## Name: **Ketan Kunkalikar**

## Roll: **21CSE1016**

## Lab Subject: **Security Lab**

## Topic: **DES Algorithm**

## Instructor: **Dr. Meenakshi Panda**

## Date: **23th October 2024**

## What is the DES **algorithm**? How does it work?

## The Data Encryption Standard (DES) is a block cipher that works on 64-bit data blocks using a 56-bit effective key

## **Main points:**

## **Block Size**: DES processes data in blocks of 64 bits.

## **Key Size**: It uses a 64-bit key, but effectively only 56 bits are used for encryption (8 bits are used for parity).

## **Rounds**: DES performs 16 rounds of processing on the data block, each involving a series of permutations and substitutions.

## **Key Components of DES:**

## **Initial Permutation (IP)**: The input block is permuted using a predefined table.

## **Key Schedule**: The 64-bit key is transformed into a series of 16 subkeys.

## **Rounds**: Each round consists of:

## Expansion: The 32-bit half-block is expanded to 48 bits.

## Key Mixing: The expanded half-block is XORed with a subkey.

## Substitution: The result is passed through S-boxes.

## Permutation: The output is permuted.

## Final Mixing: The result is XORed with the left half and swapped.

## **Final Permutation (IP⁻¹)**: The final output is permuted to produce the ciphertext.

// IMPLEMENT THE ENCRYPTION AND DECRYPTION PROCEDURE FOR THE DES ALGORITHM

#include <bits/stdc++.h>

using namespace std;

// DES Tables

const int IP[] = {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};

const int IP\_1[] = {40, 8, 48, 16, 56, 24, 64, 32,

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

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

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

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

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

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

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

const int PC\_1[] = {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};

const int PC\_2[] = {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};

const int E[] = {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};

const int S\_BOX[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}},

};

const int P[] = {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};

// Function to perform initial permutation

bitset<64> initial\_permutation(const bitset<64>& plain\_text) {

    bitset<64> permuted;

    for(int i = 0; i < 64; i++)

        permuted[63-i] = plain\_text[64-IP[i]];

    cout << "After initial permutation: " << permuted.to\_string() << endl;

    return permuted;

}

// Function to generate 16 subkeys

vector<bitset<48>> generate\_subkeys(const bitset<64>& key) {

    cout << "\nGenerating subkeys..." << endl;

    vector<bitset<48>> subkeys;

    bitset<56> permuted\_choice\_1;

    // PC-1

    for(int i = 0; i < 56; i++)

        permuted\_choice\_1[55-i] = key[64-PC\_1[i]];

    bitset<28> C = bitset<28>((permuted\_choice\_1 >> 28).to\_ulong());

    bitset<28> D = bitset<28>((permuted\_choice\_1 & bitset<56>((1LL << 28) - 1)).to\_ulong());

    cout << "Initial C0: " << C.to\_string() << endl;

    cout << "Initial D0: " << D.to\_string() << endl;

    // Generate 16 subkeys

    for(int round = 0; round < 16; round++) {

        // Left circular shift

        int shift = (round < 2 || round == 8 || round == 15) ? 1 : 2;

        C = (C << shift) | (C >> (28-shift));

        D = (D << shift) | (D >> (28-shift));

        // Combine C and D

        bitset<56> combined;

        for(int i = 0; i < 28; i++) {

            combined[i] = D[i];

            combined[i+28] = C[i];

        }

        // PC-2

        bitset<48> subkey;

        for(int i = 0; i < 48; i++)

            subkey[47-i] = combined[56-PC\_2[i]];

        subkeys.push\_back(subkey);

        cout << "Subkey " << round+1 << ": " << subkey.to\_string() << endl;

    }

    return subkeys;

}

// Function to expand 32 bits to 48 bits

bitset<48> expansion(const bitset<32>& input) {

    bitset<48> output;

    for(int i = 0; i < 48; i++)

        output[47-i] = input[32-E[i]];

    return output;

