IT NETWORK SECURITY

LAB WORK- AES ENCRYPTION

NAME: **OJASVI SINGH CHAUHAN**
ROLL NO. **R134218111**
BRANCH: BTECH CSE-**CSF-B3**
SAP ID: **500068394**

INDEX

EXPERIMENT – 3
AES ENCRYPTION

AIM: To implement Advanced Encryption Standard Encryption

INPUT: Plain Text

OUTPUT: Cipher Text

ALGORITHM:

You take the following AES steps of encryption for a 128-bit block:

1. Derive the set of round keys from the cipher key.
2. Initialize the state array with the block data (plaintext).
3. Add the initial round key to the starting state array.
4. Perform nine rounds of state manipulation.
5. Perform the tenth and final round of state manipulation.
6. Copy the final state array out as the encrypted data (ciphertext).

Each round of the encryption process requires a series of steps to alter the state array. These steps involve four types of operations called:

- SubBytes
- ShiftRows
- MixColumns
- XorRoundKey

SubBytes

This operation is a simple substitution that converts every byte into a different value. AES defines a table of 256 values for the substitution. You work through the 16 bytes of the state array, use each byte as an index into the 256-byte substitution table, and replace the byte with the value from the substitution table.

ShiftRows

As the name suggests, ShiftRows operates on each row of the state array. Each row is rotated to the right by a certain number of bytes

MixColumns

This operation is the most difficult, both to explain and perform. Each column of the state array is processed separately to produce a new column. The new column replaces the old one. The processing involves a matrix multiplication.

XorRoundKey

After the MixColumns operation, the XorRoundKey is very simple indeed and hardly needs its own name. This operation simply takes the existing state array, XORs the value of the appropriate round key, and replaces the state array with the result. It is done once before the rounds start and then once per round, using each of the round keys in turn.

