Assignment No. 6

PRN: 2020BTECS00025
Name: Sahil Santosh Otari
Course: CNS Lab
Batch: B2

Title: Data Encryption Algorithm

Aim: To study encryption and decryption using Data Encryption Algorithm.

Theory:

- AES is a block cipher.
- The key size can be 128/192/256 bits.
- Encrypts data in blocks of 128 bits each.

AES performs operations on bytes of data rather than in bits. Since the block size is 128 bits, the cipher processes 128 bits (or 16 bytes) of the input data at a time.
The number of rounds depends on the key length as follows :

128 bit key – 10 rounds
192 bit key – 12 rounds
256 bit key – 14 rounds

Creation of Round keys :
A Key Schedule algorithm is used to calculate all the round keys from the key. So the initial key is used to create many different round keys which will be used in the corresponding round of the encryption.

Each round comprises 4 steps :

1. SubBytes
2. ShiftRows
3. MixColumns
4. Add Round Key

SubBytes :
This step implements the substitution.

In this step each byte is substituted by another byte. Its performed using a lookup table also called the S-box. This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte which is a compliment of the current byte. The result of this step is a 16 byte (4 x 4) matrix like before.

ShiftRows :

This step is just as it sounds. Each row is shifted a particular number of times.

- The first row is not shifted
- The second row is shifted once to the left.
- The third row is shifted twice to the left.
- The fourth row is shifted thrice to the left.

MixColumns :

This step is basically a matrix multiplication. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result.

Add Round Keys :

Now the resultant output of the previous stage is XOR-ed with the corresponding round key. Here, the 16 bytes is not considered as a grid but just as 128 bits of data.

Decryption:

The stages of each round in decryption is as follows :

- Add round key
- Inverse MixColumns
- ShiftRows
- Inverse SubByte

Inverse MixColumns :

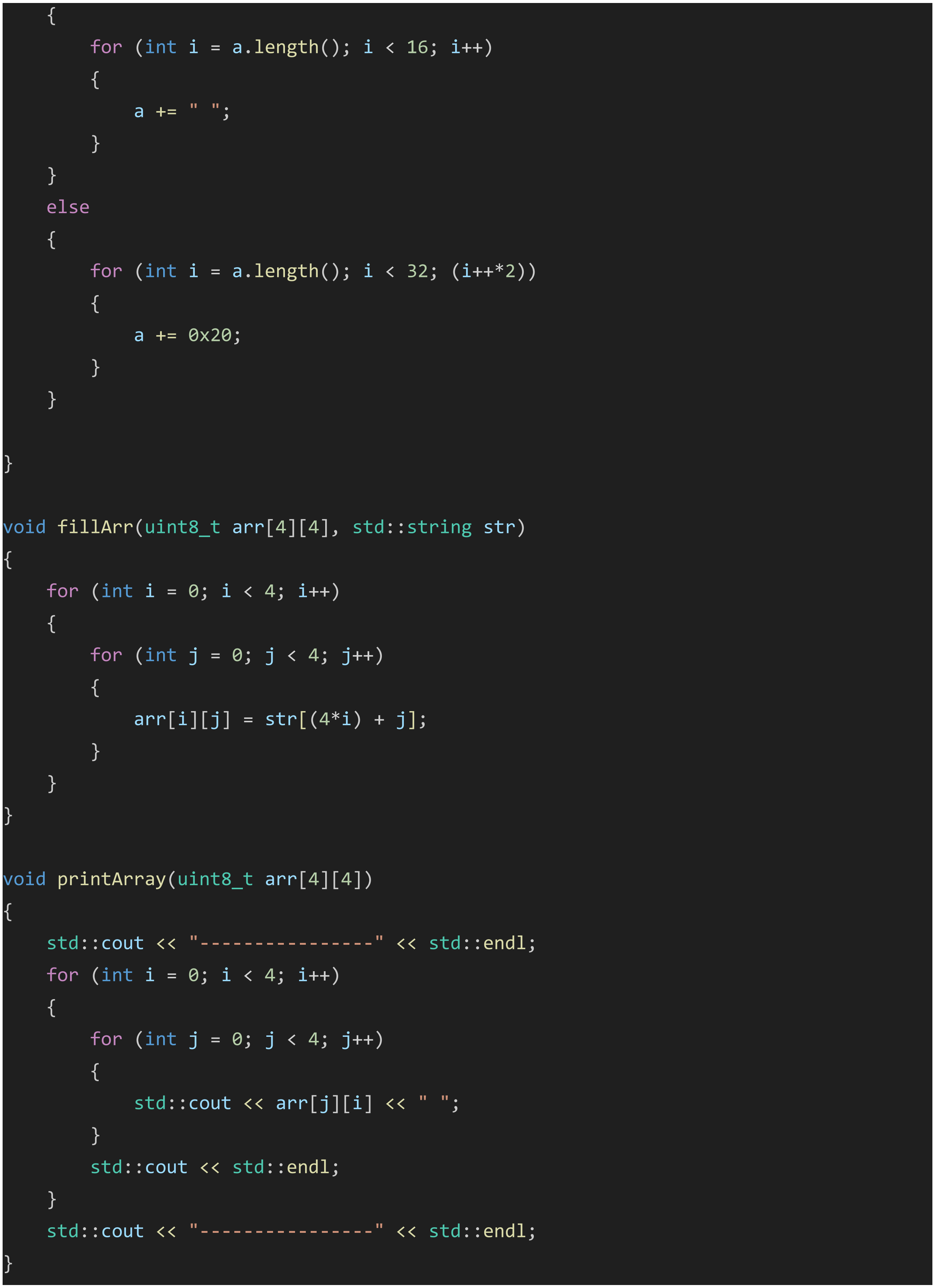
This step is similar to the MixColumns step in encryption, but differs in the matrix used to carry out the operation.

Inverse MixColumns :

