Information and Network Security
2CSDE54
Practical 3

21BCE020

Columnar Transposition Cipher

```
#include <bits/stdc++.h>
using namespace std;

string encrypt(char msg[], int key)
{
    int count = 0;
    for (int i = 0; msg[i]; i++)
        if (msg[i] != ' ')
            msg[count++] = msg[i];
    msg[count] = '\0';
    int msg_len = strlen(msg);
    char CT_matrix[key][key];
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < key; j++)
        {
            CT_matrix[i][j] = '*';
        }
    }
    int k = 0;
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < key; j++)
        {
            if (k < msg_len)
            {
                CT_matrix[i][j] = msg[k];
                k++;
            }
            else
            {
                CT_matrix[i][j] = '*';
            }
        }
```

```
    }
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < key; j++)
        {
            if (CT_matrix[i][j] == '\0')
                break;
            cout << CT_matrix[i][j] << " ";
        }
        cout << endl;
    }
    string cipher_text = "";
    cout << "CT: ";
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < key; j++)
        {
            if (CT_matrix[j][i] != '*')
            {
                cipher_text = cipher_text + CT_matrix[j][i];
                cout << CT_matrix[j][i] << "";
            }
        }
    }
    fstream fp;
    fp.open("rail_op.txt", fstream::out);
    for (int i = 0; i < cipher_text.length(); i++){fp <<
cipher_text[i];}
    return cipher_text;
}
void decrypt(string msg, int key)
{
    int count = 0;
    for (int i = 0; msg[i]; i++)
        if (msg[i] != ' ')
            msg[count++] = msg[i];
    msg[count] = '\0';
    char dex[key][key];
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < key; j++)
        {
```

```
            dex[i][j] = '*';
        }
    }
    int msg_len = msg.length();
    cout << "\n";
    int k = 0;
    int x = msg_len % key;
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < key; j++)
        {
            if (i <= (x - 1) || j <= (key - 2))
            {
                dex[j][i] = msg[k];
                k++;
            }
            else
            {
                dex[j][i] = '*';
            }
        }
    }
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < key; j++)
        {

            if (dex[i][j] == '\0')
                break;
            cout << dex[i][j] << " ";
        }
        cout << endl;
    }
    string plain_text = " ";
    cout << "PT: ";
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < key; j++)
        {
            if (dex[i][j] != '*')
            {
                plain_text = plain_text + dex[j][i];
```

```
                cout << dex[i][j] << "";
            }
        }
    }
}

int main()
{
    char str[100000];
    int no_of_chars = 0;
    int i = 0;
    ifstream myfile("rail_ip.txt");
    while (!myfile.eof())
    {
        myfile.get(str[i]);
        i++;
        no_of_chars++;
    }
    string op_str;
    int l;
    cout << "Length of key: ";
    cin >> l;
    op_str = encrypt(str, l);
    decrypt(op_str, l);
    return 0;
}
```

O/P

col_transposition.cpp 2 | rail_ip.txt | rail_op.txt

rail_ip.txt

```
NOTHING IN THE WORLD IS MORE DANGEROUS THAN SINCERE IGNORANCE AND CONSCIENTIOUS STUPIDITY
```

col_transposition.cpp 2 | rail_ip.txt | rail_op.txt

rail_op.txt

```
NTSESNDTIOHMRIOCIDTEOONROOIHWRUCANUTIOESENSSYNRDTRCCSGLAHEEITIDNAIAEUNIGNGNNP
```

```
Length of key: 9
N O T H I N G I N
T H E W O R L D I
S M O R E D A N G
E R O U S T H A N
S I N C E R E I G
N O R A N C E A N
D C O N S C I E N
T I O U S S T U P
I D I T Y * * * *
CT: NTSESNDTIOHMRIOCIDTEOONROOIHWRUCANUTIOESENSSYNRDTRCCSGLAHEEITIDNAIAEUNIGNGNNP
N O T H I N G I N
T H E W O R L D I
S M O R E D A N G
E R O U S T H A N
S I N C E R E I G
N O R A N C E A N
D C O N S C I E N
T I O U S S T U P
I D I T Y * * * *
PT: NOTHINGINTHEWORLDISMOREDANGEROUSTHANSINCEREIGNORANCEANDCONSCIENTIOUSSTUPIDITY
```

