National Institute of Technology, Tiruchirappalli

Department of Computer Applications

Information Security Lab

Lab 2

Submitted to:

Dr. Ghanshyam Bopche

Submitted by:

Akriti Upadhyay

205120007

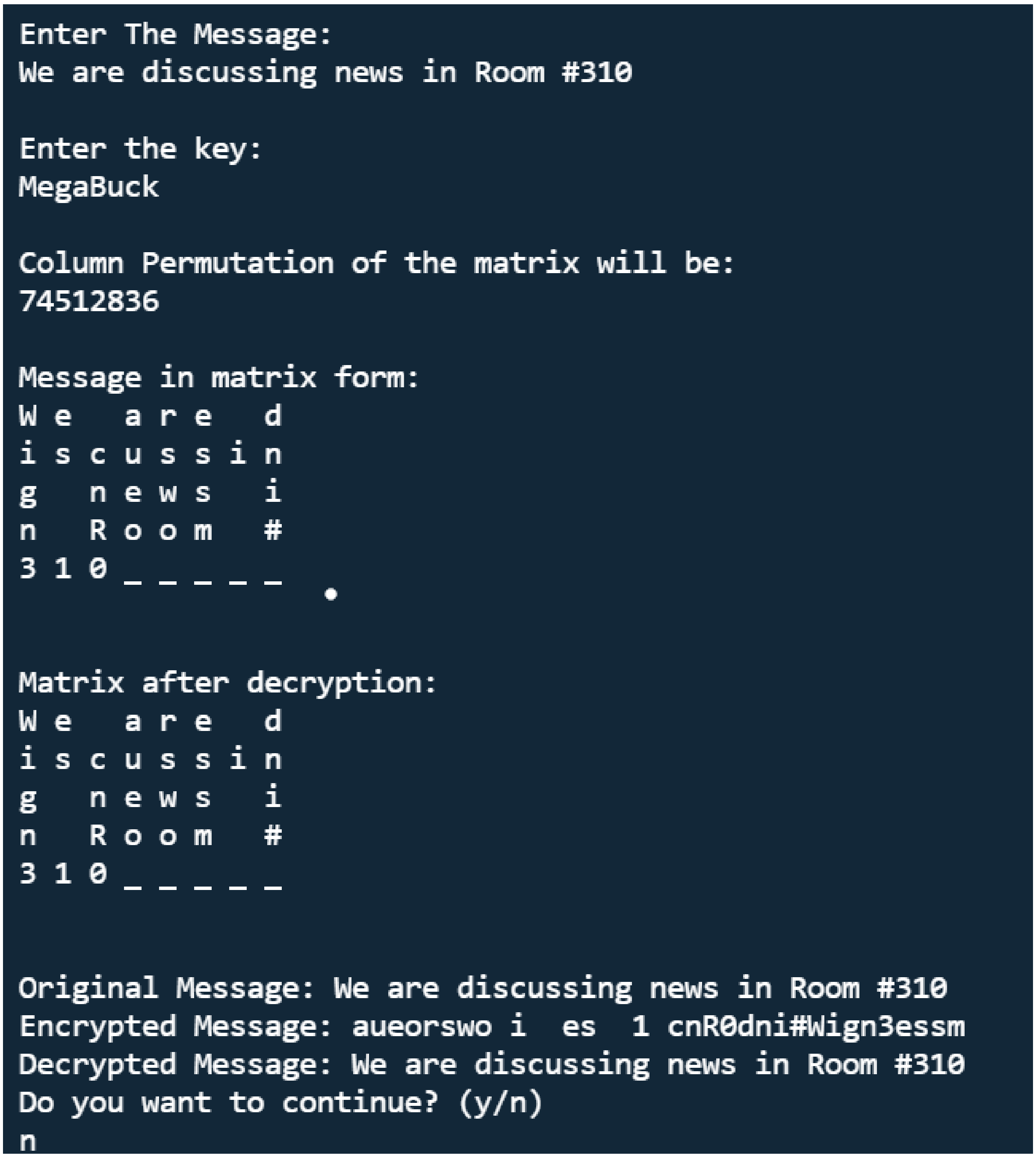
MCA – 2nd Year

Classical Encryption Algos (cond…)