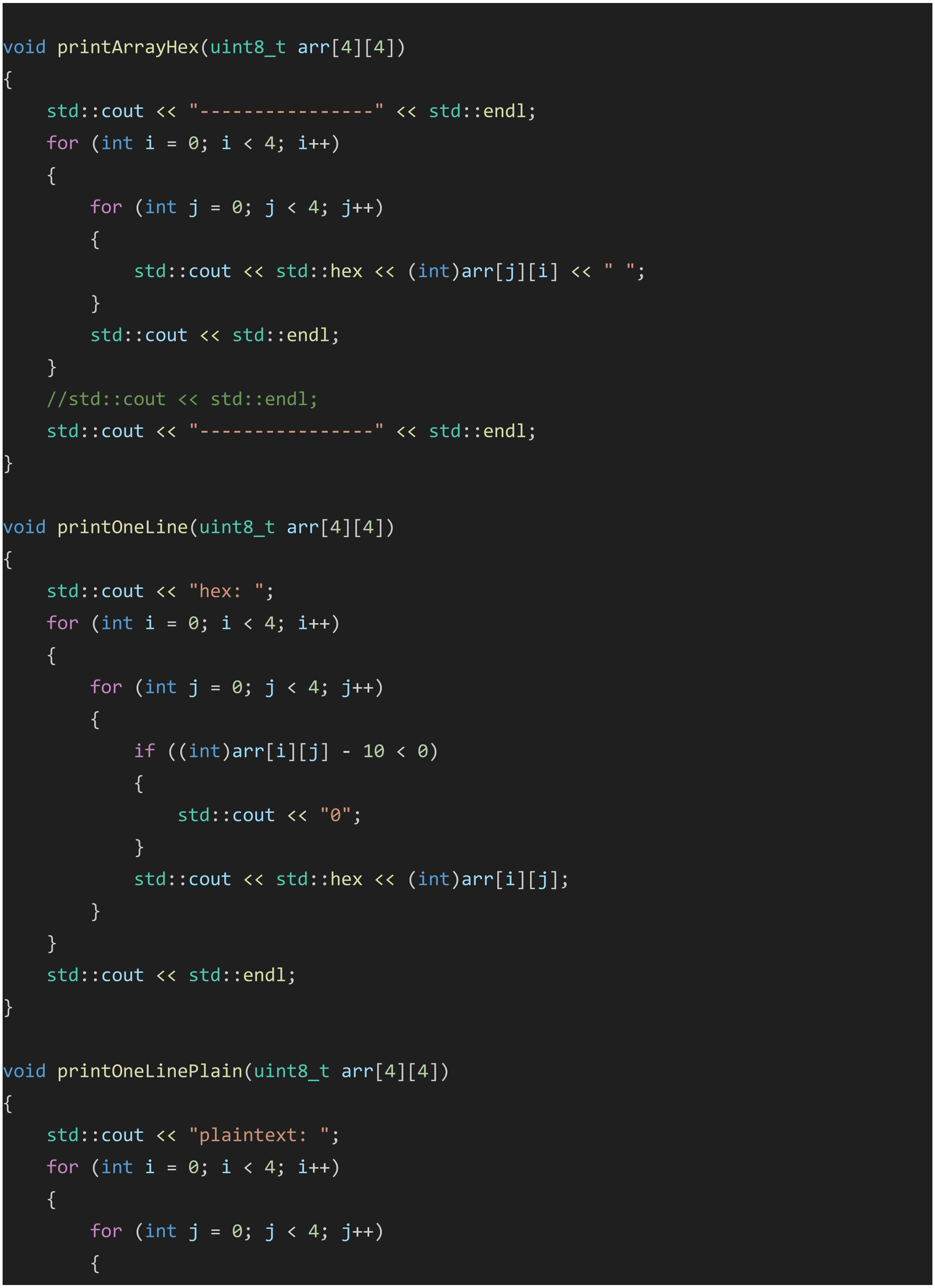
This step is similar to the MixColumns step in encryption, but differs in the matrix used to carry out the operation.

Code:

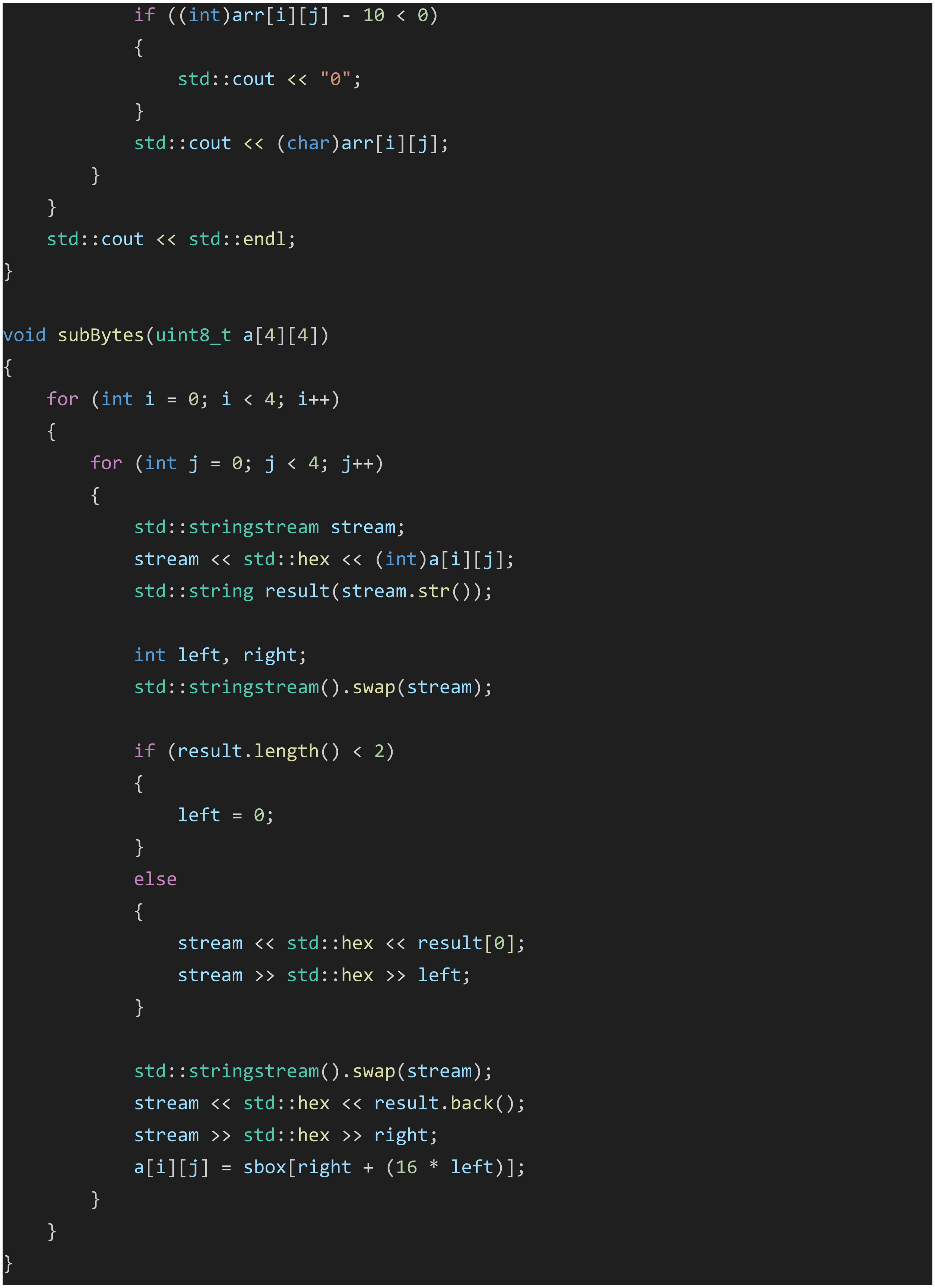
```
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <string.h>

static const uint8_t sbox[256] = {
         //0      1     2       3     4    5     6     7      8    9     A      B    C
D     E     F
        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 };

static const uint8_t rsbox[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 };

void fillString(std::string &a, bool isPlaintext)
{
    if (isPlaintext)
```



