```
%-------------------------------------------------------------------------
%------------------- AES implementation using MATLAB ----------------------
% This is the main program which calls the functions for executing each step
%                    included in AES algorithm.
%-------------------------------------------------------------------------

input_data = input_data_trial1; % This function gets the 16 bytes of data
                                % in their decimal equivalent into
                                % input_data

key_data = key_original;        % This function gets the 16-byte key in its
                                % decimal equivalent into key_data

state_array = add_round_key0(input_data, key_data); % This function adds the
                                % first round key and draws the first state
                                % matrix. The function is given input 16
                                % bytes and 16-byte key as the input.


state_array_hex = dec2hex(state_array); % This function converts the decimal
                                % values in state array into hex value for
                                % inspection and display purpose.

%-------------------------------------------------------------------------
%------------------------ KEY EXPANSION ALGORITHM ------------------------
%-------------------------------------------------------------------------
Rcon= [01 00 00 00];            % Rcon is the Round constant of the first
                                % round of key expansion Algorithm

R1keys_dec = key_expansion_round1(key_data, Rcon);  % this function will
                                % produce word4 to word7 of Key Schedule

R1keys_hex = dec2hex(R1keys_dec);  % hex representation of word4 to word7
                                % of the key schedule

Rcon123;                        % This function will return Round key for
                                % all the key expansion rounds

key_shedule_dec (1, :) = R1keys_dec;  % key_schedule_dec will now store all
                                % the key expansion schedule.

for i=2:10
key_shedule_dec(i,:)=key_expansion_round2(key_shedule_dec(i-1,:),Rcon2(i,:));
end
                                % The above for loop will make key schedule
                                % and store it. The function
                                % key_expansion_round2 gives back 4 words
                                % at a time of the key schedule. It takes
                                % inputs as the earlier 4 words of the
                                % schedule and Rcon of that round of key
                                % expansion.

key_schedule_hex = dec2hex (key_shedule_dec (i, :));  % hex representation
                                % of the whole key schedule.
```

```
%----------------------------------------------------------------------------
%----------------------- AES ENCRYPTION ---------------------------------
%----------------------------------------------------------------------------


%----------------------------------------------------------------------------
% The following loop are the Round 1 to Round 9 of encryption. Each round
% has 4 steps viz Byte substitution, shifting rows, mixing columns and
% adding round key. Each function
%----------------------------------------------------------------------------

for i=1:9
%------------------------ Byte substitution -------------------------------
state_array = byte_sub(state_array);    % returns new state array after byte
                                        % substitution.
state_array_hex = dec2hex(state_array,2);

%------------------------- Shifting Rows ----------------------------------
state_array = shift_row(state_array);  % returns new state array after shift
                                        % row operation on the input state array.
state_array_hex = dec2hex(state_array,2);

%--------------------------- Mixing Columns ------------------------------
state_array = mix_columns(state_array); % returns new state array after mix
                                        % column operation on the input state array.
state_array_hex = dec2hex(state_array,2);

%---------------------------- Add round key -------------------------------
state_array=xoring_function_trial1(state_array, key_shedule_dec (i, :));
                                        % returns new state array after adding
                                        % round key to the input state array.
                                        % The function takes state array and
                                        % corresponding words from key schedule.
state_array_hex = dec2hex(state_array,2);
end

%----------------------------------------------------------------------------
% Round 10 of AES encryption has only 3 steps viz Byte substitution, Shift
% row and adding round key
state_array=byte_sub(state_array);  % Byte substitution
state_array_hex = dec2hex(state_array,2);

state_array = shift_row(state_array);     % SHIFTING ROWS
state_array_hex = dec2hex(state_array,2);


state_array=xoring_function_trial1(state_array, key_shedule_dec (10, :));
                                                            % add round key
state_array_hex = dec2hex(state_array,2);

%----------------------------------------------------------------------------
key_shedule_out (1, :)=key_data; % rearranging the key schedule and placing
key_shedule_out (2:11, :)=key_shedule_dec;  % them in key_schedule out
%----------------------------------------------------------------------------
st= ['Cipher text   :',char(state_array)]; % Displaying the cipher text
disp(st)
```

```
%------------------------------------------------------------------------
%-------------------------- AES DECRYPTION ---------------------------------
%------------------------------------------------------------------------


%-------------------------- ADD ROUND KEY --------------------------------
state_array=xoring_function_trial1(state_array, key_shedule_out (11, :));
state_array_hex = dec2hex(state_array,2);
%------------------------------------------------------------------------


%------------------- Round 1 to Round 9 of decryption ----------------------
 for i=10: -1:2
%---------------- Inverse Shifting Rows -----------------------------------
state_array = inv_shift_row(state_array);
state_array_hexrow = dec2hex(state_array,2);  % returns new state array
                                % after inverse shifting rows operation on
                                % the input state array.
