Cryptography and Network Security Lab
Digital Assignment 2
22BCE3939
Karan Sehgal

1. Without using library functions develop a menu-driven code to simulate the following symmetric algorithms.
i. S-DES or DES
ii. AES
NOTE: The program should have sufficient test cases to perform data validation of the input and should run for any type (Binary and Hexadecimal) and for any length of input.

Pseudocode:

Cryptography and Network Security
Digital Assignment 2
22BCE3939
Karan Sehgal

Pseudo code :-

```
// function for SDES
FUNCTION  SDES_ENCRYPTION (plaintext,
                                  key)
   // Step 1 : Generate SubKeys
   (K1 , K2) = Generate SubKeys (Key)

   // Step 2 : Initial Permutation (IP)
     STATE = Initial Permutation ( plain text)

   // Step 3 : First Round
    Left, right = Split (STATE)
    Temp = Feistel Function (Right, K1) XOR left
    left = right
    right = temp
   // Step 4 : Second Right(Round)
     Temp = Fiestel Function (Right, K2) XOR
                                         left
     left = right
     right = temp
    Ciphertext = Inverse IP ( left + right)

   return Ciphertext
```

22BCE3939

// Sub-functions for SDES

FUNCTION GenerateSubKeys (Key)

Perform permutation P10 on Key

Split into two halves

Perform left shift (LS-1) on both halves

K1 = permute 8 (left + right)

Perform lef shift (LS-2) on both halves

K2 = permute 8 (left + right)

return {K1, K2}

FUNCTION FiestelFunction (data, subkey)

Expanded = ExpansionPerm (data)

XOR_res = Expanded XOR subkey

SBX_Output = ApplySBoxes (XOR_res)

perm_res = PermutationP4 (SBX_Output)

return perm_res

// function for RC4 22BCE3939

```
FUNCTION RC4_KeyScheduling(Key)
    FOR i from 0 to 255:
        S[i] = i
    j = 0
    FOR i from 0 to 255:
        j = (j + S[i] + Key[i MOD Key-length])
                                        MOD 256
        swap (S[i], S[j])
    Return S

FUNCTION RC4_GenerateStream (S, datalen)
    i = 0
    j = 0
    Output = []
    For k from 0 to datalen:
        i = (i + 1) MOD 256
        j = (j + S[i]) MOD 256
        swap (S[i], S[j])
        t = (S[i] + S[j]) MOD 256
        Output[k] = S[t]

    Return Output.
```

22 BCE 3939

FUNCTION RC4_Encrypt (plaintext, key)

S= RC4_KeyScheduling (key)

STREAM = RC4_Generate Stream (S, Length (plaintext))

FOR i from 0 to Length (plaintext):

ciphertext [i] = plaintext[i] XOR stream [i]

Return ciphertext

FUNCTION RC4_Decrypt (ciphertext, key)

S = RC4_Key scheduling (key)

STREAM = RC4_Generate Stream (S, Length (ciphertext))

FOR i from 0 to length ([iphertext)

plaintext [i] = ciphertext[i] XOR stream [i]

return plaintoxt

// AES Encryption 22BCE3939

```
AES_ENCRYPTION( plaintext , key)
  STATE = Convert Plain Text to Mat ( plaintext )
  Round Keys = Key Expansion (key)
  STATE = Add Round key ( STATE ,
                          Round keys [0] )
  FOR rounds from 1 to NR-1 do:
     STATE = Sub Bytes ( STATE)
     STATE = Shift Rows ( STATE)
     STATE = Mix Columns ( STATE )
     STATE = Add Round key ( STATE,
                             Round keys [rounds]
  // for Final Round
  STATE = SubBytes (STATE)
  STATE = ShiftRows ( STATE)
  STATE = Add Round Key (STATE, Roundkeys[NR])
  Ciphertext = Convert StateMat to Text (STATE)
  return Ciphertext)
```

// Sub-functions for AES 22BCE3939

Function ConvertPlaintext-toMat ('plaintext')

CONVERT plaintext to a 4X4 byte Matrix
(column major order)

return StateMatrix

FUNCTION KeyExpansion (key)

Generate Nr+1 round keys using
Rijndael scheduling mechanism
// Rot Word & SubWord Generate
Compute rot word for (i%4==0)

Shift the word

Substitute using SBox

XOR with RCond[i] array

// XOR with previous round key

round[j] = round[j] XOR
round[i-4][j]

// Store the keys in round keys.

roundKeys.append(round)

FUNCTION SubBytes (key)

For each byte in state key :
Replace byte with corresponding value from S-Box

return updated state.