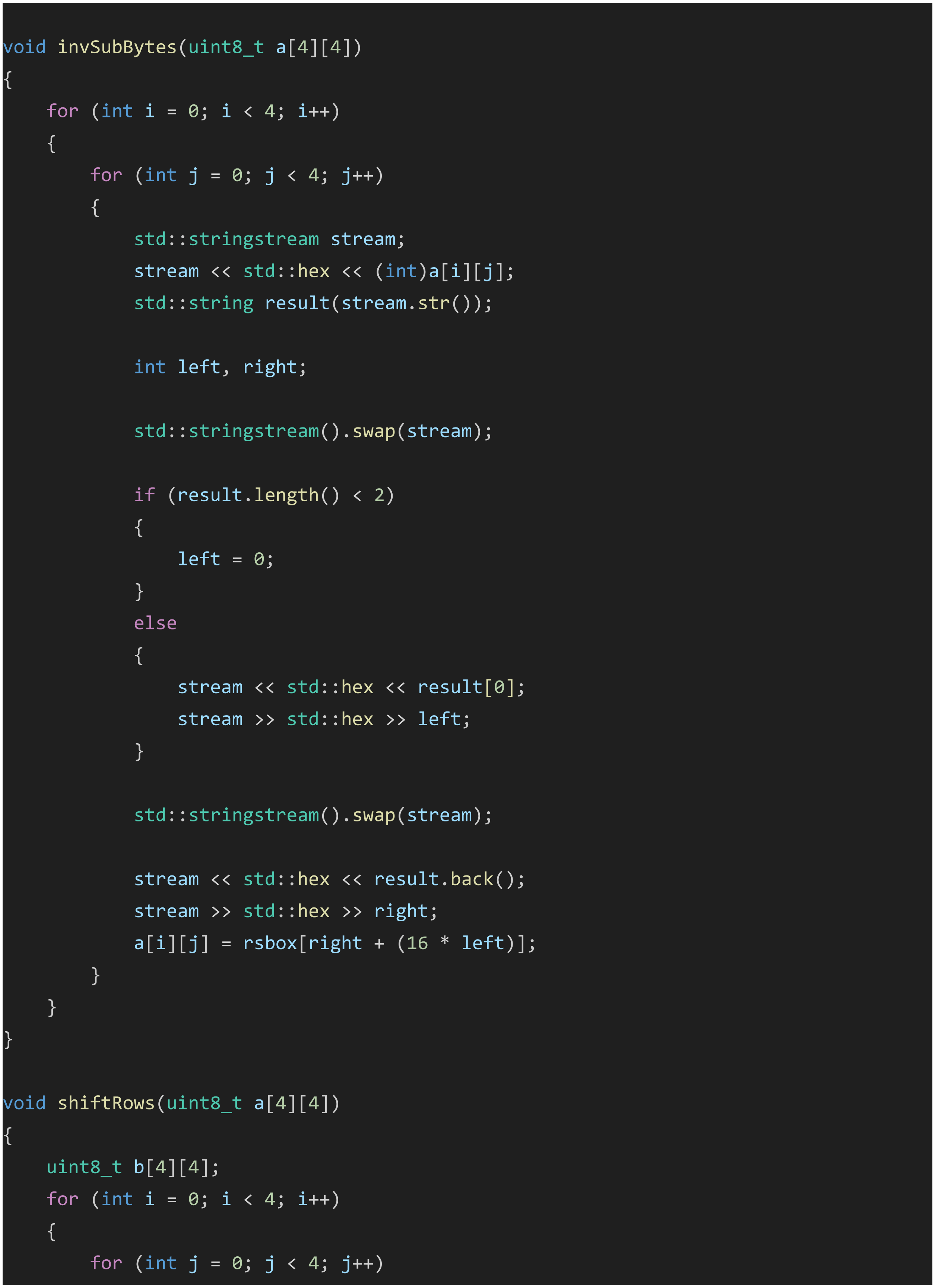
```
    {
        for (int i = a.length(); i < 16; i++)
        {
            a += " ";
        }
    }
    else
    {
        for (int i = a.length(); i < 32; (i++*2))
        {
            a += 0x20;
        }
    }

}

void fillArr(uint8_t arr[4][4], std::string str)
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            arr[i][j] = str[(4*i) + j];
        }
    }
}

void printArray(uint8_t arr[4][4])
{
    std::cout << "----------------" << std::endl;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            std::cout << arr[j][i] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << "----------------" << std::endl;
}
```



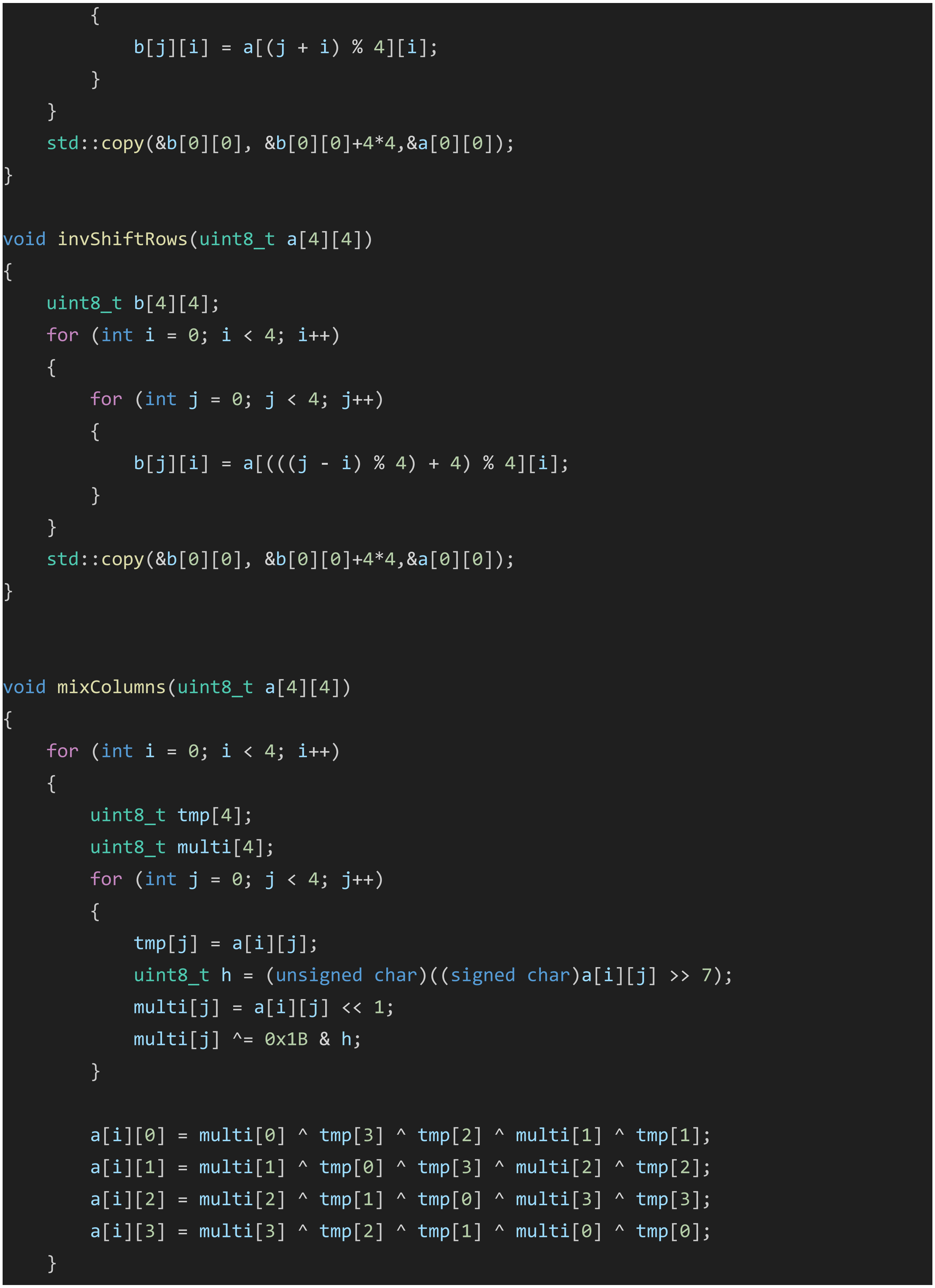
```
void printArrayHex(uint8_t arr[4][4])
{
    std::cout << "----------------" << std::endl;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            std::cout << std::hex << (int)arr[j][i] << " ";
        }
        std::cout << std::endl;
    }
    //std::cout << std::endl;
    std::cout << "----------------" << std::endl;
}

void printOneLine(uint8_t arr[4][4])
{
    std::cout << "hex: ";
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if ((int)arr[i][j] - 10 < 0)
            {
                std::cout << "0";
            }
            std::cout << std::hex << (int)arr[i][j];
        }
    }
    std::cout << std::endl;
}

void printOneLinePlain(uint8_t arr[4][4])
{
    std::cout << "plaintext: ";
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
```