5. Transpositional Cipher:

```cpp
#include<iostream>
#include<vector>
#include<algorithm>
#include<math.h>
using namespace std;

// ENCRYPTION
string encryption(string msg, string key){
    int msgLen = msg.length();
    int keyLen = key.length();

    // creating matrix to store the msg
    int row = ceil(msgLen/(float)keyLen);
    int col = keyLen;
    char matrix[row][col];
    for(int i=0, k=0; i<row; i++){
        for(int j=0; j<col; j++){
            if(k < msgLen)
                matrix[i][j] = msg[k++];
            else
                matrix[i][j] = '_';
        }
    }

    // Printing the matrix
    cout<<"\n\nMessage in matrix form: \n";
    for(int i=0; i<row; i++){
        for(int j=0; j<col; j++){
            cout<<matrix[i][j]<<" ";
        }
        cout<<endl;
    }

    // Order of picking the columns for cipher text
    vector<int> order(keyLen);
    for(int i=0; i<key.size(); i++){
        order[key[i]-'0'-1] = i;
    }
    // generating cipher text
    string output = "";
    for(int i=0; i<order.size(); i++){
        for(int j=0; j<row; j++){
            if(matrix[j][order[i]] != '_'){
                output += matrix[j][order[i]];
            }
        }
    }
    return output;
}

// DECRYPTION
string decryption(string msg, string key){
    string output = "";
    int msgLen = msg.length();
    int keyLen = key.length();

    // creating matrix to store the msg
    int row = ceil(msgLen/(float)keyLen);
    int col = keyLen;
    int empty_cells = keyLen - (msgLen%keyLen);
    char matrix[row][col];

    // assigning default value to cols which are supposed to be empty
    for(int i=row-1, j=col-1; empty_cells && j>=0; j--){
        matrix[i][j] = '_';
        empty_cells--;
    }

    // Order of storing the encrypted msg coloumn-wise in the matrix
    vector<int> order(keyLen);
    for(int i=0; i<key.size(); i++){
        order[key[i]-'0'-1] = i;
    }

    // storing the encrypted msg in matrix
    for(int i=0, k=0; i<order.size(); i++){
        for(int j=0; j<row; j++){
            if(matrix[j][order[i]] != '_'){
                matrix[j][order[i]] = msg[k++];
            }
        }
    }

    // Printing the matrix and retrieving original msg
    cout<<"\n\nMatrix after decryption: \n";
    for(int i=0; i<row; i++){
        for(int j=0; j<col; j++){
            cout<<matrix[i][j]<<" ";
            if(matrix[i][j]!='_')
                output += matrix[i][j];
        }
        cout<<endl;
    }

    return output;
}

int main(){
    while(1){
        string msg;
        cout<<"\nEnter The Message: \n";
        cin.ignore();
        getline(cin,msg);

        string key;
        cout<<"\nEnter the key: \n";
        cin>>key;

        // Finding the column permutation of the matrix
        transform(key.begin(), key.end(), key.begin(), ::toupper);
        string sortedKey = key;
        sort(sortedKey.begin(), sortedKey.end());
        string colPermutation = "";
        for(int i=0; i<key.size(); i++){
            for(int j=0; j<sortedKey.size(); j++){
                if(key[i] == sortedKey[j])
                    colPermutation += to_string(j+1);
            }
        }
        cout<<"\nColumn Permutation of the matrix will be: 
\n"<<colPermutation;
```

```
        string encrypt = encryption(msg, colPermutation);
        string decrypt = decryption(encrypt, colPermutation);
        cout<<"\n\nOriginal Message: "<<msg;
        cout<<"\nEncrypted Message: "<<encrypt;
        cout<<"\nDecrypted Message: "<<decrypt;

        char choice;
        cout<<"\nDo you want to continue? (y/n)\n";
        cin>>choice;
        if(choice == 'n' || choice == 'N')
            break;
    }
    return 0;
}
```



```
Enter The Message:
We are discussing news in Room #310

Enter the key:
MegaBuck

Column Permutation of the matrix will be:
74512836

Message in matrix form:
W e   a r e   d
i s c u s s i n
g   n e w s   i
n   R o o m   #
3 1 0 _ _ _ _ _


Matrix after decryption:
W e   a r e   d
i s c u s s i n
g   n e w s   i
n   R o o m   #
3 1 0 _ _ _ _ _


Original Message: We are discussing news in Room #310
Encrypted Message: aueorswo i  es  1 cnR0dni#Wign3essm
Decrypted Message: We are discussing news in Room #310
Do you want to continue? (y/n)
n
```

