0 | 1 Writeup

Name : 0 | 1

Description : All I can see is 0's and 1's

Difficulty : Hard

Points : 500

Domain : Cryptography

Author : Ankush Kaudi

Files Given : binary.zip which on extraction gives following

1) encrypt.c
2) up_down.py
3) justBinaries.txt
4) isThisFileUseful.txt

The text files have binary strings. Analyzing up_down.py, which is a simple program which replaces 1 to 0 and 0 to 1.

encrypt.c is a C program seems like an encryption algorithm.

Analyzing the code, we can find many details. To list a few

Some one dimensional arrays IP[], E[], p[], FP[] Two dimensional arrays from S1[] to S8[]

Arrays PC1[] and PC2[] and some more global arrays.

There are many functions few of them are

XOR() -> performs XOR operations

Functions to convert plaintext to binary string

Create16keys -> function to generate key from key.txt file

Encryption and encrypt function to encrypt the plain text

After analyzing all the function, we can conclude that the given encryption algorithm is a DES (Data Encryption Standards) algorithm which was widely used in past decade or before.

DES works by taking a 64-bit block of plaintext and applying a series of mathematical operations to it using a 56-bit secret key. The algorithm consists of 16 rounds of encryption, where each round applies a different combination of substitution and permutation operations to the input data.

Steps in DES :

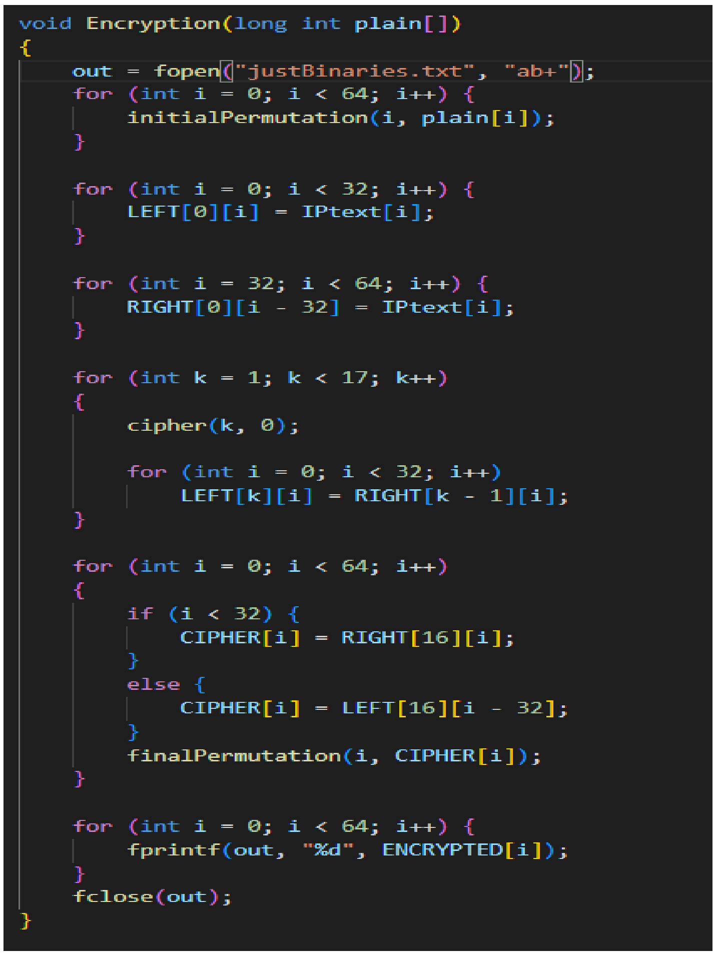
Key Generation: The 56-bit secret key is transformed into 16 round keys, each 48 bits long. This is done by performing a series of operations on the original key, including permutation, shifting, and compression.

Initial Permutation: The 64-bit plaintext block is permuted according to a fixed table to produce an initial permutation.

