Text

Description automatically generated

**T.Y.B.Tech (CSE)**

Information Security

**Lab Assignment No – A2**

**Name: Aniruddha Shende**

**Roll number: PE04**

**Batch: E1**

**Panel: E**

Table, letter

Description automatically generated

Diagram, schematic

Description automatically generated

A piece of paper with writing

Description automatically generated with low confidence

Diagram, schematic

Description automatically generated

Letter

Description automatically generated

**Program Code :**

// Name : Aniruddha Shende

// Roll no : PE04

// Batch : E1

// Panel : E

#include <iostream>

#include <vector>

#include <string>

#include <iomanip>

using namespace std;

int S0[4][4] = {

{1, 0, 3, 2},

{3, 2, 1, 0},

{0, 2, 1, 3},

{3, 1, 3, 2}};

int S1[4][4] = {

{0, 1, 2, 3},

{2, 0, 1, 3},

{3, 0, 1, 0},

{2, 1, 0, 3}};

typedef std::vector<int> int\_vec;

#define TWIDTH 25

int P10[10] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

int P8[8] = {6, 3, 7, 4, 8, 5, 10, 9};

int P4[4] = {2, 4, 3, 1};

int IP[8] = {2, 6, 3, 1, 4, 8, 5, 7};

int INVERSE\_IP[8] = {4, 1, 3, 5, 7, 2, 8, 6};

int EP[8] = {4, 1, 2, 3, 2, 3, 4, 1};

string vector\_to\_string(int\_vec v)

{

string s = "";

for (int i = 0; i < v.size(); i++)

{

s += to\_string(v[i]);

s += " ";

}

return s;

}

void print\_vector(int\_vec &v)

{

for (int i = 0; i < v.size(); i++)

{

cout << v[i] << " ";

}

cout << endl;

}

void print\_table\_header()

{

cout << "|| " << setw(TWIDTH) << left << "Operation"

<< " || " << setw(TWIDTH) << "O/P"

<< " ||" << endl;

cout << "|| " << setw(TWIDTH) << left << " "

<< " || " << setw(TWIDTH) << " "

<< " ||" << endl;

}

void print\_table\_row(vector<string> row)

{

cout << "->" << setw(TWIDTH) << row[0] << "-> "

<< setw(TWIDTH) << row[1] << "->"

<< endl;

}

class Bits

{

public:

int size;

int\_vec bits, original\_bits;

Bits(int \_size)

{

size = \_size;

bits.resize(size);

original\_bits.resize(8);

}

void set\_bit(int\_vec \_bits)

{

if (\_bits.size() != size)

{

cout << "Error: Size of bits is not equal to size of the object" << endl;

return;

}

bits = \_bits;

}

Bits \*RLshift()

{

int start = size / 2;

int temp = bits[start];

for (int i = start; i < size - 1; i++)

{

bits[i] = bits[i + 1];

}

bits[size - 1] = temp;

return this;

}

Bits \*LLshift()

{

int end = (size / 2) - 1;

int temp = bits[0];

for (int i = 0; i < end; i++)

{

bits[i] = bits[i + 1];

}

bits[end] = temp;

return this;

}

Bits \*Lshift()

{

this

->RLshift()

->LLshift();

print\_table\_row({"LeftShift", vector\_to\_string(bits)});

return this;

}

Bits \*P10()

{

if (size != 10)

{

cout << "Error: P10 on non 10-bit data" << endl;

return this;

}

int\_vec temp(10);

for (int i = 0; i < 10; i++)

{

temp[i] = bits[::P10[i] - 1];

}

bits = temp;

print\_table\_row({"P10", vector\_to\_string(bits)});

return this;

}

Bits \*P8()

{

int\_vec temp(8);

for (int i = 0; i < 8; i++)

{

temp[i] = bits[::P8[i] - 1];

}

bits = temp;

size = 8;

print\_table\_row({"P8", vector\_to\_string(bits)});

return this;

}

Bits \*P4()

{

if (size != 8)

{

cout << "Error: Size of bits is not equal to 8 bit" << endl;

return this;

