Q1 Team Name

0 Points

Cryptophilic

Q2 Commands

10 Points

List all the commands in sequence used from the start screen of this level to the end of the level. (Use -> to separate the commands)

enter -> jump -> jump -> back -> pull -> back -> back -> enter -> wave -> back -> thrnxxtzy -> read -> 1347215420976590298452739 57 -> c -> read -> password -> c

(After we got the wand, we went back to Level 3 to free the spirit, then entered all the commands to clear level 3 again. After that, we again got to level 4, where we could enter any plaintext and receive the corresponding encypted ciphertext)

Q3 CryptoSystem

5 Points

What cryptosystem was used at this level? Please be precise.

6-Round DES (Block Cipher)

Q4 Analysis

80 Points

Knowing which cryptosystem has been used at this level, give a detailed description of the cryptanalysis used to figure out the password. (Use Latex wherever required. If your solution is not readable, you will lose marks. If necessary, the file upload option in this question must be used TO SHARE IMAGES ONLY.)

First, we pulled the magic wand from the lake. Then we went back to Level 3 to free the spirit. After that, we came to level 4, and when we typed read, the spirit read from the glass panel. Since it said to type password, we got the ciphertext: 'gdpdgdkhhjsgnlmkkjogidkodkfphilf'. So we need to decrypt this.

The spirit gave us hints that it was encrypted with DES. As mentioned by the spirit, 4-Round DES would be very easy and 10-Round DES would be very difficult to break, so we assumed it to be 6-Round DES. (If 6-Round DES would fail, we would have then tried to break 4-Round DES). Since it was given that 2 letters represent 1 byte, and DES has a block size of 8 bytes, therefore 16 letters would represent 1 block size. Because only 16 letters can be represented by 4 bits, we would have to use plaintexts of 16 letters.

So we sent in enough random plaintexts and analyzed the outputs by frequency analysis, we found out that the 16 letters : 'd' to 's' were used in the outputs. So we mapped these letters to 0-15 using 4 bits:
(d- 0000, e- 0001, f- 0010, ..., t- 1110, s-1111).

Since we could get the encrypted ciphertext for any plaintext we sent, we decided to use **Chosen plaintext attack** using differential cryptanalysis to break this 6-Round DES. For 6-Round DES, we would need (r-2 = 4) 4-Round characteristic. We used the characteristic:

$$(405C\bar{0}, 0400\bar{0}, \tfrac{1}{4}, 0400\bar{0}, 0054\bar{0}, \tfrac{5}{128}, 0054\bar{0}, \bar{0}\bar{0}, 1, \bar{0}\bar{0}, 0054\bar{0}, \tfrac{5}{128}, 0054\bar{0}, 0400\bar{0})$$

as given in lectures. It's probability is $\frac{1}{4} \times \frac{5}{128} \times 1 \times \frac{5}{128} = 0.000381$, so we need approximately 1 lakh plaintext-ciphertext pairs.

Generating 1 lakh input-output pairs :

We need input pairs with XOR = $IP^{-1}(405C\bar{0}0400\bar{0}) = 0000901010005000$, so when we apply Initital Permutation on them, their XOR would be equal to our required value.

We use our code **Input-Output-gen.cpp** to generate and store 1 lakh such inputs in a file: 'gen_inputs.txt' , and their corresponding outputs in a file: 'gen_outputs.txt'.

Processing the Outputs :

After this step, we used the code **Output-processing.cpp**, which converted the outputs into their bit representation (using bit representation for each letter as stated above), used Initial Permutation on them to undo its inverse IP, and then swapped the left and right parts to generate the output after the Round 6 of DES.