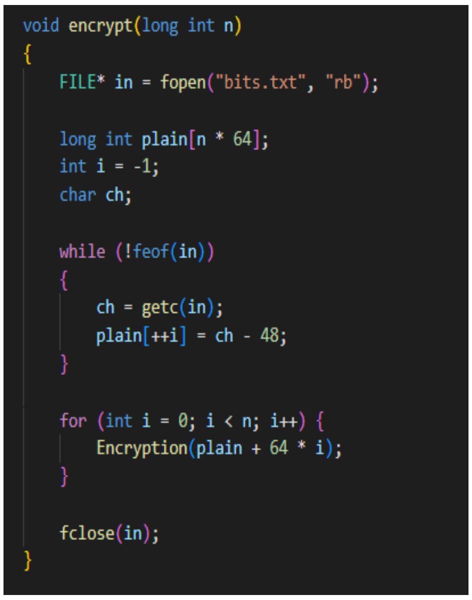
Data Encryption: The plaintext block is then divided into two 32-bit halves, and each half is processed through a series of 16 rounds of encryption. In each round, the right half is expanded to 48 bits and combined with the round key using XOR. The resulting 48-bit value is then processed through a series of substitution and permutation operations, which substitute and shuffle the bits to produce a new 32-bit value. The output of the substitution and permutation operations is then XORed with the left half of the plaintext block to produce the new right half.

Final Permutation: After all 16 rounds of encryption, the two 32-bit halves are swapped and combined to produce a 64-bit ciphertext block. This block is then permuted according to a fixed table to produce the final ciphertext.

To decrypt a message encrypted with DES, the same algorithm is applied in reverse order, using the same 56-bit secret key.



```c
void Encryption(long int plain[])
{
    out = fopen("justBinaries.txt", "ab+");
    for (int i = 0; i < 64; i++) {
        initialPermutation(i, plain[i]);
    }

    for (int i = 0; i < 32; i++) {
        LEFT[0][i] = IPtext[i];
    }

    for (int i = 32; i < 64; i++) {
        RIGHT[0][i - 32] = IPtext[i];
    }

    for (int k = 1; k < 17; k++)
    {
        cipher(k, 0);

        for (int i = 0; i < 32; i++)
            LEFT[k][i] = RIGHT[k - 1][i];
    }

    for (int i = 0; i < 64; i++)
    {
        if (i < 32) {
            CIPHER[i] = RIGHT[16][i];
        }
        else {
            CIPHER[i] = LEFT[16][i - 32];
        }
        finalPermutation(i, CIPHER[i]);
    }

    for (int i = 0; i < 64; i++) {
        fprintf(out, "%d", ENCRYPTED[i]);
    }
    fclose(out);
}
```



```c
void encrypt(long int n)
{
    FILE* in = fopen("bits.txt", "rb");

    long int plain[n * 64];
    int i = -1;
    char ch;

    while (!feof(in))
    {
        ch = getc(in);
        plain[++i] = ch - 48;
    }

    for (int i = 0; i < n; i++) {
        Encryption(plain + 64 * i);
    }

    fclose(in);
}
```

The above two functions encrypt the plaintext(which is the flag)

The challenge should have been developed as

1) The flag must have been encrypted using DES algorithm
2) The encrypted bits of binary string must have been flipped.

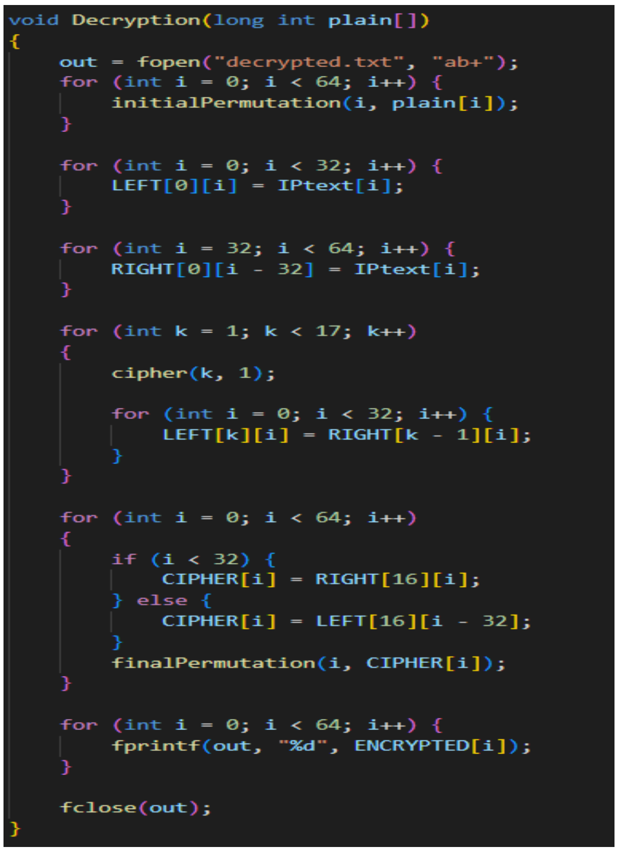
To solve this challenge, the given binary strings must be flipped again to obtain the actual encrypted flag and the key.