}

bitset<32> s\_box\_substitution(const bitset<48>& input) {

    bitset<32> output;

    int pos = 0;

    for(int i = 0; i < 8; i++) {

        int row = input[47-6\*i]\*2 + input[47-6\*i-5];

        int col = (input[47-6\*i-1]\*8) + (input[47-6\*i-2]\*4) +

                 (input[47-6\*i-3]\*2) + input[47-6\*i-4];

        int val = S\_BOX[i][row][col];

        bitset<4> bits(val);

        for(int j = 0; j < 4; j++)

            output[31-pos-j] = bits[3-j];

        pos += 4;

    }

    return output;

}

// Function to perform permutation

bitset<32> permutation(const bitset<32>& input) {

    bitset<32> output;

    for(int i = 0; i < 32; i++)

        output[31-i] = input[32-P[i]];

    return output;

}

// Feistel function

bitset<32> f\_function(const bitset<32>& R, const bitset<48>& subkey) {

    cout << "\nFeistel function:" << endl;

    cout << "Input R: " << R.to\_string() << endl;

    bitset<48> expanded = expansion(R);

    cout << "After expansion: " << expanded.to\_string() << endl;

    bitset<48> xored = expanded ^ subkey;

    cout << "After XOR with subkey: " << xored.to\_string() << endl;

    bitset<32> substituted = s\_box\_substitution(xored);

    cout << "After S-box substitution: " << substituted.to\_string() << endl;

    bitset<32> permuted = permutation(substituted);

    cout << "After permutation: " << permuted.to\_string() << endl;

    return permuted;

}

bitset<64> des\_process(bitset<64>& input\_text, const bitset<64>& key, bool encrypt = true) {

    cout << "\nStarting DES " << (encrypt ? "encryption" : "decryption") << "..." << endl;

    cout << (encrypt ? "Plain" : "Cipher") << " text: " << input\_text.to\_string() << endl;

    cout << "Key: " << key.to\_string() << endl;

    // Generate subkeys

    vector<bitset<48>> subkeys = generate\_subkeys(key);

    // For decryption, reverse the subkey order

    if (!encrypt) {

        cout << "\nReversing subkey order for decryption..." << endl;

        reverse(subkeys.begin(), subkeys.end());

    }

    // Initial permutation

    bitset<64> current\_bits = initial\_permutation(input\_text);

    // Split into left and right halves

    bitset<32> left = bitset<32>((current\_bits >> 32).to\_ulong());

    bitset<32> right = bitset<32>((current\_bits & bitset<64>((1LL << 32) - 1)).to\_ulong());

    // 16 rounds

    for(int round = 0; round < 16; round++) {

        cout << "\nRound " << round+1 << ":" << endl;

        cout << "L: " << left.to\_string() << endl;

        cout << "R: " << right.to\_string() << endl;

        bitset<32> temp = right;

        right = left ^ f\_function(right, subkeys[round]);

        left = temp;

    }

    // Combine right and left (reversed)

    bitset<64> pre\_output(((unsigned long long)right.to\_ulong() << 32) | left.to\_ulong());

    // Final permutation

    bitset<64> output;

    for(int i = 0; i < 64; i++)

        output[63-i] = pre\_output[64-IP\_1[i]];

    cout << "\nFinal " << (encrypt ? "cipher" : "plain") << " text: " << output.to\_string() << endl;

    return output;

}

bitset<64> des\_encrypt(bitset<64>& plain\_text, const bitset<64>& key) {

    return des\_process(plain\_text, key, true);

}

bitset<64> des\_decrypt(bitset<64>& cipher\_text, const bitset<64>& key) {

    return des\_process(cipher\_text, key, false);

}

int main() {

    string input\_text, input\_key;

    unsigned long long hex\_text, hex\_key;

    cout << "Enter plain text in hexadecimal: ";

    cin >> input\_text;

    cout << "Enter key in hexadecimal: ";

    cin >> input\_key;