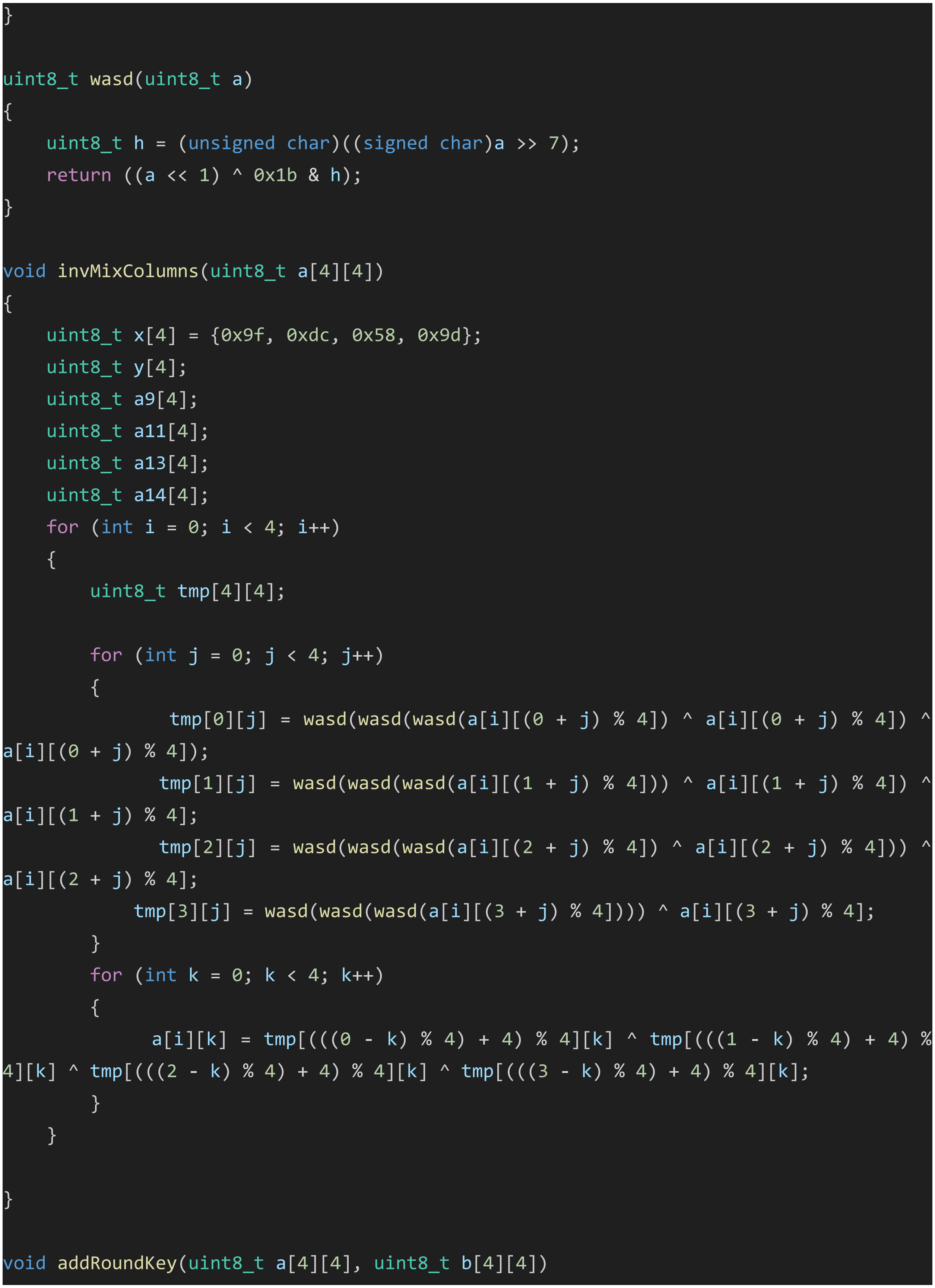
```
            if ((int)arr[i][j] - 10 < 0)
            {
                std::cout << "0";
            }
            std::cout << (char)arr[i][j];
        }
    }
    std::cout << std::endl;
}

void subBytes(uint8_t a[4][4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            std::stringstream stream;
            stream << std::hex << (int)a[i][j];
            std::string result(stream.str());

            int left, right;
            std::stringstream().swap(stream);

            if (result.length() < 2)
            {
                left = 0;
            }
            else
            {
                stream << std::hex << result[0];
                stream >> std::hex >> left;
            }

            std::stringstream().swap(stream);
            stream << std::hex << result.back();
            stream >> std::hex >> right;
            a[i][j] = sbox[right + (16 * left)];
        }
    }
}
```