Encrypted flag (after flipping) :
0000101111101010100000000100011000001101100101111011100100100101100110100111
1100000000101100011100111111001110010000000001011110111001001001110100101
1011010011000000111100010000000011011011101010110101101110001111110101110111
0111000100111110000110010111111101100110110001101011111111010011001110000000
0111110111011101100101110110111101101001111001010101101101000101010101011100
01100111011101

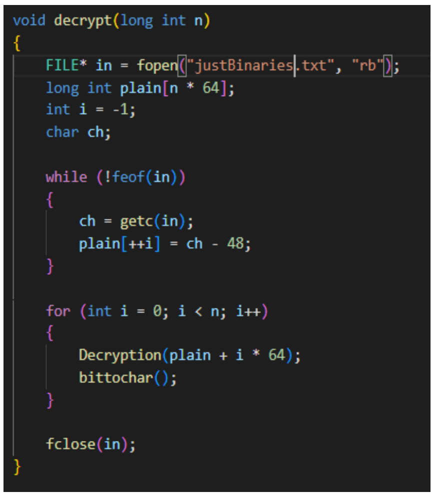
Key (after flipping) :
0110101001110101011100110111010001100001011010110110010101111001

Now, we have to reverse the process of encryption with the same parameters as it is.

So replacing the Encryption() and encrypt() function with the Decryption() and decrypt() functions will give out the flag. The Decryption() and decrypt() functions are as follows:



```c
void Decryption(long int plain[])
{
    out = fopen("decrypted.txt", "ab+");
    for (int i = 0; i < 64; i++) {
        initialPermutation(i, plain[i]);
    }

    for (int i = 0; i < 32; i++) {
        LEFT[0][i] = IPtext[i];
    }

    for (int i = 32; i < 64; i++) {
        RIGHT[0][i - 32] = IPtext[i];
    }

    for (int k = 1; k < 17; k++)
    {
        cipher(k, 1);

        for (int i = 0; i < 32; i++) {
            LEFT[k][i] = RIGHT[k - 1][i];
        }
    }

    for (int i = 0; i < 64; i++)
    {
        if (i < 32) {
            CIPHER[i] = RIGHT[16][i];
        } else {
            CIPHER[i] = LEFT[16][i - 32];
        }
        finalPermutation(i, CIPHER[i]);
    }

    for (int i = 0; i < 64; i++) {
        fprintf(out, "%d", ENCRYPTED[i]);
    }

    fclose(out);
}
```



```
void decrypt(long int n)
{
    FILE* in = fopen("justBinaries.txt", "rb");
    long int plain[n * 64];
    int i = -1;
    char ch;

    while (!feof(in))
    {
        ch = getc(in);
        plain[++i] = ch - 48;
    }

    for (int i = 0; i < n; i++)
    {
        Decryption(plain + i * 64);
        bittochar();
    }

    fclose(in);
}
```

After executing, the revised code the decrypted flag (in binary) will be stored in decrypted.txt. The decrypted binary string can be converted back to plain text. After converting the binary string to plain text, flag can be obtained.

Flag is : VishwaCTF{3v3ryth1ng_15_m3s5y_4r0und_h3r3....}