    // Convert hex string to integer

    hex\_text = stoull(input\_text, nullptr, 16);

    hex\_key = stoull(input\_key, nullptr, 16);

    // Create bitsets from the hex values

    bitset<64> plain\_text(hex\_text);

    bitset<64> key(hex\_key);

    cout << "\n=== ENCRYPTION ===" << endl;

    bitset<64> cipher\_text = des\_encrypt(plain\_text, key);

    cout << "\n=== DECRYPTION ===" << endl;

    bitset<64> decrypted\_text = des\_decrypt(cipher\_text, key);

    cout << "\nResults in hexadecimal:" << endl;

    cout << "Original plain text: " << uppercase << input\_text << endl;

    cout << "Key: " << uppercase << input\_key << endl;

    cout << "Cipher text: ";

    for(int i = 60; i >= 0; i -= 4) {

        bitset<4> bits;

        for(int j = 0; j < 4; j++)

            bits[j] = cipher\_text[i+j];

        cout << hex << uppercase << bits.to\_ulong();

    }

    cout << endl;

    cout << "Decrypted text: ";

    for(int i = 60; i >= 0; i -= 4) {

        bitset<4> bits;

        for(int j = 0; j < 4; j++)

            bits[j] = decrypted\_text[i+j];

        cout << hex << uppercase << bits.to\_ulong();

    }

    cout << endl;

    return 0;