%---------------- Inverse Byte substitution -------------------------------
state_array = inv_byte_sub(state_array);
state_array_hexsub = dec2hex(state_array,2);  % returns new state array
                                % after inverse Byte substitution operation
                                % on the input state array.
%----------------- Add Round key ------------------------------------------
state_array = xoring_function_trial1(state_array, key_shedule_out (i, :));
state_array_hexkey = dec2hex(state_array,2);    % returns new state array
                                % after adding round key on the input state
                                    array.
%----------------- Mixing column ------------------------------------------
state_array = inv_mix_columns(state_array);
state_array_hexcolumn = dec2hex(state_array,2); % returns new state array
                                % after mixing column operation on the
                                % input state array.
 end
%------------------------------------------------------------------------



%--------------- Round 10 of AES Decryption -------------------------------
%---------------- Inverse Shifting Rows -----------------------------------
state_array = inv_shift_row(state_array);
state_array_hex = dec2hex(state_array,2);
%---------------- Inverse Byte Substitution -------------------------------
state_array = inv_byte_sub(state_array);
state_array_hex = dec2hex(state_array,2);
%---------------- Add Round Key -------------------------------------------
state_array = xoring_function_trial1(state_array, key_shedule_out (1, :));
state_array_hex = dec2hex(state_array,2);
%------------------------------------------------------------------------

state_array_char = ['Decrypted text:', char(state_array)];
disp(state_array_char)              % Displaying the cipher text
```

```
%-------------------------------------------------------------------------
%------------ FUNCTION USED IN AES ENCRYPTION AND DECRYPTION --------------
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
%---------------- Function for entering plain text ------------------------
%------------------------------------------------------------------------

function [input_dec_string] = input_data_trial1
input_dec_string = unicode2native ('Two One Nine Two','utf-8');
% Converts the input string of data "Plain text" into its decimal equivalent
disp ('Plain text    :Prathamesh CSUF1')

%-------------------------------------------------------------------------
%---------------- Function for entering plain text ------------------------
%------------------------------------------------------------------------

function [key_dec_string] = key_original
key_dec_string = unicode2native ('Thats my Kung Fu','utf-8');
% Converts the encryption key into its decimal equivalent
disp ('KEY           :AES ENCRYPTION  ')

%-------------------------------------------------------------------------
%-------------- Function for adding round key i.e. W0-W3 ------------------
%------------------------------------------------------------------------

function[state_array] = add_round_key0(input_data, key_data)
% this function will add round key to the input data i.e. W0-W3 of key data
out=xoring_function_trial1(input_data, key_data);
% the function, xor's the input data and key data to give the 1st state
% array
out_hex=dec2hex(out);
state_array=out;

%------------------------------------------------------------------------
%-------------------- Function for XORING FUNCTION ------------------------
%------------------------------------------------------------------------

  function[xoring_output] = xoring_function_trial1(data1, data2)
% This function will return xor the data1 and data2
% It first converts the data into binary values and the xor's them
% Converts back to decimal as return value to the function
leng=length(data1);
    for i=1:leng
        var1=hexToBinaryVector(dec2hex(data1(i)),8);
        var2=hexToBinaryVector(dec2hex(data2(i)),8);
        xoring1=binaryVectorToHex(bitxor(var1,var2));
        out1(i)=hex2dec(xoring1);
    end
xoring_output=out1;
```

```
%--------------------------------------------------------------------------