22BCE3939

FUNCTION ShiftRows (STATE):

Row 0 remains unchanged

Row 1 shifts left by 1 pos^n

Row 2 shifts left by 2 pos^n

Row 3 shifts left by 3 pos^n

Return updated State

FUNCTION MixColums (STATTE)

FOR each column in STATE:

Apply MixColums transformation using Galois field multiplicatn

Return updated STATE

FUNCTION AddRoundKey (STATE, round)

FOR each byte in STATE do:

STATE[byte] = STATE[byte] XOR round[byte]

return updated State

FUNCTION ConvertStateMatToText (STATE)

Convert 4x4 matrix back into ciphertext

return ciphertext

Source Code:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <bitset>
#include <sstream>
#include <stdexcept>
#include <algorithm>

using namespace std;

// S-DES Constants
const int P10[10] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
const int P8[8] = {6, 3, 7, 4, 8, 5, 10, 9};
const int IP[8] = {2, 6, 3, 1, 4, 8, 5, 7};
const int EP[8] = {4, 1, 2, 3, 2, 3, 4, 1};
const int P4[4] = {2, 4, 3, 1};

const int S0[4][4] = {
    {1, 0, 3, 2},
    {3, 2, 1, 0},
    {0, 2, 1, 3},
    {3, 1, 3, 2}
};

const int S1[4][4] = {
    {0, 1, 2, 3},
    {2, 0, 1, 3},
    {3, 0, 1, 0},
    {2, 1, 0, 3}
};

// AES S-Box (useful for encryption)
const unsigned char SBOX[256] = {
    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
};

// Corresponding Inverse S-Box (useful for decryption)
const unsigned char INV_SBOX[256] = {
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
```

```
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf,
0xce, 0xf0, 0xb4, 0xe6, 0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9,
0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7,
0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb,
0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12,
0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5,
0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb,
0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69,
0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
};

class AES {
private:
    vector<vector<unsigned char>> state;
    vector<vector<unsigned char>> roundKeys;
    int Nr; // Number of rounds

    void printState(const string& label) {
        cout << label << ":" << endl;
        for(const auto& row : state) {
            for(unsigned char byte : row) {
                cout << hex << setw(2) << setfill('0') <<
static_cast<int>(byte) << " ";
            }
            cout << endl;
        }
        cout << endl;
    }

    void subBytes() {
        for(auto& row : state) {
            transform(row.begin(), row.end(), row.begin(),
                [](unsigned char byte) { return SBOX[byte]; });
        }
        printState("After SubBytes");
    }
```

```
void shiftRows() {
    // First row remains unchanged
    // Second row shifts left by 1
    rotate(state[1].begin(), state[1].begin() + 1, state[1].end());

    // Third row shifts left by 2
    rotate(state[2].begin(), state[2].begin() + 2, state[2].end());

    // Fourth row shifts left by 3 (or right by 1)
    rotate(state[3].begin(), state[3].begin() + 3, state[3].end());

    printState("After ShiftRows");
}

unsigned char gmul(unsigned char a, unsigned char b) {
    unsigned char result = 0;
    while (b) {
        if (b & 1) result ^= a;
        bool hi_bit_set = (a & 0x80);
        a <<= 1;
        if (hi_bit_set) a ^= 0x1B; // x^8 + x^4 + x^3 + x + 1
        b >>= 1;
    }
    return result;
}

void mixColumns() {
    vector<vector<unsigned char>> temp = state;
    for (int c = 0; c < 4; ++c) {
        state[0][c] = gmul(temp[0][c], 2) ^ gmul(temp[1][c], 3) ^
                      temp[2][c] ^ temp[3][c];
        state[1][c] = temp[0][c] ^ gmul(temp[1][c], 2) ^
                      gmul(temp[2][c], 3) ^ temp[3][c];
        state[2][c] = temp[0][c] ^ temp[1][c] ^
                      gmul(temp[2][c], 2) ^ gmul(temp[3][c], 3);
        state[3][c] = gmul(temp[0][c], 3) ^ temp[1][c] ^
                      temp[2][c] ^ gmul(temp[3][c], 2);
    }
    printState("After MixColumns");
}

void addRoundKey(int round) {
    for(int i = 0; i < 4; ++i) {
        for(int j = 0; j < 4; ++j) {
```

