**Lab 2**

**Lab 2.1: Write a program to implement the DES key generation process to generate subkeys. Also, show the subkeys generated at each round.**

**Algorithm:**

DES key generation process to generate subkeys:

1. Input:
   1. Accept a 64-bit key as input.
2. Initial permutation (PC1):
   1. Apply the PC1 (Permuted Choice 1) table to permute the 64-bit key, rearranging its bits to create a 56-bit key.
3. Key splitting:
   1. Divide the 56-bit key into two halves of 28 bits each, C0 and D0.
4. Round key generation loop:
5. Repeat the following steps for 16 rounds:
   1. Left shift:
      1. Shift C0 and D0 left by 1 or 2 bits according to the round number (specified in the DES specification).
   2. Permutation (PC2):
      1. Combine C0 and D0 back into a 56-bit key.
      2. Apply the PC2 (Permuted Choice 2) table to select 48 bits from the 56-bit key, forming the subkey for this round.
   3. Output:
      1. Display or store the generated 48-bit subkey.
6. Output:
   1. After 16 rounds, you'll have 16 distinct 48-bit subkeys.

**Source Code:**

#include<iostream>

#include<string>

#include<bitset>

using namespace std;

string round\_keys[16];

// circular left shift by one

string C\_L\_Shift\_Once(string key\_chunk)

{

string shifted = "";

for (int i = 1; i < 28; i++)

{

shifted += key\_chunk[i];

}

shifted += key\_chunk[0];

return shifted;

}

// circular left shift by two

string C\_L\_Shift\_Twice(string key\_chunk)

{

string shifted = "";

for (int i = 0; i < 2; i++)

{

for (int j = 1; j < 28; j++)

{

shifted += key\_chunk[j];

}

shifted += key\_chunk[0];

key\_chunk = shifted;

shifted = "";

}

return key\_chunk;

}

void key\_generate(string key)

{

// initial permutation table to convert the key in 56bits

int ip[56] = {

57,49,41,33,25,17,9,

1,58,50,42,34,26,18,

10,2,59,51,43,35,27,

19,11,3,60,52,44,36,

63,55,47,39,31,23,15,

7,62,54,46,38,30,22,

14,6,61,53,45,37,29,

21,13,5,28,20,12,4

};

// compression permutation table to compress the key in 48bits

int cp[48] = {

14,17,11,24,1,5,

3,28,15,6,21,10,

23,19,12,4,26,8,

16,7,27,20,13,2,

41,52,31,37,47,55,

30,40,51,45,33,48,

44,49,39,56,34,53,

46,42,50,36,29,32

};

// compressing the Key to 56 bit using compression permutation table

string perm\_key ="";

for(int i = 0; i < 56; i++)

{

perm\_key+= key[ip[i]-1];

}

// dividing the the 56 key into two part

string left = perm\_key.substr(0, 28);

string right = perm\_key.substr(28, 56);

// generating 16 round key

for (int i = 0; i < 16; i++)

{

// one left circular for 1, 2, 9, 16

if (i == 0 || i == 1|| i == 8 || i == 15)

{

left = C\_L\_Shift\_Once(left);

right = C\_L\_Shift\_Once(right);

}

else

{

left = C\_L\_Shift\_Twice(left);

right = C\_L\_Shift\_Twice(right);

}

// key chunks are combined

string combined\_key = left + right;

string round\_key = "";

for (int i = 0; i < 48; i++)

{

round\_key += combined\_key[cp[i]-1];

}

round\_keys[i] = round\_key;

cout << "Key "<< i+1 << ":" << round\_keys[i] << endl;

}

}

string TextToBinaryString(string words)

{

string binaryString = "";

for (char& \_char : words) {

binaryString +=bitset<8>(\_char).to\_string();

}

return binaryString;

}

int main()

{

string key, Plain\_Text, key\_bin;

cout << "Enter the key to encrypt" << endl;

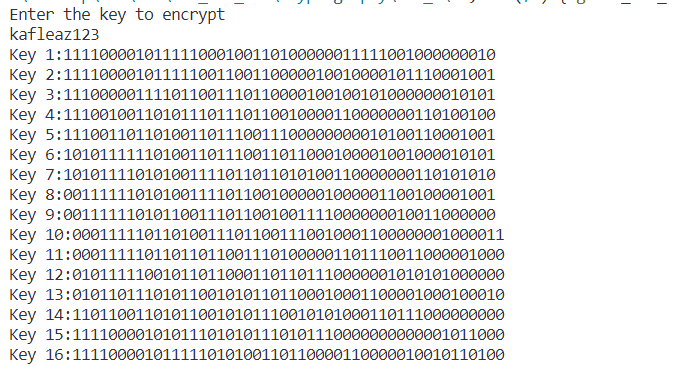
cin >> key;

key\_bin = TextToBinaryString(key).substr(0, 64);

key\_generate(key\_bin);