%-------------- Function for KEY EXPANSION 1ST ROUND KEY ------------------
%-------------------------------------------------------------------------

function [round1_key_dec] =key_expansion_round1(key_data, Rcon)

%----- For calculating the "g" function getting Word 3 form key_data ------

left=key_data (13:16);          % This is the word 3 form the key schedule
left_hex=dec2hex(left);

%----------- shifting one-byte left circular rotation on Wi+3 ------------
left_1=left (1);
left_main=zeros (1,1);
for i=2:4
   left_main(i-1)=left(i);
end
left_main(4) =left_1;
left_main_hex=dec2hex(left_main);

%------------ entering sbox and substituting byte from s box --------------
s_box;                           % this function will return a sbox1 matrix
                                 % from where we can substitute byte
for i=1:4                        % This loop will substitute byte form sbox
    temp=dec2hex(left_main(i));
    row=hex2dec (temp (1)) +1;
    column=hex2dec (temp (2)) +1;
    substituted_byte_dec(i)= sbox1(row, column);
end
substituted_byte_hex=dec2hex(substituted_byte_dec);

%----------- Add round key and generating G function "gw3" ----------------
%-------------- Rcon= [01 00 00 00] for the first round --------------------
%------------------- Rcon=[RC(i), 0x00, 0x00, 0x00] ------------------------
h1=hexToBinaryVector (dec2hex (Rcon (1)),8);
h2=hexToBinaryVector (dec2hex (substituted_byte_dec (1)),8);

gw31=hex2dec (binaryVectorToHex (bitxor (h1, h2))); % first element RC[i]

gw3_dec= [gw31 substituted_byte_dec (2:4)]; % RC[i]
gw3_hex=dec2hex(gw3_dec);

%----------------- Calculating Wi+4 = Wi xor gWi+3 ------------------------
%----- Calculating Word 4, Word 5, Word 6 word \=7 of key schedule ---------
w4_dec=xoring_function_trial1(gw3_dec, key_data (1:4));
w4_hex=dec2hex(w4_dec);          % W4 = W1 xor gWi+3

w5_dec=xoring_function_trial1(w4_dec, key_data (5:8));
w5_hex=dec2hex(w5_dec);          % W5 = W1 xor W4

w6_dec=xoring_function_trial1(w5_dec, key_data (9:12));
w6_hex=dec2hex(w6_dec);         % W6 = W2 xor W5

w7_dec=xoring_function_trial1(w6_dec, key_data (13:16));
w7_hex=dec2hex(w7_dec);          % W7 = W3 xor W6

round1_key_dec= [w4_dec w5_dec w6_dec w7_dec];
```

```
%-----------------------------------------------------------------------------
Rcon
Rcon2= [     01 00 00 00;
             02 00 00 00;
             04 00 00 00;
             08 00 00 00;
             16 00 00 00;
             32 00 00 00;
             64 00 00 00;
            128 00 00 00;
             27 00 00 00;
             54 00 00 00];

%----------------------------------------------------------------------------
%-------------- Function for KEY EXPANSION ALGORITHM -----------------------
%----------------------------------------------------------------------------

function [round1_key_dec] =key_expansion_round2(key_data, Rcon)

left = key_data (13:16);            % Getting the last word from the 4 words
left_hex = dec2hex(left);

%-----------------------------------------------------------------------------
%------------ Shifting one bye left circular rotation on Wi+3 ------------
%-----------------------------------------------------------------------------
left_1=left (1);
left_main=zeros (1,1);
for i=2:4
   left_main(i-1) =left(i);
end
left_main (4) =left_1;
left_main_hex=dec2hex(left_main);

%-----------------------------------------------------------------------------
%------------------- Byte Substituting from s-box -------------------------
%-----------------------------------------------------------------------------
s_box;
for i=1:4
temp=dec2hex(left_main(i),2);
row=hex2dec (temp (1)) +1;
column12=hex2dec (temp (2)) +1;
substituted_byte_dec(i)= sbox1(row, column12);
end
substituted_byte_hex=dec2hex(substituted_byte_dec);

%-----------------------------------------------------------------------------
%--------- Add round key and generating G function "g(Wi+3)"---------------
%-----------------------------------------------------------------------------
h1=hexToBinaryVector(dec2hex(Rcon(1)),8);
h2=hexToBinaryVector(dec2hex(substituted_byte_dec(1)),8);

gw31=hex2dec(binaryVectorToHex(bitxor(h1,h2)));% first element RC[i]

gw3_dec=[gw31 substituted_byte_dec(2:4) ]; %Rcon[i]
gw3_hex=dec2hex(gw3_dec);