```
                state[i][j] ^= roundKeys[round*4 + i][j];
            }
        }
        printState("After AddRoundKey");
    }

    void keyExpansion(const vector<unsigned char>& key) {
        roundKeys.clear();
        // Simplified key expansion for 128-bit key
        for(int i = 0; i < 44; ++i) {
            vector<unsigned char> roundKey(4, 0);
            if(i < 4) {
                // First 4 words are directly from the key
                for(int j = 0; j < 4; ++j) {
                    roundKey[j] = key[i*4 + j];
                }
            } else {
                // Subsequent words are XORed
                roundKey = roundKeys[i-1];
                if(i % 4 == 0) {
                    // RotWord and SubWord
                    rotate(roundKey.begin(), roundKey.begin() + 1,
roundKey.end());
                    transform(roundKey.begin(), roundKey.end(),
roundKey.begin(),
                        [](unsigned char byte) { return SBOX[byte]; });
                    // XOR with round constant
                    roundKey[0] ^= (1 << ((i/4 - 1) % 10));
                }

                // XOR with previous round key
                for(int j = 0; j < 4; ++j) {
                    roundKey[j] ^= roundKeys[i-4][j];
                }
            }
            roundKeys.push_back(roundKey);
        }
    }

public:
    AES(const vector<unsigned char>& key) : Nr(10) {
        // Initialize state and perform key expansion
        state = vector<vector<unsigned char>>(4, vector<unsigned
char>(4));
```

```
        keyExpansion(key);
    }

    vector<unsigned char> encrypt(const vector<unsigned char>&
plaintext) {
        // Initialize state from plaintext
        for(int i = 0; i < 4; ++i) {
            for(int j = 0; j < 4; ++j) {
                state[j][i] = plaintext[i*4 + j];
            }
        }
        printState("Initial State");

        // Initial round key
        addRoundKey(0);

        // Main rounds
        for(int round = 1; round < Nr; ++round) {
            cout<< "\nRound " << round << ":" << endl;
            subBytes();
            shiftRows();
            mixColumns();
            addRoundKey(round);
        }

        // Final round (no MixColumns)
        subBytes();
        shiftRows();
        addRoundKey(Nr);

        // Convert state back to vector
        vector<unsigned char> ciphertext;
        for(int i = 0; i < 4; ++i) {
            for(int j = 0; j < 4; ++j) {
                ciphertext.push_back(state[j][i]);
            }
        }
        return ciphertext;
    }

    // Helper method to convert hex string to bytes
    static vector<unsigned char> hexToBytes(const string& hex) {
        vector<unsigned char> bytes;
        for(size_t i = 0; i < hex.length(); i += 2) {
```

```
        string byteString = hex.substr(i, 2);
        unsigned char byte = stoi(byteString, nullptr, 16);
        bytes.push_back(byte);
    }
    return bytes;
  }

  // Helper method to convert bytes to hex string
  static string bytesToHex(const vector<unsigned char>& bytes)
{
    stringstream ss;
    ss << hex << setfill('0');
    for(unsigned char byte : bytes) {
        ss << setw(2) << static_cast<int>(byte);
    }
    return ss.str();
  }
};


class SDES {
private:
  string key;
  vector<string> subKeys;

  string permute(const string& input, const int* pattern, int
patternSize) {
    string output;
    for(int i = 0; i < patternSize; i++) {
        output += input[pattern[i] - 1];
    }
    return output;
  }

  string leftShift(const string& s, int positions) {
    return s.substr(positions) + s.substr(0, positions);
  }

  void generateSubKeys() {
    cout << "Generating Subkeys:" << endl;
    string permuted = permute(key, P10, 10);
    string left = permuted.substr(0, 5);
    string right = permuted.substr(5, 5);
```

```
        left = leftShift(left, 1);
        right = leftShift(right, 1);
        subKeys.push_back(permute(left + right, P8, 8));
        cout << "Subkey 1: " << subKeys.back() << endl;

        left = leftShift(left, 2);
        right = leftShift(right, 2);
        subKeys.push_back(permute(left + right, P8, 8));
        cout << "Subkey 2: " << subKeys.back() << endl;
    }

    string sBox(const string& input, const int sbox[4][4]) {
        int row = (input[0] - '0') * 2 + (input[3] - '0');
        int col = (input[1] - '0') * 2 + (input[2] - '0');
        return bitset<2>(sbox[row][col]).to_string();
    }

    string fFunction(const string& right, const string& subkey) {
        string expanded = permute(right, EP, 8);
        cout << "Expanded Right: " << expanded << endl;

        string xored;
        for(size_t i = 0; i < expanded.length(); i++) {
            xored += (expanded[i] != subkey[i]) ? '1' : '0';
        }
        cout << "XOR with Subkey: " << xored << endl;

        string s0Result = sBox(xored.substr(0, 4), S0);
        string s1Result = sBox(xored.substr(4, 4), S1);
        string combined = s0Result + s1Result;
        cout << "S-Box Output: " << combined << endl;

        return permute(combined, P4, 4);
    }

public:
    SDES(const string& inputKey) : key(inputKey) {
        generateSubKeys();
    }

    string encrypt(const string& plaintext) {
        string current = permute(plaintext, IP, 8);
        cout << "Initial Permutation: " << current << endl;
```

