CRYPTOGRAPHY LABORATORY FILE
CS-511

Submitted to:
Dr. Samayveer Singh
Assistant Professor
Department of Computer Science

Submitted by:
Shashi Shekhar Azad
Roll No.: 23203029
M. Tech. CSE 1st Semester

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DR. B. R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR

Assignment 5

Write a program to implement the encryption and decryption process of Advanced Encryption Standard (AES).

Code:

```
#include <stdio.h>
#include <stdlib.h>
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};

unsigned char revSBox[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};

unsigned char Rcon[255] = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
    0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
    0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
    0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d,
    0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab,
    0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,
    0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25,
    0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,
    0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
    0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,
    0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a,
    0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,
    0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
    0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
    0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
    0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
    0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f,
    0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5,
    0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,
    0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb};

/// Funcetion declaration
void rotate(unsigned char *word);
void core(unsigned char *word, int iteration);
void expandKey(unsigned char *expandedKey, unsigned char *key, int size, size_t
expandedKeySize);
```

```
void subBytes(unsigned char *state);
void shiftRows(unsigned char *state);
void shiftRow(unsigned char *state, unsigned char nbr);
void addRoundKey(unsigned char *state, unsigned char *roundKey);
unsigned char galoisMultiplication(unsigned char a, unsigned char b);
void mixColumns(unsigned char *state);
void mixColumn(unsigned char *column);
void aesRound(unsigned char *state, unsigned char *roundKey);
void createRoundKey(unsigned char *expandedKey, unsigned char *roundKey);
void aesProcess(unsigned char *state, unsigned char *expandedKey, int roundNum);
int aesEncryption(unsigned char *input, unsigned char *output, unsigned char *key, int size);
void reverseSubBytes(unsigned char *state);
void reverseShiftRows(unsigned char *state);
void reverseShiftRow(unsigned char *state, unsigned char nbr);
void reverseMixColumns(unsigned char *state);
void reverseMixColumn(unsigned char *column);
void reverseAesRound(unsigned char *state, unsigned char *roundKey);
void reverseAesProcess(unsigned char *state, unsigned char *expandedKey, int roundNum);
int aesDecryption(unsigned char *input, unsigned char *output, unsigned char *key, int size);

int main()
{
    int size = 16, i;
    int expandedKeySize = 176;
    unsigned char key[16];
    unsigned char plainText[16];
    unsigned char ciphertext[16];
    unsigned char decryptedtext[16];
    unsigned char expandedKey[expandedKeySize];
    printf("\nEnter 16-byte(128-bit) key: ");
    for (i = 0; i < 16; i++)
    {
        scanf("%c", &key[i]);
    }
```

```
    printf("\nEnter 16-byte(128-bit) plain text: ");
    for (i = 0; i < 16; i++)
    {
        scanf(" %c", &plainText[i]);
    }
    expandKey(expandedKey, key, size, expandedKeySize);
    printf("\nExpanded Key (hexadecimal):\n");
    for (i = 0; i < expandedKeySize; i++)
    {
        printf("%2.2x%c", expandedKey[i], ((i + 1) % 16) ? ' ' : '\n');
    }

    printf("\nPlain Text: ");
    for (i = 0; i < 16; i++)
    {
        printf("%c", plainText[i]);
    }
    aesEncryption(plainText, ciphertext, key, size);
    printf("\n\nEnciphered Text : ");
    for (i = 0; i < 16; i++)
    {
        printf("%c", ciphertext[i]);
    }
    aesDecryption(ciphertext, decryptedtext, key, size);
    printf("\n\nDeciphered Text: ");
    for (i = 0; i < 16; i++)
    {
        printf("%c", decryptedtext[i]);
    }
    return 0;
}
void rotate(unsigned char *word)
{
    unsigned char c;
    int i;
```

```
    c = word[0];
    for (i = 0; i < 3; i++)
        word[i] = word[i + 1];
    word[3] = c;
}
void core(unsigned char *word, int iteration)
{
    int i;
    rotate(word);
    for (i = 0; i < 4; ++i)
    {
        word[i] = sBox[word[i]];
    }
    word[0] = word[0] ^ Rcon[iteration];
}
void expandKey(unsigned char *expandedKey, unsigned char *key, int size, size_t
expandedKeySize)
{
    int currentSize = 0;
    int rconIteration = 1;
    int i;
    unsigned char t[4] = {0};
    for (i = 0; i < size; i++)
        expandedKey[i] = key[i];
    currentSize += size;
    while (currentSize < expandedKeySize)
    {
        for (i = 0; i < 4; i++)
        {
            t[i] = expandedKey[(currentSize - 4) + i];
        }
        if (currentSize % size == 0)
        {
            core(t, rconIteration++);
        }
```

```
        for (i = 0; i < 4; i++)
        {
            expandedKey[currentSize] = expandedKey[currentSize - size] ^ t[i];
            currentSize++;
        }
    }
}

void subBytes(unsigned char *state)
{
    int i;
    for (i = 0; i < 16; i++)
        state[i] = sBox[state[i]];
}
void shiftRows(unsigned char *state)
{
    int i;
    for (i = 0; i < 4; i++)
        shiftRow(state + i * 4, i);
}
void shiftRow(unsigned char *state, unsigned char nbr)
{
    int i, j;
    unsigned char temp;
    for (i = 0; i < nbr; i++)
    {
        temp = state[0];
        for (j = 0; j < 3; j++)
            state[j] = state[j + 1];
        state[3] = temp;
    }
}
void addRoundKey(unsigned char *state, unsigned char *roundKey)
{
    int i;
```

```
    for (i = 0; i < 16; i++)
        state[i] = state[i] ^ roundKey[i];
}

unsigned char galoisMultiplication(unsigned char a, unsigned char b)
{
    unsigned char p = 0;
    unsigned char counter;
    unsigned char hi_bit_set;
    for (counter = 0; counter < 8; counter++)
    {
        if ((b & 1) == 1)
            p ^= a;
        hi_bit_set = (a & 0x80);
        a <<= 1;
        if (hi_bit_set == 0x80)
            a ^= 0x1b;
        b >>= 1;
    }
    return p;
}
void mixColumns(unsigned char *state)
{
    int i, j;
    unsigned char column[4];
    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            column[j] = state[(j * 4) + i];