}

**Output:**



**Lab 2.2** **Implement the initial permutation of the DES algorithm by applying the initial permutation to a given plaintext block and observe the result.:**

**Algorithm:**

1. Define the IP table:
   1. Create a list or array specifying the new position for each bit in the permuted output.
   2. The IP table for DES is fixed and can be found in cryptography references.
2. Input plaintext block:
   1. Obtain the 64-bit plaintext block to be permuted.
3. Apply IP:
   1. Iterate through the 64 bits of the plaintext block:
      1. For each bit, look up its new position in the IP table.
      2. Place the bit at that new position in the output block.
4. Observe the permuted block:
   1. The resulting 64-bit block is the output of the initial permutation.

**Source Code:**

#include<iostream>

#include<string>

#include<bitset>

using namespace std;

void initial\_permutation(string Plain\_Text)

{

//Assuming each character is 1 bit instead of 1 byte

int initial\_perm[64] = {

58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7

};

string Encrypt;

for(int i=0;i<64;i++){

Encrypt += Plain\_Text[initial\_perm[i]];

}

cout << Encrypt;

cout<<"\n";

}

string TextToBinaryString(string words)

{

string binaryString = "";

for (char& \_char : words) {

binaryString +=bitset<8>(\_char).to\_string();

}

return binaryString;

}

int main()

{

string Plain\_Text, Plain\_Text\_bin;

cout << "Enter the Plain text to encrypt" << endl;

cin >> Plain\_Text;

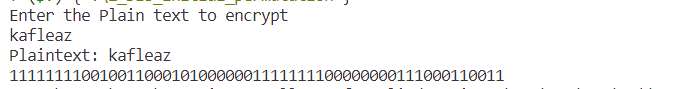
cout<<"Plaintext: " << Plain\_Text <<endl;

Plain\_Text\_bin = TextToBinaryString(Plain\_Text).substr(0, 64);

initial\_permutation(Plain\_Text\_bin);

}

**Output:**



**Lab 2.3: Write a program to apply the round function to a given 32-bit data and subkey, and display the intermediate results.**

**Algorithm:**

1. Expand data: Apply the E-box permutation to the 32-bit data, resulting in a 48-bit expanded block.
2. XOR with subkey: Perform bitwise XOR between the expanded data and the 48-bit subkey.
3. Split into blocks: Divide the 48-bit output into 8 blocks of 6 bits each.
4. Substitute blocks: Apply S-boxes:
   1. For each 6-bit block:
      1. Use the corresponding S-box (S1 to S8) to replace the block with its 4-bit output.
      2. Access the S-box using the first and last bits of the 6-bit block as row and column indices.
5. Permute substituted blocks: Apply the P-box permutation to rearrange the 32 bits resulting from the S-box substitutions.
6. Return output: The final 32-bit output after the P-box is the round function's output (Right Half).

**Source Code:**

#include<iostream>

#include<string>

#include<bitset>

#include<cmath>

using namespace std;

string convertDecimalToBinary(int decimal)

{

string binary;

while (decimal != 0)

{

if (decimal % 2 == 0) {

binary = "0" + binary;

}

else

{

binary = "1" + binary;

}

decimal = decimal / 2;

}

while(binary.length() < 4){

binary = "0" + binary;

}

return binary;

}

int convertBinaryToDecimal(string binary)

{

int decimal = 0;

int counter = 0;

int size = binary.length();

for(int i = size-1; i >= 0; i--)

{

if(binary[i] == '1'){

decimal += pow(2, counter);

}

counter++;

}

return decimal;

}

string Xor(string a, string b){

string result = "";

int size = b.size();

for(int i = 0; i < size; i++){

if(a[i] != b[i]){

result += "1";

}

else{

result += "0";

}

}

return result;

}

void round\_function(string Right\_Plain\_text, const string Round\_key) {

int expansion\_table[48] = {

32,1,2,3,4,5,4,5,

6,7,8,9,8,9,10,11,

12,13,12,13,14,15,16,17,

16,17,18,19,20,21,20,21,

22,23,24,25,24,25,26,27,

28,29,28,29,30,31,32,1

};

int substition\_boxes[8][4][16]=

{{

14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,

0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,

4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,

15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13

},

{

15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,

3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,

0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,

13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9

},

{

10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,

13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,

13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,

1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12

},

{

7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,

13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,

10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,

3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14

},

{

2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,

14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,

4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,

11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3

},

{

12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,

10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,

9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,

4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13

},

{

4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,

13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,

1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,

6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12

},

{

13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,

1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,

7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,

2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11

}};