}

## **Output: NOTE: I COULD NOT SHARE SS SINCE OUTPUT LOG WAS LONG.**

## **ketan@Arondight MINGW64 ~/Desktop/sem 7/Security Lab/09\_23\_oct\_absent**

## **$ g++ q1.cpp -o q1 && ./q1**

## **Enter plain text in hexadecimal: 123456ABCD132536**

## **Enter key in hexadecimal: AABB09182736CCDD**

## **=== ENCRYPTION ===**

## **Starting DES encryption...**

## **Plain text: 0001001000110100010101101010101111001101000100110010010100110110**

## **Key: 1010101010111011000010010001100000100111001101101100110011011101**

## **Generating subkeys...**

## **Initial C0: 1100001111000000001100111010**

## **Initial D0: 0011001111110000110011111010**

## **Subkey 1: 000110010100110011010000011100101101111010001100**

## **Subkey 2: 010001010110100001011000000110101011110011001110**

## **Subkey 3: 000001101110110110100100101011001111010110110101**

## **Subkey 4: 110110100010110100000011001010110110111011100011**

## **Subkey 5: 011010011010011000101001111111101100100100010011**

## **Subkey 6: 110000011001010010001110100001110100011101011110**

## **Subkey 7: 011100001000101011010010110111011011001111000000**

## **Subkey 8: 001101001111100000100010111100001100011001101101**

## **Subkey 9: 100001001011101101000100011100111101110011001100**

## **Subkey 10: 000000100111011001010111000010001011010110111111**

## **Subkey 11: 011011010101010101100000101011110111110010100101**

## **Subkey 12: 110000101100000111101001011010100100101111110011**

## **Subkey 13: 100110011100001100010011100101111100100100011111**

## **Subkey 14: 001001010001101110001011110001110001011111010000**

## **Subkey 15: 001100110011000011000101110110011010001101101101**

## **Subkey 16: 000110000001110001011101011101011100011001101101**

## **After initial permutation: 0001010010100111110101100111100000011000110010100001100010101101**

## **Round 1:**

## **L: 00010100101001111101011001111000**

## **R: 00011000110010100001100010101101**

## **Feistel function:**

## **Input R: 00011000110010100001100010101101**

## **After expansion: 100011110001011001010100000011110001010101011010**

## **After XOR with subkey: 100101100101101010000100011111011100101111010110**

## **After S-box substitution: 10000000000000000000000000000000**

## **After permutation: 00000000100000000000000000000000**

## **Round 2:**

## **L: 00011000110010100001100010101101**

## **R: 00010100001001111101011001111000**

## **Feistel function:**

## **Input R: 00010100001001111101011001111000**

## **After expansion: 000010101000000100001111111010101100001111110000**

## **After XOR with subkey: 010011111110100101010111111100000111111100111110**

## **After S-box substitution: 01100000000000000000000000000000**

## **After permutation: 00000000000000001000001000000000**

## **Round 3:**

## **L: 00010100001001111101011001111000**

## **R: 00011000110010101001101010101101**

## **Feistel function:**

## **Input R: 00011000110010101001101010101101**

## **After expansion: 100011110001011001010101010011110101010101011010**

## **After XOR with subkey: 100010011111101111110001111000111010000011101111**

## **After S-box substitution: 00010000000000000000000000000000**

## **After permutation: 00000000000000000000000000000010**

## **Round 4:**

## **L: 00011000110010101001101010101101**

## **R: 00010100001001111101011001111010**

## **Feistel function:**

## **Input R: 00010100001001111101011001111010**

## **After expansion: 000010101000000100001111111010101100001111110100**

## **After XOR with subkey: 110100001010110000001100110000011010110100010111**

## **After S-box substitution: 10010000000000000000000000000000**

## **After permutation: 00000000100000000000000000000010**

## **Round 5:**

## **L: 00010100001001111101011001111010**

## **R: 00011000010010101001101010101111**

## **Feistel function:**

## **Input R: 00011000010010101001101010101111**

## **After expansion: 100011110000001001010101010011110101010101011110**

## **After XOR with subkey: 111001101010010001111100101100011001110001001101**

## **After S-box substitution: 10100000000000000000000000000000**

## **After permutation: 00000000100000000000001000000000**

## **Round 6:**

## **L: 00011000010010101001101010101111**

## **R: 00010100101001111101010001111010**

## **Feistel function:**

## **Input R: 00010100101001111101010001111010**

## **After expansion: 000010101001010100001111111010101000001111110100**

## **After XOR with subkey: 110010110000000110000001011011011100010010101010**

## **After S-box substitution: 11000000000000000000000000000000**