```

```
%-----------------------------------------------------------------------------
%---------------- Calculating Wi+4 from Wi xor g(Wi+3) -------------------
%-----------------------------------------------------------------------------
w4_dec=xoring_function_trial1(gw3_dec,key_data(1:4));
w4_hex=dec2hex(w4_dec);

w5_dec=xoring_function_trial1(w4_dec,key_data(5:8));
w5_hex=dec2hex(w5_dec);

w6_dec=xoring_function_trial1(w5_dec,key_data(9:12));
w6_hex=dec2hex(w6_dec);

w7_dec=xoring_function_trial1(w6_dec,key_data(13:16));
w7_hex=dec2hex(w7_dec);
round1_key_dec=[w4_dec w5_dec w6_dec w7_dec];


%-----------------------------------------------------------------------------
%----------------- Function for Byte Substitution -----------------------
%-----------------------------------------------------------------------------

function [state_array] =byte_sub(input_state_array)
% This function will return byte substituted state array of input state array
s_box;                      % this function will return sbox1 matrix

% This loop will substitute each byte of state array by its equivalent in
sbox
for i=1:16
    temp=dec2hex(input_state_array(i),2);
    rowi=hex2dec(temp(1))+1;          % row index from each byte of state array
    columni=hex2dec(temp(2))+1;       % column index
    aes_byte_dec(i)= sbox1(rowi, columni);    % byte substitution form sbox
                                             % using row and column index
end
state_array=aes_byte_dec;


%-----------------------------------------------------------------------------

sbox1 = [
 99, 124, 119, 123, 242, 107, 111, 197,  48,   1, 103,  43, 254, 215, 171, 118,
202, 130, 201, 125, 250,  89,  71, 240, 173, 212, 162, 175, 156, 164, 114, 192,
183, 253, 147,  38,  54,  63, 247, 204,  52, 165, 229, 241, 113, 216,  49,  21,
  4, 199,  35, 195,  24, 150,   5, 154,   7,  18, 128, 226, 235,  39, 178, 117,
  9, 131,  44,  26,  27, 110,  90, 160,  82,  59, 214, 179,  41, 227,  47, 132,
 83, 209,   0, 237,  32, 252, 177,  91, 106, 203, 190,  57,  74,  76,  88, 207,
208, 239, 170, 251,  67,  77,  51, 133,  69, 249,   2, 127,  80,  60, 159, 168,
 81, 163,  64, 143, 146, 157,  56, 245, 188, 182, 218,  33,  16, 255, 243, 210,
205,  12,  19, 236,  95, 151,  68,  23, 196, 167, 126,  61, 100,  93,  25, 115,
 96, 129,  79, 220,  34,  42, 144, 136,  70, 238, 184,  20, 222,  94,  11, 219,
224,  50,  58,  10,  73,   6,  36,  92, 194, 211, 172,  98, 145, 149, 228, 121,
231, 200,  55, 109, 141, 213,  78, 169, 108,  86, 244, 234, 101, 122, 174,   8,
186, 120,  37,  46,  28, 166, 180, 198, 232, 221, 116,  31,  75, 189, 139, 138,
112,  62, 181, 102,  72,   3, 246,  14,  97,  53,  87, 185, 134, 193,  29, 158,
225, 248, 152,  17, 105, 217, 142, 148, 155,  30, 135, 233, 206,  85,  40, 223,
140, 161, 137,  13, 191, 230,  66, 104,  65, 153,  45,  15, 176,  84, 187,  22];
```

```
%-------------------------------------------------------------------------
%----------------- Function for SHIFTING ROWS ----------------------------
%-------------------------------------------------------------------------
function[state_array] =shift_row(input_state_array)

ST_ARY_DEC=reshape(input_state_array,4,4);  % Converts the 16-byte array
                                            % into 4x4 matrix
%--------------- This loop will shift row 2, 3 and 4 ----------------------
for i=2:4
    ST_ARY_DEC (i, :) = circshift(ST_ARY_DEC(i,:),-i+1,2);
end
state_array=reshape(ST_ARY_DEC,1,16);    % Converts the 4x4 matrix into
                                         % 16-bytes array
ST_ARY_HEX=dec2hex(ST_ARY_DEC,2);


%-------------------------------------------------------------------------
%----------------- Function for MIXING COLUMN ----------------------------
%-------------------------------------------------------------------------
function [addop]=mix_columns(ST_ARY_DEC)

column_operation= [02 01 01 03 03 02 01 01 01 03 02 01 01 01 03 02];


step=1;

% The following loop is used for matrix multiplication over GF (2^8)
% Between column_operation matrix and state array
for j=1:4
% Dot_operation is multiplication over GF (2^8) function
% this loop is used for multiplying row of column_operation matrix with
% corresponding column of state array.