        }
        mixColumn(column);
        for (j = 0; j < 4; j++)
        {
            state[(j * 4) + i] = column[j];
```

```
        }
    }
}
void mixColumn(unsigned char *column)
{
    unsigned char dupColumn[4];
    int i;
    for (i = 0; i < 4; i++)
    {
        dupColumn[i] = column[i];
    }
    column[0] = galoisMultiplication(dupColumn[0], 2) ^
                galoisMultiplication(dupColumn[3], 1) ^
                galoisMultiplication(dupColumn[2], 1) ^
                galoisMultiplication(dupColumn[1], 3);

    column[1] = galoisMultiplication(dupColumn[1], 2) ^
                galoisMultiplication(dupColumn[0], 1) ^
                galoisMultiplication(dupColumn[3], 1) ^
                galoisMultiplication(dupColumn[2], 3);
    column[2] = galoisMultiplication(dupColumn[2], 2) ^
                galoisMultiplication(dupColumn[1], 1) ^
                galoisMultiplication(dupColumn[0], 1) ^
                galoisMultiplication(dupColumn[3], 3);
    column[3] = galoisMultiplication(dupColumn[3], 2) ^
                galoisMultiplication(dupColumn[2], 1) ^
                galoisMultiplication(dupColumn[1], 1) ^
                galoisMultiplication(dupColumn[0], 3);
}
void aesRound(unsigned char *state, unsigned char *roundKey)
{
    subBytes(state);
    shiftRows(state);
    mixColumns(state);
    addRoundKey(state, roundKey);
```

```
}
void createRoundKey(unsigned char *expandedKey, unsigned char *roundKey)
{
    int i, j;
    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
            roundKey[(i + (j * 4))] = expandedKey[(i * 4) + j];
    }
}
void aesProcess(unsigned char *state, unsigned char *expandedKey, int roundNum)
{
    int i = 0;
    unsigned char roundKey[16];
    createRoundKey(expandedKey, roundKey);
    addRoundKey(state, roundKey);
    for (i = 1; i < roundNum; i++)
    {
        createRoundKey(expandedKey + 16 * i, roundKey);
        aesRound(state, roundKey);
    }
    createRoundKey(expandedKey + 16 * roundNum, roundKey);
    subBytes(state);
    shiftRows(state);
    addRoundKey(state, roundKey);
}
int aesEncryption(unsigned char *input, unsigned char *output, unsigned char *key, int size)
{
    int expandedKeySize;
    int roundNum;
    unsigned char *expandedKey;
    unsigned char block[16];
    int i, j;
    expandedKeySize = (16 * (roundNum + 1));
    expandedKey = (unsigned char *)malloc(expandedKeySize * sizeof(unsigned char));
```

```
    if (expandedKey == NULL)
    {
        printf("Unable to allocate memory!");
        return 1;
    }
    else
    {
        for (i = 0; i < 4; i++)
        {
            for (j = 0; j < 4; j++)
                block[(i + (j * 4))] = input[(i * 4) + j];
        }
        expandKey(expandedKey, key, size, expandedKeySize);
        aesProcess(block, expandedKey, roundNum);
        for (i = 0; i < 4; i++)
        {
            for (j = 0; j < 4; j++)
                output[(i * 4) + j] = block[(i + (j * 4))];
        }
        free(expandedKey);
        expandedKey = NULL;
    }
    return 0;
}
void reverseSubBytes(unsigned char *state)
{
    int i;
    for (i = 0; i < 16; i++)
        state[i] = revSBox[state[i]];
}
void reverseShiftRows(unsigned char *state)
{
    int i;
    for (i = 0; i < 4; i++)
        reverseShiftRow(state + i * 4, i);
```

```
}
void reverseShiftRow(unsigned char *state, unsigned char nbr)
{
    int i, j;
    unsigned char temp;
    for (i = 0; i < nbr; i++)
    {
        temp = state[3];
        for (j = 3; j > 0; j--)
            state[j] = state[j - 1];
        state[0] = temp;
    }
}
void reverseMixColumns(unsigned char *state)
{
    int i, j;
    unsigned char column[4];
    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            column[j] = state[(j * 4) + i];
        }
        reverseMixColumn(column);
        for (j = 0; j < 4; j++)
        {
            state[(j * 4) + i] = column[j];
        }
    }
}

void reverseMixColumn(unsigned char *column)
{
    unsigned char dupColumn[4];
    int i;
```

```
    for (i = 0; i < 4; i++)
    {
        dupColumn[i] = column[i];
    }
    column[0] = galoisMultiplication(dupColumn[0], 14) ^
                galoisMultiplication(dupColumn[3], 9) ^
                galoisMultiplication(dupColumn[2], 13) ^
                galoisMultiplication(dupColumn[1], 11);
    column[1] = galoisMultiplication(dupColumn[1], 14) ^
                galoisMultiplication(dupColumn[0], 9) ^
                galoisMultiplication(dupColumn[3], 13) ^
                galoisMultiplication(dupColumn[2], 11);
    column[2] = galoisMultiplication(dupColumn[2], 14) ^
                galoisMultiplication(dupColumn[1], 9) ^
                galoisMultiplication(dupColumn[0], 13) ^
                galoisMultiplication(dupColumn[3], 11);
    column[3] = galoisMultiplication(dupColumn[3], 14) ^
                galoisMultiplication(dupColumn[2], 9) ^
                galoisMultiplication(dupColumn[1], 13) ^
                galoisMultiplication(dupColumn[0], 11);
}
void reverseAesRound(unsigned char *state, unsigned char *roundKey)
{
    reverseShiftRows(state);
    reverseSubBytes(state);
    addRoundKey(state, roundKey);
    reverseMixColumns(state);
}
void reverseAesProcess(unsigned char *state, unsigned char *expandedKey, int roundNum)
{
    int i = 0;
    unsigned char roundKey[16];
    createRoundKey(expandedKey + 16 * roundNum, roundKey);
    addRoundKey(state, roundKey);
    for (i = roundNum - 1; i > 0; i--)
```

```
    {
        createRoundKey(expandedKey + 16 * i, roundKey);
        reverseAesRound(state, roundKey);
    }
    createRoundKey(expandedKey, roundKey);
    reverseShiftRows(state);
    reverseSubBytes(state);
    addRoundKey(state, roundKey);
}
int aesDecryption(unsigned char *input, unsigned char *output, unsigned char *key, int size)
{
    int i, j;
    int expandedKeySize;
    int roundNum;
    unsigned char *expandedKey;
    unsigned char block[16];
    expandedKeySize = (16 * (roundNum + 1));
    expandedKey = (unsigned char *)malloc(expandedKeySize * sizeof(unsigned char));
    if (expandedKey == NULL)
    {
        printf("Unable to allocate memory!");
        return 1;
    }
    else
    {
        for (i = 0; i < 4; i++)
        {
            for (j = 0; j < 4; j++)
                block[(i + (j * 4))] = input[(i * 4) + j];
        }
        expandKey(expandedKey, key, size, expandedKeySize);
        reverseAesProcess(block, expandedKey, roundNum);
        for (i = 0; i < 4; i++)
        {
            for (j = 0; j < 4; j++)
```

```
            output[(i * 4) + j] = block[(i + (j * 4))];
        }
        free(expandedKey);
        expandedKey = NULL;
    }
    return 0;
}
```

Output:

```
PS D:\DATAs\NITJ\CryptoLab> gcc aesEncryption.c
PS D:\DATAs\NITJ\CryptoLab> ./a

Enter 16-byte(128-bit) key: 1234567891234567

Enter 16-byte(128-bit) plain text: ABCDEFGHIJKLMNOP

Expanded Key (hexadecimal):
31 32 33 34 35 36 37 38 39 31 32 33 34 35 36 37
a6 37 a9 2c 93 01 9e 14 aa 30 ac 27 9e 05 9a 10
cf 8f 63 27 5c 8e fd 33 f6 be 51 14 68 bb cb 04
21 90 91 62 7d 1e 6c 51 8b a0 3d 45 e3 1b f6 41
86 d2 12 73 fb cc 7e 22 70 6c 43 67 93 77 b5 26
63 07 e5 af 98 cb 9b 8d e8 a7 d8 ea 7b d0 6d cc
33 3b ae 8e ab f0 35 03 43 57 ed e9 38 87 80 25
64 f6 91 89 cf 06 a4 8a 8c 51 49 63 b4 d6 c9 46
12 2b cb 04 dd 2d 6f 8e 51 7c 26 ed e5 aa ef ab
a5 f4 a9 dd 78 d9 c6 53 29 a5 e0 be cc 0f 0f 15
e5 82 f0 96 9d 5b 36 c5 b4 fe d6 7b 78 f1 d9 6e

Plain Text: ABCDEFGHIJKLMNOP

Enciphered Text : □╪9"□ú■ ï□ÆP╖XT≤

Deciphered Text: ABCDEFGHIJKLMNOP
```