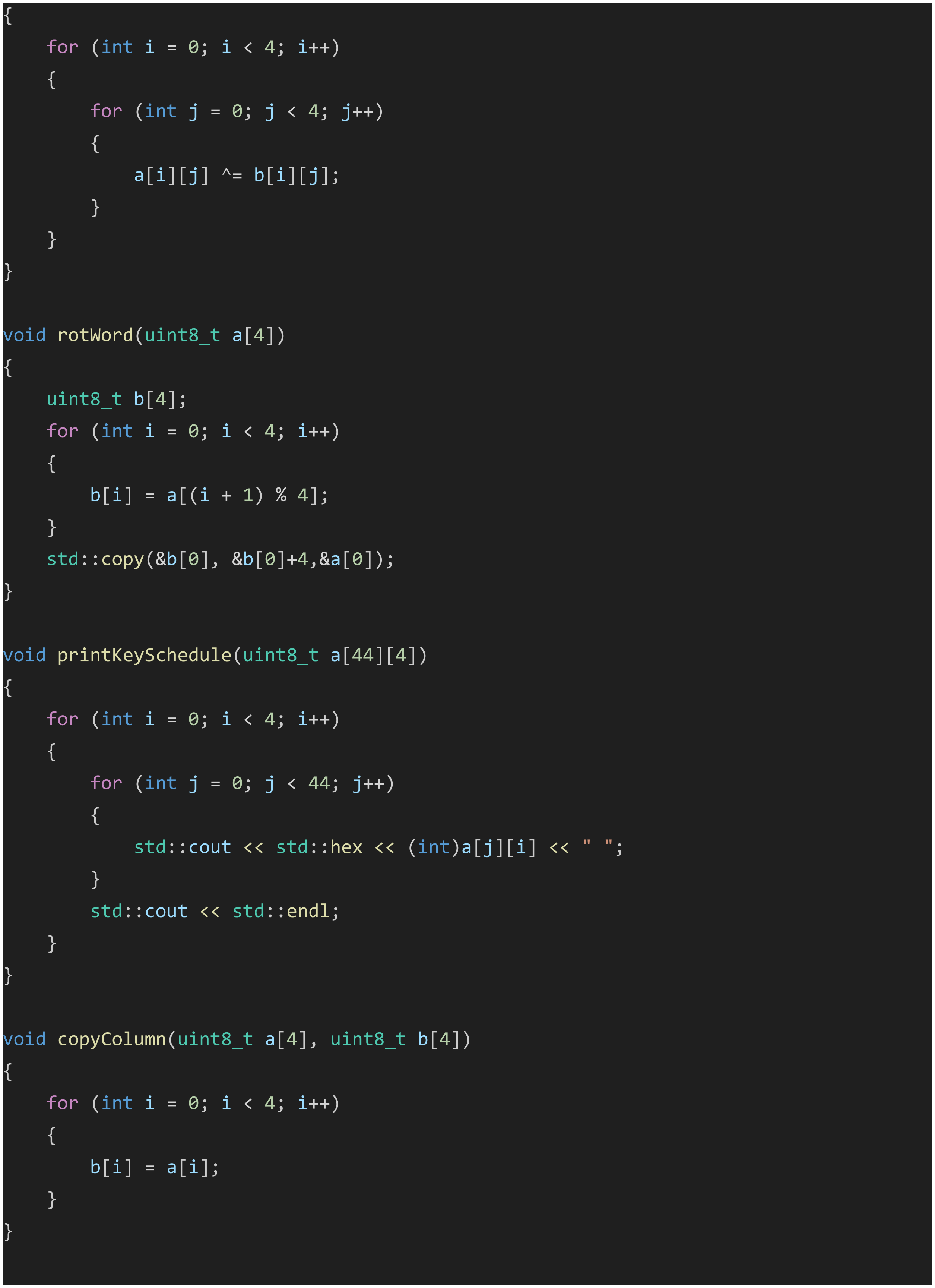
```
void invSubBytes(uint8_t a[4][4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            std::stringstream stream;
            stream << std::hex << (int)a[i][j];
            std::string result(stream.str());

            int left, right;

            std::stringstream().swap(stream);

            if (result.length() < 2)
            {
                left = 0;
            }
            else
            {
                stream << std::hex << result[0];
                stream >> std::hex >> left;
            }

            std::stringstream().swap(stream);

            stream << std::hex << result.back();
            stream >> std::hex >> right;
            a[i][j] = rsbox[right + (16 * left)];
        }
    }
}

void shiftRows(uint8_t a[4][4])
{
    uint8_t b[4][4];
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
```



```
        {
            b[j][i] = a[(j + i) % 4][i];
        }
    }
    std::copy(&b[0][0], &b[0][0]+4*4,&a[0][0]);
}

void invShiftRows(uint8_t a[4][4])
{
    uint8_t b[4][4];
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            b[j][i] = a[(((j - i) % 4) + 4) % 4][i];
        }
    }
    std::copy(&b[0][0], &b[0][0]+4*4,&a[0][0]);
}


void mixColumns(uint8_t a[4][4])
{
    for (int i = 0; i < 4; i++)
    {
        uint8_t tmp[4];
        uint8_t multi[4];
        for (int j = 0; j < 4; j++)
        {
            tmp[j] = a[i][j];
            uint8_t h = (unsigned char)((signed char)a[i][j] >> 7);
            multi[j] = a[i][j] << 1;
            multi[j] ^= 0x1B & h;
        }

        a[i][0] = multi[0] ^ tmp[3] ^ tmp[2] ^ multi[1] ^ tmp[1];
        a[i][1] = multi[1] ^ tmp[0] ^ tmp[3] ^ multi[2] ^ tmp[2];
        a[i][2] = multi[2] ^ tmp[1] ^ tmp[0] ^ multi[3] ^ tmp[3];
        a[i][3] = multi[3] ^ tmp[2] ^ tmp[1] ^ multi[0] ^ tmp[0];
    }
```