  for i=1:4
      dot_op1(i,:,j)=dot_operation(column_operation(step),ST_ARY_DEC(i));
      dot_op2(i,:,j)=dot_operation(column_operation(step),ST_ARY_DEC(i+4));
      dot_op3(i,:,j)=dot_operation(column_operation(step),ST_ARY_DEC(i+8));
      dot_op4(i,:,j)=dot_operation(column_operation(step),ST_ARY_DEC(i+12));
      step=step+4;
  end
    if(j==1)
      step=2;
    elseif(j==2)
      step=3;
    elseif(j==3)
      step=4;
    end

  % These add_bit is used for adding the result of dot operation to form each
  % byte of s'matrix i.e. output of mix column operation.
   add_op1(j)=add_bits(dot_op1(:,:,j));
   add_op2(j)=add_bits(dot_op2(:,:,j));
   add_op3(j)=add_bits(dot_op3(:,:,j));
   add_op4(j)=add_bits(dot_op4(:,:,j));
end

  addop= [add_op1 add_op2 add_op3 add_op4];
```

```
%----------------------------------------------------------------------
%----------------- Function for DOT_OPERATION ---------------------------
%----------------------------------------------------------------------
function [operation_output] =dot_operation (input1, input2)

field=hexToBinaryVector (dec2hex (27),8); %field representation 0x1b

%--------- If multiplication (dot operation) by x01 over GF (2^8) -----------
if      input1==01                    %
    operation_output=input2;     % The input remains the same.

%-------------- If multiplication by 0x02 over GF (2^8) --------------------
elseif input1==02
    in2=dec2hex(input2,2);        % first the binary value of the input byte
    input2_bin=hexToBinaryVector(in2,8);     % is shifted one bit left
    output_bin (1, :)=input2_bin (1,2:8);
    output_bin (1,8) =0;

    if input2_bin (1) == 1         % If the 1st bit of input is 1 then it is
                                                              % xored
    output_bin= bitxor (output_bin, field) ;  % with field representation
    end                                      % 0x1b

   op1=binaryVectorToHex(output_bin);
   output_dec = hex2dec(op1);

   operation_output=output_dec;

%-------------- If multiplication by 0x03 over GF (2^8) --------------------
elseif input1==03
    in2=dec2hex(input2,2);
    input2_bin=hexToBinaryVector(in2,8);     % input2 as binary (state matrix)

    output_bin1(1, :)=input2_bin (1,2:8);      % shifting bits to left
    output_bin1(1,8) =0;

    if input2_bin (1) ==1                  % If the 1st bit of input is 1 then
       output_bin1= bitxor (output_bin1, field) ; % xoring with 0x1b
    end
    op=bitxor (output_bin1, input2_bin);
    operation_output=hex2dec(binaryVectorToHex(op));
end



%----------------------------------------------------------------------
%----------------- Function for Add Bits --------------------------------
%----------------------------------------------------------------------
function [main_output] =add_bits(input)
% This function is used to add bytes in matrix multiplication.
% the input is 4 bytes of data after dot_operation
% the output is byte addition of the 4 input bytes
    for i=1:4
       temp=dec2hex(input(i),2);
       input_bin (i,:)= hexToBinaryVector(temp,8);
    end
```

```
   b1= bitxor(input_bin(1,:),input_bin(2,:));
   b2= bitxor(input_bin(3,:),input_bin(4,:));
   b3= bitxor(b1, b2);
   main_output_bin = binaryVectorToHex(b3);

   main_output=hex2dec(main_output_bin);


function [state_array] =inv_byte_sub(input_state_array)

inv_sbox;                    % this returns inverse _s_box matrix

% This loop will substitute each byte of state array by its equivalent in
% inverse _s_box matrix
for i=1:16
    temp=dec2hex(input_state_array(i),2);
    rowi=hex2dec (temp (1)) +1;      % row index from each byte of state array
    columni=hex2dec (temp (2)) +1;      % column index
    aes_byte_dec(i)= inv_s_box (rowi, columni);    % byte substitution from
                                                % inv_sbox using row and column index
end
state_array=aes_byte_dec;


%-------------------------------------------------------------------------
%----------------- Function for INVERSE MIXING COLUMN --------------------
%-------------------------------------------------------------------------

function [addop]=inv_mix_columns(ST_ARY_DEC)

column_operation= [14 09 13 11 11 14 09 13 13 11 14 09 09 13 11 14];
% [0E 0B 0D 09
% 09 0E 0B 0D
% 0D 09 0E 0B
% 0B 0D 09 0E]

% this matrix is used for matrix multiplication over GF (2^8) while inverse
% mixing columns

step=1;

% The following loop is used for matrix multiplication over GF (2^8)
% Between column_operation matrix and state array
for j=1:4
% INV_Dot_operation is multiplication over GF (2^8) function
% this loop is used for multiplying row of column_operation matrix with
% corresponding column of state array.
