Network Security Assignment

Ajit Kumar (M21CS017)

Question1.(a) .Write a script from scratch that implements the full DES [Encryption as well as Decryption]. Modify substitution operation, permutation operation, XOR operation, block size, key generation, and calculate the avalanche effect for this implementation of DES to standard DES. For the encryption key, your script should prompt the user for a keyboard entry.

Answer: Des algorithm contains three stages:

- Initial Permutation
- Key_generation
- 16 round
- Inverse Permutation

Initial Permutation:

It changes the order of original message. Below is implementation of initial permutation:

```
// variable use to store initial permutation result
string first_permutation_string = "";
//permutation table
int initial_permutation_table[64] = {
57,49,41,33,25,17,9,1,
59,51,43,35,27,19,11,3,
61,53,45,37,29,21,13,5,
63,55,47,39,31,23,15,7,
56,48,40,32,24,16,8,0,
58,50,42,34,26,18,10,2,
60,52,44,36,28,20,12,4,
62,54,46,38,30,22,14,6
};
// iterating over the initial_permutation_table
for(int i = 0; i < 64; i++){
```

Key_generation:

- PC1
- PC2
- 64 to 56 bit
- 56 to 48 bit

```
void key_generator(string key){

    // The Pc1 table to compress the key from 64 to 56
    int pc_1[56] = {
    56,48,40,32,24,16,8,
    0,57,49,41,33,25,17,
    9,1,58,50,42,34,26,
    18,10,2,59,51,443,35,
    62,54,46,38,30,22,14,
    6,61,53,45,37,29,21,
    13,5,60,52,44,36,28,
    20,12,4,27,19,11,3
    };
    // The PC2 table to compress the key size from 56 to 48
    int pc_2[48] = {
    13,16,10,23,0,4,
    2,27,14,5,20,9,
    22,18,11,3,25,7,
    15,6,26,19,12,1,
    40,51,30,36,46,54,
    29,39,50,44,32,47,
    43,48,38,55,33,52,
    45,41,49,35,28,31
    };

    // variable to store permute string
    string permutation_string_key ="";
    // iteration over the pc1 table
    for(int a = 0; a < 56; a++){
```

Continue..

```
    // Generating all the sixteen keys
    for(int b=0; b<16; b++){

//shift the key value by one in round 1,2,9,16
        if(b == 0 || b == 1 || b==8 || b==15 ){
            // calling to shift_left_byone function for left string
            left_message= shift_left_byone(left_message);
             // calling to shift_left_byone function for right string
            right_message=shift_left_byone(right_message);
        }

//shift the ket vlue by two for round except 1,2,9,16
        else{
            // calling to shift_left_twice function for left string
            left_message= shift_left_twice(left_message);
            // calling to shift_left_twice function for right string
            right_message= shift_left_twice(right_message);
        }
    // combinning the left and right string
    string combined_key = left_message + right_message;
    // variable use to store key generated at particular phase
    string round_key = "";


// transpose the key bits byusing PC2
    for(int c = 0; c < 48; c++){
        //after transpose storing into round_key variable
        round_key += combined_key[pc_2[c]];
```

16 round:

16 round has many internal function.

- 32-bit string passing through expansion box
- Performing the x-or
- 48-bit data passing through substitution box
- Permute the right string
- Swapping the left and right string

32 bit string passes through expansion box

```
    // variable store the expended right string
    string right_expanded_string = "";
int expansion_table[48] = {
31,0,1,2,3,4,3,4,
5,6,7,8,7,8,9,10,
11,12,12,12,13,14,15,16,
15,16,17,18,19,20,19,20,
21,22,23,24,23,24,25,26,
27,28,27,28,29,30,31,0
};

//half right expansion of plane text to perform x-or between key and right string
    for(int a = 0; a < 48; a++) {
```

Performing X-or:

```
// Function to compute xor between two strings
string Xor(string a, string b){
    // variable use to store result
    string result = "";
    // size store the length of b string
    int size = b.size();
    //iterating over both string
    for(int i = 0; i < size; i++){
        // if both are not equal then result store the value 1(x-or properties)
        if(a[i] != b[i]){
            //result store the value 1
            result += "1";
        }
        else{
            // if both are equal then storing 0
            result += "0";
        }
```

Permute the right string:

```
string first_permutation_string2 ="";
// iterating over the permutation_tab
for(int a = 0; a< 32; a++){
    //performing permutation for each element
    first_permutation_string2 += final_res[permutation_tab[a]];
```