## **After permutation: 00000000100000001000000000000000**

## **Round 7:**

## **L: 00010100101001111101010001111010**

## **R: 00011000110010100001101010101111**

## **Feistel function:**

## **Input R: 00011000110010100001101010101111**

## **After expansion: 100011110001011001010100000011110101010101011110**

## **After XOR with subkey: 111111111001110010000110110100101110011010011110**

## **After S-box substitution: 11010000000000000000000000000000**

## **After permutation: 00000000100000001000000000000010**

## **Round 8:**

## **L: 00011000110010100001101010101111**

## **R: 00010100001001110101010001111000**

## **Feistel function:**

## **Input R: 00010100001001110101010001111000**

## **After expansion: 000010101000000100001110101010101000001111110000**

## **After XOR with subkey: 001111100111100100101100010110100100010110011101**

## **After S-box substitution: 00010000000000000000000000000000**

## **After permutation: 00000000000000000000000000000010**

## **Round 9:**

## **L: 00010100001001110101010001111000**

## **R: 00011000110010100001101010101101**

## **Feistel function:**

## **Input R: 00011000110010100001101010101101**

## **After expansion: 100011110001011001010100000011110101010101011010**

## **After XOR with subkey: 000010111010110100010000011111001000100110010110**

## **After S-box substitution: 01000000000000000000000000000000**

## **After permutation: 00000000000000001000000000000000**

## **Round 10:**

## **L: 00011000110010100001101010101101**

## **R: 00010100001001111101010001111000**

## **Feistel function:**

## **Input R: 00010100001001111101010001111000**

## **After expansion: 000010101000000100001111111010101000001111110000**

## **After XOR with subkey: 000010001111011101011000111000100011011001001111**

## **After S-box substitution: 01000000000000000000000000000000**

## **After permutation: 00000000000000001000000000000000**

## **Round 11:**

## **L: 00010100001001111101010001111000**

## **R: 00011000110010101001101010101101**

## **Feistel function:**

## **Input R: 00011000110010101001101010101101**

## **After expansion: 100011110001011001010101010011110101010101011010**

## **After XOR with subkey: 111000100100001100110101111000000010100111111111**

## **After S-box substitution: 00110000000000000000000000000000**

## **After permutation: 00000000000000000000001000000010**

## **Round 12:**

## **L: 00011000110010101001101010101101**

## **R: 00010100001001111101011001111010**

## **Feistel function:**

## **Input R: 00010100001001111101011001111010**

## **After expansion: 000010101000000100001111111010101100001111110100**

## **After XOR with subkey: 110010000100000011100110100000001000100000000111**

## **After S-box substitution: 11000000000000000000000000000000**

## **After permutation: 00000000100000001000000000000000**

## **Round 13:**

## **L: 00010100001001111101011001111010**

## **R: 00011000010010100001101010101101**

## **Feistel function:**

## **Input R: 00011000010010100001101010101101**

## **After expansion: 100011110000001001010100000011110101010101011010**

## **After XOR with subkey: 000101101100000101000111100110001001110001000101**

## **After S-box substitution: 01110000000000000000000000000000**

## **After permutation: 00000000000000001000001000000010**

## **Round 14:**

## **L: 00011000010010100001101010101101**

## **R: 00010100001001110101010001111000**

## **Feistel function:**

## **Input R: 00010100001001110101010001111000**

## **After expansion: 000010101000000100001110101010101000001111110000**

## **After XOR with subkey: 001011111001101010000101011011011001010000100000**

## **After S-box substitution: 00100000000000000000000000000000**

## **After permutation: 00000000000000000000001000000000**

## **Round 15:**

## **L: 00010100001001110101010001111000**

## **R: 00011000010010100001100010101101**

## **Feistel function:**

## **Input R: 00011000010010100001100010101101**

## **After expansion: 100011110000001001010100000011110001010101011010**

## **After XOR with subkey: 101111000011001010010001110101101011011000110111**

## **After S-box substitution: 01110000000000000000000000000000**

## **After permutation: 00000000000000001000001000000010**

## **Round 16:**

## **L: 00011000010010100001100010101101**

## **R: 00010100001001111101011001111010**

## **Feistel function:**

## **Input R: 00010100001001111101011001111010**

## **After expansion: 000010101000000100001111111010101100001111110100**

## **After XOR with subkey: 000100101001110101010010100111110000010110011001**

## **After S-box substitution: 11010000000000000000000000000000**

## **After permutation: 00000000100000001000000000000010**

## **Final cipher text: 0010000100111011101010010101011111001110001000110001101000011101**

## **=== DECRYPTION ===**

## **Starting DES decryption...