// The permutation table

int permutation\_tab[32] = {

16,7,20,21,29,12,28,17,

1,15,23,26,5,18,31,10,

2,8,24,14,32,27,3,9,

19,13,30,6,22,11,4,25

};

// Apply the expansion permutation.The right half of the plain text is expanded

string right\_expanded="";

for(int i = 0; i < 48; i++) {

right\_expanded += Right\_Plain\_text[expansion\_table[i]];

}

// XOR with the round key.

string xored = Xor(Round\_key, right\_expanded);

string res = "";

// Apply the S-boxes.

string S\_box\_outputs(32, '\0');

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

string row1= xored.substr(i\*6,1) + xored.substr(i\*6 + 5,1);

int row = convertBinaryToDecimal(row1);

string col1 = xored.substr(i\*6 + 1,1) + xored.substr(i\*6 + 2,1) + xored.substr(i\*6 + 3,1) + xored.substr(i\*6 + 4,1);;

int col = convertBinaryToDecimal(col1);

int val = substition\_boxes[i][row][col];

res += convertDecimalToBinary(val);

}

// Apply the P-box permutation.

string perm2 ="";

for(int i = 0; i < 32; i++){

perm2 += res[permutation\_tab[i]-1];

}

cout << perm2 <<endl;

}

string TextToBinaryString(string words)

{

string binaryString = "";

for (char& \_char : words) {

binaryString +=bitset<8>(\_char).to\_string();

}

return binaryString;

}

int main()

{

string key, Plain\_Text, key\_bin, binary\_plaintext, ciphertext;

cout << "Enter the Plain text to encrypt" << endl;

cin >> Plain\_Text;

binary\_plaintext = TextToBinaryString(Plain\_Text).substr(0, 64);

string left\_half = binary\_plaintext.substr(0, 32);

string right\_half = binary\_plaintext.substr(32, 32);

for (int i = 0; i < 16; i++)

{

// Get the round key.

string round\_key = "";

for (int j = 0; j < 48; j++)

{

round\_key += binary\_plaintext[48 \* i + j];

}

//cout << "Round " << i + 1 << " Key: ";

// Apply the round function.

round\_function(right\_half, round\_key);

// Swap the left and right halves.

string temp = left\_half;

left\_half = right\_half;

right\_half = temp;

}

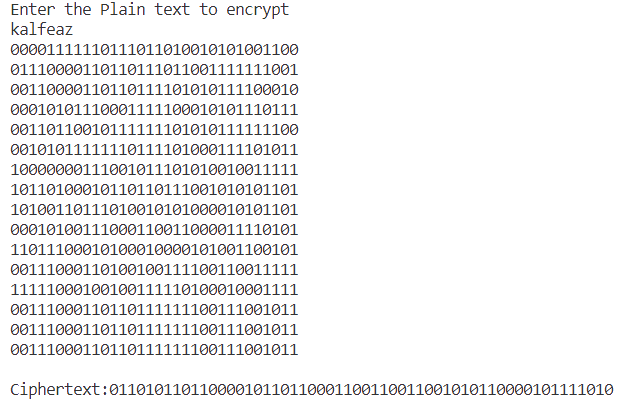
ciphertext = left\_half + right\_half;

cout<<"\n";

cout <<"Ciphertext:" <<ciphertext << endl;

}

**Output:**



**Lab 2.4: Implement the IDEA key scheduling algorithm to generate subkeys from the main encryption key.**

**Algorithm:**

1. Input:
   1. The 128-bit main encryption key (K)
2. Initialization:
   1. Divide K into eight 16-bit subkeys: K1, K2, ..., K8
   2. Create a 128-bit key register (KR) and initialize it with K
3. 3. Key Scheduling Loop:
   1. Repeat the following steps eight times:
      1. Rotate: Rotate the 16 bits in KR left by 25 bits (equivalent to a left circular shift by 9 bits)
4. Split:
   1. Divide KR into eight 16-bit subkeys: KR1, KR2, ..., KR8
5. Output Subkeys:
   1. Use KR1, KR2, ..., KR8 as the subkeys for the next eight rounds of IDEA encryption
6. Output:
   1. The 52 subkeys generated from the key scheduling algorithm

**Source Code:**

#include<iostream>

#include<string>

#include<bitset>

using namespace std;

const int KEY\_SIZE = 128;

const int ROUNDS = 8;

// IDEA round keys as strings

string subkeys[ROUNDS][6];