48 bit data passing through substitution boxes:

```
// dividing the result into eight parts and here passed substitution box and
//size is reduce to four bit from six bit
    for(int a=0;a<8; a++){
        // Finding row and column indices to lookup the
        // substituition box
        // calculation of row to use substitution box
        string row1= xored_res.substr(a*6,1) + xored_res.substr(a*6 + 5,1);
        //converting the row from binary string to decimal
        int row = convertBinaryToDecimal(row1);
        // calculation of column to use in substitution box
        string col1 = xored_res.substr(a*6 + 1,1) + xored_res.substr(a*6 + 2,1) + xored_res.substr(a*6 + 3,1) + xored_res.s
        //converting the binary value from binary to decimal
        int col = convertBinaryToDecimal(col1);
        //substitution occur
        int val = substition_boxes[a][row][col];
```

Swapping the left and right string:

```
    //Swap of left and right string
  if(a < 15){
      swap(left,right);
```

Inverse Permutation:

```
  // inverse_permutation table
 int inverse_permutation_table[64]= {
      39,7,47,15,55,23,63,31,
 38,6,46,14,54,22,62,30,
 37,5,45,13,53,21,61,29,
 36,4,44,12,52,20,60,28,
 35,3,43,11,51,19,59,27,
 34,2,42,10,50,18,58,26,
 33,1,41,9,49,17,57,25,
 32,0,40,8,48,16,56,24
 };
 // variable use to store cipher text
 string ciphertext_string ="";

erforming inverse permutation
 for(int w = 0; w< 64; w++){
```

In Modified DES , to calculate the avalanchel effect changed the following things:

- substitution box,
- initial_permutation box (also changed the Inverse_Intial_permutation table),
- X-or operation,
- block size

Plane_Text:
1110101010101010101010000000000001101010101010100000111111010100

Key:
1010101010111011000010010001100000100111001101101100110011011101

Output Of Standard_DES:

```
[Running] cd "c:\Users\DELL\Desktop\" && g++ abc.cpp -o abc && "c:\Users\DELL\Desktop\"abc
Original_message: 1110101010101010101010000000000001101010101010100000111111010100
Incrypted_message : 1010001000110100100001100010010000001001011101101101110001010010
Decrypted message:1110101010101010101010000000000001101010101010100000111111010100
```

Output Of Modified_DES:

```
[Running] cd "g:\Network Security\DES and AES\" && g++ DES.Avalance.cpp -o DES.Avalance && "g:\Network Security\DES and AE
Original_message: 1110101010101010101010000000000001101010101010100000111111010100
Incrypted_message : 1011001011101111110101001110100011000111110110101000101110110100
Decrypted message:1110101010101010101010000000000001101010101010100000111111010100
Plain text encryoted and decrypted perform successfully.
```

Cipher_Text in Standard_DES:

```
1010001000110100100001100010010000001001011101101101110001010010
```

Cipher_Text in Modified_DES:

```
1011001011101111110101001110100011000111110110101000101110110100
```

Avalanche Effect:

Total number of different bit=33

Avalanche Effect=(33/64)*100

=**51.5625**

Question1.(b) DES encryption was broken. What is your opinion about the vulnerable points in DES? DES was cracked, does it make the DES an unimportant encryption technique?

Ans:

In 1999 DES encryption was broken. Des has been vulnerable for sometimes. Security is very much important for preventing the cyber attack and once It was break then we should stop using this. DES is become weaker with time.

Even it is weak but it doesn't means it is unimportant. In future might be we came up with new algorithm which is more secure than AES and it could be modification of Standard DES.

Question2. Write a program to implement AES from scratch. Implementation involves the following four steps in each round:

- **byte-by-byte substitution**
- **shifting of the rows of the state array**
- **mixing of the columns**
- **the addition of the round key**

Modify AES implementation in terms of XOR, shift row, and column mixing. Calculate

the avalanche effect.

Ans:

AES algorithm:

- Symmetric key block cipher
- Fixed block size
- It is more secure than DES
- Number of bit in key depend upon round

Rounds	Number of bit in Key
10	128
12	192
14	256

Step In AES:

- Key Expansion according to the round
- Substitute Bytes
- Shift Rows
- Add Round Key

Key Expansion according to the round:

```
void fun_expansion_key(bitset<8> key[4*key_size_in_word], bitset<32>w[4*(total_round+1)])
{
    // a variable declaration of type bitset
    bitset<32> mal;
    // declaration and initialization of varible
    int lol= 0;

// starting four is input key as s
    while(lol < key_size_in_word)
    {
        // calling to converter_word function
        w[lol] = converter_word(key[4*lol], key[4*lol+1], key[4*lol+2], key[4*lol+3]);
        //increament the varible
        ++lol;
    }
  // storing the key size in lol variable
    lol= key_size_in_word;
    // iteration
    while(lol < 4*(total_round+1))
    {

        //Record the previous word
        mal = w[lol-1];
        //cheking position is divisible by key size or not
        if(lol % key_size_in_word == 0)
        {
        // if true the abolve condition the store into w[]
            w[lol] = w[lol-key_size_in_word] ^ substitution_fun_word(left_shift_by_one(mal)) ^ Round_constant[lol/key_s
        }
```