Finding the Round 6 key : k_6

Notations for our analysis:
-> $L_k R_k$: output after k^{th} round of DES
-> R_5 and R'_5 : right halves of output after 5^{th} round of DES
-> For the 8 S-boxes :

-> $E(R_5) = \alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7\alpha_8 = \alpha$
$E(R'_5) = \alpha'_1\alpha'_2\alpha'_3\alpha'_4\alpha'_5\alpha'_6\alpha'_7\alpha'_8 = \alpha'$
where $|\alpha_i| = |\alpha'_i| = 6$
-> For L_0R_0 and $L'_0R'_0$: $L_0 \oplus L'_0 = 405C\bar{0}$ and $R_0 \oplus R'_0 = 04000\bar{0}$
-> $k_6 = k_{6,1}k_{6,2}k_{6,3}k_{6,4}k_{6,5}k_{6,6}k_{6,7}k_{6,8}$ (from the 8 S-boxes).
-> Let $\beta_i = \alpha_i \oplus k_{6,i}$; $\beta'_i = \alpha'_i \oplus k_{6,i}$ where $|\beta_i| = |\beta'_i| = 6$

If we have a given plaintext $L_0R_0, L'_0R'_0$ which has XOR value = $405C\bar{0}04000\bar{0}$, we can find its output after round 6 of DES : $L_6R_6, L'_6R'_6$.
$\because L_6 = R_5$, output of Round 6 expansion is : $E(R_5), E(R'_5)$;
$\because L_5 \oplus L'_5 = R_4 \oplus R'_4 = 04000\bar{0}$ (probability = 0.000381)
We can find $\gamma = \gamma_1\gamma_2\gamma_3\gamma_4\gamma_5\gamma_6\gamma_7\gamma_8 = S(\beta) \oplus S(\beta') = P^{-1}(L_5 \oplus L'_5 \oplus R_6 \oplus R'_6)$ with probability = 0.000381
$\therefore$ We have $\alpha_i, \alpha'_i, \beta_i \oplus \beta'_i (= \alpha_i \oplus \alpha'_i)$ and value of $\gamma = S(\beta) \oplus S(\beta')$ with probability = 0.000381

-> Using our code **Round6-KeyFinder.cpp** : we generated all possible (β_i, β'_i) for which $\beta_i \oplus \beta'_i = \alpha_i \oplus \alpha'_i$ and $S(\beta_i) \oplus S(\beta'_i) = \gamma_i$ for all the 8 S-boxes.
-> Then we added its corresponding key : $k = \alpha_i \oplus \beta_i$ to a hashmap where all possible $k_{6,i}$ were listed.
-> We did this for all 1 lakh pairs, and then selected the key with highest frequency as $k_{6,i}$.
-> We obtained these keys : 101101, 110011, 010101, 011010, 011011, 110110, 000101, 111110

We recorded the frequencies of the most-frequent and second most-frequent elements in the hashmap for the 8 S-boxes $(1 <= i <= 8)$:
i = 1 : 101101 - 8077, 6670
i = 2 : 110011 - 8494, 6641
i = 3 : 010101 - 6521, 6461
i = 4 : 011010 - 6438, 6400
i = 5 : 011011 - 9248, 6898
i = 6 : 110110 - 8532, 6714
i = 7 : 000101 - 8342, 6696
i = 8 : 111110 - 8612, 6820

Finding the Actual Key : Using our code **KeyFinder.cpp**

For i = 3 and i = 4 : difference between most-frequent and second most-frequent is not much. So we discarded $k_{6,3}$ and $k_{6,4}$, and mapped the other bits to the main key using key scheduling algorithm.

$\therefore$ Now we have 6 blocks out of the 8 blocks. So we have 36 bits (6 keys of size 6 bits) out of the required 56 bits.
-> Using a known plaintext : "qosppgjqphiporfs" and its corresponding ciphertext : "jispdnkkpngkqnii" , we used brute force to find all the 2^{20} possibilities for the remaining 20 bits.
-> After applying PC_1, we found the key as :
01101110010111100111101100001001011011111010010101010011

Decrypting the Password :

-> Since the spirit gave us hint that we need to type 'password' , we used its encrypted text : 'gdpdgdkhhjsgnlmkkjogidkodkfphilf'.
-> It is 128 bit long, so we divided it into two halves of 64 bits : 'gdpdgdkhhjsgnlmk' and 'kjogidkodkfphilf' .
-> As we know the key now, we decrypted both of them using out DES decryption algorithm.
-> When we converted the decrypted password to alphabets (4 bits for each alphabet) and entered it, the level was not cleared.
-> Then we assumed ASCII representation (8 bits for each alphabet) and got : "ovtskudfkc000000". Still, the level was not cleared.
-> Then we removed the zeroes at the end of 'ovtskudfkc000000' , because they would have been added to make text length a multiple of the size of the block. This time, the level was cleared.