// Function to rotate a string left by n bits

string rotateLeft(const string& str, int n) {

return str.substr(n) + str.substr(0, n);

}

// IDEA key scheduling function

void IDEA\_key\_schedule(string main\_key) {

for (int round = 0; round < ROUNDS; ++round) {

for (int i = 0; i < 6; ++i) {

subkeys[round][i] = main\_key.substr(0, 16);

main\_key = rotateLeft(main\_key, 25);

}

}

}

string TextToBinaryString(string words)

{

string binaryString = "";

for (char& \_char : words) {

binaryString +=bitset<8>(\_char).to\_string();

}

return binaryString;

}

int main() {

string main\_key, key\_bin;

cout << "Enter the key to encrypt" << endl;

cin >> main\_key;

key\_bin = TextToBinaryString(main\_key).substr(0, 128);

IDEA\_key\_schedule(key\_bin);

// Output the generated subkeys

cout << "Generated Subkeys:" << endl;

for (int round = 0; round < ROUNDS; ++round) {

cout << "Round " << round + 1 << ": ";

for (int i = 0; i < 6; ++i) {

cout << subkeys[round][i] << " ";

}

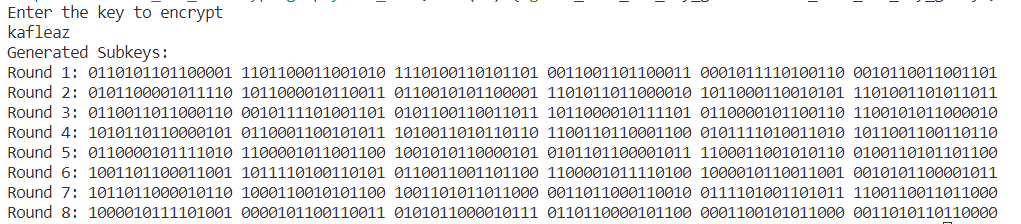
cout << endl;

}

return 0;

}

**Output:**



**Lab 2.5: Write a program to implement the AES SubBytes and ShiftRows operations for encryption. Apply these operations to a given state matrix and show the results.**

**Algorithm:**

SubBytes:

1. Input: A 4x4 state matrix representing the current state of the data.
2. Process each byte:
   1. Iterate through each byte in the state matrix.
   2. Use the S-box (a fixed 16x16 lookup table) to substitute the current byte with the corresponding value in the S-box.

ShiftRows:

1. Shift rows cyclically:
   1. Row 1: Keep it unchanged.
   2. Row 2: Shift all bytes one position to the left.
   3. Row 3: Shift all bytes two positions to the left.
   4. Row 4: Shift all bytes three positions to the left.

**Source Code:**

#include<iostream>

#include<vector>

#include<string>

#include<bitset>

using namespace std;

// S-box for SubBytes operation (hexadecimal representation)

const unsigned char sBox[256] = {

// S-box values (in hexadecimal)

0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,

0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,

0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,

0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,

0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,

0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,

0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,

0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,

0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,

0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,

0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,

0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,

0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,

0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,

0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,

0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16

};

// Shift positions for ShiftRows operation

const int shiftPositions[4][4] = {

{0, 1, 2, 3},

{1, 2, 3, 0},

{2, 3, 0, 1},

{3, 0, 1, 2}

};

// Function to apply SubBytes operation

void SubBytes(vector<vector<uint8\_t>>& state) {

for (int row = 0; row < 4; row++) {

for (int col = 0; col < 4; col++) {

state[row][col] = sBox[state[row][col]];

}

}

}

// Function to apply ShiftRows operation

void ShiftRows(vector<vector<uint8\_t>>& state) {

vector<vector<uint8\_t>> tempState = state;

for (int row = 0; row < 4; row++) {

for (int col = 0; col < 4; col++) {

state[row][col] = tempState[row][shiftPositions[row][col]];

}

}

}

// Function to convert a 128-bit hexadecimal string to a 4x4 state matrix

vector<vector<uint8\_t>> HexStringToStateMatrix(const string& hexString) {

vector<vector<uint8\_t>> state(4, vector<uint8\_t>(4, 0));

if (hexString.length() < 32) {

cout << "Input must be a 128-bit hexadecimal string." << endl;

return state;

}

for (int i = 0; i < 32; i += 2) {

string byteStr = hexString.substr(i, 2);

state[i / 8][i % 8 / 2] = stoul(byteStr, nullptr, 16);

}

return state;

}

// Function to display the state matrix

void DisplayState(const vector<vector<uint8\_t>>& state) {

for (int row = 0; row < 4; row++) {

for (int col = 0; col < 4; col++) {

cout << hex << static\_cast<int>(state[row][col]) << " ";