  for i=1:4
      dot_op1(i,:,j)=inv_dot_operation(column_operation(step),ST_ARY_DEC(i));

dot_op2(i,:,j)=inv_dot_operation(column_operation(step),ST_ARY_DEC(i+4));

dot_op3(i,:,j)=inv_dot_operation(column_operation(step),ST_ARY_DEC(i+8));

dot_op4(i,:,j)=inv_dot_operation(column_operation(step),ST_ARY_DEC(i+12));
      step=step+4;
  end
```

```
    if(j==1)
      step=2;
    elseif(j==2)
      step=3;
    elseif(j==3)
      step=4;
    end

  % These add_bit is used for adding the result of dot operation to form each
  % byte of s'matrix i.e. output of mix column operation.
   add_op1(j)=add_bits(dot_op1(:,:,j));
   add_op2(j)=add_bits(dot_op2(:,:,j));
   add_op3(j)=add_bits(dot_op3(:,:,j));
   add_op4(j)=add_bits(dot_op4(:,:,j));
end

  addop = [add_op1 add_op2 add_op3 add_op4];

%--------------------------------------------------------------------------
%----------------- Function for INVERSE DOT OPERATION --------------------
%--------------------------------------------------------------------------
function [output]=inv_dot_operation (input1, input2)

    field=hexToBinaryVector (dec2hex (27),8); %field representation

    in2=dec2hex(input2,2);
    input2_bin=hexToBinaryVector(in2,8); % converting to binary

%-------- If multiplication (dot operation) by 0x09 over GF (2^8) -----------
if    input1==09       % in × 0x09=(((in×2) ×2) ×2) +in

    p1=gf_mul(input2);  % gf_mul multiplies 0x02 with input2 over GF (2^8)
    p2=gf_mul(p1);
    p3=gf_mul(p2);
    p4=xoring_function_trial1(p3, input2);
    output=p4;

%-------- If multiplication (dot operation) by 0x0b over GF (2^8) -----------
elseif input1==11   % in × 0x0b =(((in×2) ×2) ×2) +in

    p1=gf_mul(input2);  % gf_mul multiplies 0x02 with input2 over GF (2^8)
    p2=gf_mul(p1);
    p3=xoring_function_trial1(p2,input2);
    p4=gf_mul(p3);
    p5=xoring_function_trial1(p4,input2);
    output=p5;

%-------- If multiplication (dot operation) by 0x0d over GF (2^8) -----------
elseif input1==13   % in × 0x0d =(((in×2) ×2) ×2) +in
    p1=gf_mul(input2);  % gf_mul multiplies 0x02 with input2 over GF (2^8)
    p2=xoring_function_trial1(p1,input2);
    p3=gf_mul(p2);
    p4=gf_mul(p3);
    p5=xoring_function_trial1(p4,input2);
    output=p5;
```

```
%-------- If multiplication (dot operation) by 0x0e over GF (2^8) -----------
elseif input1==14   % in × 0x0e =(((in×2) ×2) ×2) +in
    p1=gf_mul(input2);  % gf_mul multiplies 0x02 with input2 over GF(2^8)
    p2=xoring_function_trial1(p1,input2);
    p3=gf_mul(p2);
    p4=xoring_function_trial1(p3,input2);
    p5=gf_mul(p4);
    output=p5;

end


%--------------------------------------------------------------------------
%------------------ Function for GF_MUL -----------------------------------
%--------------------------------------------------------------------------
function [out_data] = gf_mul(input2)

field=hexToBinaryVector (dec2hex (27),8);%field representation

    in2=dec2hex(input2,2);
    input2_bin=hexToBinaryVector(in2,8);    % converting into binary

    output_bin(1,:)=input2_bin(1,2:8);
    output_bin (1,8)=0;                      % shifting left

    if input2_bin(1)==1
    output_bin= bitxor(output_bin, field) ;   % xoring with 0x1b
    end

   op1=binaryVectorToHex(output_bin);
   output_dec = hex2dec(op1);               % converting to decimal

   out_data=output_dec;
```

Assignment #5 – AES Prathamesh Ovalekar
CWID: 805668530 EGCP 543

OUTPUT SCREEN: