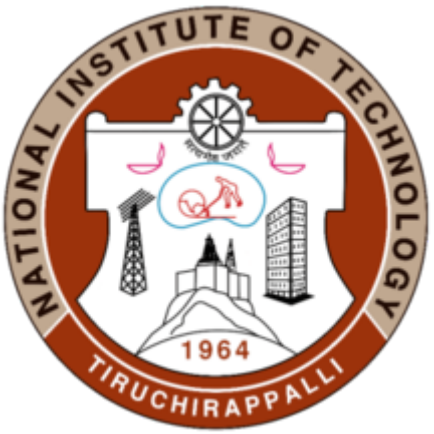
National Institute of Technology, Tiruchirappalli



Department of Computer Applications

Information Security Lab

Lab 1-4

Submitted to:

Dr. Ghanshyam Bopche

Submitted by:

Akriti Upadhyay

205120007

MCA – 2nd Year

Lab 1:

Classical Encryption Algos

1. Caesar Cipher:

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

int main(){
    while(1){
        string msg;
        cout<<"\nEnter the message: \n";
        cin.ignore();
        getline(cin, msg);
        int key;
        cout<<"Enter the key: ";
        cin>>key;
        char choice;
        cout<<"What you want to do with the message?\n1. Encrypt
message\n2. Decrypt message\n";
        cin>>choice;
        // ENCRYPTION
        if(choice == '1'){
            for(int i=0; i<msg.size(); i++){
                //encrypting lower case letters
                if(msg[i]>='a' && msg[i]<='z'){
                    msg[i] = (((msg[i]-'a') + key) % 26) + 'a';
                }
                //encrypting upper case letters
                else if(msg[i]>='A' && msg[i]<='Z'){
                    msg[i] = (((msg[i]-'A') + key) % 26) + 'A';
                }
            }
            cout<<"\nEncrypted Message: "<<msg<<"\n\n";
        }
        // DECRYPTION
        else if(choice == '2'){
            for(int i=0; i<msg.size(); i++){
                //decrypting lower case letters
                if(msg[i]>='a' && msg[i]<='z'){
```

```
                    msg[i] = ((((msg[i]-'a') - key) + 26) % 26) + 'a';
                }
                //encrypting upper case letters
                else if(msg[i]>='A' && msg[i]<='Z'){
                    msg[i] = ((((msg[i]-'A') - key) + 26) % 26) + 'A';
                }
            }
            cout<<"\nDecrypted Message: "<<msg<<"\n\n";
        }
        else{
            cout<<"Invalid Choice, Try again!";
        }
        cout<<"Do you want to continue? (y/n)\n";
        cin>>choice;
        if(choice == 'n')
            break;
    }
    return 0;
}
```

```
Enter the message:
 Hello World I am Akriti Upadhyay
Enter the key: 6
What you want to do with the message?
1. Encrypt message
2. Decrypt message
1

Encrypted Message: Nkrru Cuxrj O gs Gqxozo Avgjnege

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

Enter the message:
Nkrru Cuxrj O gs Gqxozo Avgjnege
Enter the key: 6
What you want to do with the message?
1. Encrypt message
2. Decrypt message
2

Decrypted Message: Hello World I am Akriti Upadhyay
```

---

2. Double Transposition Cipher:

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

vector<vector<char>> encrypt_step_1(vector<vector<char>> matrix,
vector<int> row_permute, int row){
    vector<vector<char>> temp_matrix(row);
    for(int i=0; i<row; i++){
        temp_matrix[i] = matrix[row_permute[i]-1];
    }
    return temp_matrix;
}

vector<vector<char>> encrypt_step_2(vector<vector<char>> matrix,
vector<int> col_permute, int row, int col){
    vector<vector<char>> temp_matrix(row,vector<char>(col));
    for(int i=0; i<col; i++){
        for(int j=0; j<row; j++){
            temp_matrix[j][i] = matrix[j][col_permute[i]-1];
        }
    }
    return temp_matrix;
}

vector<vector<char>> decrypt_step_1(vector<vector<char>> matrix,
vector<int> col_permute, int row, int col){
    vector<vector<char>> temp_matrix(row,vector<char>(col));
    for(int i=0; i<col; i++){
        for(int j=0; j<row; j++){
            temp_matrix[j][col_permute[i]-1] = matrix[j][i];
        }
    }
    return temp_matrix;
}


vector<vector<char>> decrypt_step_2(vector<vector<char>> matrix,
vector<int> row_permute, int row){
    vector<vector<char>> temp_matrix(row);
    for(int i=0; i<row; i++){
```

```
        temp_matrix[row_permute[i]-1] = matrix[i];
    }
    return temp_matrix;
}

void print_matrix(vector<vector<char>> matrix){
    int r = matrix.size();
    int c = matrix[0].size();
    for(int i=0; i<r; i++){
        for(int j=0; j<c; j++){
            if(matrix[i][j] != '0')
                cout<<matrix[i][j]<<" ";
        }
        cout << "\n";
    }
    cout<<"\n";
}

string print_message(vector<vector<char>> matrix){
    string msg;
    int r = matrix.size();
    int c = matrix[0].size();
    for(int i=0; i<r; i++){
        for(int j=0; j<c; j++){
            if(matrix[i][j] != '0')
                msg.push_back(matrix[i][j]);
        }
    }
    return msg;
}


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

        // Forming a grid of string
        int col = ceil(sqrt(len));
        int row = sqrt(len);
```

```
        if(row*col<len)
            row = col;
        vector<vector<char>> matrix(row,vector<char>(col,'0'));

        for(int i=0, k=0; i<row && k<len; i++){
            for(int j=0; j<col && k<len; j++){
                matrix[i][j] = msg[k++];
            }
        }
        cout << "\nOriginal message in form of a Grid: \n";
        print_matrix(matrix);

        // Generating random permutations for rearranging rows and
columns
        vector<int> row_permute(row);
        for(int i=0; i<row; i++)
            row_permute[i] = i+1;
        for(int i=1; i<row; i++)
            swap(row_permute[i], row_permute[rand()%i]);

        vector<int> col_permute(col);
        for(int i=0; i<col; i++)
            col_permute[i] = i+1;
        for(int i=1; i<col; i++)
            swap(col_permute[i], col_permute[rand()%i]);

        //Rearranging rows
        matrix = encrypt_step_1(matrix, row_permute, row);
        cout<<"Matrix after rearranging rows [Encryption Step 1]: \n";
        print_matrix(matrix);

        //Rearranging Cols
        matrix = encrypt_step_2(matrix,col_permute,row,col);
        cout<<"Matrix after rearranging columns [Encryption Step 2]:
\n";
        print_matrix(matrix);

        // Encrypted Msg
        cout<<"Encrypted Message: "<<print_message(matrix)<<"\n";

        //Getting back the original columns
        matrix = decrypt_step_1(matrix,col_permute,row,col);
```

```
        cout << "\nMatrix after getting back original column
arrangement [Decryption Step 1]: \n";
        print_matrix(matrix);

        //Getting back the original rows
        matrix = decrypt_step_2(matrix,row_permute,row);
        cout << "Matrix after getting back original row arrangement
[Decryption Step 2]: \n";
        print_matrix(matrix);

        // Decrypted Msg
        cout<<"Decrypted Message: "<<print_message(matrix)<<"\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:
WE ARE DISCUSSING NEWS

Original message in form of a Grid:
W E   A R
E   D I S
C U S S I
N G   N E
W S

Matrix after rearranging rows [Encryption Step 1]:
W S
W E   A R
E   D I S
C U S S I
N G   N E

Matrix after rearranging columns [Encryption Step 2]:
S W
  R A E W
D S I   E
S I S U C
  E N G N

Encrypted Message: SW RAEWDSI ESISUC ENGN

Matrix after getting back original column arrangement [Decryption Step 1]:
W S
W E   A R
E   D I S
C U S S I
N G   N E

Matrix after getting back original row arrangement [Decryption Step 2]:
W E   A R
E   D I S
C U S S I
N G   N E
W S

Decrypted Message: WE ARE DISCUSSING NEWS
```

---

3. Monoalphabetic Substitutional Cipher:

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

int main(){
    unordered_map<char,char> map = {
        {'A', 'Q'}, {'B', 'W'}, {'C', 'E'}, {'D', 'R'}, {'E', 'T'},
{'F', 'Y'}, {'G', 'U'}, {'H', 'I'}, {'I', 'O'}, {'J', 'P'}, {'K', 'A'},
```