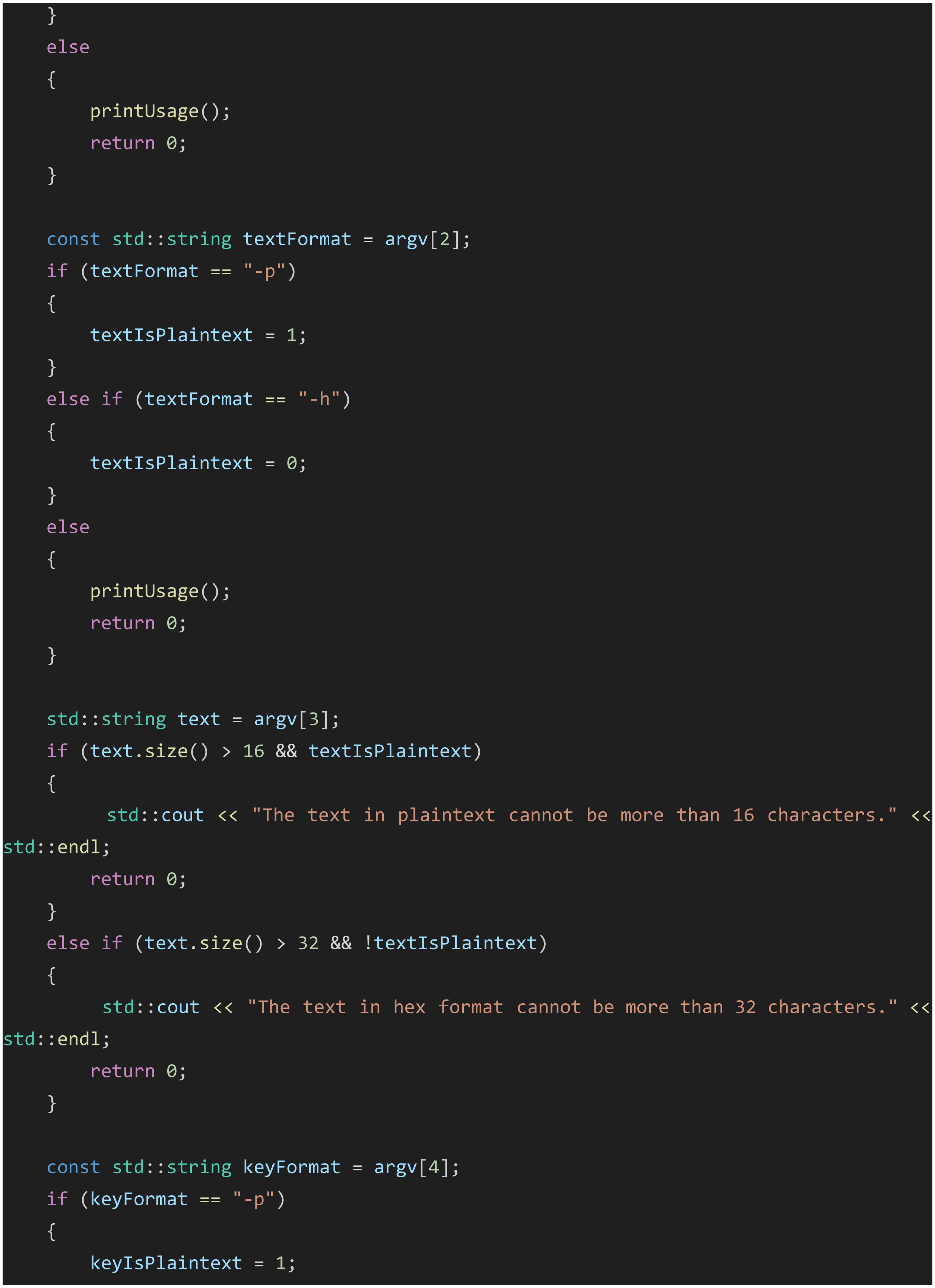
```
}

uint8_t wasd(uint8_t a)
{
    uint8_t h = (unsigned char)((signed char)a >> 7);
    return ((a << 1) ^ 0x1b & h);
}

void invMixColumns(uint8_t a[4][4])
{
    uint8_t x[4] = {0x9f, 0xdc, 0x58, 0x9d};
    uint8_t y[4];
    uint8_t a9[4];
    uint8_t a11[4];
    uint8_t a13[4];
    uint8_t a14[4];
    for (int i = 0; i < 4; i++)
    {
        uint8_t tmp[4][4];

        for (int j = 0; j < 4; j++)
        {
               tmp[0][j] = wasd(wasd(wasd(a[i][(0 + j) % 4]) ^ a[i][(0 + j) % 4]) ^
a[i][(0 + j) % 4]);
              tmp[1][j] = wasd(wasd(wasd(a[i][(1 + j) % 4])) ^ a[i][(1 + j) % 4]) ^
a[i][(1 + j) % 4];
              tmp[2][j] = wasd(wasd(wasd(a[i][(2 + j) % 4]) ^ a[i][(2 + j) % 4])) ^
a[i][(2 + j) % 4];
            tmp[3][j] = wasd(wasd(wasd(a[i][(3 + j) % 4]))) ^ a[i][(3 + j) % 4];
        }
        for (int k = 0; k < 4; k++)
        {
             a[i][k] = tmp[(((0 - k) % 4) + 4) % 4][k] ^ tmp[(((1 - k) % 4) + 4) %
4][k] ^ tmp[(((2 - k) % 4) + 4) % 4][k] ^ tmp[(((3 - k) % 4) + 4) % 4][k];
        }
    }

}

void addRoundKey(uint8_t a[4][4], uint8_t b[4][4])
```