**

## **Cipher text: 0010000100111011101010010101011111001110001000110001101000011101**

## **Key: 1010101010111011000010010001100000100111001101101100110011011101**

## **Generating subkeys...**

## **Initial C0: 1100001111000000001100111010**

## **Initial D0: 0011001111110000110011111010**

## **Subkey 1: 000110010100110011010000011100101101111010001100**

## **Subkey 2: 010001010110100001011000000110101011110011001110**

## **Subkey 3: 000001101110110110100100101011001111010110110101**

## **Subkey 4: 110110100010110100000011001010110110111011100011**

## **Subkey 5: 011010011010011000101001111111101100100100010011**

## **Subkey 6: 110000011001010010001110100001110100011101011110**

## **Subkey 7: 011100001000101011010010110111011011001111000000**

## **Subkey 8: 001101001111100000100010111100001100011001101101**

## **Subkey 9: 100001001011101101000100011100111101110011001100**

## **Subkey 10: 000000100111011001010111000010001011010110111111**

## **Subkey 11: 011011010101010101100000101011110111110010100101**

## **Subkey 12: 110000101100000111101001011010100100101111110011**

## **Subkey 13: 100110011100001100010011100101111100100100011111**

## **Subkey 14: 001001010001101110001011110001110001011111010000**

## **Subkey 15: 001100110011000011000101110110011010001101101101**

## **Subkey 16: 000110000001110001011101011101011100011001101101**

## **Reversing subkey order for decryption...**

## **After initial permutation: 0001100011001010100110001010111100010100001001111101011001111010**

## **Round 1:**

## **L: 00011000110010101001100010101111**

## **R: 00010100001001111101011001111010**

## **Feistel function:**

## **Input R: 00010100001001111101011001111010**

## **After expansion: 000010101000000100001111111010101100001111110100**

## **After XOR with subkey: 000100101001110101010010100111110000010110011001**

## **After S-box substitution: 11010000000000000000000000000000**

## **After permutation: 00000000100000001000000000000010**

## **Round 2:**

## **L: 00010100001001111101011001111010**

## **R: 00011000010010100001100010101101**

## **Feistel function:**

## **Input R: 00011000010010100001100010101101**

## **After expansion: 100011110000001001010100000011110001010101011010**

## **After XOR with subkey: 101111000011001010010001110101101011011000110111**

## **After S-box substitution: 01110000000000000000000000000000**

## **After permutation: 00000000000000001000001000000010**

## **Round 3:**

## **L: 00011000010010100001100010101101**

## **R: 00010100001001110101010001111000**

## **Feistel function:**

## **Input R: 00010100001001110101010001111000**

## **After expansion: 000010101000000100001110101010101000001111110000**

## **After XOR with subkey: 001011111001101010000101011011011001010000100000**

## **After S-box substitution: 00100000000000000000000000000000**

## **After permutation: 00000000000000000000001000000000**

## **Round 4:**

## **L: 00010100001001110101010001111000**

## **R: 00011000010010100001101010101101**

## **Feistel function:**

## **Input R: 00011000010010100001101010101101**

## **After expansion: 100011110000001001010100000011110101010101011010**

## **After XOR with subkey: 000101101100000101000111100110001001110001000101**

## **After S-box substitution: 01110000000000000000000000000000**

## **After permutation: 00000000000000001000001000000010**

## **Round 5:**

## **L: 00011000010010100001101010101101**

## **R: 00010100001001111101011001111010**

## **Feistel function:**

## **Input R: 00010100001001111101011001111010**

## **After expansion: 000010101000000100001111111010101100001111110100**

## **After XOR with subkey: 110010000100000011100110100000001000100000000111**

## **After S-box substitution: 11000000000000000000000000000000**

## **After permutation: 00000000100000001000000000000000**

## **Round 6:**

## **L: 00010100001001111101011001111010**

## **R: 00011000110010101001101010101101**

## **Feistel function:**

## **Input R: 00011000110010101001101010101101**

## **After expansion: 100011110001011001010101010011110101010101011010**

## **After XOR with subkey: 111000100100001100110101111000000010100111111111**

## **After S-box substitution: 00110000000000000000000000000000**