```
{'L', 'S'}, {'M', 'D'}, {'N', 'F'}, {'O', 'G'}, {'P', 'H'}, {'Q', 'J'},
{'R', 'K'}, {'S', 'L'}, {'T', 'Z'}, {'U', 'X'}, {'V', 'C'}, {'W', 'V'},
{'X', 'B'}, {'Y', 'N'}, {'Z', 'M'},
        {'a', 'q'}, {'b', 'w'}, {'c', 'e'}, {'d', 'r'}, {'e', 't'},
{'f', 'y'}, {'g', 'u'}, {'h', 'i'}, {'i', 'o'}, {'j', 'p'}, {'k', 'a'},
{'l', 's'}, {'m', 'd'}, {'n', 'f'}, {'o', 'g'}, {'p', 'h'}, {'q', 'j'},
{'r', 'k'}, {'s', 'l'}, {'t', 'z'}, {'u', 'x'}, {'v', 'c'}, {'w', 'v'},
{'x', 'b'}, {'y', 'n'}, {'z', 'm'}, {' ', '$'}
    };
    while(1){
        string msg;
        cout<<"\nEnter the message: \n";
        cin.ignore();
        getline(cin, msg);
        char choice;
        cout<<"What you want to do with the message?\n1. Encrypt
message\n2. Decrypt message\n";
        cin>>choice;
        // ENCRYPTION
        if(choice == '1'){
            for(int i=0; i<msg.size(); i++){
                msg[i] = map[msg[i]];
            }
            cout<<"\nEncrypted Message: "<<msg<<"\n\n";
        }
        // DECRYPTION
        else if(choice == '2'){
            for(int i=0; i<msg.size(); i++){
                // find the key in map with value msg[i], and assign it
to msg[i]
                for(auto &it:map){
                    if(it.second == msg[i]){
                        msg[i] = it.first;
                        break;
                    }
                }
            }
            cout<<"\nDecrypted Message: "<<msg<<"\n\n";
        }
        else{
            cout<<"Invalid Choice, Try again!";
        }
        cout<<"Do you want to continue? (y/n)\n";
```

```
        cin>>choice;
        if(choice == 'n')
            break;
    }
    return 0;
}
```

```
Enter the message:
 Hello World I am Akriti Upadhyay
What you want to do with the message?
1. Encrypt message
2. Decrypt message
1

Encrypted Message: Itssg$Vgksr$O$qd$Qakozo$Xhqrinqn

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

Enter the message:
Itssg$Vgksr$O$qd$Qakozo$Xhqrinqn
What you want to do with the message?
1. Encrypt message
2. Decrypt message
2

Decrypted Message: Hello World I am Akriti Upadhyay

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

---

4. Polyalphabetic Substitutional Cipher:

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

string generateKey(string msg, string keyWord){
    int sizeKeyword = keyWord.size();
    int msgSize = msg.size();
    // if keyword is longer than the msg
    if(sizeKeyword > msgSize){
        return keyWord.substr(0, msgSize);
    }
    // msg >= keyword
    for(int i=sizeKeyword; i<msg.size(); i++){
```

```
        keyWord.push_back(keyWord[i%sizeKeyword]);
    }
    return keyWord;
}
string encryption(string msg, string key) {
    string output;
    for(int i=0; i<msg.size(); i++){
        // converting to range 0-25
        char x = (msg[i] + key[i]) % 26;
        // convert to range of ASCII Alphabet (65-90 | A-Z)
        if(msg[i] == ' ')
            x = ' ';
        else
            x = x + 'A';
        output.push_back(x);
    }
    return output;
}
string decryption(string encrypt, string key) {
    string output;
    int j=0;
    for (int i=0; i<encrypt.size(); i++) {
        // converting to range 0-25
        char x = (encrypt[i] - key[i] + 26) % 26;
        // convert to range of ASCII Alphabet (65-90 | A-Z)
        if(encrypt[i] == ' ')
            x = ' ';
        else
            x = x + 'A';
        output.push_back(x);
    }
    return output;
}

int main() {
    while(1){
        string keyWord;
        cout<<"\nEnter the key word (In Capital Letters): \n";
        cin>>keyWord;
        string msg;
        cout<<"\nEnter the Message (In Capital Letters): \n";
        cin.ignore();
        getline(cin,msg);
```

```
        string key = generateKey(msg, keyWord);
        string encrypt = encryption(msg, key);
        string decrypt = decryption(encrypt, key);
        cout<<"\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 key word (In Capital Letters):
MEGABUCK

Enter the Message (In Capital Letters):
WE ARE DISCUSSING NEWS

Original Message: WE ARE DISCUSSING NEWS
Encrypted Message: II ASY NUWIUTMKXS TEXM
Decrypted Message: WE ARE DISCUSSING NEWS
Do you want to continue? (y/n)
Y

Enter the key word (In Capital Letters):
ZEBRA

Enter the Message (In Capital Letters):
HI I AM AKRITI UPADHYAY

Original Message: HI I AM AKRITI UPADHYAY
Encrypted Message: GM Z ZQ RKQMUZ TTBUHXEZ
Decrypted Message: HI I AM AKRITI UPADHYAY
Do you want to continue? (y/n)
N
```

Lab 2:

Classical Encryption Algos (cond…)

5. Transpositional Cipher:

```
#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):

```cpp
#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){
```

```
        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:
000111011111010001010010111111100011110001010110110010101111010101111111 0

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):
000111011111010001010010111111100011110001010110110010101111010101111111 0

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
```

Lab 3:

