Reg no: 22BCE3799

Apurba Koirala

Lab Assessment 4 Cryptography and Network Security

AES

a. Key Expansion

Code:

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <sstream>

using namespace std;

const uint8_t Rcon[10] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36 };

const uint8_t 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
};

vector<uint8_t> hexStringToBytes(const string &hex) {
    vector<uint8_t> bytes;
    for (size_t i = 0; i < hex.length(); i += 2) {
        string byteStr = hex.substr(i, 2);
        bytes.push_back(stoul(byteStr, nullptr, 16));
    }
    return bytes;
```

```
}

void keyExpansion(const vector<uint8_t> &key, vector<vector<uint8_t>> &roundKeys) {
    roundKeys.resize(11, vector<uint8_t>(16));
    for (int i = 0; i < 16; i++)
        roundKeys[0][i] = key[i];

    for (int round = 1; round <= 10; round++) {
        vector<uint8_t> temp = { roundKeys[round - 1][13], roundKeys[round - 1][14], roundKeys[round - 1][15],
roundKeys[round - 1][12] };

        for (int i = 0; i < 4; i++)
            temp[i] = sbox[temp[i]];

        temp[0] ^= Rcon[round - 1];

        for (int i = 0; i < 16; i++)
            roundKeys[round][i] = roundKeys[round - 1][i] ^ temp[i % 4];
    }
}

void printRoundKeys(const vector<vector<uint8_t>> &roundKeys) {
    for (int round = 0; round <= 10; round++) {
        cout << "Round " << round << " Key: ";
        for (uint8_t byte : roundKeys[round])
            cout << hex << setw(2) << setfill('0') << (int)byte << " ";
        cout << endl;
    }
}

int main() {
    string keyHex;
    cout << "Enter a 16-byte AES key (32 hex characters): ";
    cin >> keyHex;

    vector<uint8_t> key = hexStringToBytes(keyHex);
    vector<vector<uint8_t>> roundKeys;
```

```
    keyExpansion(key, roundKeys);
    printRoundKeys(roundKeys);

    return 0;
}
```

Output:

```
Enter a 16-byte AES key (32 hex characters): 3243f6a8885a308d313198a2e0370734
Round 0 Key: 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
Round 1 Key: a9 86 ee a8 13 9f 28 8d aa f4 80 a2 7b f2 1f 34
Round 2 Key: ab 46 f6 89 11 5f 30 ac a8 34 98 83 79 32 07 15
Round 3 Key: 8c 83 af 3f 36 9a 69 1a 8f f1 c1 35 5e f7 5e a3
Round 4 Key: 84 db af 67 3e c2 69 42 87 a9 c1 6d 56 af 5e fb
Round 5 Key: 94 83 af d6 2e 9a 69 f3 97 f1 c1 dc 46 f7 5e 4a
Round 6 Key: b4 db 79 8c 0e c2 bf a9 b7 a9 17 86 66 af 88 10
Round 7 Key: f4 db b3 bf 4e c2 75 9a f7 a9 dd b5 26 af 42 23
Round 8 Key: 74 f7 95 48 ce ee 53 6d 77 85 fb 42 a6 83 64 d4
Round 9 Key: 6f b4 95 48 d5 ad 53 6d 6c c6 fb 42 bd c0 64 d4
Round a Key: 59 f7 95 48 e3 ee 53 6d 5a 85 fb 42 8b 83 64 d4
```

b. Initial Transformation

Code:

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <sstream>

using namespace std;

vector<uint8_t> hexStringToBytes(const string &hex) {
    vector<uint8_t> bytes;
    for (size_t i = 0; i < hex.length(); i += 2) {
```

```
        string byteStr = hex.substr(i, 2);
        bytes.push_back(stoul(byteStr, nullptr, 16));
    }
    return bytes;
}

vector<vector<uint8_t>> createStateMatrix(const vector<uint8_t> &bytes) {
    vector<vector<uint8_t>> state(4, vector<uint8_t>(4));
    int index = 0;
    for (int col = 0; col < 4; col++)
        for (int row = 0; row < 4; row++)
            state[row][col] = bytes[index++];
    return state;
}

void addRoundKey(vector<vector<uint8_t>> &state, const vector<uint8_t> &roundKey) {
    for (int col = 0; col < 4; col++)
        for (int row = 0; row < 4; row++)
            state[row][col] ^= roundKey[row + col * 4];
}

void printState(const vector<vector<uint8_t>> &state) {
    for (const auto &row : state) {
        for (uint8_t byte : row)
            cout << hex << setw(2) << setfill('0') << (int)byte << " ";
        cout << endl;
    }
}

int main() {
    string plaintextHex, keyHex;
    cout << "Enter a 16-byte plaintext (32 hex characters): ";
    cin >> plaintextHex;
    cout << "Enter the first round key (from key expansion, 32 hex characters): ";
    cin >> keyHex;

    vector<uint8_t> plaintext = hexStringToBytes(plaintextHex);
```

```
    vector<uint8_t> key = hexStringToBytes(keyHex);

    vector<vector<uint8_t>> state = createStateMatrix(plaintext);
    addRoundKey(state, key);

    cout << "After Initial AddRoundKey:\n";
    printState(state);

    return 0;
}
```