```
        for(int round = 0; round < 2; round++) {
            string left = current.substr(0, 4);
            string right = current.substr(4, 4);

            string fResult = fFunction(right, subKeys[round]);

            string newRight;
            for(int i = 0; i < 4; i++) {
                newRight += (left[i] != fResult[i]) ? '1' : '0';
            }
            cout << "Round " << round + 1 << " - Left: " << left << "
Right: " << right << " NewRight: " << newRight << endl;

            current = (round == 0) ? (right + newRight) : (newRight +
right);
        }

        return permute(current, IP, 8);
    }

    string decrypt(const string& ciphertext) {
        string current = permute(ciphertext, IP, 8);
        cout << "Initial Permutation: " << current << endl;

        for(int round = 0; round < 2; round++) {
            string left = current.substr(0, 4);
            string right = current.substr(4, 4);

            string fResult = fFunction(right, subKeys[1 - round]);

            string newRight;
            for(int i = 0; i < 4; i++) {
                newRight += (left[i] != fResult[i]) ? '1' : '0';
            }
            cout << "Round " << round + 1 << " - Left: " << left << "
Right: " << right << " NewRight: " << newRight << endl;

            current = (round == 0) ? (right + newRight) : (newRight +
right);
        }

        return permute(current, IP, 8);
    }
```

```
    static string hexToBinary(const string& hex) {
        stringstream binary;
        for(char c : hex) {
            int value = (c >= 'A') ? (c - 'A' + 10) : (c - '0');
            binary << bitset<4>(value);
        }
        return binary.str();
    }

    static string binaryToHex(const string& binary) {
        stringstream hex;
        hex << std::hex << std::setfill('0');
        for(size_t i = 0; i < binary.length(); i += 4) {
            string chunk = binary.substr(i, 4);
            hex << std::setw(1) << std::stoi(chunk, nullptr, 2);
        }
        return hex.str();
    }
};


class RC4 {
public:
    string processToHex(const string& input, const string& keyHex,
bool encrypt = true) {
        vector<unsigned char> inputBytes = hexToBytes(input);
        vector<unsigned char> keyBytes = hexToBytes(keyHex);

        vector<int> S(256);
        for(int k = 0; k < 256; k++) S[k] = k;

        int j = 0;
        cout << "\nKey-Scheduling Algorithm (KSA) steps:" << endl;
        for(int k = 0; k < 256; k++) {
            j = (j + S[k] + keyBytes[k % keyBytes.size()]) % 256;
            swap(S[k], S[j]);
            cout << "S[" << k << "] swapped with S[" << j << "]" <<
endl;
        }

        vector<unsigned char> outputBytes;
        int i = 0;
        j = 0;
```

```
        string processType = encrypt ? "Encryption" : "Decryption";
        cout << "\nPseudo-Random Generation Algorithm (PRGA)
steps ("
             << processType << "):" << endl;

        for(size_t index = 0; index < inputBytes.size(); index++) {
            i = (i + 1) % 256;
            j = (j + S[i]) % 256;
            swap(S[i], S[j]);
            int k = S[(S[i] + S[j]) % 256];
            outputBytes.push_back(inputBytes[index] ^ k);

            cout << "Step " << index + 1 << ": i=" << i << ", j=" << j
                 << ", Key Stream Byte=" << hex << setw(2) <<
setfill('0')
                 << static_cast<int>(k) << dec << " XOR with "
                 << static_cast<int>(inputBytes[index]) << " = "
                 << static_cast<int>(outputBytes.back()) << endl;
        }

        return bytesToHex(outputBytes);
    }

private:
    vector<unsigned char> hexToBytes(const string& hex) {
        vector<unsigned char> bytes;
        for(size_t i = 0; i < hex.length(); i += 2) {
            string byteString = hex.substr(i, 2);
            unsigned char byte = stoi(byteString, nullptr, 16);
            bytes.push_back(byte);
        }
        return bytes;
    }

    string bytesToHex(const vector<unsigned char>& bytes) {
        stringstream ss;
        ss << hex << setfill('0');
        for(unsigned char byte : bytes) {
            ss << setw(2) << static_cast<int>(byte);
        }
        return ss.str();
    }
};
```