}

cout << endl;

}

}

string BinaryToHex(string binaryStr) {

string hexStr = "";

int numBits = binaryStr.size();

// Ensure the binary string length is a multiple of 4 (for easy conversion to hex)

int padding = (4 - numBits % 4) % 4;

string paddedBinaryStr = string(padding, '0') + binaryStr;

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

string nibble = paddedBinaryStr.substr(i, 4);

int decimalValue = stoi(nibble, nullptr, 2);

hexStr += "0123456789ABCDEF"[decimalValue];

}

return hexStr;

}

string TextToBinaryString(string words)

{

string binaryString = "";

for (char& \_char : words) {

binaryString +=bitset<8>(\_char).to\_string();

}

return binaryString;

}

int main() {

string key, Plain\_Text, key\_bin, binary\_plaintext, ciphertext, hex\_plainText;

cout << "Enter the Plain text to encrypt (text must be of 128-bit)" << endl;

cin >> Plain\_Text;

binary\_plaintext = TextToBinaryString(Plain\_Text).substr(0, 128);

hex\_plainText = BinaryToHex(binary\_plaintext);

vector<vector<uint8\_t>> state = HexStringToStateMatrix(hex\_plainText);

if (state.empty()) {

return 1;

}

cout << "Original State:" << endl;

DisplayState(state);

cout << "After SubBytes:" << endl;

SubBytes(state);

DisplayState(state);

cout << "After ShiftRows:" << endl;

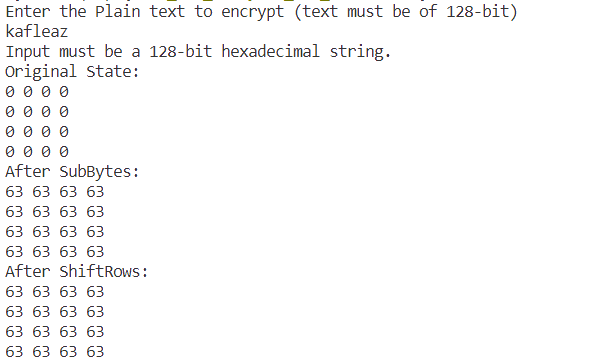
ShiftRows(state);

DisplayState(state);

return 0;

}

**Output:**



**Lab 2.6: Write a program to implement the AES MixColumns operation for encryption. Apply the operation to a given state matrix and round key, and show the results.**

**Algorithm:**

1. Input: A 4x4 state matrix representing the current state of the data.
2. Process each column:
3. For each column in the state matrix:
   1. Treat the column as a 4-term polynomial over the finite field GF(2^8).
   2. Multiply the polynomial by a fixed 4x4 matrix (defined in AES specifications) using special finite field multiplication rules.
   3. Replace the original column with the resulting column.

**Source Code:**

#include <stdio.h>

#include <stdint.h>

// AES MixColumns operation

void mixColumns(uint8\_t state[4][4]) {

uint8\_t tmp[4];

for (int c = 0; c < 4; c++) {

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

tmp[i] = state[i][c];

}

state[0][c] = (uint8\_t)(tmp[0] ^ tmp[1] ^ tmp[2] ^ tmp[3]);

uint8\_t t = tmp[0] ^ tmp[1];

state[0][c] ^= 0x02 \* tmp[0] ^ 0x03 \* t;

state[1][c] = (uint8\_t)(t ^ tmp[2] ^ tmp[3]);

t = tmp[1] ^ tmp[2];

state[1][c] ^= 0x02 \* tmp[1] ^ 0x03 \* t;

state[2][c] = (uint8\_t)(t ^ tmp[0] ^ tmp[3]);

t = tmp[2] ^ tmp[3];

state[2][c] ^= 0x02 \* tmp[2] ^ 0x03 \* t;

state[3][c] = (uint8\_t)(t ^ tmp[1] ^ tmp[0]);

t = tmp[3] ^ tmp[0];

state[3][c] ^= 0x02 \* tmp[3] ^ 0x03 \* t;

}

}

void displayState(uint8\_t state[4][4]) {

printf("State Matrix:\n");

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

for (int j = 0; j < 4; j++) {

printf("%02X ", state[i][j]);

}

printf("\n");

}

int main() {

// Example state matrix (4x4)

uint8\_t state[4][4] = {

{0x32, 0x88, 0x31, 0xe0},

{0x43, 0x5a, 0x31, 0x37},

{0xf6, 0x30, 0x98, 0x07},

{0xa8, 0x8d, 0xa2, 0x34}

};