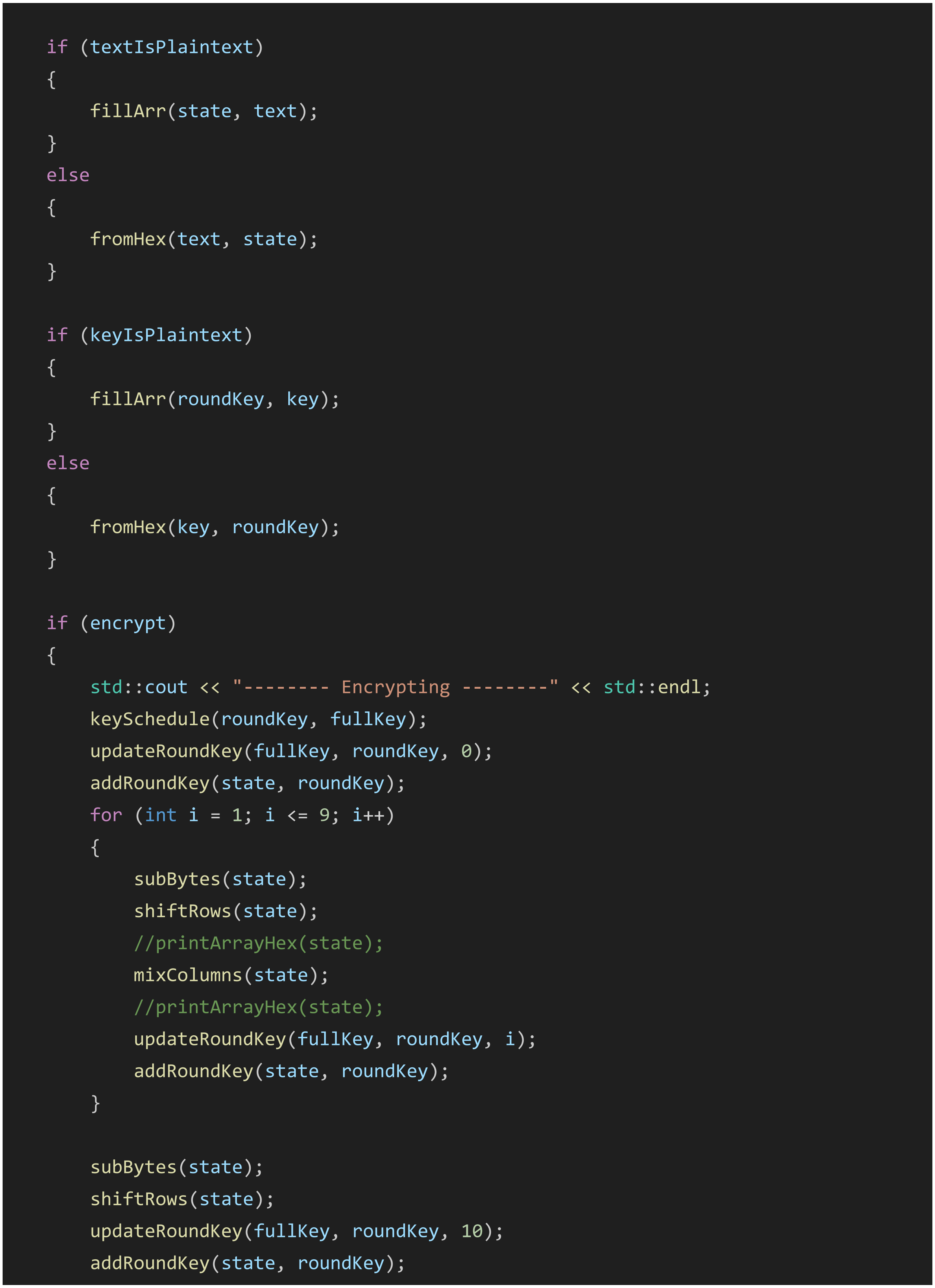
```
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            a[i][j] ^= b[i][j];
        }
    }
}

void rotWord(uint8_t a[4])
{
    uint8_t b[4];
    for (int i = 0; i < 4; i++)
    {
        b[i] = a[(i + 1) % 4];
    }
    std::copy(&b[0], &b[0]+4,&a[0]);
}

void printKeySchedule(uint8_t a[44][4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 44; j++)
        {
            std::cout << std::hex << (int)a[j][i] << " ";
        }
        std::cout << std::endl;
    }
}

void copyColumn(uint8_t a[4], uint8_t b[4])
{
    for (int i = 0; i < 4; i++)
    {
        b[i] = a[i];
    }
}
```

```cpp
void subBytesRow(uint8_t a[4])
{
    for (int i = 0; i < 4; i++)
    {
        std::stringstream stream;
        stream << std::hex << (int)a[i];
        std::string result(stream.str());

        int left, right;

        std::stringstream().swap(stream);

        if (result.length() < 2)
        {
            left = 0;
        }
        else
        {
            stream << std::hex << result[0];
            stream >> std::hex >> left;
        }

        std::stringstream().swap(stream);

        stream << std::hex << result.back();
        stream >> std::hex >> right;
        a[i] = sbox[right + (16 * left)];
    }
}



void keySchedule(uint8_t cipherKey[4][4], uint8_t ok[44][4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            ok[i][j] = cipherKey[i][j];
        }
    }

    static const uint8_t rcon[10] = { 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36};

    for (int i = 4; i < 44; i++)
    {
        uint8_t tmpRcon[4] = {0, 0, 0, 0};
        uint8_t tmp[4];
        copyColumn(ok[i - 1], tmp);


        if (i % 4 == 0)
        {
            tmpRcon[0] = rcon[(i/4) - 1];
            rotWord(tmp);
            subBytesRow(tmp);
        }

        for (int j = 0; j < 4; j++)
        {
             ok[i][j] = (i % 4 == 0) ? (ok[i - 4][j] ^ tmp[j] ^ tmpRcon[j]) : (ok[i - 4][j] ^ tmp[j]);
        }
    }
}

void updateRoundKey(uint8_t a[44][4], uint8_t b[4][4], unsigned int round)
{
    if (round > 10)
    {
        std::cout << "The round cannot be larger than 10" << std::endl;
        exit(1);
    }
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j <4; j++)
        {
            b[i][j] = a[i + (4*round)][j];
        }
```

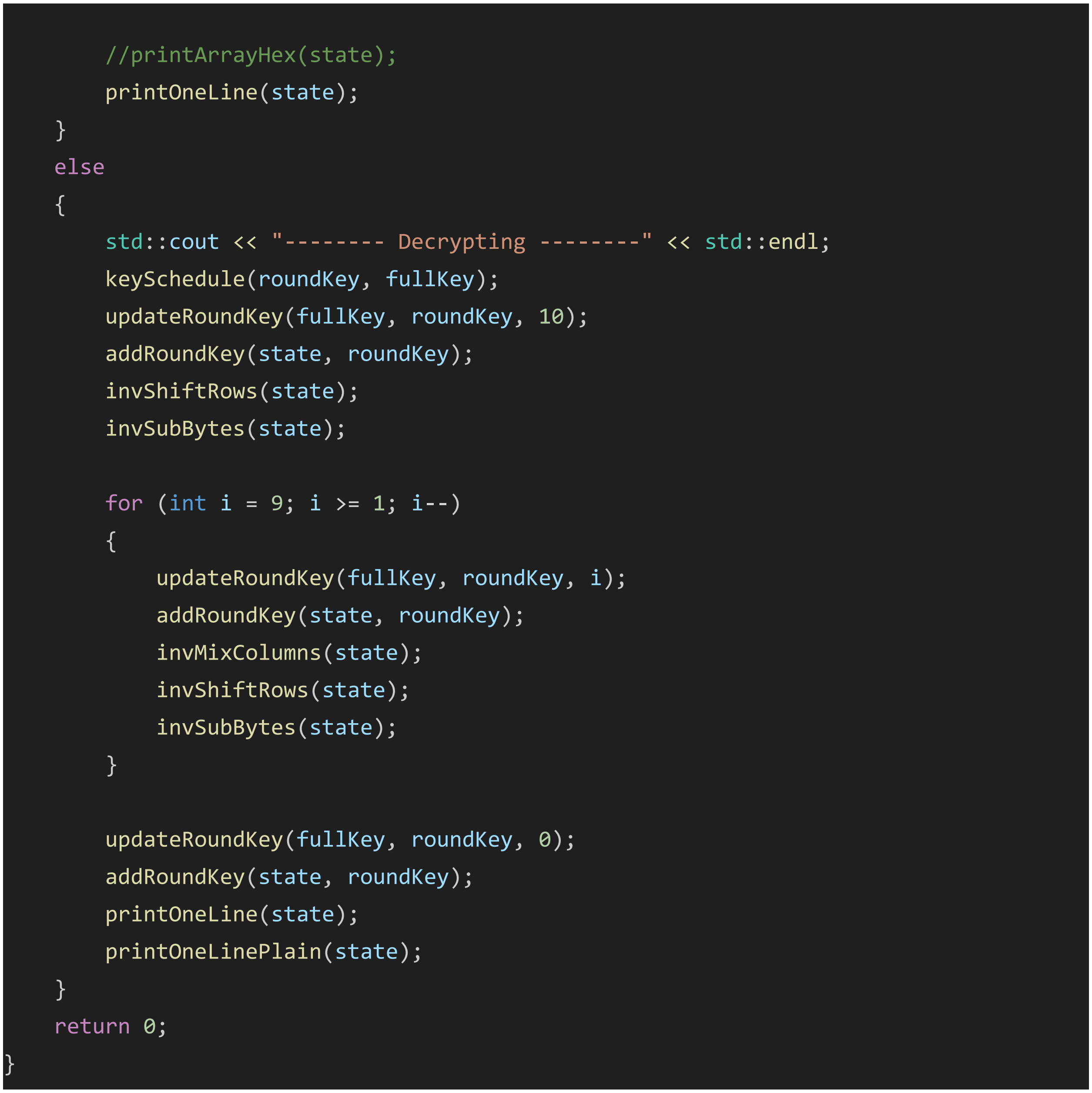
```cpp
    }
}

void fromHex(std::string str, uint8_t ret[4][4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            ret[i][j] = std::stoi(str.substr((2*j)+(8*i), 2), 0, 16);
        }
    }
}

void printUsage()
{
       std::cout << "Usage: ./aes encrypt/decrypt -p/-h <text> -p/-h <key>" <<
std::endl;
}

int main(int argc, char** argv)
{
    bool encrypt = -1;
    bool textIsPlaintext = -1;
    bool keyIsPlaintext = -1;

    if (argc != 6)
    {
        printUsage();
        return 0;
    }

    const std::string needsAName = argv[1];
    if (needsAName == "encrypt")
    {
        encrypt = 1;
    }
    else if (needsAName == "decrypt")
    {
        encrypt = 0;
```



```
    }
    else
    {
        printUsage();
        return 0;
    }

    const std::string textFormat = argv[2];
    if (textFormat == "-p")
    {
        textIsPlaintext = 1;
    }
    else if (textFormat == "-h")
    {
        textIsPlaintext = 0;
    }
    else
    {
        printUsage();
        return 0;
    }

    std::string text = argv[3];
    if (text.size() > 16 && textIsPlaintext)
    {
         std::cout << "The text in plaintext cannot be more than 16 characters." <<
std::endl;
        return 0;
    }
    else if (text.size() > 32 && !textIsPlaintext)
    {
         std::cout << "The text in hex format cannot be more than 32 characters." <<
std::endl;
        return 0;
    }

    const std::string keyFormat = argv[4];
    if (keyFormat == "-p")
    {
        keyIsPlaintext = 1;
    }
    else if (keyFormat == "-h")
    {
        keyIsPlaintext = 0;
    }
    else
    {
        printUsage();
        return 0;
    }

    std::string key = argv[5];
    if (key.size() > 16 && keyIsPlaintext)
    {
          std::cout << "The key in plaintext cannot be more than 16 characters." <<
std::endl;
        return 0;
    }
    else if (key.size() > 32 && !keyIsPlaintext)
    {
         std::cout << "The key in hex format cannot be more than 32 characters." <<
std::endl;
        return 0;
    }

    if (text.size() < 16 && textIsPlaintext)
    {
        fillString(text, textIsPlaintext);
    }
    else if (text.size() < 32 && !textIsPlaintext)
    {
        fillString(text, textIsPlaintext);
    }

    std::cout << "Text: " << text << std::endl;
    std::cout << "Key:  " << key << std::endl;

    uint8_t fullKey[44][4];
    uint8_t state[4][4];
    uint8_t roundKey[4][4];
```