```
class SymmetricEncryptionTool {
private:
    SDES sdes;
    RC4 rc4;

public:
    SymmetricEncryptionTool() : sdes("1010101010") {}

    void runMainMenu() {
        while(true) {
            cout << "\n=== Symmetric Encryption Tool ===\n";
            cout << "1. S-DES Encryption\n";
            cout << "2. S-DES Decryption\n";
            cout << "3. RC4 Encryption\n";
            cout << "4. RC4 Decryption\n";
            cout << "5. AES Encryption\n";
            cout << "6. Exit\n";
            cout << "Enter your choice (1-6): ";
            int choice;
            cin >> choice;

            switch(choice) {
                case 1: {
                    string plaintext, key;
                    cout << "Enter plaintext (8-bit binary or 2-digit hex):
";
                    cin >> plaintext;
                    cout << "Enter key (10-bit binary or 3-digit hex): ";
                    cin >> key;

                    if(plaintext.length() == 2) plaintext =
SDES::hexToBinary(plaintext);
                    if(key.length() == 3) key = SDES::hexToBinary(key);

                    SDES currentSdes(key);
                    string encrypted = currentSdes.encrypt(plaintext);
                    cout << "Encrypted (binary): " << encrypted << endl;
                    cout << "Encrypted (hex): " <<
SDES::binaryToHex(encrypted) << endl;
                    break;
                }
                case 2: {
                    string ciphertext, key;
```

```
            cout << "Enter ciphertext (8-bit binary or 2-digit hex):
";
            cin >> ciphertext;
            cout << "Enter key (10-bit binary or 3-digit hex): ";
            cin >> key;

            if(ciphertext.length() == 2) ciphertext =
SDES::hexToBinary(ciphertext);
            if(key.length() == 3) key = SDES::hexToBinary(key);

            SDES currentSdes(key);
            string decrypted = currentSdes.decrypt(ciphertext);
            cout << "Decrypted (binary): " << decrypted << endl;
            cout << "Decrypted (hex): " <<
SDES::binaryToHex(decrypted) << endl;
            break;
        }
        case 3: {
            string input, key;
            cout << "Enter input in hex: ";
            cin >> input;
            cout << "Enter key in hex: ";
            cin >> key;

            string result = rc4.processToHex(input, key, true);
            cout << "Encrypted Result: " << result << endl;
            break;
        }
        case 4: {
            string input, key;
            cout << "Enter input in hex: ";
            cin >> input;
            cout << "Enter key in hex: ";
            cin >> key;

            string result = rc4.processToHex(input, key, false);
            cout << "Decrypted Result: " << result << endl;
            break;
        }
        case 5: {
            string input, keyHex;
            cout << "Enter 128-bit input in hex (32 hex
characters): ";
            cin >> input;
```

```
                cout << "Enter 128-bit key in hex (32 hex characters):
";
                cin >> keyHex;

                try {
                    vector<unsigned char> inputBytes =
AES::hexToBytes(input);
                    vector<unsigned char> keyBytes =
AES::hexToBytes(keyHex);

                    AES aes(keyBytes);
                    vector<unsigned char> ciphertext =
aes.encrypt(inputBytes);

                    cout << "Ciphertext (hex): "
                         << AES::bytesToHex(ciphertext) << endl;
                } catch(const exception& e) {
                    cerr << "Error: " << e.what() << endl;
                }
                break;
            }
            case 6:
                cout << "Exiting...\n";
                return;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }
  }
};

int main() {
    try {
        SymmetricEncryptionTool tool;
        tool.runMainMenu();
    } catch(const exception& e) {
        cerr << "Error: " << e.what() << endl;
        return 1;
    }

    return 0;
}
```

Output: (For S-DES)

1) Handling Binary Input:

```
=== Symmetric Encryption Tool ===
1. S-DES Encryption
2. S-DES Decryption
3. RC4 Encryption
4. RC4 Decryption
5. AES Encryption
6. Exit
Enter your choice (1-6): 1
Enter plaintext (8-bit binary or 2-digit hex): 00000000
Enter key (10-bit binary or 3-digit hex): 1010101010
Generating Subkeys:
Subkey 1: 11100100
Subkey 2: 01010011
Initial Permutation: 00000000
Expanded Right: 00000000
XOR with Subkey: 11100100
S-Box Output: 1110
Round 1 - Left: 0000 Right: 0000 NewRight: 1011
Expanded Right: 11010111
XOR with Subkey: 10000100
S-Box Output: 0010
Round 2 - Left: 0000 Right: 1011 NewRight: 0010
Encrypted (binary): 00100111
Encrypted (hex): 27
```

2) Handling Hex input:

```
=== Symmetric Encryption Tool ===
1. S-DES Encryption
2. S-DES Decryption
3. RC4 Encryption
4. RC4 Decryption
5. AES Encryption
6. Exit
Enter your choice (1-6): 1
Enter plaintext (8-bit binary or 2-digit hex): AB
Enter key (10-bit binary or 3-digit hex): 2CC
Generating Subkeys:
Subkey 1: 00100111
Subkey 2: 01111010
Initial Permutation: 00110111
Expanded Right: 10111110
XOR with Subkey: 10011001
S-Box Output: 1110
Round 1 - Left: 0011 Right: 0111 NewRight: 1000
Expanded Right: 01000001
XOR with Subkey: 00111011
S-Box Output: 1001
Round 2 - Left: 0111 Right: 1000 NewRight: 0010
Encrypted (binary): 00100010
Encrypted (hex): 22
```

3)Decryption:

```
=== Symmetric Encryption Tool ===
1. S-DES Encryption
2. S-DES Decryption
3. RC4 Encryption
4. RC4 Decryption
5. AES Encryption
6. Exit
Enter your choice (1-6): 2
Enter ciphertext (8-bit binary or 2-digit hex): 2C
Enter key (10-bit binary or 3-digit hex): ABB
Generating Subkeys:
Subkey 1: 11101100
Subkey 2: 11010011
Initial Permutation: 01100010
Expanded Right: 00010100
XOR with Subkey: 11000111
S-Box Output: 0111
Round 1 - Left: 0110 Right: 0010 NewRight: 1000
Expanded Right: 01000001
XOR with Subkey: 10101101
S-Box Output: 1000
Round 2 - Left: 0010 Right: 1000 NewRight: 0011
Decrypted (binary): 00101010
Decrypted (hex): 2a
```

4) Invalid key and plain text:

```
=== Symmetric Encryption Tool ===
1. S-DES Encryption
2. S-DES Decryption
3. RC4 Encryption
4. RC4 Decryption
5. AES Encryption
6. Exit
Enter your choice (1-6): 1
Enter plaintext (8-bit binary or 2-digit hex): 1010101011010110
Enter key (10-bit binary or 3-digit hex): 1010101
Invalid plaintext /  key length!
```

OUTPUT: (RC4)

1) Encryption:

```
=== Symmetric Encryption Tool ===
1. S-DES Encryption
2. S-DES Decryption
3. RC4 Encryption
4. RC4 Decryption
5. AES Encryption
6. Exit
Enter your choice (1-6): 3
Enter input in hex: 00112233
Enter key in hex: 0123456789

Key-Scheduling Algorithm (KSA) steps:
S[0] swapped with S[1]
S[1] swapped with S[36]
S[2] swapped with S[107]
S[3] swapped with S[213]
S[4] swapped with S[98]
S[5] swapped with S[104]
S[6] swapped with S[145]
S[7] swapped with S[221]
S[8] swapped with S[76]
S[9] swapped with S[222]
S[10] swapped with S[233]
S[11] swapped with S[23]
S[12] swapped with S[104]
S[13] swapped with S[220]
S[14] swapped with S[115]
S[15] swapped with S[131]
S[16] swapped with S[182]
S[17] swapped with S[12]
S[18] swapped with S[133]
S[19] swapped with S[33]
S[20] swapped with S[54]
S[21] swapped with S[110]
S[22] swapped with S[201]
S[23] swapped with S[59]
S[24] swapped with S[220]
S[25] swapped with S[246]
S[26] swapped with S[51]
S[27] swapped with S[147]
S[28] swapped with S[22]
S[29] swapped with S[188]
S[30] swapped with S[219]
S[31] swapped with S[29]
S[32] swapped with S[130]
```

```
S[218] swapped with S[136]
S[219] swapped with S[212]
S[220] swapped with S[26]
S[221] swapped with S[68]
S[222] swapped with S[146]
S[223] swapped with S[216]
S[224] swapped with S[180]
S[225] swapped with S[150]
S[226] swapped with S[155]
S[227] swapped with S[195]
S[228] swapped with S[14]
S[229] swapped with S[215]
S[230] swapped with S[190]
S[231] swapped with S[200]
S[232] swapped with S[85]
S[233] swapped with S[24]
S[234] swapped with S[139]
S[235] swapped with S[119]
S[236] swapped with S[134]
S[237] swapped with S[255]
S[238] swapped with S[147]
S[239] swapped with S[11]
S[240] swapped with S[252]
S[241] swapped with S[16]
S[242] swapped with S[255]
S[243] swapped with S[89]
S[244] swapped with S[237]
S[245] swapped with S[188]
S[246] swapped with S[248]
S[247] swapped with S[52]
S[248] swapped with S[180]
S[249] swapped with S[54]
S[250] swapped with S[49]
S[251] swapped with S[79]
S[252] swapped with S[132]
S[253] swapped with S[232]
S[254] swapped with S[145]
S[255] swapped with S[60]