CODE:

```
#include<stdio.h>
#define Nb 4
int Nr=0;
int Nk=0;
unsigned char in[16], out[16], state[4][4];
unsigned char RoundKey[240];
unsigned char Key[32];
int getSBoxValue(int num)
{
   int 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 };
   return sbox[num];
}
int 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  };

void KeyExpansion()
{
   int i,j;
   unsigned char temp[4],k;

   for(i=0;i<Nk;i++)
   {
      RoundKey[i*4]=Key[i*4];
      RoundKey[i*4+1]=Key[i*4+1];
      RoundKey[i*4+2]=Key[i*4+2];
      RoundKey[i*4+3]=Key[i*4+3];
   }

   while (i < (Nb * (Nr+1)))
   {
      for(j=0;j<4;j++)
      {
         temp[j]=RoundKey[(i-1) * 4 + j];
      }
      if (i % Nk == 0)
      {

         {
            k = temp[0];
            temp[0] = temp[1];
            temp[1] = temp[2];
            temp[2] = temp[3];
            temp[3] = k;
         }
         {
            temp[0]=getSBoxValue(temp[0]);
            temp[1]=getSBoxValue(temp[1]);
            temp[2]=getSBoxValue(temp[2]);
            temp[3]=getSBoxValue(temp[3]);
         }
         temp[0] =  temp[0] ^ Rcon[i/Nk];
      }
      else if (Nk > 6 && i % Nk == 4)
      {

         {
            temp[0]=getSBoxValue(temp[0]);
            temp[1]=getSBoxValue(temp[1]);
            temp[2]=getSBoxValue(temp[2]);
            temp[3]=getSBoxValue(temp[3]);
         }
      }
      RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
      RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
      RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
      RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
      i++;
   }
}

void AddRoundKey(int round)
{
   int i,j;
   for(i=0;i<4;i++)
   {
      for(j=0;j<4;j++)
      {
         state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb + j];
      }
   }
}

void SubBytes()
{
   int i,j;
   for(i=0;i<4;i++)
   {
      for(j=0;j<4;j++)
      {
         state[i][j] = getSBoxValue(state[i][j]);
      }
   }
}
```

```
void ShiftRows()
{
   unsigned char temp;

   temp=state[1][0];
   state[1][0]=state[1][1];
   state[1][1]=state[1][2];
   state[1][2]=state[1][3];
   state[1][3]=temp;

   temp=state[2][0];
   state[2][0]=state[2][2];
   state[2][2]=temp;
   temp=state[2][1];
   state[2][1]=state[2][3];
   state[2][3]=temp;

   temp=state[3][0];
   state[3][0]=state[3][3];
   state[3][3]=state[3][2];
   state[3][2]=state[3][1];
   state[3][1]=temp;
}

#define xtime(x)   ((x<<1) ^ (((x>>7) & 1) * 0x1b))

void MixColumns()
{
   int i;
   unsigned char Tmp,Tm,t;
   for(i=0;i<4;i++)
   {
      t=state[0][i];
      Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i] ;
      Tm = state[0][i] ^ state[1][i] ; Tm = xtime(Tm); state[0][i] ^= Tm ^ Tmp ;
      Tm = state[1][i] ^ state[2][i] ; Tm = xtime(Tm); state[1][i] ^= Tm ^ Tmp ;
      Tm = state[2][i] ^ state[3][i] ; Tm = xtime(Tm); state[2][i] ^= Tm ^ Tmp ;
      Tm = state[3][i] ^ t ; Tm = xtime(Tm); state[3][i] ^= Tm ^ Tmp ;
   }
}

void Cipher()
{
   int i,j,round=0;

   for(i=0;i<4;i++)
```

```
    {
        for(j=0;j<4;j++)
        {
            state[j][i] = in[i*4 + j];
        }
    }

    AddRoundKey(0);

    for(round=1;round<Nr;round++)
    {
        SubBytes();
        ShiftRows();
        MixColumns();
        AddRoundKey(round);
    }

    SubBytes();
    ShiftRows();
    AddRoundKey(Nr);

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            out[i*4+j]=state[j][i];
        }
    }
}
void main()
{
    int i;

    while(Nr!=128 && Nr!=192 && Nr!=256)
    {
        printf("Enter the length of Key(128, 192 or 256 only): ");
        scanf("%d",&Nr);
    }

    Nk = Nr / 32;
    Nr = Nk + 6;

    unsigned char temp[16] = {0x00  ,0x01  ,0x02  ,0x03  ,0x04  ,0x05  ,0x06  ,0x07
,0x08  ,0x09  ,0x0a  ,0x0b  ,0x0c  ,0x0d  ,0x0e  ,0x0f};
    unsigned char temp2[16]= {0x00  ,0x11  ,0x22  ,0x33  ,0x44  ,0x55  ,0x66  ,0x77
,0x88  ,0x99  ,0xaa  ,0xbb  ,0xcc  ,0xdd  ,0xee  ,0xff};
```