Keyword Columnar Transposition

```
#include <iostream>
#include <fstream>
#include <map>
using namespace std;

string const key = "HACK";
map<int, int> key_map;

void permutation_order() {for (int i = 0; i < key.length();
i++){key_map[key[i]] = i;}}


string read_data(const string &filename) {
    ifstream file(filename);
    string content;
    getline(file, content);
    file.close();
    return content;
}


void write_data(const string &filename, const string &content)
{
    ofstream file(filename);
```

```
    file << content;
    file.close();
}

string encryptMessage(string msg) {
    int row, col, j;
    string cipher = "";
    //calc col length
    col = key.length();
    //calc max row length
    row = msg.length() / col;
    if (msg.length() % col)
        row += 1;
    char matrix[row][col];
    for (int i = 0, k = 0; i < row; i++) {
        for (int j = 0; j < col;) {
            if (msg[k] == '\0') {
                //padding to char
                matrix[i][j] = '_';
                j++;
            }

            if (isalpha(msg[k]) || msg[k] == ' ') {
                //only space and char to mat
                matrix[i][j] = msg[k];
                j++;
            }
            k++;
        }
    }

    for (map<int, int>::iterator ii = key_map.begin(); ii !=
key_map.end(); ++ii)
    {
        j = ii->second;
        //CT from col text using key
        for (int i = 0; i < row; i++)
        {
            if (isalpha(matrix[i][j]) || matrix[i][j] == ' '
|| matrix[i][j] == '_')
                cipher += matrix[i][j];
        }
```

```
    }

    return cipher;
}

string decryptMessage(string cipher) {
    int col = key.length(); //col length
    int row = cipher.length() / col; //max row length
    char CT_matrix[row][col];
    //add char to mat
    for (int j = 0, k = 0; j < col; j++)
        for (int i = 0; i < row; i++)
            CT_matrix[i][j] = cipher[k++];
    //update order of key for dec
    int index = 0;
    for (map<int, int>::iterator ii = key_map.begin(); ii !=
key_map.end(); ++ii)
        ii->second = index++;
    //arrange matrix col-wise to permutation order
    char decCipher[row][col];
    map<int, int>::iterator ii = key_map.begin();
    int k = 0;
    for (int l = 0, j; key[l] != '\0'; k++) {
        j = key_map[key[l++]];
        for (int i = 0; i < row; i++) {
            decCipher[i][k] = CT_matrix[i][j];
        }
    }
    string msg = ""; //msg from matrix
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            if (decCipher[i][j] != '_')
                msg += decCipher[i][j];
        }
    }
    return msg;
}

int main(void) {
    string msgFile = "keyword_inp.txt";
    string keyFile = "keyword_key.txt";
    string msg = read_data(msgFile);
```

```
    string key = read_data(keyFile);
    permutation_order();
    string cipher = encryptMessage(msg);
    write_data("keyword_enc_op.txt", cipher);
    string decryptedMsg = decryptMessage(cipher);
    write_data("keyword_dec_op.txt", decryptedMsg);
    return 0;
}
```

O/P

```
rail_fence_col.cpp 7   keyword_inp.txt ×   keyword_enc_op.txt   keyword_dec_op.txt
keyword_inp.txt
1  THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG
```

```
rail_fence_col.cpp 7   keyword_inp.txt   keyword_enc_op.txt ×   keyword_dec_op.txt
keyword_enc_op.txt
1  HU WOUDEHADEIBNXM REZOTQKOFJEVTL  CR  PO  YG
```

```
rail_fence_col.cpp 7   keyword_inp.txt   keyword_enc_op.txt   keyword_dec_op.txt ×
keyword_dec_op.txt
1  THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG
```

ONE PAD CIPHER

```
#include <bits/stdc++.h>
using namespace std;

string encrypt(string text, string key)
{
    string CT = "";
    int cipher[key.length()];

    for (int i = 0; i < key.length(); i++)
    {
        cipher[i] = text.at(i) - 'A' + key.at(i) - 'A';
        if (cipher[i] > 25)
        {
            cipher[i] = cipher[i] - 26;
        }
    }
```

```
    for (int i = 0; i < key.length(); i++)
    {
        int x = cipher[i] + 'A';
        CT += (char)x;
    }

    return CT;
}

static string decrypt(string s, string key)
{
    string PT = "";
    int plain[key.length()];

    for (int i = 0; i < key.length(); i++)
    {
        plain[i] = s.at(i) - 'A' - (key.at(i) - 'A');
        if (plain[i] < 0)
        {
            plain[i] = plain[i] + 26;
        }
    }

    for (int i = 0; i < key.length(); i++)
    {
        int x = plain[i] + 'A';
        PT += (char)x;
    }

    return PT;
}

int main()
{
    string PT;
    ifstream inputFile("pad_ip.txt");
    if (inputFile.is_open())
    {
        getline(inputFile, PT);
        inputFile.close();
    }
    else
```

```
    {
        cout << "Error";
        return 1;
    }

    string key;
    ifstream keyFile("key.txt");
    if (keyFile.is_open())
    {
        getline(keyFile, key);
        keyFile.close();
    }
    else
    {
        cout << "Error";
        return 1;
    }

    for (int i = 0; i < PT.length(); i++)
    {
        PT[i] = toupper(PT[i]);
    }
    for (int i = 0; i < key.length(); i++)
    {
        key[i] = toupper(key[i]);
    }

    string encryptedText = encrypt(PT, key);

    ofstream outputFile("pad_op.txt");
    if (outputFile.is_open())
    {
        outputFile << "Cipher text: " << encryptedText <<
endl;
        outputFile << "Original message: " <<
decrypt(encryptedText, key);
        outputFile.close();
    }
    else
    {
        cout << "Error";
        return 1;
```

```
    }

    return 0;
}
```

O/P

one_pad.cpp 2 | pad_ip.txt | key.txt | pad_op.txt

pad_ip.txt

```
IF
```

one_pad.cpp 2 | pad_ip.txt | key.txt | pad_op.txt

key.txt

```
PH
```

one_pad.cpp 2 | pad_ip.txt | key.txt | pad_op.txt

pad_op.txt

```
Cipher text: XM
Original message: IF
```