Pseudo-Random Generation Algorithm (PRGA) steps (Encryption):
Step 1: i=1, j=36, Key Stream Byte=1b XOR with 0 = 27
Step 2: i=2, j=143, Key Stream Byte=35 XOR with 17 = 36
Step 3: i=3, j=100, Key Stream Byte=63 XOR with 34 = 65
Step 4: i=4, j=49, Key Stream Byte=2a XOR with 51 = 25
Encrypted Result: 1b244119
```

It takes 256 iterations to generate key stream.

2) Decryption:

```
=== Symmetric Encryption Tool ===
1. S-DES Encryption
2. S-DES Decryption
3. RC4 Encryption
4. RC4 Decryption
5. AES Encryption
6. Exit
Enter your choice (1-6): 4
Enter input in hex: 1b244119
Enter key in hex: 0123456789

Key-Scheduling Algorithm (KSA) steps:
S[0] swapped with S[1]
S[1] swapped with S[36]
S[2] swapped with S[107]
S[3] swapped with S[213]
S[4] swapped with S[98]
S[5] swapped with S[104]
S[6] swapped with S[145]
S[7] swapped with S[221]
S[8] swapped with S[76]
S[9] swapped with S[222]
S[10] swapped with S[233]
S[11] swapped with S[23]
S[12] swapped with S[104]
S[13] swapped with S[220]
S[14] swapped with S[115]
S[15] swapped with S[131]
S[16] swapped with S[182]
S[17] swapped with S[12]
S[18] swapped with S[133]
S[19] swapped with S[33]
S[20] swapped with S[54]
S[21] swapped with S[110]
S[22] swapped with S[201]
S[23] swapped with S[59]
S[24] swapped with S[220]
S[25] swapped with S[246]
S[26] swapped with S[51]
S[27] swapped with S[147]
S[28] swapped with S[22]
S[29] swapped with S[188]
S[30] swapped with S[219]
S[31] swapped with S[29]
S[32] swapped with S[130]
```

```
S[226] swapped with S[155]
S[227] swapped with S[195]
S[228] swapped with S[14]
S[229] swapped with S[215]
S[230] swapped with S[190]
S[231] swapped with S[200]
S[232] swapped with S[85]
S[233] swapped with S[24]
S[234] swapped with S[139]
S[235] swapped with S[119]
S[236] swapped with S[134]
S[237] swapped with S[255]
S[238] swapped with S[147]
S[239] swapped with S[11]
S[240] swapped with S[252]
S[241] swapped with S[16]
S[242] swapped with S[255]
S[243] swapped with S[89]
S[244] swapped with S[237]
S[245] swapped with S[188]
S[246] swapped with S[248]
S[247] swapped with S[52]
S[248] swapped with S[180]
S[249] swapped with S[54]
S[250] swapped with S[49]
S[251] swapped with S[79]
S[252] swapped with S[132]
S[253] swapped with S[232]
S[254] swapped with S[145]
S[255] swapped with S[60]

Pseudo-Random Generation Algorithm (PRGA) steps (Decryption)
Step 1: i=1, j=36, Key Stream Byte=1b XOR with 27 = 0
Step 2: i=2, j=143, Key Stream Byte=35 XOR with 36 = 17
Step 3: i=3, j=100, Key Stream Byte=63 XOR with 65 = 34
Step 4: i=4, j=49, Key Stream Byte=2a XOR with 25 = 51
Decrypted Result: 00112233
```

OUTPUT:(AES Encryption)

```
=== Symmetric Encryption Tool ===
1. S-DES Encryption
2. S-DES Decryption
3. RC4 Encryption
4. RC4 Decryption
5. AES Encryption
6. Exit
Enter your choice (1-6): 5
Enter 128-bit input in hex (32 hex characters): 00112233445566778899AABBCCDDEEFF
Enter 128-bit key in hex (32 hex characters): 000102030405060708090A0B0C0D0E0F
Initial State:
00 44 88 cc
11 55 99 dd
22 66 aa ee
33 77 bb ff