// Example round key (4x4)

uint8\_t roundKey[4][4] = {

{0x2b, 0x28, 0xab, 0x09},

{0x7e, 0xae, 0xf7, 0xcf},

{0x15, 0xd2, 0x15, 0x4f},

{0x16, 0xa6, 0x88, 0x3c}

};

printf("Original State:\n");

displayState(state);

// Apply MixColumns operation

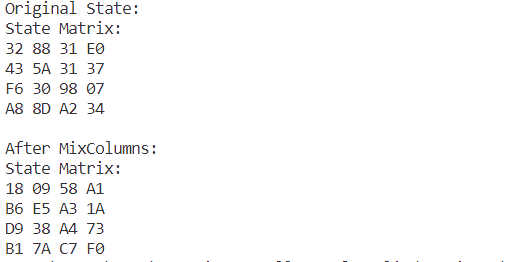
mixColumns(state);

printf("\nAfter MixColumns:\n");

displayState(state);

return 0;

}**Output:**



**Lab 3**

**Lab 3.1: Write a program to implement the Miller-Rabin primality test. Test it with various values of 'n'.**

**Algorithm:**

bool isPrime(int n, int k)

1. Handle base cases for n < 3
2. If n is even, return false.
3. Find an odd number d such that n-1 can be written as d\*2r. Note that since n is odd, (n-1) must be even and r must be greater than 0.
4. Do following k times
   * if (millerTest(n, d) == false)
   * return false
5. Return true.

**Source Code:**

#include <iostream>

#include <stdlib.h>

using namespace std;

long long mulmod(long long, long long, long long);

long long modulo(long long, long long, long long);

bool Miller(long long, int);

int main()

{

// generateHeader("Program to implement Miller Rabin primality test");

do

{

int iteration = 10;

long long num;

cout << "Enter integer to test primality: ";

cin >> num;

if (Miller(num, iteration))

cout << num << " is prime" << endl;

else

cout << num << " is not prime" << endl;

char choice;

cout << "Do you want to continue? (y/n): ";

cin >> choice;

if (choice == 'n' || choice == 'N')

break;

} while (true);

cin.get();

return 0;

}

long long mulmod(long long a, long long b, long long m)

{

long long x = 0,

y = a % m;

while (b > 0)

{

if (b % 2 == 1)

{

x = (x + y) % m;

}

y = (y \* 2) % m;

b /= 2;

}

return x % m;

}

long long modulo(long long base, long long e, long long m)

{

long long x = 1;

long long y = base;

while (e > 0)

{

if (e % 2 == 1)

x = (x \* y) % m;

y = (y \* y) % m;

e = e / 2;

}

return x % m;

}

bool Miller(long long p, int iteration)

{

if (p < 2)

{

return false;

}

if (p != 2 && p % 2 == 0)

{

return false;

}

long long s = p - 1;

while (s % 2 == 0)

{

s /= 2; }

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

{

long long a = rand() % (p - 1) + 1, temp = s;

long long mod = modulo(a, temp, p);

while (temp != p - 1 && mod != 1 && mod != p - 1)

{

mod = mulmod(mod, mod, p);

temp \*= 2;

}

if (mod != p - 1 && temp % 2 == 0)

{

return false;

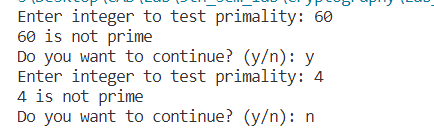
}

}

return true;

}

**Output:**



**Lab 3.2: Calculate φ(n) (Euler's Totient Function) for a given positive integer 'n.' Verify its correctness for multiple values of 'n.**

**Algorithm:**

**Source Code:**

#include <iostream>

using namespace std;

void computeTotient(int);

int main()

{

// generateHeader("Program to compute Totient value.");

int n;

do

{

cout << "Enter a positive integer: ";

cin >> n;

computeTotient(n);

cout << "Do you want to continue? (y/n): ";

char ch;

cin >> ch;

if (ch == 'n' || ch == 'N')

break;

} while (true);

return 0;

}

void computeTotient(int n)

{

long long phi[n + 1];

for (int i = 1; i <= n; i++)

phi[i] = i; // indicates not evaluated yet and initializes for product formula.

for (int p = 2; p <= n; p++)

{

// If phi[p] is not computed already, then number p is prime

if (phi[p] == p)

{

phi[p] = p - 1;

for (int i = 2 \* p; i <= n; i += p)

{

// Add contribution of p to its multiple i by multiplying with (1 - 1/p)

phi[i] = (phi[i] / p) \* (p - 1);

}

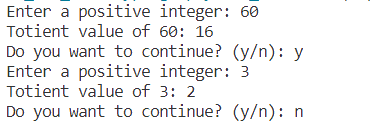
}

}

cout << "Totient value of " << n << ": " << phi[n] << endl;

}

**Output:**



**Lab 3.3: Write a program to apply Fermat's Little Theorem to check if a given number, is a probable prime.**

**Algorithm:**

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#define ll long long

ll modulo(ll base, ll exponent, ll mod) {

ll x = 1;

ll y = base;

while (exponent > 0) {

if (exponent % 2 == 1)

x = (x \* y) % mod;

y = (y \* y) % mod;

exponent = exponent / 2;

}

return x % mod;

}

int Fermat(ll p, int iterations) {

int i;

if (p == 1) {

return 0;

}

for (i = 0; i < iterations; i++) {

ll a = rand() % (p - 1) + 1;

if (modulo(a, p - 1, p) != 1) {

return 0;

}

}

return 1;

}

int main() {

int iteration = 50;

ll num;

printf("Enter integer to test primality: ");

scanf("%lld", &num);

if (Fermat(num, iteration) == 1)

printf("%lld is probably prime ", num);

else

printf("%lld is not prime ", num);

return 0;

}

**Output:**



**Lab 3.4: Write a program to generate a public and private key using RSA algorithm. Also, encrypt a message “CAB College” and again decrypt it using the algorithm.**

**Algorithm:**

**STEP-1**: Select two co-prime numbers as p and q.

**STEP-2**: Compute n as the product of p and q.

**STEP-3**: Compute (p-1)\*(q-1) and store it in z.

**STEP-4**: Select a random prime number e that is less than that of z.

**STEP-5**: Compute the private key, d as e \* mod-1(z).

**STEP-6**: The cipher text is computed as messagee \* mod n.

**STEP-7**: Decryption is done as cipherd mod n.

**Source Code:**

#include <bits/stdc++.h>

using namespace std;

set<int> prime;

int public\_key;

int private\_key;

int n;

void primefiller()

{

vector<bool> seive(250, true);

seive[0] = false;

seive[1] = false;

for (int i = 2; i < 250; i++) {

for (int j = i \* 2; j < 250; j += i) {

seive[j] = false;

}

}

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

if (seive[i])

prime.insert(i);

}

}

int pickrandomprime()

{

int k = rand() % prime.size();

auto it = prime.begin();

while (k--)

it++;

int ret = \*it;

prime.erase(it);

return ret;

}

void setkeys()

{

int prime1 = pickrandomprime();

int prime2 = pickrandomprime();

n = prime1 \* prime2;

int fi = (prime1 - 1) \* (prime2 - 1);

int e = 2;

while (1) {

if (\_\_gcd(e, fi) == 1)

break;

e++;

}

public\_key = e;

int d = 2;

while (1) {

if ((d \* e) % fi == 1)

break;

d++;

}

private\_key = d;

}

long long int encrypt(double message)

{

int e = public\_key;

long long int encrpyted\_text = 1;

while (e--) {

encrpyted\_text \*= message;

encrpyted\_text %= n;

}

return encrpyted\_text;

}

long long int decrypt(int encrpyted\_text)

{

int d = private\_key;

long long int decrypted = 1;

while (d--) {

decrypted \*= encrpyted\_text;

decrypted %= n;

}

return decrypted;

}

vector<int> encoder(string message)

{

vector<int> form;

for (auto& letter : message)

form.push\_back(encrypt((int)letter));

return form;

}

string decoder(vector<int> encoded)

{

string s;

for (auto& num : encoded)

s += decrypt(num);

return s;

}

int main()

{

primefiller();

setkeys();

string message = "CAB College";

vector<int> coded = encoder(message);

cout << "Initial message:\n" << message;

cout << "\n\nThe encoded message(encrypted by public "

"key)\n";

for (auto& p : coded)

cout << p;

cout << "\n\nThe decoded message(decrypted by private "

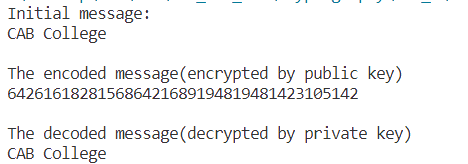
"key)\n";

cout << decoder(coded) << endl;

return 0;