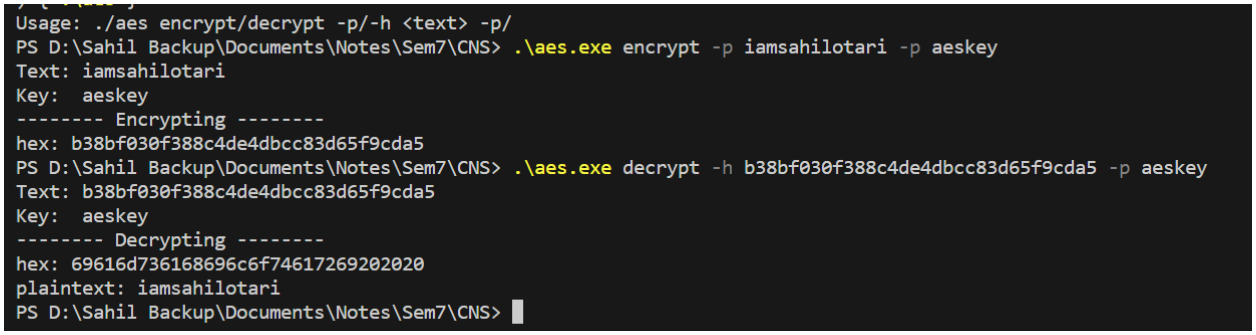


```
if (textIsPlaintext)
{
    fillArr(state, text);
}
else
{
    fromHex(text, state);
}

if (keyIsPlaintext)
{
    fillArr(roundKey, key);
}
else
{
    fromHex(key, roundKey);
}

if (encrypt)
{
    std::cout << "-------- Encrypting --------" << std::endl;
    keySchedule(roundKey, fullKey);
    updateRoundKey(fullKey, roundKey, 0);
    addRoundKey(state, roundKey);
    for (int i = 1; i <= 9; i++)
    {
        subBytes(state);
        shiftRows(state);
        //printArrayHex(state);
        mixColumns(state);
        //printArrayHex(state);
        updateRoundKey(fullKey, roundKey, i);
        addRoundKey(state, roundKey);
    }

    subBytes(state);
    shiftRows(state);
    updateRoundKey(fullKey, roundKey, 10);
    addRoundKey(state, roundKey);
```



```
        //printArrayHex(state);
        printOneLine(state);
    }
    else
    {
        std::cout << "-------- Decrypting --------" << std::endl;
        keySchedule(roundKey, fullKey);
        updateRoundKey(fullKey, roundKey, 10);
        addRoundKey(state, roundKey);
        invShiftRows(state);
        invSubBytes(state);

        for (int i = 9; i >= 1; i--)
        {
            updateRoundKey(fullKey, roundKey, i);
            addRoundKey(state, roundKey);
            invMixColumns(state);
            invShiftRows(state);
            invSubBytes(state);
        }

        updateRoundKey(fullKey, roundKey, 0);
        addRoundKey(state, roundKey);
        printOneLine(state);
        printOneLinePlain(state);
    }
    return 0;
}
```

Output:



```
Usage: ./aes encrypt/decrypt -p/-h <text> -p/
PS D:\Sahil Backup\Documents\Notes\Sem7\CNS> .\aes.exe encrypt -p iamsahilotari -p aeskey
Text: iamsahilotari
Key:  aeskey
-------- Encrypting --------
hex: b38bf030f388c4de4dbcc83d65f9cda5
PS D:\Sahil Backup\Documents\Notes\Sem7\CNS> .\aes.exe decrypt -h b38bf030f388c4de4dbcc83d65f9cda5 -p aeskey
Text: b38bf030f388c4de4dbcc83d65f9cda5
Key:  aeskey
-------- Decrypting --------
hex: 69616d736168696c6f74617269202020
plaintext: iamsahilotari
PS D:\Sahil Backup\Documents\Notes\Sem7\CNS> █
```

Advantages:

- It uses higher length key sizes such as 128, 192 and 256 bits for encryption. Hence, it makes AES algorithm more robust against hacking.
- It is the most common security protocol used for wide variety of applications such as wireless communication, financial transactions, e-business, encrypted data storage etc.
- It is one of the most widely used commercial and open source solutions across the world.
- For 128 bit, about 2128 attempts are needed to break. This makes it very difficult to hack it, as a result it is a very safe protocol.

Limitations:

- Every block is always encrypted in the same way.
- Hard to implement with software.
- AES in counter mode is complex to implement in software, taking both performance and security into considerations.