Symmetric Key Cryptosystem Algos

1. A5/1:

```
#include<iostream>
#include<vector>
#include<string>
using namespace std;
#define SIZEX 19
#define SIZEY 22
#define SIZEZ 23


// GETTING CHARACTER FROM DECIMAL VAL (Bin -> Dec)
char getCharacter(string binary){
    char reqChar;
    // Bin -> Dec
    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 = char(dec_val);
    return reqChar;
}



// converting Binary (7 bits) to corresponding ASCII character
string BinToChar(string stream){
    string msg = "";
    int i=0;
    while(i<stream.size()){
        // combining 7 bits at a time to retrieve the corresponding
character from the binary form (0-125)
```

```
        string binOfChar = "";
        for(int j=i; j<i+7; j++){
            binOfChar += stream[j];
        }
        i = i+7;
        msg += getCharacter(binOfChar);
    }
    return msg;
}


// Generate a random 64-bit key
string generatekey(){
    string key = "";
    for(int i=0; i<64; i++)
        key += to_string(rand()%2);
    return key;
}




// Print the registers
void printRegister(vector<int> reg){
    for(int i=0; i<reg.size(); i++)
        cout<<reg[i]<<" ";
    cout<<"\n";
}




// Allocating registers with Key
void allocateRegisters(vector<int> &x, vector<int> &y, vector<int> &z,
string key){
    // x
    for(int i=0; i<SIZEX; i++)
        x[i] = key[i]-'0';
    // y
    for(int i=19; i<SIZEY+19; i++)
        y[i-19] = key[i]-'0';
    // z
    for(int i=41; i<SIZEZ+41; i++)
```

```
        z[i-41] = key[i]-'0';

}




// Majority vote function
int majorityVote(int x, int y, int z){
    int m;
    if(x == 0){
        if(y == 0 || z == 0)
            m = 0;
        else
            m = 1;
    }
    else{
        if(y == 1 || z == 1)
            m = 1;
        else
            m = 0;
    }
    return m;
}




// Shifting registers by 1 position (towards Right)
void shiftRegister(vector<int> &reg, const int t){
    for(int j=reg.size()-1; j>=0; j--){
        if(j == 0)
            reg[j] = t;
        else
            reg[j] = reg[j-1];
    }
}




// Functioning of registers X, Y, Z
```

```
void registerFunctioning(vector<int> &x, vector<int> &y, vector<int>
&z, const int m){
    if(x[8] == m){
        int t = x[13] ^ x[16] ^ x[17] ^ x[18];
        shiftRegister(x, t);
    }
    if(y[10] == m){
        int t = y[20] ^ y[21];
        shiftRegister(y, t);
    }
    if(z[10] == m){
        int t = z[20] ^ z[21] ^ z[22];
        shiftRegister(z, t);
    }
}




// Converting ASCII (decimal number) to 8-bit Binary string
string asciiToBin(int num){
    string bin = "";
    for(int i = 1<<6; i>0; i=i/2){
        if((num & i) != 0)
            bin += to_string(1);
        else
            bin += to_string(0);
    }
    return bin;
}




// Convert Msg string to Binary stream
string generateMsgStream(string msg){
    string stream = "";
    cout<<"\n";
    for(int i=0; i<msg.size(); i++){
        int ascii = (int)msg[i];
        string temp = asciiToBin(ascii);
        stream += temp;
        cout<<msg[i]<<" => "<<ascii<<" => "<<temp<<endl;
```

```
    }
    return stream;
}




// COMPUTATION INVOLVED IN ENCRYPTION/DECRYPTION
string computation(string input_stream, vector<int>x, vector<int>y,
vector<int>z){
    string output_stream = "";
    for(int i=0; i<input_stream.size(); i++){
        int m = majorityVote(x[8], y[10], z[10]);

        // doing operations on Array
        registerFunctioning(x, y, z, m);

        // keyStreamBit
        int s = x[18] ^ y[21] ^ z[22];

        // XOR of keystream bit and cipher character bit
        output_stream += ((input_stream[i]-'0') ^ s)+'0';
    }
    return output_stream;
}




// ENCRYPTION
string encryption(const string msg, vector<int> &x, vector<int> &y,
vector<int> &z, const string key){
    cout<<"\nPlain Text: "<<msg<<endl;
    string msgStream = generateMsgStream(msg);
    cout<<"\nPlain Text as Binary stream: \n"<<msgStream<<"\n";

    // allocating registers x, y, z with key
    allocateRegisters(x,y,z,key);
    cout<<"\n\nIntially Registers are (Encryption):\n";
    cout<<"X:\n";
    printRegister(x);
    cout<<"Y:\n";
    printRegister(y);
```

```
    cout<<"Z:\n";
    printRegister(z);

    cout<<"\nEncrypting.......\n";
    // encrypting
    string cipherStream = computation(msgStream,x,y,z);
    cout<<"\nCipher Stream:\n"<<cipherStream<<endl;
    string cipherText = BinToChar(cipherStream);
    return cipherText;
}



// DECRYPTION
string decryption(const string cipherText, vector<int> &x, vector<int>
&y, vector<int> &z, const string key){
    cout<<"\nCipher Text: "<<cipherText<<endl;
    string cipherStream = generateMsgStream(cipherText);
    cout<<"\nCipher Text as Binary stream: "<<cipherStream<<endl;

    // allocating registers x, y, z with key
    allocateRegisters(x,y,z,key);
    cout<<"\n\nIntially Registers are (Decryption):\n";
    cout<<"X:\n";
    printRegister(x);
    cout<<"Y:\n";
    printRegister(y);
    cout<<"Z:\n";
    printRegister(z);


    cout<<"\nDecrypting.......\n";
    // decrypting
    string msgStream = computation(cipherStream,x,y,z);
    cout<<"\nMessage Stream:\n"<<msgStream<<endl;
    string plainText = BinToChar(msgStream);
    return plainText;
}
```

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

        // Linear feedback shift registers (LFSRs)
        vector<int> x(SIZEX), y(SIZEY), z(SIZEZ);

        // Key (64 bit)
        string k;
        k = generatekey();
        cout<<"\nKey (64 Bit): \n"<<k<<endl;

cout<<"_____
_____";

        // Encryption
        cout<<"\nENCRYPTION:\n";
        string encrypt = encryption(msg,x,y,z,k);
        cout<<"\nCipher Text: "<<encrypt<<endl;

cout<<"_____
_____";

        // Decryption
        cout<<"\nDECRYPTION:\n";
        string decrypt = decryption(encrypt,x,y,z,k);
        cout<<"\nPlain Text: "<<decrypt<<endl;

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:
Secret Message!

Key (64 Bit):
1000100011111111111010000010010101010111001000010100101100001101
```

```
ENCRYPTION:

Plain Text: Secret Message!

S => 83 => 1010011
e => 101 => 1100101
c => 99 => 1100011
r => 114 => 1110010
e => 101 => 1100101
t => 116 => 1110100
  => 32 => 0100000
M => 77 => 1001101
e => 101 => 1100101
s => 115 => 1110011
s => 115 => 1110011
a => 97 => 1100001
g => 103 => 1100111
e => 101 => 1100101
! => 33 => 0100001

Plain Text as Binary stream:
10100111100101110001111100101100101111010001000001001101110010111100111110011110000111001111100101010100001


Intially Registers are (Encryption):
X:
1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
Y:
0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 0
Z:
0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 1 1 0 1

Encrypting.......
```

```
Cipher Stream:
00000101111010110101011111110101100001000001001000001111000000010011101100001110101000101111101
11001011000

Cipher Text: ☻zj~X H▲Θ↔CT/\X
_____________________________________________________________________________________________
DECRYPTION:

Cipher Text: ☻zj~X H▲Θ↔CT/\X

☻ => 2 => 0000010
z => 122 => 1111010
j => 106 => 1101010
~ => 126 => 1111110
X => 88 => 1011000
  => 32 => 0100000
H => 72 => 1001000
▲ => 30 => 0011110
Θ => 1 => 0000001
↔ => 29 => 0011101
C => 67 => 1000011
T => 84 => 1010100
/ => 47 => 0101111
\ => 92 => 1011100
X => 88 => 1011000

Cipher Text as Binary stream: 00000101111010110101011111110101100001000001001000001111000000010
01110110000111010100010111111011001011000


Intially Registers are (Decryption):
X:
1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1
Y:
0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 0 1 1 1 0
Z:
0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 0 0 0 0 1 1 0 1

Decrypting.......
```

```
Message Stream:
10100111100101110001111100101100101111010001000001001101110010111100111110011111000011100111110
01010100001

Plain Text: Secret Message!
_____________________________________________________________________________________________

Original Message: Secret Message!
Encrypted Message: ☻zj~X H▲Θ↔CT/\X
Decrypted Message: Secret Message!
_____________________________________________________________________________________________
Do you want to continue? (y/n)
n
```

---

2. RC4:

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

```
void printVector(vector<int> v){
    for(int i=0; i<v.size(); i++)
        cout<<v[i]<<" ";
    cout<<endl;
}

int main(){
    while(1){
        // Input Plain Text
        string msg;
        cout<<"\nEnter Plain Text: \n";
        cin.ignore();
        getline(cin, msg);

        int ptSize = msg.size();
        vector<int> plainTextArray(ptSize);
        for(int i=0; i<ptSize; i++){
            plainTextArray[i] = msg[i];
        }

        // Input Key
        string key;
        cout<<"\nEnter Key: \n";
        cin>>key;
        int keySize = key.size();
        // size(keyArray) = size(plainTextArray)
        vector<int> keyArray(ptSize);
        for(int i=0; i<ptSize; i++){
            keyArray[i] = key[i%keySize];
        }




cout<<"_____
_____";
        cout<<"\nBEHIND THE SCENES:\n";


        cout<<"\nPlain Text Array:\n";
```

```
        printVector(plainTextArray);
        cout<<"\nKey Array:\n";
        printVector(keyArray);


        // CREATING THE TABLE
        // S[] - state vector
        // T[] - key vector (Temporary array)
        // size(S) = size(T)
        // Assume size of S (let it be double of KeyArray)
        vector<int> S(keyArray.size()*2);
        vector<int> T(S.size());
        // Intialising S array
        for(int i=0; i<S.size(); i++){
            S[i] = i;
        }
        // Initialising T array with key
        for(int i=0; i<T.size(); i++){
            T[i] = keyArray[i % keyArray.size()];
        }
        cout<<"\nStateVector(S):\n";
        printVector(S);
        cout<<"\nTempArray(T):\n";
        printVector(T);



        // Step 1: KEY SCHEDULING
        // No. of Iterations = size of state(S) array
        for(int i=0, j=0; i<S.size(); i++){
            j = (j + S[i] + T[i]) % S.size();
            swap(S[i], S[j]);
        }
        cout<<"\nS after step 1 (key scheduling):\n";
        printVector(S);



        // Step 2: STREAM GENERATION
        // No. of Iterations = size of key
        vector<int> newKeyArray(keyArray.size());
        // New key - used for encryption and decryption
        int i=0, j=0;
```

```
        while(i<keyArray.size()){
            i = (i + 1) % S.size();
            j = (j + S[i]) % S.size();
            swap(S[i], S[j]);
            int t = (S[i]+S[j]) % S.size();
            newKeyArray[i] = S[t];
            i++;
        }
        cout<<"\nS after step 2 (stream generation):\n";
        printVector(S);
        cout<<"\nnewKeyArray:\n";
        printVector(newKeyArray);



        // Step 3: Encryption
        // PT XOR NewKey = CT

cout<<"_____________________________________________________________________
_____________________";
        cout<<"\nEncrypting......\n";
        vector<int> cipherText(plainTextArray.size());
        for(int i=0; i<plainTextArray.size(); i++){
            cipherText[i] = plainTextArray[i] ^ newKeyArray[i];
        }
        cout<<"\nCipher Text Array:\n";
        printVector(cipherText);
        string cipher = "";
        for(int i=0; i<cipherText.size(); i++){
            cipher += char(cipherText[i]);
        }
        cout<<"\nCipher Text: "<<cipher<<endl;



        // Step 3: Decryption
        // CT XOR NewKey = PT

cout<<"_____________________________________________________________________
_____________________";
        cout<<"\nDecrypting......\n";
        vector<int> decryptedPlainTextArray(cipherText.size());
        for(int i=0; i<cipherText.size(); i++){
```

```
            decryptedPlainTextArray[i] = cipherText[i] ^
newKeyArray[i];
        }
        cout<<"\nPlain text Array (After Decryption):\n";
        printVector(decryptedPlainTextArray);
        string decrypt = "";
        for(int i=0; i<decryptedPlainTextArray.size(); i++){
            decrypt += char(decryptedPlainTextArray[i]);
        }
        cout<<"\nDecrypted Plain Text: "<<decrypt;


cout<<"\n_____
_____";
        cout<<"\n\nOriginal Message: "<<msg;
        cout<<"\nEncrypted Message: "<<cipher;
        cout<<"\nDecrypted Message: "<<decrypt;