}

**Output:**



**Lab 3.5: Write a program to calculate the Key for two persons using the Diffie Hellman Key exchange**

**Algorithm:**

**STEP-1**: Both Alice and Bob shares the same public keys g and p.

**STEP-2**: Alice selects a random public key a.

**STEP-3**: Alice computes his secret key A as ga mod p.

**STEP-4**: Then Alice sends A to Bob.

**STEP-5**: Similarly Bob also selects a public key b and computes his secret key as B and sends the same back to Alice.

**STEP-6**: Now both of them compute their common secret key as the other one’s secret key power of a mod p

**Source Code:**

#include <cmath>

#include <iostream>

using namespace std;

// Power function to return value of a ^ b mod P

long long int power(long long int a, long long int b,long long int P)

{

if (b == 1)

return a;

else

return (((long long int)pow(a, b)) % P);

}

// Driver program

int main()

{

long long int P, G, x, a, y, b, ka, kb;

// Both the persons will be agreed upon the

// public keys G and P

P = 23; // A prime number P is taken

cout << "The value of P : " << P << endl;

G = 5; // A primitive root for P, G is taken

cout << "The value of G : " << G << endl;

// Alice will choose the private key a

a = 4; // a is the chosen private key

cout << "The private key a for Alice : " << a << endl;

x = power(G, a, P); // gets the generated key

// Bob will choose the private key b

b = 3; // b is the chosen private key

cout << "The private key b for Bob : " << b << endl;

y = power(G, b, P); // gets the generated key

ka = power(y, a, P); // Secret key for Alice

kb = power(x, b, P); // Secret key for Bob

cout << "Secret key for the Alice is : " << ka << endl;

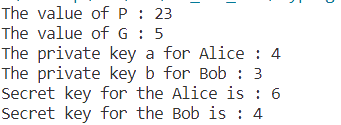
cout << "Secret key for the Bob is : " << kb << endl;

return 0;

}

**Output:**

// This code is contributed by Pranay Arora



**Lab 4**

**Lab 4.1: Write a program to implement MD4 and MD5 algorithm using library functions.**

**MD5Algorithm:**

**STEP-1**: Read the 128-bit plain text.

**STEP-2**: Divide into four blocks of 32

**STEP-3**: Compute the functions f, g, h and i with operations such as, rotations, permutations, etc,.

**STEP-4**: The output of these functions are combined together as F and performed circular shifting and then given to key round.

**STEP-5**: Finally, right shift of ‘s’ times are performed and the results are combined together to produce the final output.

**MD5Algorithm:**

**MD4 Source Code:**

import hashlib

text = 'Hello There'

hashObject = hashlib.new('md4', text.encode('utf-8'))

digest = hashObject.hexdigest()

print("The hexadecimal equivalent of hash using MD4 is \n: ", end ="")

print(digest)

**Output:**



**MD5 Source Code:**

import hashlib

str2hash = "Master Kenobi"

result = hashlib.md5(str2hash.encode())

print("The hexadecimal equivalent of hash using MD5 is \n: ", end ="")

print(result.hexdigest())

**Output:**



**Lab 4.2: Write a program to implement SHA-1 and SHA-2 algorithm using library functions.**

**SHA-1 Algorithm:**

**STEP-1**: Read the 256-bit key values.

**STEP-2**: Divide into five equal-sized blocks named A, B, C, D and E.

**STEP-3**: The blocks B, C and D are passed to the function F.

**STEP-4**: The resultant value is permuted with block E.

**STEP-5**: The block A is shifted right by ‘s’ times and permuted with the result of step-4.

**STEP-6**: Then it is permuted with a weight value and then with some other key pair and taken as the first block.

**STEP-7**: Block A is taken as the second block and the block B is shifted by ‘s’ times and taken as the third block.

**STEP-8**: The blocks C and D are taken as the block D and E for the final output.

**SHA-1 Algorithm:**

**SHA-1 Source Code:**

import hashlib

text = 'Hello There'

result = hashlib.sha1(text.encode())

print("The hexadecimal equivalent of hash using SHA-1 is : ", end ="")

print(result.hexdigest())

**SHA-1 Output:**



**SHA-2 Source Code:**

**S**

**SHA-2 Output:**

**Lab 4.3: Download SSL (Digital Certificate) of a website and analyze its content.**

**Lab 4.2: Write a malicious logic code (Trojan Horse/Virus) program that performs some malicious code.**

**Procedure:**

**Source Code:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main()

{

ofstream fout;

string line = "aaa";

string txt = ".txt";

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

string name = to\_string(i);

string fname = name + txt;

// Open the file outside the loop

fout.open(fname);

// Execute a loop if the file is successfully opened

while (fout) {

for (int j = 0; j <= 10; j++) {

// Generate large text content, you can modify this as needed

fout << "This is line " << j << " in file " << i << endl;

}

// Write line in the file

//fout << line << endl;

fout.close();

}

}

}

**Output:**