Substitute Bytes:

```
void substitution_byte_function(bitset<8> matrix[4*4])
{
   // taking a variable to iterate over susbstitution box
int w=0;
    while(w<16)
{
    // calculation of row
        int row = matrix[w][7]*8 + matrix[w][6]*4 + matrix[w][5]*2 + matrix[w][4];
        // calculation of column
        int col = matrix[w][3]*8 + matrix[w][2]*4 + matrix[w][1]*2 + matrix[w][0];
        //string into matrix according to row and column
        matrix[w] = substitution_box[row][col];
        //increament
        w++;
    }
}
```

Shift Rows:

```
void shifting_left_row_function(bitset<8> matrix[4*4])
{
// left shift by one of second row
    bitset<8> mal = matrix[4];
    // performing left shift operation
    for(int w=0; w<3; ++w)
        matrix[w+4] = matrix[w+5];
    matrix[7] = mal;

// move two places of third line of circle
    for(int w=0; w<2; ++w)
    {
        // taking a temp varible and store the element at position w+8
        mal = matrix[w+8];
        //w+8 position occupied by w+10
        matrix[w+8] = matrix[w+10];
        // w+10 position is filled by temp variable for shifting operation
        matrix[w+10] = mal;
    }
   // move three places of third line of circle
    mal = matrix[15];
    for(int u=3; u>0; --u)
    // storing u+11 element to u+12
        matrix[u+12] = matrix[u+11];
        //stroing temporory varible to at position 12
    matrix[12] = mal;
}
```

Mix Column:

```
// definition of mix_column function to perform mix_column operation
void mix_column_function(bitset<8> matrix[4*4])
{
    // declaration of array of size 4 of type bitset
    bitset<8> array[4];
    //iteration
    for(int a=0; a<4; ++a)
    {
        // internal iteration
        for(int b=0; b<4; ++b)
        // storing into array temp
            array[b] = matrix[a+b*4];
        // performing the relavant operation to perform mix_coumn function
        matrix[a] = multiplication_by_gf(0x02, array[0]) ^ multiplication_by_gf(0x03, array[1]) ^ arr
        // stroing at a+4 position
        matrix[a+4] = array[0] ^ multiplication_by_gf(0x02, array[1]) ^ multiplication_by_gf(0x03, ar
        // stroing at a+8 position
        matrix[a+8] = array[0] ^ array[1] ^ multiplication_by_gf(0x02, array[2]) ^ multiplication_by_
        // storing at a+12 position
```

Add Round_Key:

```
// definition of adding key function
void adding_round_key_function(bitset<8> matrix[4*4], bitset<32>ka
{
    // iteration
    for(int a=0; a<4; ++a)
    {
        // perform the right shift then store in kate1
        bitset<32> kate1 = kate[a] >> 24;
        //perform the left shift followed by right shift then stor
        bitset<32> kate2 = (kate[a] << 8) >> 24;
         //perform the left shift followed by right shift then sto
        bitset<32> kate3 = (kate[a] << 16) >> 24;
        //perform the left shift followed by right shift then stor
        bitset<32> kate4 = (kate[a] << 24) >> 24;
          // performing the x-or then store at position a
        matrix[a] = matrix[a] ^ bitset<8>(kate1.to_ulong());
        // performing the x-or then store at position a+4
        matrix[a+4] = matrix[a+4] ^ bitset<8>(kate2.to_ulong());
        // performing the x-or then store at position a+8
        matrix[a+8] = matrix[a+8] ^ bitset<8>(kate3.to_ulong());
```

In Modified AES , to calculate the Avalanche effect changed the following things:

- Shift Row
- Column mixing

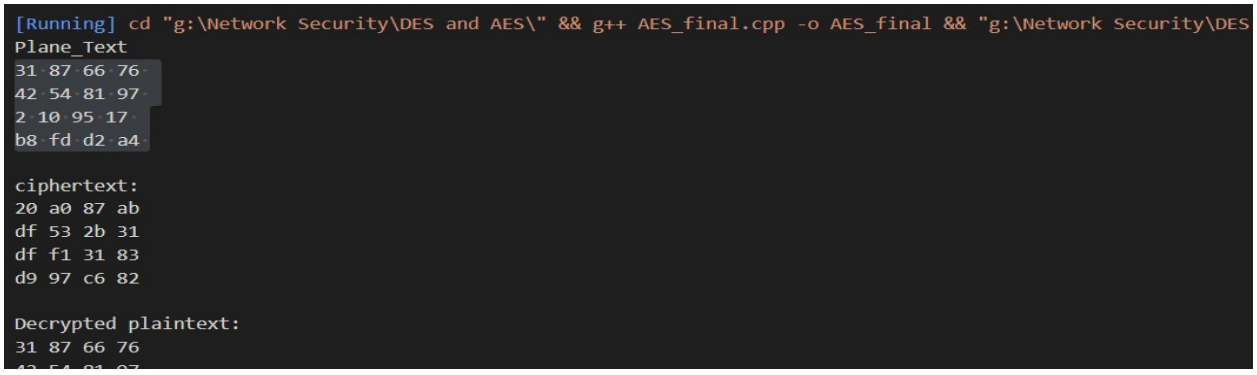
Plane Text In Hexadecimal:

```
31 87 66 76
42 54 81 97
2 10 95 17
b8 fd d2 a4
```