After AddRoundKey:
00 45 8a cf
15 50 9f da
2a 6f a0 e5
3f 7a b5 f0


Round 1:
After SubBytes:
63 6e 7e 8a
59 53 db 57
e5 a8 e0 d9
75 da d5 8c

After ShiftRows:
63 6e 7e 8a
53 db 57 59
e0 d9 e5 a8
8c 75 da d5

After MixColumns:
5f 06 3a 99
72 c6 3e 0e
64 83 8d fc
15 5a 9f c5

After AddRoundKey:
89 ac 4e 64
a0 69 4c f4
be 25 f5 0d
c3 f1 e9 3b
```

```
Round 2:
After SubBytes:
a7 91 2f 43
e0 f9 29 bf
ae 3f e6 d7
2e a1 1e e2

After ShiftRows:
a7 91 2f 43
f9 29 bf e0
e6 d7 ae 3f
e2 2e a1 1e

After MixColumns:
41 bb 8b 9c
9d 8f 02 c7
b4 7f 2f ff
32 0a 39 26

After AddRoundKey:
f7 29 44 97
f9 b2 bf 36
0a e4 ea ff
5a 3a 8a d8


Round 3:
After SubBytes:
68 a5 1b 88
99 37 08 05
67 69 87 16
be 80 7e 61

After ShiftRows:
68 a5 1b 88
37 08 05 99
87 16 67 69
61 be 80 7e

After MixColumns:
6f e1 de ac
f5 31 38 64
e9 58 4b 41
ca 8d 54 8f

After AddRoundKey:
d9 1e aa e2
27 f3 f1 db
85 01 47 fe
ce e4 eb ce
```

```
Round 4:
After SubBytes:
35 72 ac 98
cc 0d a1 b9
97 7c a0 bb
8b 69 e9 8b

After ShiftRows:
35 72 ac 98
0d a1 b9 cc
a0 bb 97 7c
8b 8b 69 e9

After MixColumns:
56 2c 6d f1
5f 76 0e 76
e5 38 9b 8c
ff 81 13 ca

After AddRoundKey:
11 db 9a 4d
ca 43 30 75
1c 54 a9 30
02 84 9e 37


Round 5:
After SubBytes:
82 b9 b8 e3
74 1a 04 9d
9c 20 d3 04
77 5f 0b 9a

After ShiftRows:
82 b9 b8 e3
1a 04 9d 74
d3 04 9c 20
9a 77 5f 0b

After MixColumns:
78 16 14 6a
42 ca 79 60
90 2c e7 ca
7b 3e 6c 7c

After AddRoundKey:
44 bc b7 82
eb 55 e4 8b
c0 df 48 9d
d6 c8 4e d6
```

```
Round 6:
After SubBytes:
1b 65 a9 13
e9 fc 69 3d
ba 9e 52 5e
f6 e8 2f f6

After ShiftRows:
1b 65 a9 13
fc 69 3d e9
52 5e ba 9e
f6 f6 e8 2f

After MixColumns:
8d d9 5c b7
f8 a3 ee 4c
42 b1 d8 ac
74 6f ac 1c

After AddRoundKey:
d3 e0 53 ca
0f 05 7c da
e5 e4 e5 6d
7e cc b3 77


Round 7:
After SubBytes:
66 e1 ed 74
76 6b 10 57
d9 69 d9 3c
f3 4b 6d f5

After ShiftRows:
66 e1 ed 74
6b 10 57 76
d9 3c d9 69
f5 f3 4b 6d

After MixColumns:
5d 26 aa 76
35 76 78 4e
a0 87 ce 67
e9 e9 34 59

After AddRoundKey:
49 df da 6c
d6 29 9a c2
e4 8d 11 2a
a7 40 f4 7f
```

```
Round 8:
After SubBytes:
3b 9e 57 50
f6 a5 b8 25
69 5d 82 e5
5c 09 bf d2

After ShiftRows:
3b 9e 57 50
a5 b8 25 f6
82 e5 69 5d
d2 5c 09 bf

After MixColumns:
d2 4d a1 43
25 9d af ff
ec 13 bb c6
d5 5c a7 3e

After AddRoundKey:
95 0e 26 76
81 81 ca 46
0c 05 01 32
7b e3 dd ec


Round 9:
After SubBytes:
2a ab f7 38
0c 0c 74 5a
fe 6b 7c 23
21 11 c1 ce

After ShiftRows:
2a ab f7 38
0c 74 5a 0c
7c 23 fe 6b
ce 21 11 c1

After MixColumns:
f2 d3 f4 ce
78 07 4b 5c
97 fa 79 ba
89 f3 84 b6

After AddRoundKey:
bd 4a c6 1f
93 82 1c 34
9c 69 94 26
2c df 13 f8
```

```
After SubBytes:
7a d6 b4 c0
dc 13 9c 18
de f9 22 f7
71 9e 7d 41

After ShiftRows:
7a d6 b4 c0
13 9c 18 dc
22 f7 de f9
41 71 9e 7d

After AddRoundKey:
44 c7 a9 17
c6 08 52 63
fc f0 79 da
3a 5a ae 10

Ciphertext (hex): 44c6fc3ac708f05aa95279ae1763da10
```

All 10 rounds of encryption along with Key generation for AES-128.