-> So, we found our password to be : **ovtskudfkc**

References :
1) https://drive.google.com/drive/folders/1sDzYN7F-SWqDEzGybbKg6KSMzLwLFd2c?usp=sharing
2) https://medium.com/lotus-fruit/breaking-des-using-differential-cryptanalysis-958e8118ff41

No files uploaded

Q5 Password
5 Points

What was the password used to clear this level?

ovtskudfkc

Q6 Codes
0 Points

Unlike previous assignments, this time it is MANDATORY that you upload the codes used in the cryptanalysis. If you fail to do so, you will be given 0 marks for the entire assignment.

Input-Output-gen.cpp

Download

```cpp
#include <bits/stdc++.h>
using namespace std;

#define endl '\n'

mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int getRand(int l, int r)
{
    uniform_int_distribution<int> uid(l, r);
    return uid(rng);
}

void getINPUTS()
{
    uint64_t diff = 0x405c000004000000;

    ofstream fout("gen_inputs.txt");
    int IP[] = {
        58, 50, 42, 34, 26, 18, 10, 2,
        60, 52, 44, 36, 28, 20, 12, 4,
        62, 54, 46, 38, 30, 22, 14, 6,
        64, 56, 48, 40, 32, 24, 16, 8,
        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};

    uint64_t rdiff = 0;
    for (int i = 0; i < 64; i++)
    {
        uint64_t cur = ((diff >> (63 - i)) & 1);
        int to = IP[i];
        rdiff |= (cur << (64 - to));
    }

    cout << hex << setw(16) << setfill('0') << rdiff << endl;

    int L = (1e5/2);
    for (int i = 0; i < L; i++)
    {
        uint64_t first = 0;
        string s = "";
        for (int j = 0; j < 16; j++)
        {
            int k = getRand(0, 15);
            first = (first << 4) + k;
            s += char('d' + k);
        }
        fout << s << endl;
        s = "";
        uint64_t second = (first ^ rdiff);
        for (int j = 0; j < 16; j++)
        {
            int num = 0;
            for (int k = 0; k < 4; k++)
            {
                int at = 4 * j + k;
                int cur = ((second >> (63 - at)) & 1);
                num += (cur << (3 - k));
            }
            s += char('d' + num);
        }
        fout << s << endl;
    }
}

void getOUTPUTS()
{
    ifstream fin("gen_inputs.txt");
    ofstream fout("commands_record.txt");

    fout << "Cryptophilic" << endl;
    fout << "team3" << endl;
    fout << 4 << endl;

    fout << "read" << endl;

    string s;
    while (fin >> s)
    {
        fout << s << endl;
        fout << 'c' << endl;
    }

    fout << "back" << endl;
    fout << "exit" << endl;

    fout.close();
    fin.close();

    system("ssh students@172.27.26.188 < commands_record.txt > out");
    system("grep --no-group-separator -A 1 \"Slowly, a new text starts appearing on the screen. It reads ...\" out | grep --no-group-separator -v \"Slowly, a new text starts appearing on the screen. It reads ...\" | tr -d \"\\t\" > gen_outputs.txt");
}

int main()
{

    getINPUTS();
    getOUTPUTS();

    return 0;
}
```

Output-processing.cpp

Download

```cpp
#include <bits/stdc++.h>

using namespace std;

using bits64 = bitset<64>;
using bits32 = bitset<32>;
using bits48 = bitset<48>;
using bits56 = bitset<56>;
using bits28 = bitset<28>;

int IP[] = {
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    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};

int IP_inv[] = {
    40, 8, 48, 16, 56, 24, 64, 32,
    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};

bits64 apply_ip(bits64 in, bool inv = false)
{
    bits64 out;
    for (int i = 63; i >= 0; i--)
    {
        int j = (inv ? IP_inv[63 - i] : IP[63 - i]) - 1;
        out[i] = in[63 - j];
    }
    return out;
}

pair<bits32, bits32> get_LR(bits64 in)
{
    bits32 L, R;
    for (int i = 63, j = 31; i >= 32; i--, j--)
    {
        L[j] = in[i];
        R[j] = in[i - 32];
    }
    return make_pair(L, R);
}
bits64 join(bits32 L, bits32 R)
{
    bits64 out;
    for (int i = 31; i >= 0; i--)
    {
        out[i + 32] = L[i];
        out[i] = R[i];
    }
    return out;
}

int main()
{

    ifstream fin("gen_outputs.txt");
    ofstream fout("gen_out_bits.txt");

    string s;
    while (fin >> s)
    {
        assert((int)s.length() == 16);
        bits64 bits = 0;
        for (int i = 0; i < 16; i++)
        {
            char c = s[i];
            int num = c - 'd';
            assert(num < 16 and num >= 0);
            bits = (bits << 4) | bits64(num);
        }
        bits64 actual = apply_ip(bits);
        auto [R, L] = get_LR(actual);

        fout << L << R << endl;
    }

    return 0;
}
```