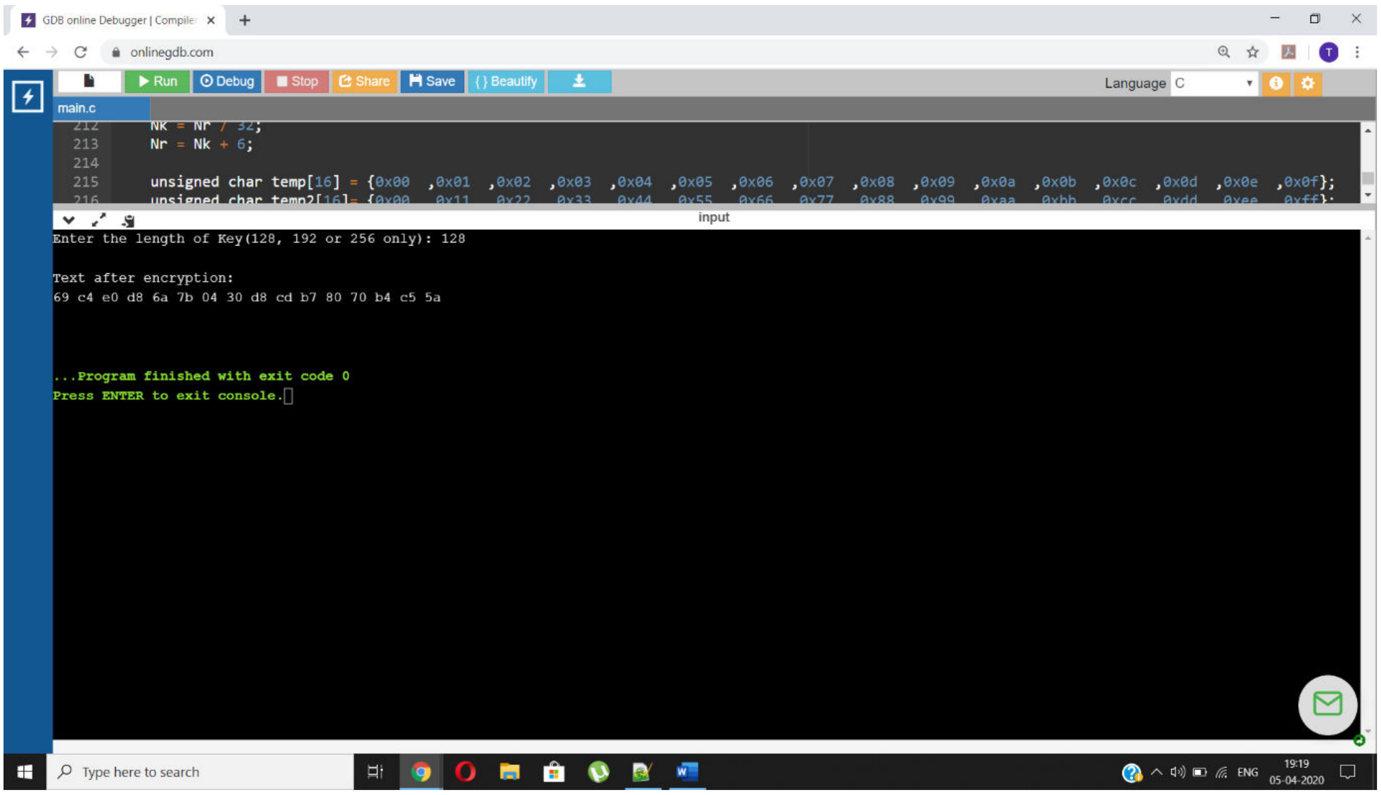
```
    for(i=0;i<Nk*4;i++)
    {
        Key[i]=temp[i];
        in[i]=temp2[i];
    }

    KeyExpansion();

    Cipher();

    printf("\nText after encryption:\n");
    for(i=0;i<Nk*4;i++)
    {
        printf("%02x ",out[i]);
    }
    printf("\n\n");
}
```

<u>OUTPUT SNAPSHOT:</u>



```
    Nk = Nr / 32;
    Nr = Nk + 6;

    unsigned char temp[16] = {0x00  ,0x01  ,0x02  ,0x03  ,0x04  ,0x05  ,0x06  ,0x07  ,0x08  ,0x09  ,0x0a  ,0x0b  ,0x0c  ,0x0d  ,0x0e  ,0x0f};
    unsigned char temp2[16]= {0x00  ,0x11  ,0x22  ,0x33  ,0x44  ,0x55  ,0x66  ,0x77  ,0x88  ,0x99  ,0xaa  ,0xbb  ,0xcc  ,0xdd  ,0xee  ,0xff};
```

```
Enter the length of Key(128, 192 or 256 only): 128

Text after encryption:
69 c4 e0 d8 6a 7b 04 30 d8 cd b7 80 70 b4 c5 5a



...Program finished with exit code 0
Press ENTER to exit console.
```