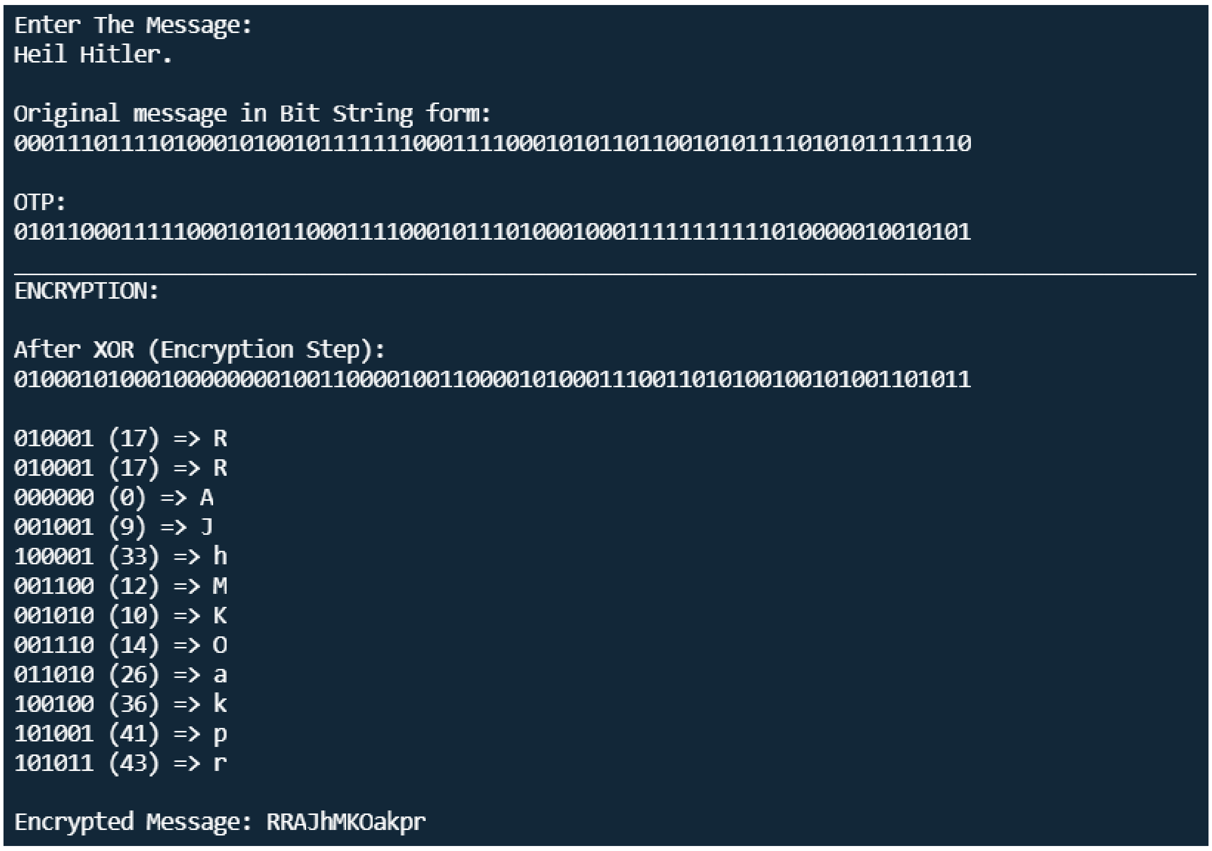
---

6. One Time Pad (OTP):

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

// GETTING BINARY FORM OF A DECIMAL NUMBER
string getBinary(int num){
    string bin = "";
    for(int i = 1<<5; i>0; i=i/2){
```

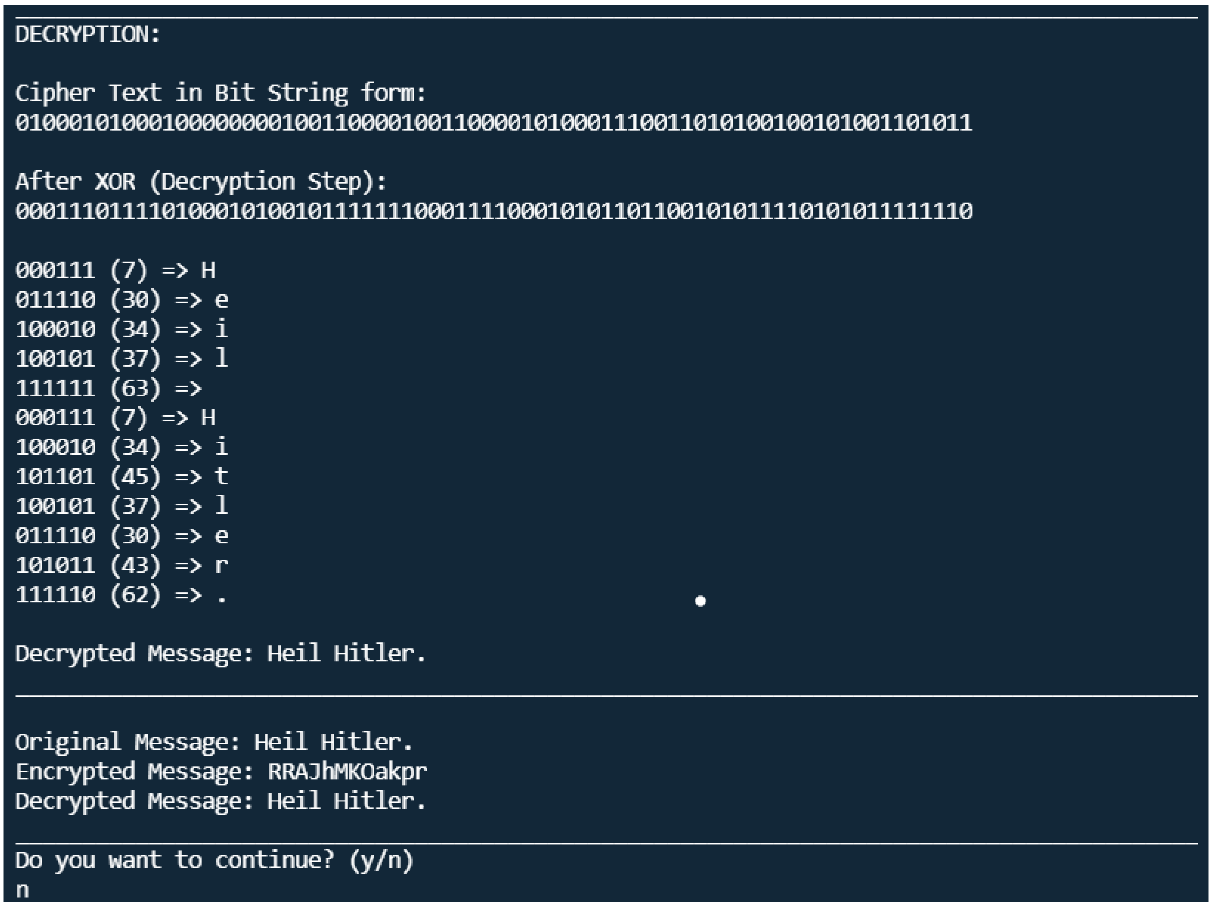
```cpp
        if((num & i) != 0)
            bin += to_string(1);
        else
            bin += to_string(0);
    }
    return bin;
}

// GENERATING BIT STRING FROM THE MSG
string generateBitString(string msg, string map){
    string bit = "";
    // cout<<"\nBinary representation of message characters:\n";
    for(int i=0; i<msg.length(); i++){
        int index = map.find(msg[i]);
        string binary = getBinary(index);
        bit += binary;
        // cout<<msg[i]<<" => "<<getBinary(index)<<" ("<<index<<")\n";
    }
    return bit;
}

// GENERATING A RANDOM OTP
string generateOTP(int len){
    string otp = "";
    for(int i=0; i<len; i++)
        otp += to_string(rand()%2);
    return otp;
}


// GETTING CHARACTER FROM DECIMAL VAL (Bin -> Dec)
char getCharacter(string binary, string map){
    char reqChar;
    int dec_val = 0;
    // initial val of base = 1 (2^0)
    int base = 1;
    int temp = stoi(binary);
    while(temp){
        int last_digit = temp%10;
        temp /= 10;
        dec_val += last_digit * base;
        base *= 2;
    }
    reqChar = map[dec_val];
    cout<<binary<<" ("<<dec_val<<") => "<<reqChar<<endl;
    return reqChar;
}


// XOR OF TWO BINARY STRINGS
string getXOR(string msg, string otp){
    string XOR = "";
    for(int i=0; i<msg.size(); i++){
        XOR += to_string((msg[i]-'0') ^ (otp[i]-'0'));
    }
    return XOR;
}

// GENERATE MSG FROM XOR STRING
string XORToMsg(string XOR, string map){
    // getting cipher string from the XOR string obtained
    // 2^5 = 32 (we are representing binary form in 6 bits) (0 - 63)
    string msg = "";
    int i=0;
    while(i<XOR.size()){
        // combining 6 bits at a time to retrieve the corresponding
character from the binary form
        string binOfChar = "";
        for(int j=i; j<i+6; j++){
            binOfChar += XOR[j];
        }
        i = i+6;
        msg += getCharacter(binOfChar, map);
    }
    return msg;
}

// ENCRYPTION
string encryption(string msg, string otp, string map){
    string XOR = getXOR(msg, otp);
    cout<<"\nAfter XOR (Encryption Step):\n"<<XOR<<"\n\n";

    string output = XORToMsg(XOR, map);
    cout<<"\nEncrypted Message: "<<output<<endl;
    return output;
}
```

```
// DECRYPTION
string decryption(string msg, string otp, string map){
    string bitString = generateBitString(msg, map);
    cout<<"\nCipher Text in Bit String form:\n"<<bitString<<endl;

    string XOR = getXOR(bitString, otp);
    cout<<"\nAfter XOR (Decryption Step):\n"<<XOR<<"\n\n";

    string output = XORToMsg(XOR, map);
    cout<<"\nDecrypted Message: "<<output<<endl;
    return output;
}

int main(){
    while(1){
        string map =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890. ";
        string msg;
        cout<<"\nEnter The Message: \n";
        cin.ignore();
        getline(cin,msg);

        // generate bit string
        string bitString = generateBitString(msg, map);
        cout<<"\nOriginal message in Bit String form:\n"<<bitString;
        // generate OTP
        string otp = generateOTP(bitString.length());
        cout<<"\n\nOTP: \n"<<otp<<"\n";


cout<<"_________________________________________________________________
_________________________";
        cout<<"\nENCRYPTION:\n";
        string encrypt = encryption(bitString, otp, map);

cout<<"_________________________________________________________________
_________________________";
        cout<<"\nDECRYPTION:\n";
        string decrypt = decryption(encrypt, otp, map);
cout<<"_________________________________________________________________
_________________________";

        cout<<"\n\nOriginal Message: "<<msg;
        cout<<"\nEncrypted Message: "<<encrypt;
        cout<<"\nDecrypted Message: "<<decrypt;

cout<<"\n_______________________________________________________________
___________________________";

        char choice;
        cout<<"\nDo you want to continue? (y/n)\n";
        cin>>choice;
        if(choice == 'n' || choice == 'N')
            break;
    }
    return 0;
}
```



```
Enter The Message:
Heil Hitler.

Original message in Bit String form:
000111011110100010100101111111000111100010101101100101011110101011111110

OTP:
010110001111100010101100011110001011101000100011111111111010000010010101
```

```
ENCRYPTION:

After XOR (Encryption Step):
010001010001000000001001100001001100001010001110011010100100101001101011

010001 (17) => R
010001 (17) => R
000000 (0) => A
001001 (9) => J
100001 (33) => h
001100 (12) => M
001010 (10) => K
001110 (14) => O
011010 (26) => a
100100 (36) => k
101001 (41) => p
101011 (43) => r

Encrypted Message: RRAJhMKOakpr
```



```
DECRYPTION:

Cipher Text in Bit String form:
010001010001000000001001100001001100001010001110011010100100101001101011

After XOR (Decryption Step):
000111011110100010100101111111000111100010101101100101011110101011111110

000111 (7) => H
011110 (30) => e
100010 (34) => i
100101 (37) => l
111111 (63) =>
000111 (7) => H
100010 (34) => i
101101 (45) => t
100101 (37) => l
011110 (30) => e
101011 (43) => r
111110 (62) => .

Decrypted Message: Heil Hitler.
```

```
Original Message: Heil Hitler.
Encrypted Message: RRAJhMKOakpr
Decrypted Message: Heil Hitler.
```

```
Do you want to continue? (y/n)
n
```

X-X-X-X