## **After permutation: 00000000000000000000001000000010**

## **Round 7:**

## **L: 00011000110010101001101010101101**

## **R: 00010100001001111101010001111000**

## **Feistel function:**

## **Input R: 00010100001001111101010001111000**

## **After expansion: 000010101000000100001111111010101000001111110000**

## **After XOR with subkey: 000010001111011101011000111000100011011001001111**

## **After S-box substitution: 01000000000000000000000000000000**

## **After permutation: 00000000000000001000000000000000**

## **Round 8:**

## **L: 00010100001001111101010001111000**

## **R: 00011000110010100001101010101101**

## **Feistel function:**

## **Input R: 00011000110010100001101010101101**

## **After expansion: 100011110001011001010100000011110101010101011010**

## **After XOR with subkey: 000010111010110100010000011111001000100110010110**

## **After S-box substitution: 01000000000000000000000000000000**

## **After permutation: 00000000000000001000000000000000**

## **Round 9:**

## **L: 00011000110010100001101010101101**

## **R: 00010100001001110101010001111000**

## **Feistel function:**

## **Input R: 00010100001001110101010001111000**

## **After expansion: 000010101000000100001110101010101000001111110000**

## **After XOR with subkey: 001111100111100100101100010110100100010110011101**

## **After S-box substitution: 00010000000000000000000000000000**

## **After permutation: 00000000000000000000000000000010**

## **Round 10:**

## **L: 00010100001001110101010001111000**

## **R: 00011000110010100001101010101111**

## **Feistel function:**

## **Input R: 00011000110010100001101010101111**

## **After expansion: 100011110001011001010100000011110101010101011110**

## **After XOR with subkey: 111111111001110010000110110100101110011010011110**

## **After S-box substitution: 11010000000000000000000000000000**

## **After permutation: 00000000100000001000000000000010**

## **Round 11:**

## **L: 00011000110010100001101010101111**

## **R: 00010100101001111101010001111010**

## **Feistel function:**

## **Input R: 00010100101001111101010001111010**

## **After expansion: 000010101001010100001111111010101000001111110100**

## **After XOR with subkey: 110010110000000110000001011011011100010010101010**

## **After S-box substitution: 11000000000000000000000000000000**

## **After permutation: 00000000100000001000000000000000**

## **Round 12:**

## **L: 00010100101001111101010001111010**

## **R: 00011000010010101001101010101111**

## **Feistel function:**

## **Input R: 00011000010010101001101010101111**

## **After expansion: 100011110000001001010101010011110101010101011110**

## **After XOR with subkey: 111001101010010001111100101100011001110001001101**

## **After S-box substitution: 10100000000000000000000000000000**

## **After permutation: 00000000100000000000001000000000**

## **Round 13:**

## **L: 00011000010010101001101010101111**

## **R: 00010100001001111101011001111010**

## **Feistel function:**

## **Input R: 00010100001001111101011001111010**

## **After expansion: 000010101000000100001111111010101100001111110100**

## **After XOR with subkey: 110100001010110000001100110000011010110100010111**

## **After S-box substitution: 10010000000000000000000000000000**

## **After permutation: 00000000100000000000000000000010**

## **Round 14:**

## **L: 00010100001001111101011001111010**

## **R: 00011000110010101001101010101101**

## **Feistel function:**

## **Input R: 00011000110010101001101010101101**

## **After expansion: 100011110001011001010101010011110101010101011010**

## **After XOR with subkey: 100010011111101111110001111000111010000011101111**

## **After S-box substitution: 00010000000000000000000000000000**

## **000001000000000**

## **Round 16:**

## **L: 00010100001001111101011001111000**

## **R: 00011000110010100001100010101101**

## **Feistel function:**

## **Input R: 00011000110010100001100010101101**

## **After expansion: 100011110001011001010100000011110001010101011010**

## **After XOR with subkey: 100101100101101010000100011111011100101111010110**

## **After S-box substitution: 10000000000000000000000000000000**

## **After permutation: 00000000100000000000000000000000**

## **Final plain text: 0001001000110100010101101010101111001101000100110010010100110110**

## **Results in hexadecimal:**

## **Original plain text: 123456ABCD132536**

## **Key: AABB09182736CCDD**

## **Cipher text: 213BA957CE231A1D**

## **Decrypted text: 123456ABCD132536**

## **ketan@Arondight MINGW64 ~/Desktop/sem 7/Security Lab/09\_23\_oct\_absent**