Key in Hexadecimal:

```
{0x22, 0x73, 0x95, 0x86,
 0x25, 0xfe, 0xf2, 0xa6,
 0x4b, 0x77, 0xa5, 0xe8,
 0x19, 0x3f, 0x45, 0x37};
```

Output In Standard AES:

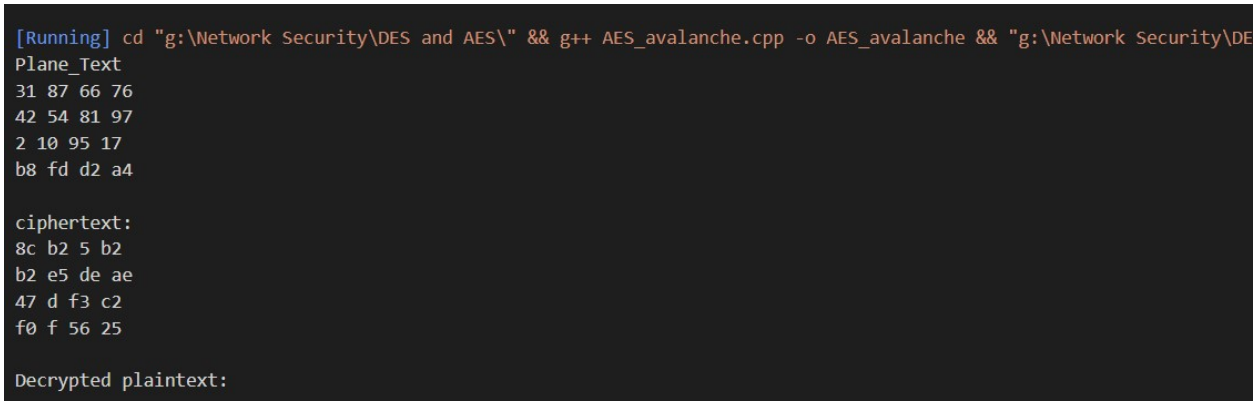


```
[Running] cd "g:\Network Security\DES and AES\" && g++ AES_final.cpp -o AES_final && "g:\Network Security\DES
Plane_Text
31 87 66 76
42 54 81 97
2 10 95 17
b8 fd d2 a4

ciphertext:
20 a0 87 ab
df 53 2b 31
df f1 31 83
d9 97 c6 82

Decrypted plaintext:
31 87 66 76
```

Output in Modified AES:



```
[Running] cd "g:\Network Security\DES and AES\" && g++ AES_avalanche.cpp -o AES_avalanche && "g:\Network Security\DE
Plane_Text
31 87 66 76
42 54 81 97
2 10 95 17
b8 fd d2 a4

ciphertext:
8c b2 5 b2
b2 e5 de ae
47 d f3 c2
f0 f 56 25

Decrypted plaintext:
```

Avalanche Effect :

Cipher_Text in Standard AES:

```
20 a0 87 ab

df 53 2b 31

df f1 31 83

d9 97 c6 82
```

Cipher Text In Modified AES:

```
8c b2 5 b2
b2 e5 de ae
47 d f3 c2
f0 f 56 25
```

Binary Representation of Standard AES Cipher Text:

00100000101000001000011110101011110111110101001100101011001100011
10111111111000100110001100000111101100110010111110001101000001 0

Binary Representation of Modified AES Cipher Text:

10001100101100100101101100101011001011100101110111101010111001000
11111011111001111000010111100001111010101100010010 1

Avalanche Effect:

Number of bit that are different=70

Avalanche effect=(70/128)*100

=54.6875

Thank You

Reference:

https://programmer.group/c-implementation-of-aes-encryption-algorithms.html#:~:text=AES%20algorithm%20(Rijndael%20algorithm)%20is,%2C%20%22AES%2D256%22.

https://www.educative.io/edpresso/how-to-implement-the-des-algorithm-in-cpp