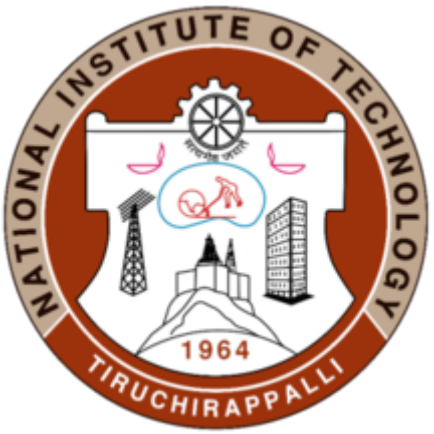
# National Institute of Technology, Tiruchirappalli




## Department of Computer Applications

# Information Security Lab

## *Lab 4*

Submitted to:

Dr. Ghanshyam Bopche

Submitted by:

Akriti Upadhyay

205120007

MCA – 2nd Year

# 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
```

---

**X-X-X-X**