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

```
Enter Plain Text:
Secret Message!

Enter Key:
grape
_____________________________________________________________________________________
BEHIND THE SCENES:

Plain Text Array:
83 101 99 114 101 116 32 77 101 115 115 97 103 101 33

Key Array:
103 114 97 112 101 103 114 97 112 101 103 114 97 112 101

StateVector(S):
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

TempArray(T):
103 114 97 112 101 103 114 97 112 101 103 114 97 112 101 103 114 97 112 101 103 114 97 112 101 103 114 97 112 101

S after step 1 (key scheduling):
13 8 4 12 27 23 5 29 22 3 7 15 2 19 18 17 20 21 25 0 1 9 28 10 11 14 16 6 26 24

S after step 2 (stream generation):
15 22 4 1 27 19 5 2 8 17 7 13 29 10 18 16 20 21 25 0 12 9 28 23 11 14 3 6 26 24

newKeyArray:
0 13 0 19 0 2 0 22 0 12 0 26 0 1 0
```

```
_____________________________________________________________________________________
Encrypting......

Cipher Text Array:
83 104 99 97 101 118 32 91 101 127 115 123 103 100 33

Cipher Text: Shcaev [es{gd!
_____________________________________________________________________________________
Decrypting......

Plain text Array (After Decryption):
83 101 99 114 101 116 32 77 101 115 115 97 103 101 33

Decrypted Plain Text: Secret Message!
_____________________________________________________________________________________

Original Message: Secret Message!
Encrypted Message: Shcaev [es{gd!
Decrypted Message: Secret Message!

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

Lab 4:
Asymmetric Key Cryptosystem Algos

1. RSA:

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

int n, z, e, d;
// Vector to store ASCII values of characters from A-Z
vector<int> P;



// GCD(a,b)
int gcd(int a, int b)
{
    return b == 0 ? a : gcd(b, a % b);
}

// Print the registers
void printRegister(vector<int> reg){
    for(int i=0; i<reg.size(); i++)
        cout<<reg[i]<<" ";
    cout<<"\n";
}


// ENCRYPTION
string encryption(string plain, int p, int q)
{
    // STEP 2:
    n = p*q;

    // STEP 3:
    z = (p-1)*(q-1);

    // STEP 4:
    // choose (e) relatively prime to z (ie, GCD(e, z) = 1)
    for(int i=2;i<z;i++){
        if(gcd(z,i) == 1)
```

```
        {
            e = i;
            break;
        }
    }

    // STEP 5:
    // find d (multiplicative inverse of e % z), ie, (e*d)%z = 1
    for(int i=2;;i++){
        if((e*i)%z == 1)
        {
            d = i;
            break;
        }
    }

    // output values
    cout<<"\ne = "<<e<<",    "<<"n = "<<n<<",    "<<"d = "<<d<<endl;


    // Encrypting
    // Public key: {n, e}
    // C = P^e % n
    string result, res, cipher;

    // converting text form of plain text chars to ASCII equivalent
(converting range 65-90 to 0-25)
    for(int i=0; i<plain.length(); i++)
    {
        // if char is b/w A-Z
        if((int)plain[i]>64 && (int)plain[i]<91)
            // bringing value in range 0-25
            res.push_back(plain[i]-65);
        // keep space as it is
        if(plain[i]==' ')
            res.push_back(' ');
    }


    // encryption process => P^e % n
    for(int i=0; i<res.length(); i++)
    {
        long long z = 1; // P^e
```

```
        if(res[i] == ' ')
        {
            result.push_back(' ');
            P.push_back((int)res[i]);
        }
        else
        {
            // P^e
            for(int j=0; j<e; j++)
                z = ((int)res[i]) * z;
            // store in P to differentiate b/w numbers like 7, 33,
where 7%26 = 33%26 = 7 (storing 1 in P, if z%n > 26 | storing 0, if z%n
< 26)
            P.push_back((z % n)/26);
            result.push_back((z % n)%26);
        }
    }


    // changing ASCII values to alphabetic characters (converting range
0-25 to 65-90)
    for(int i=0; i<result.length() ;i++)
    {
        if((int)result[i]>=0 && (int)result[i]<26)
            cipher.push_back(result[i]+65);
        if(result[i]==' ')
            cipher.push_back(' ');
    }
    return cipher;
}



// DECRYPTION
string decryption(string cipher)
{
    // Decryption
    // Private key: {n, d}
    // P = C^d % n

    string result, res, final;
```

```
    // converting text form of cipher text chars to ASCII equivalent
(converting range 65-90 to 0-25)
    for(int i=0; i<cipher.length(); i++)
    {
        if(cipher[i]>64 && cipher[i]<91)
            // add P[i]*26 if > 26
            res.push_back((cipher[i]-65) + (P[i]*26));
        if(cipher[i] == ' ')
            res.push_back(' ');
    }

    // decryption process => C^d % n
    for(int i=0; i<res.length(); i++)
    {
        long long z = 1; // C^d
        if(res[i] == ' ')
            result.push_back(' ');
        else
        {
            // C^d
            for(int j=0; j<d; j++)
                z = ((int)res[i]) * z;
            result.push_back(z % n);
        }
    }

    // changing ASCII values to alphabetic characters (converting range
0-25 to 65-90)
    for(int i=0; i<result.length(); i++)
    {
        if((int)result[i]>=0 && (int)result[i]<26)
            final.push_back(result[i] + 65);
        if(result[i] == ' ')
            final.push_back(' ');
    }

    return final;
}



int main()
{
```

```
    // Taking plain text from user
    string plain;
    cout<<"\nEnter Plain Text: "<<endl;
    getline(cin,plain);


    // STEP 1:
    // take large prime numbers: p, q
    int p;
    int q;
    cout<<"\nEnter two prime numbers p and q (such that, p*q < 40):
\n";
    cin>>p>>q;

    // Encryption
    string cipher = encryption(plain,p,q);
    cout<<"\nCipher Text: "<<cipher<<endl;

    // Decryption
    string decrypt = decryption(cipher);
    cout<<"Decrypted Plain Text: "<<decrypt<<"\n\n";

    return 0;
}
```

```
Enter Plain Text:
AKRITI UPADHYAY

Enter two prime numbers p and q (such that, p*q < 40):
3
11

e = 3,   n = 33,   d = 7

Cipher Text: AKDRCR OJABNEAE
Decrypted Plain Text: AKRITI UPADHYAY
```