▾ Round6-KeyFinder.cpp Download

```cpp
#include <bits/stdc++.h>

using namespace std;

using bits64 = bitset<64>;
using bits32 = bitset<32>;
using bits48 = bitset<48>;
using bits56 = bitset<56>;
using bits28 = bitset<28>;

int IP[] = {
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    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};

int
```

```
int IP_inv[] = {
    40, 8, 48, 16, 56, 24, 64, 32,
    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};

bits64 apply_ip(bits64 in, bool inv = false)
{
    bits64 out;
    for (int i = 63; i >= 0; i--)
    {
        int j = (inv ? IP_inv[63 - i] : IP[63 - i]) - 1;
        out[i] = in[63 - j];
    }
    return out;
}

pair<bits32, bits32> get_LR(bits64 in)
{
    bits32 L, R;
    for (int i = 63, j = 31; i >= 32; i--, j--)
    {
        L[j] = in[i];
        R[j] = in[i - 32];
    }
    return make_pair(L, R);
}

int E[] = {
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32, 1};

bits48 apply_E(bits32 in)
{
    bits48 out;
    for (int i = 47; i >= 0; i--)
    {
        int j = E[47 - i] - 1;
        out[i] = in[31 - j];
    }
    return out;
}

int S1[4][16] = {
    14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
    0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
    4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
    15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13};

int S2[4][16] = {
    15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
    3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
    0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
    13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9};

int S3[4][16] = {
    10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
    13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
    13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
    1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12};

int S4[4][16] = {
    7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
    13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
    10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
    3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14};

int S5[4][16] = {
    2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
    14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
    4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
    11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3};

int S6[4][16] = {
    12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
    10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
    9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
    4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13};

int S7[4][16] = {
    4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
    13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
    1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
    6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12};

int S8[4][16] = {
    13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
    1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
    7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
    2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11};

bits32 apply_S(bits48 in)
{
    bits32 out;
    int i = 47, j = 31;
    for (auto S : {S1, S2, S3, S4, S5, S6, S7, S8})
    {
        int row = 0, col = 0;
        for (int k = i - 1; k > i - 5; k--)
        {
            col = col * 2 + in[k];
```

```
        }
        row = in[i] * 2 + in[i - 5];

        bitset<4> num = S[row][col];
        for (int k = j; k >= j - 3; k--)
        {
            out[k] = num[3 - (j - k)];
        }
        i -= 6;
        j -= 4;
    }
    return out;
}

int P[] = {
    16, 7, 20, 21, 29, 12, 28, 17,
    1, 15, 23, 26, 5, 18, 31, 10,
    2, 8, 24, 14, 32, 27, 3, 9,
    19, 13, 30, 6, 22, 11, 4, 25};

int P_inv[32];

void calc_P_inv()
{
    for (int i = 0; i < 32; i++)
    {
        int to = P[i] - 1;
        P_inv[to] = i + 1;
    }
}

bits32 apply_P(bits32 in, bool inv = false)
{
    bits32 out;
    for (int i = 31; i >= 0; i--)
    {
        int j = (inv ? P_inv[31 - i] : P[31 - i]) - 1;
        out[i] = in[31 - j];
    }
    return out;
}

using bits4 = bitset<4>;
using bits6 = bitset<6>;

bits4 calc(bits6 in, int i)
{
    bits4 out;
    int cur = 0;
    for (auto S : {S1, S2, S3, S4, S5, S6, S7, S8})
    {
        if (i == cur)
        {
            int row = 0, col = 0;
            for (int k = 4; k > 0; k--)
            {
                col = col * 2 + in[k];
            }
            row = in[5] * 2 + in[0];

            bitset<4> num = S[row][col];
            for (int k = 3; k >= 0; k--)
            {
                out[k] = num[k];
            }
            return out;
        }
        cur = cur + 1;
    }
    assert(false);
    return out;
}
#define endl '\n'

int main()
{
    ifstream fin("gen_out_bits.txt");
    calc_P_inv();

    int N = 1e5;

    unordered_map<bitset<6>, int> maybe[8];

    for (int _ = 0; _ < N; _++)
    {
        bitset<32> out1L, out1R, out2L, out2R;
        fin >> out1L >> out1R >> out2L >> out2R;

        // outputs after expansion in the last round
        bitset<48> out1E = apply_E(out1L);
        bitset<48> out2E = apply_E(out2L);
        bitset<48> inxorS = (out1E ^ out2E);

        // cout << inxorS << endl;

        bitset<32> xorL = bitset<32>(0x04000000);
        bitset<32> xorP = (out1R ^ out2R ^ xorL);
        bitset<32> outxorS = apply_P(xorP, true);

        for (int i = 0; i < 8; i++)
        {
            int st = i * 6;
            for (int j = 0; j < (1 << 6); j++)
            {
                bitset<6> in1 = j;
                bitset<6> in2 = 0;
                for (int k = st; k - st < 6; k++)
                {
                    in2[5 - k + st] = (in1[5 - k + st] ^ inxorS[47 - k]);
                }
                bitset<4> out1 = calc(in1, i);
```

```
                bitset<4> out2 = calc(in2, i);
                bitset<4> got = (out1 ^ out2);
                bool ok = 1;
                for (int k = i * 4; k - i * 4 < 4; k++)
                {
                    ok &= (outxorS[31 - k] == got[3 - k + i * 4]);
                }

                if (ok)
                {
                    bitset<6> cand;
                    for (int k = st; k - st < 6; k++)
                    {
                        cand[5 - k + st] = in1[5 - k + st] ^ out1E[47 - k];
                    }
                    maybe[i][cand]++;
                }
            }
        }
    }

    bitset<48> key = 0;

    for (int i = 0; i < 8; i++)
    {
        int cur_mx = 0;
        bitset<6> ans;
        vector<int> cnts;
        for (auto [bs, cnt] : maybe[i])
        {
            cnts.push_back(cnt);
            if (cnt > cur_mx)
            {
                cur_mx = cnt;
                ans = bs;
            }
        }

        sort(cnts.begin(), cnts.end());
        reverse(cnts.begin(), cnts.end());

        key = (key << 6);
        for (int j = 0; j < 6; j++)
        {
            key[j] = ans[j];
        }

        cout << i + 1 << ' ' << ans << ' ' << cnts[0] << ' ' << cnts[1] << endl;
    }

    cout << key << endl;

    return 0;
}
```

