {
            text[i] = keyMatrix[r1][(c1 + 1) % 5];
            text[i + 1] = keyMatrix[r2][(c2 + 1) % 5];
        } else if (c1 == c2) {
            text[i] = keyMatrix[(r1 + 1) % 5][c1];
            text[i + 1] = keyMatrix[(r2 + 1) % 5][c2];
        } else {
            text[i] = keyMatrix[r1][c2];

```

```

        text[i + 1] = keyMatrix[r2][c1];
    }
}

void postProcessText(string &text) {
    string result = "";
    for (size_t i = 0; i < text.length(); i++) {
        if (i > 0 && text[i] == 'x' && text[i - 1] == text[i + 1]) {
            continue;
        }
        result += text[i];
    }
    text = result;
}

void decryptText(string &text, char keyMatrix[5][5]) {
    for (size_t i = 0; i < text.length(); i += 2) {
        int r1, c1, r2, c2;
        findPosition(keyMatrix, text[i], r1, c1);
        findPosition(keyMatrix, text[i + 1], r2, c2);

        if (r1 == r2) {
            text[i] = keyMatrix[r1][(c1 - 1 + 5) % 5];
            text[i + 1] = keyMatrix[r2][(c2 - 1 + 5) % 5];
        } else if (c1 == c2) {
            text[i] = keyMatrix[(r1 - 1 + 5) % 5][c1];
            text[i + 1] = keyMatrix[(r2 - 1 + 5) % 5][c2];
        } else {
            text[i] = keyMatrix[r1][c2];
            text[i + 1] = keyMatrix[r2][c1];
        }
    }
    postProcessText(text);
}

void printKeyMatrix(char keyMatrix[5][5]) {
    cout << "Key Matrix:\n";
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            cout << keyMatrix[i][j] << " ";
        }
        cout << endl;
    }
}

int main() {
    string plaintext, key;

    cout << "Enter Plaintext: ";
    cin >> plaintext;

```

```

cout << "Enter Key: ";
cin >> key;

toLowerCase(plaintext);
toLowerCase(key);

char keyMatrix[5][5];
generateKeyMatrix(key, keyMatrix);
prepareText(plaintext);

printKeyMatrix(keyMatrix);

string ciphertext = plaintext;
encryptText(ciphertext, keyMatrix);
cout << "Cipher text: " << ciphertext << endl;

decryptText(ciphertext, keyMatrix);
cout << "Decrypted text: " << ciphertext << endl;

return 0;
}

```

Output:

```

Enter Plaintext: canicallnow
Enter Key: winterholidays
Key Matrix:
w i n t e
r h o l d
a y s b c
f g k m p
q u v x z
Cipher text: aytnaybtotr
Decrypted text: canicallnow

```

c. Vigenère Cipher

```
#include <iostream>
#include <string>
using namespace std;

string generateKey(string text, string key) {
    int textLength = text.size();
    int keyLength = key.size();
    for (int i = 0; key.size() < textLength; i++) {
        key.push_back(key[i % keyLength]);
    }
    return key;
}

string toUpperCase(string str) {
    for (size_t i = 0; i < str.size(); i++) {
        if (str[i] >= 'a' && str[i] <= 'z') {
            str[i] = str[i] - 'a' + 'A';
        }
    }
    return str;
}

string encryptText(string text, string key) {
    string cipherText;
    for (size_t i = 0; i < text.size(); i++) {
        char encryptedChar = (text[i] + key[i] - 2 * 'A') % 26 + 'A';
        cipherText.push_back(encryptedChar);
    }
    return cipherText;
}

string decryptText(string cipherText, string key) {
    string originalText;
    for (size_t i = 0; i < cipherText.size(); i++) {
        char decryptedChar = (cipherText[i] - key[i] + 26) % 26 + 'A';
        originalText.push_back(decryptedChar);
    }
    return originalText;
}

int main() {
    string text, keyword;
    cout << "Enter the text: ";
    cin >> text;
    cout << "Enter the keyword: ";
    cin >> keyword;

    text = toUpperCase(text);
```

```

keyword = toUpperCase(keyword);

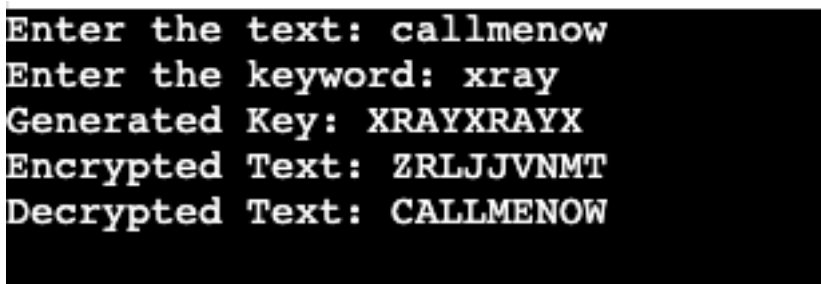
string key = generateKey(text, keyword);
cout << "Generated Key: " << key << endl;

string cipherText = encryptText(text, key);
cout << "Encrypted Text: " << cipherText << endl;

string originalText = decryptText(cipherText, key);
cout << "Decrypted Text: " << originalText << endl;

return 0;
}
Output:

```



```

Enter the text: callmenow
Enter the keyword: xray
Generated Key: XRAYXRAYX
Encrypted Text: ZRLJJVNMT
Decrypted Text: CALLMENOW

```

d. Hill Cipher

```

#include <iostream>
#include <vector>
#include <string>
using namespace std;

vector<int> performMatrixMultiplication(vector<vector<int>> &matrix,
vector<int> &vectorInput, int mod) {
    int size = matrix.size();
    vector<int> result(size, 0);

    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            result[row] += matrix[row][col] * vectorInput[col];
        }
        result[row] = (result[row] % mod + mod) % mod;
    }
    return result;
}

string encryptMessage(string plainText, vector<vector<int>> &keyMatrix,
int matrixSize) {

```

```

string encryptedText = "";

while (plainText.size() % matrixSize != 0) {
    plainText += 'X';
}

for (size_t i = 0; i < plainText.size(); i += matrixSize) {
    vector<int> letterVector(matrixSize);
    for (int j = 0; j < matrixSize; j++) {
        letterVector[j] = plainText[i + j] - 'A';
    }

    vector<int> encryptedVector =
performMatrixMultiplication(keyMatrix, letterVector, 26);

    for (int j = 0; j < matrixSize; j++) {
        encryptedText += (encryptedVector[j] + 'A');
    }
}

return encryptedText;
}

int main() {
    string plainText;
    cout << "Enter the message to encrypt (uppercase letters only): ";
    cin >> plainText;

    int matrixSize;
    cout << "Enter key matrix size: ";
    cin >> matrixSize;

    vector<vector<int>> encryptionKey(matrixSize, vector<int>(matrixSize));
    cout << "Enter the elements of key matrix:\n";
    for (int i = 0; i < matrixSize; i++) {
        for (int j = 0; j < matrixSize; j++) {
            cin >> encryptionKey[i][j];
        }
    }

    string encryptedText = encryptMessage(plainText, encryptionKey,
matrixSize);
    cout << "Encrypted message: " << encryptedText << endl;

    return 0;
}

```


}

Enter the message to encrypt (uppercase letters only): TOMORROW

Enter key matrix size: 2

Enter the elements of key matrix:

1

2

2

1

Encrypted message: VAOMZZGY