---

2. Diffie-Hellman:

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

```
// Array of first 26 prime numbers
// A-Z => 26 alphabets (only Uppercase)
// Pick value of 'g' according to the ASCII value of alphabet character
int
g[26]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83
,89,97,101};


void diffie_helman(string key, int pvtA, int pvtB, int prime)
{
    string a, b, sendToB, sendToA;
    // sendToB = g^x % n
    // b = sendToB ^ y => (g^x % n)^y => shared key of B
    // sendToA = g^y % n
    // a = sendToA ^ x => (g^y % n)^x => shared key of A


    // computing, sendToB = g^x % n
    for(int i=0; i<key.length(); i++)
    {
        long long z = 1;  // z = g^x
        // keep spaces as it is
        if(key[i] == ' ')
            sendToB.push_back(' ');
        else
        {
            // if ascii(char) is in range A-Z
            if(key[i]>64 && key[i]<91){
                // g^x
                for(int j=0; j<pvtA; j++)
                    z = z * g[key[i]-65];
            }
            sendToB.push_back(z % prime);
        }
    }

    // computing b = sendToB ^ y
    for(int i=0; i<sendToB.length(); i++)
    {
        long long z = 1;
        // keep space as it is
        if(sendToB[i] == ' ')
```

```
            b.push_back(' ');
        else
        {
            // sendTOB ^ y
            for(int j=0; j<pvtB; j++)
                z = z * sendToB[i];
            // ascii to alphabet
            b.push_back(((z % prime) % 26) + 65);
        }
    }

    // computing, sendToA = g^y % n
    for(int i=0; i<key.length(); i++)
    {
        long long z = 1;
        if(key[i] == ' ')
            sendToA.push_back(' ');
        else
        {
            if(key[i]>64 && key[i]<91){
                // g^y
                for(int j=0; j<pvtB; j++)
                    z = z * g[key[i]-65];
            }
            sendToA.push_back(z % prime);
        }
    }

    // computing a = sendToA ^ x
    for(int i=0; i<sendToA.length(); i++)
    {
        long long z = 1;
        if(sendToA[i] == ' ')
            a.push_back(' ');
        else
        {

            for(int j=0; j<pvtA; j++)
                z = z*sendToA[i];
            // ascii to alphabet
            a.push_back(((z % prime) % 26) + 65);
        }
    }
```

```
    cout<<"\nFinal shared key of A: "<<a<<endl;
    cout<<"Final shared key of B: "<<b<<endl;
    if(a == b)
        cout<<"Shared keys of A and B are equal\n\n";
    else
        cout<<"Shared keys of A and B are NOT equal\n\n";
}

int main()
{
    // Input the encrypted key
    string key;
    // pvtA - private key of A (x)
    // pvtB - private key of B (y)
    int pvtA, pvtB;
    cout<<"\nEnter the key to be exchanged (UPPERCASE ALPHABETS ONLY):
\n";
    getline(cin,key);

    // n - prime num
    int n;
    // Bcoz max. value we can handle is 10^100 (otherwise overflow in
datatypes)
    cout<<"\nEnter a prime number (less than 100)"<<endl;
    cin>>n;


    cout<<"\nEnter the private keys of A and B, \nNOTE: \n1.
priv_key(A) < priv_key(B)\n2. Both the private keys should be <=
10)\n";
    cout<<"\nEnter private key of A: \n";
    cin>>pvtA;    // x
    cout<<"\nEnter private key of B: \n";
    cin>>pvtB;    // y

    // Key distribution algorith
    diffie_helman(key, pvtA, pvtB, n);

    return 0;
}
```

```
Enter the key to be exchanged (UPPERCASE ALPHABETS ONLY):
SECRET KEY

Enter a prime number (less than 100)
19

Enter the private keys of A and B,
NOTE:
1. priv_key(A) < priv_key(B)
2. Both the private keys should be <= 10)

Enter private key of A:
6

Enter private key of B:
4

Final shared key of A: LBHLBH BBH
Final shared key of B: LBHLBH BBH
Shared keys of A and B are equal
```

X-X-X-X