KeyFinder.cpp

Download

```
#include <bits/stdc++.h>

using namespace std;

using bits64 = bitset<64>;
using bits32 = bitset<32>;
using bits48 = bitset<48>;
using bits56 = bitset<56>;
using bits28 = bitset<28>;

int IP[] = {
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    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};

int IP_inv[] = {
    40, 8, 48, 16, 56, 24, 64, 32,
    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};

bits64 apply_ip(bits64 in, bool inv = false)
{
    bits64 out;
    for (int i = 63; i >= 0; i--)
    {
        int j = (inv ? IP_inv[63 - i] : IP[63 - i]) - 1;
        out[i] = in[63 - j];
    }
    return out;
}

pair<bits32, bits32> get_LR(bits64 in)
{
    bits32 L, R;
    for (int i = 63, j = 31; i >= 32; i--, j--)
    {
        L[j] = in[i];
        R[j] = in[i - 32];
    }
    return make_pair(L, R);
}

int E[] = {
```

```
    32, 1, 2, 3, 4, 5,
    4, 5, 6, 7, 8, 9,
    8, 9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32, 1};

bits48 apply_E(bits32 in)
{
    bits48 out;
    for (int i = 47; i >= 0; i--)
    {
        int j = E[47 - i] - 1;
        out[i] = in[31 - j];
    }
    return out;
}

int S1[4][16] = {
    14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
    0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
    4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
    15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13};

int S2[4][16] = {
    15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
    3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
    0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
    13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9};

int S3[4][16] = {
    10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
    13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
    13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
    1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12};

int S4[4][16] = {
    7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
    13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
    10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
    3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14};

int S5[4][16] = {
    2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
    14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
    4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
    11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3};

int S6[4][16] = {
    12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
    10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
    9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
    4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13};

int S7[4][16] = {
    4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
    13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
    1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
    6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12};

int S8[4][16] = {
    13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
    1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
    7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
    2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11};

bits32 apply_S(bits48 in)
{
    bits32 out;
    int i = 47, j = 31;
    for (auto S : {S1, S2, S3, S4, S5, S6, S7, S8})
    {
        int row = 0, col = 0;
        for (int k = i - 1; k > i - 5; k--)
        {
            col = col * 2 + in[k];
        }
        row = in[i] * 2 + in[i - 5];

        bitset<4> num = S[row][col];
        for (int k = j; k >= j - 3; k--)
        {
            out[k] = num[3 - (j - k)];
        }
        i -= 6;
        j -= 4;
    }
    return out;
}

int P[] = {
    16, 7, 20, 21, 29, 12, 28, 17,
    1, 15, 23, 26, 5, 18, 31, 10,
    2, 8, 24, 14, 32, 27, 3, 9,
    19, 13, 30, 6, 22, 11, 4, 25};

int P_inv[32];

void calc_P_inv()
{
    for (int i = 0; i < 32; i++)
    {
        int to = P[i] - 1;
        P_inv[to] = i + 1;
    }
}

bits32 apply_P(bits32 in, bool inv = false)
{
```

```
    bits32 out;
    for (int i = 31; i >= 0; i--)
    {
        int j = (inv ? P_inv[31 - i] : P[31 - i]) - 1;
        out[i] = in[31 - j];
    }
    return out;
}

using vec64 = array<int, 64>;
using vec32 = array<int, 32>;
using vec48 = array<int, 48>;
using vec56 = array<int, 56>;
using vec28 = array<int, 28>;