}

int\_vec temp(8);

for (int i = 0; i < 8; i++)

{

if (i < 4)

temp[i] = bits[i];

else

temp[i] = bits[::P4[i - 4] - 1 + 4];

}

bits = temp;

print\_table\_row({"P4", vector\_to\_string(bits)});

return this;

}

Bits \*R\_xor\_with(Bits \*b)

{

if (size != 12)

{

cout << "Size of bits is not equal to 12 bit" << endl;

return this;

}

int\_vec temp(size);

for (int i = 0; i < 12; i++)

{

if (i < 4)

temp[i] = bits[i];

else

temp[i] = bits[i] == b->bits[i - 4] ? 0 : 1;

}

bits = temp;

print\_table\_row({"R\_xor\_with", vector\_to\_string(bits)});

return this;

}

Bits \*R\_xor\_with\_L()

{

if (size != 8)

{

cout << "Size of bits not equal to 8" << endl;

return this;

}

int\_vec temp(size);

for (int i = 0; i < 8; i++)

{

if (i < 4)

temp[i] = bits[i];

else

temp[i] = bits[i] == bits[i - 4] ? 0 : 1;

}

bits = temp;

for (int i = 0; i < 4; i++)

{

temp[i] = bits[i + 4];

temp[i + 4] = original\_bits[i + 4];

}

bits = temp;

print\_table\_row({"R\_xor\_with\_L", vector\_to\_string(bits)});

return this;

}

Bits \*IP()

{

if (size != 8)

{

cout << "Size of bits is not equal to 8 bit" << endl;

return this;

}

int\_vec temp(8);

for (int i = 0; i < 8; i++)

{

temp[i] = bits[::IP[i] - 1];

}

original\_bits = bits = temp;

print\_table\_row({"IP", vector\_to\_string(bits)});

return this;

}

Bits \*INVERSE\_IP()

{

if (size != 8)

{

cout << "Size of bits is not equal to 8 bit" << endl;

return this;

}

int\_vec temp(8);

for (int i = 0; i < 8; i++)

{

temp[i] = bits[::INVERSE\_IP[i] - 1];

}

bits = temp;

print\_table\_row({"INVERSE\_IP", vector\_to\_string(bits)});

return this;

}

Bits \*EP()

{

int\_vec temp(12);

for (int i = 0; i < 12; i++)

{

if (i < 4)

temp[i] = bits[i];

else

temp[i] = bits[::EP[i - 4] - 1 + 4];

}

size = 12;

bits = temp;

print\_table\_row({"EP", vector\_to\_string(bits)});

return this;

}

Bits \*shrink()

{

if (size != 12)

{

cout << "Size of bits is not equal to 12 bit" << endl;

return this;

}

int\_vec temp(8);

for (int i = 0; i < 4; i++)

{

temp[i] = bits[i];

}

// for s0

int a = bits[4], b = bits[7], c = bits[5], d = bits[6]; // ab = row , cd = col

int row = a \* 2 + b, col = c \* 2 + d;

int corresponding\_value = S0[row][col];

temp[4] = corresponding\_value / 2;

temp[5] = corresponding\_value % 2;

// for s1

a = bits[8], b = bits[11], c = bits[9], d = bits[10]; // ab = row , cd = col

row = a \* 2 + b;

col = c \* 2 + d;

corresponding\_value = S1[row][col];

temp[6] = corresponding\_value / 2;

temp[7] = corresponding\_value % 2;

size = 8;

bits = temp;

print\_table\_row({"Shrink", vector\_to\_string(bits)});

return this;

}

Bits \*swap()

{

int\_vec temp = bits;

for (int i = 0; i < 4; i++)

{

bits[i] = temp[4 + i];

bits[4 + i] = temp[i];

}

original\_bits = bits;

print\_table\_row({"Swap", vector\_to\_string(bits)});

return this;

}

Bits \*complex(Bits \*key)

{

return this->EP()->R\_xor\_with(key)->shrink()->P4()->R\_xor\_with\_L();