Output:

```
Enter a 16-byte plaintext (32 hex characters): 3243f6a8885a308d313198a2e0370734
Enter the first round key (from key expansion, 32 hex characters): 0001020304050
60708090a0b0c0d0e0f
After Initial AddRoundKey:
32 8c 39 ec
42 5f 38 3a
f4 36 92 09
ab 8a a9 3b
```

c. Round Transformation

Code:

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <sstream>
#include <cstdint>

using namespace std;

typedef vector<vector<uint8_t>> Matrix;
```

```
const uint8_t S_BOX[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
};


Matrix hexToMatrix(const string &hex) {
    Matrix state(4, vector<uint8_t>(4));
    for (int i = 0; i < 16; i++) {
        string byteString = hex.substr(i * 2, 2);
        state[i % 4][i / 4] = stoi(byteString, nullptr, 16);
    }
    return state;
}


void printMatrix(const Matrix &state) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            cout << hex << setw(2) << setfill('0') << (int)state[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}


void subBytes(Matrix &state) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] = S_BOX[state[i][j]];
        }
```

```
    }
}


void shiftRows(Matrix &state) {
    for (int i = 1; i < 4; i++) {
        vector<uint8_t> tempRow = state[i];
        for (int j = 0; j < 4; j++) {
            state[i][j] = tempRow[(j + i) % 4];
        }
    }
}

uint8_t gmul(uint8_t a, uint8_t b) {
    uint8_t p = 0;
    while (b) {
        if (b & 1) p ^= a;
        a = (a << 1) ^ ((a & 0x80) ? 0x1B : 0);
        b >>= 1;
    }
    return p;
}


void mixColumns(Matrix &state) {
    const uint8_t mix[4][4] = {
        {2, 3, 1, 1},
        {1, 2, 3, 1},
        {1, 1, 2, 3},
        {3, 1, 1, 2}
    };

    Matrix temp(4, vector<uint8_t>(4));
    for (int c = 0; c < 4; c++) {
        for (int r = 0; r < 4; r++) {
            temp[r][c] = gmul(mix[r][0], state[0][c]) ^
                         gmul(mix[r][1], state[1][c]) ^
```

```
                    gmul(mix[r][2], state[2][c]) ^
                    gmul(mix[r][3], state[3][c]);
        }
    }
    state = temp;
}


void addRoundKey(Matrix &state, const Matrix &roundKey) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] ^= roundKey[i][j];
        }
    }
}


void aesRound(Matrix &state, const Matrix &roundKey) {
    subBytes(state);
    shiftRows(state);
    mixColumns(state);
    addRoundKey(state, roundKey);
}

int main() {
    string plaintextHex, keyHex;

    cout << "Enter 16-byte plaintext in hex (32 characters): ";
    cin >> plaintextHex;
    cout << "Enter 16-byte key in hex (32 characters): ";
    cin >> keyHex;

    if (plaintextHex.length() != 32 || keyHex.length() != 32) {
        cerr << "Invalid input! Must be exactly 32 hex characters (16 bytes)." << endl;
        return 1;
    }
```

```
    Matrix state = hexToMatrix(plaintextHex);
    Matrix roundKey = hexToMatrix(keyHex);

    cout << "\nInitial State:\n";
    printMatrix(state);

    aesRound(state, roundKey);

    cout << "State after AES round:\n";
    printMatrix(state);

    return 0;
}
```

Output:

```
Enter 16-byte plaintext in hex (32 characters): 3243f6a8885a308d313198a2e0370734
Enter 16-byte key in hex (32 characters): 000102030405060708090a0b0c0d0e0f

Initial State:
32 88 31 e0
43 5a 31 37
f6 30 98 07
a8 8d a2 34

State after AES round:
87 93 28 26
5d c4 e1 35
b7 50 57 1c
e8 05 c3 11
```