int PC_1[] = {
    57, 49, 41, 33, 25, 17, 9,
    1, 58, 50, 42, 34, 26, 18,
    10, 2, 59, 51, 43, 35, 27,
    19, 11, 3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7, 62, 54, 46, 38, 30, 22,
    14, 6, 61, 53, 45, 37, 29,
    21, 13, 5, 28, 20, 12, 4};

unsigned short shifts[] = {
    1, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1};

int PC_2[] = {
    14, 17, 11, 24, 1, 5,
    3, 28, 15, 6, 21, 10,
    23, 19, 12, 4, 26, 8,
    16, 7, 27, 20, 13, 2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32};

vector<vec48> keys(16);

void getKEYS()
{
    vec56 key;
    for (int i = 55; i >= 0; i--)
    {
        key[i] = 56 - i;
    }

    vec28 C, D;
    for (int i = 55, j = 27; i >= 28; i--, j--)
    {
        C[j] = key[i];
        D[j] = key[i - 28];
    }

    for (int r = 0; r < 16; r++)
    {
        vec28 rC, rD;
        int s = shifts[r];
        for (int i = 27; i >= 0; i--)
        {
            int to = (i + s) % 28;
            rC[to] = C[i];
            rD[to] = D[i];
        }
        C = rC, D = rD;

        for (int i = 47; i >= 0; i--)
        {
            int j = PC_2[47 - i] - 1;
            j = 55 - j;
            if (j > 27)
            {
                keys[r][i] = C[j - 28];
            }
            else
            {
                keys[r][i] = D[j];
            }
        }
    }
}

void print(vec48 key)
{
    for (int i = 47; i >= 0; i -= 6)
    {
        int num = 0;
        for (int j = i; j >= i - 5; j--)
        {
            num = num * 2 + key[j];
        }
        cout << setw(2) << setfill('0') << hex << num;
    }
    cout << " ";
}

bits64 join(bits32 L, bits32 R)
{
    bits64 out;
    for (int i = 31; i >= 0; i--)
    {
        out[i + 32] = L[i];
        out[i] = R[i];
    }
    return out;
}

bits64 get(string s)
{
    assert(s.length() == 16);
```

```
    bits64 ans = 0;
    for (int i = 0; i < 16; i++)
    {
        int foo = s[i] - 'd';
        ans = (ans << 4) | bits64(foo);
    }
    return ans;
}

vector<bits48> rkeys(16);
void getKEYS2(vector<int> key)
{
    for (int r = 0; r < 16; r++)
    {
        for (int i = 47; i >= 0; i--)
        {
            int j = keys[r][i] - 1;
            rkeys[r][i] = key[55 - j];
        }
    }
}

int main()
{
    getKEYS();
    bits48 key_r6 = bits48("101101110011010101011010011011110110000101111110");

    vector<int> key(56, -1);
    for (int i = 47, cur = 0; i >= 0; i -= 6, cur++)
    {
        if (cur == 2 or cur == 3)
            continue;
        for (int j = i; j >= i - 5; j--)
        {
            int k = keys[5][j] - 1;
            key[55 - k] = key_r6[j];
        }
    }

    auto doDES = [&](bits64 in, bool inv = false) {
        bits64 LR = apply_ip(in);
        auto [L, R] = get_LR(LR);
        for (int r = 0; r < 6; r++)
        {
            auto ER = apply_E(R);
            ER ^= (inv ? rkeys[5 - r] : rkeys[r]);
            auto SR = apply_S(ER);
            auto PR = apply_P(SR);
            auto nL = R, nR = L ^ PR;
            L = nL, R = nR;
        }
        auto O = join(R, L);
        auto out = apply_ip(O, true);
        return out;
    };

    vector<int> pos;
    int cnt = 0;
    for (int i = 55; i >= 0; i--)
    {
        if (key[i] == -1)
            pos.push_back(i);
        cnt += (key[i] == -1);
    }

    calc_P_inv();

    bits64 in = get("qosppgjqphiporfs");
    bits64 out = get("jispdnkkpngkqnii");

    for (int i = 0; i < (1 << cnt); i++)
    {
        for (int j = 0; j < cnt; j++)
        {
            if ((i & (1 << j)) != 0)
            {
                key[pos[j]] = 1;
            }
            else
                key[pos[j]] = 0;
        }
        getKEYS2(key);
        if (doDES(in) == out)
        {
            cout << "The key is : " << endl;
            for (int k = 55; k >= 0; k--)
            {
                cout << key[k];
            }
            cout << endl;
            break;
        }
    }

    bits64 in21 = get("gdpdgdkhhjsgnlmk");
    bits64 in22 = get("kjogidkodkfphilf");

    for (auto in2 : {in21, in22})
    {
        auto out2 = doDES(in2, true);
        for (int i = 63; i >= 0; i -= 8)
        {
            int num = 0;
            for (int j = i; j >= i - 7; j--)
            {
                num = num * 2 + out2[j];
            }
            cout << char(num);
        }
    }
    cout << endl;
```

```
387
388     return 0;
389 }
```

Select a question.

Group M