}

void print()

{

print\_vector(bits);

}

};

class SDES

{

public:

Bits \*plaintext;

Bits \*key, \*key1, \*key2;

Bits \*ciphertext;

SDES()

{

plaintext = new Bits(8);

key = new Bits(10);

key1 = new Bits(10);

key2 = new Bits(10);

ciphertext = new Bits(8);

}

void set\_key(int\_vec \_key)

{

key->set\_bit(\_key);

key1->set\_bit(\_key);

}

void set\_plaintext(int\_vec \_plaintext)

{

plaintext->set\_bit(\_plaintext);

}

void encrypt()

{

cout << endl

<< "KEY 1:" << endl;

print\_table\_header();

Bits \*temp\_plaintext = new Bits(8);

temp\_plaintext->set\_bit(plaintext->bits);

// generate key1 and key2

key1->P10()

->Lshift();

key2->set\_bit(key1->bits); // copy key1 to key2

key1->P8();

cout << endl

<< "KEY 2:" << endl;

print\_table\_header();

key2

->Lshift()

->Lshift()

->P8();

// now start encryption

cout << endl

<< "PLain Text:" << endl;

print\_table\_header();

ciphertext = temp\_plaintext->IP()->complex(key1)->swap()->complex(key2)->INVERSE\_IP();

}

void decrypt()

{

cout << endl

<< "KEY 1:" << endl;

print\_table\_header();

Bits \*temp\_plaintext = new Bits(8);

temp\_plaintext->set\_bit(plaintext->bits);

key1->P10()

->Lshift();

key2->set\_bit(key1->bits);

key1->P8();

cout << endl

<< "KEY 2:" << endl;

print\_table\_header();

key2

->Lshift()

->Lshift()

->P8();

cout << endl

<< "PLain Text:" << endl;

print\_table\_header();

ciphertext = temp\_plaintext->IP()->complex(key2)->swap()->complex(key1)->INVERSE\_IP();

}

void display(int is\_decrypt = 0)

{

cout << endl

<< (is\_decrypt ? "Ciphertext: " : "Plaintext: ");

plaintext->print();

cout << "Key: ";

key->print();

cout << "Sub Key1: ";

key1->print();

cout << "Sub Key2: ";

key2->print();

cout << (is\_decrypt ? "Plaintext: " : "Ciphertext: ");

ciphertext->print();

cout << endl;

}

};

int main()

{

int choice = 0;

int temp\_int = 0;

int\_vec temp;

SDES encrypt\_object, decrypt\_object;

do

{

cout << "\n"

<< "1. Enter value of Key"

<< "\n"

<< "2. SDES Encrypt"

<< "\n"

<< "3. SDES Decrypt"

<< "\n"

<< "4. End program"

<< "\n"

<< "Enter choice: " << endl;

cin >> choice;

cout << endl;

switch (choice)

{

case 1:

cout << "Enter key: ";

temp.clear();

for (int i = 0; i < 10; i++)

{

cin >> temp\_int;

temp.push\_back(temp\_int);

}

encrypt\_object.set\_key(temp);

decrypt\_object.set\_key(temp);

break;

case 2:

cout << "Enter plaintext: ";

temp.clear();

for (int i = 0; i < 8; i++)

{

cin >> temp\_int;

temp.push\_back(temp\_int);

}

encrypt\_object.set\_plaintext(temp);

encrypt\_object.encrypt();

encrypt\_object.display();

break;

case 3:

cout << "Enter ciphertext: ";

temp.clear();

for (int i = 0; i < 8; i++)

{

cin >> temp\_int;

temp.push\_back(temp\_int);

}

decrypt\_object.set\_plaintext(temp);

decrypt\_object.decrypt();

decrypt\_object.display(1);

break;

case 4:

break;

default:

cout << endl

<< "Invalid Choice!";

}

cout << endl;

} while (choice != 4);

cout << endl;

return 0;

}

**Output :**

Text

Description automatically generated

Text

Description automatically generated

A picture containing text

Description automatically generated

Text

Description automatically generated