



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 -> thrmxxzy -> read -> 134721542097659029845273957 -> 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 encrypted 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: 'gdpdgdkhhjsgnlmkkjogidkodkfphlf'. 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:

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

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 Initial 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 Shows .

... or the 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 $R_0 \oplus R'_0 = 0400\bar{0}$
- > $k_6 = k_{6,1}k_{6,2}k_{6,3}k_{6,4}k_{6,5}k_{6,6}k_{6,7}k_{6,8}$ (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 = $405C\bar{0}0400\bar{0}$, we can find its output after round 6 of DES : $L_6R_6, L'_6R'_6$.

- > $L_6 = R_5$, output of Round 6 expansion is : $E(R_5), E(R'_5)$;
- > $L_5 \oplus L'_5 = R_4 \oplus R'_4 = 0400\bar{0}$ (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
- > 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 ($1 \leq i \leq 8$) :

```
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.

- > 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 : "jispdnkpngkqnii", 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 :

011011110010111100111101100001001011011111010010101010011

Decrypting the Password :

- > Since the spirit gave us hint that we need to type 'password' , we used its encrypted text : 'gdpdgdkhhjsgnlmkkjogidkodkfphifl'.
- > It is 128 bit long, so we divided it into two halves of 64 bits : 'gdpdgdkhhjsgnlmk' and 'kjogidkodkfphifl' .
- > 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](#)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define endl '\n'
5
6 mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
7 int getRand(int l, int r)
8 {
9     uniform_int_distribution<int> uid(l, r);
10    return uid(rng);
11 }
12
13 void getINPUTS()
14 {
15     uint64_t diff = 0x405c000004000000;
16
17     ofstream fout("gen_inputs.txt");
18     int IP[] = {
19         58, 50, 42, 34, 26, 18, 10, 2,
20         60, 52, 44, 36, 28, 20, 12, 4,
21         62, 54, 46, 38, 30, 22, 14, 6,
22         64, 56, 48, 40, 32, 24, 16, 8,
23         57, 49, 41, 33, 25, 17, 9, 1,
24         59, 51, 43, 35, 27, 19, 11, 3,
25         61, 53, 45, 37, 29, 21, 13, 5,
26         63, 55, 47, 39, 31, 23, 15, 7};
27
28     uint64_t rdiff = 0;
29     for (int i = 0; i < 64; i++)
30     {
31         uint64_t cur = ((diff >> (63 - i)) & 1);
32         int to = IP[i];
33         rdiff |= (cur << (64 - to));
34     }
35
36     cout << hex << setw(16) << setfill('0') << rdiff << endl;
37
38     int L = (1e5/2);
39     for (int i = 0; i < L; i++)
40     {
41         uint64_t first = 0;
42         string s = "";
43         for (int j = 0; j < 16; j++)
44         {
45             int k = getRand(0, 15);
46             first = (first << 4) + k;
47             s += char('d' + k);
48         }
49         fout << s << endl;
50         s = "";
51         uint64_t second = (first ^ rdiff);
52         for (int j = 0; j < 16; j++)
53         {
54             int num = 0;
55             for (int k = 0; k < 4; k++)
56             {
57                 int at = 4 * j + k;
58                 int cur = ((second >> (63 - at)) & 1);
59                 num += (cur << (3 - k));
60             }
61             s += char('d' + num);
62         }
63         fout << s << endl;
64     }
65 }
66
67 void getOUTPUTS()
68 {
69     ifstream fin("gen_inputs.txt");
70     ofstream fout("commands_record.txt");
71
72     fout << "Cryptophilic" << endl;
73     fout << "team3" << endl;
74     fout << 4 << endl;
75
76     fout << "read" << endl;
77
78     string s;
79     while (fin >> s)
80     {
81         fout << s << endl;
82         fout << 'c' << endl;
83     }
84
85     fout << "back" << endl;
86     fout << "exit" << endl;
87
88     fout.close();
89     fin.close();
90
91     system("ssh students@172.27.26.188 < commands_record.txt > out");
92     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");
93 }
94
95 int main()
96 {
97
98     getINPUTS();
99     getOUTPUTS();
100
101     return 0;
102 }
```

▼ Output-processing.cpp

[Download](#)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 using bits64 = bitset<64>;
6 using bits32 = bitset<32>;
7 using bits48 = bitset<48>;
8 using bits56 = bitset<56>;
9 using bits28 = bitset<28>;
10
11 int IP[] = {
12     58, 50, 42, 34, 26, 18, 10, 2,
13     60, 52, 44, 36, 28, 20, 12, 4,
14     62, 54, 46, 38, 30, 22, 14, 6,
15     64, 56, 48, 40, 32, 24, 16, 8,
16     57, 49, 41, 33, 25, 17, 9, 1,
17     59, 51, 43, 35, 27, 19, 11, 3,
18     61, 53, 45, 37, 29, 21, 13, 5,
19     63, 55, 47, 39, 31, 23, 15, 7};
20
21 int IP_inv[] = {
22     40, 8, 48, 16, 56, 24, 64, 32,
23     39, 7, 47, 15, 55, 23, 63, 31,
24     38, 6, 46, 14, 54, 22, 62, 30,
25     37, 5, 45, 13, 53, 21, 61, 29,
26     36, 4, 44, 12, 52, 20, 60, 28,
27     35, 3, 43, 11, 51, 19, 59, 27,
28     34, 2, 42, 10, 50, 18, 58, 26,
29     33, 1, 41, 9, 49, 17, 57, 25};
30
31 bits64 apply_ip(bits64 in, bool inv = false)
32 {
33     bits64 out;
34     for (int i = 63; i >= 0; i--)
35     {
36         int j = (inv ? IP_inv[63 - i] : IP[63 - i]) - 1;
37         out[i] = in[63 - j];
38     }
39     return out;
40 }
41
42 pair<bits32, bits32> get_LR(bits64 in)
43 {
44     bits32 L, R;
45     for (int i = 63, j = 31; i >= 32; i--, j--)
46     {
47         L[j] = in[i];
48         R[j] = in[i - 32];
49     }
50     return make_pair(L, R);
51 }
52 bits64 join(bits32 L, bits32 R)
53 {
54     bits64 out;
55     for (int i = 31; i >= 0; i--)
56     {
57         out[i + 32] = L[i];
58         out[i] = R[i];
59     }
60     return out;
61 }
62
63 int main()
64 {
65
66     ifstream fin("gen_outputs.txt");
67     ofstream fout("gen_out_bits.txt");
68
69     string s;
70     while (fin >> s)
71     {
72         assert((int)s.length() == 16);
73         bits64 bits = 0;
74         for (int i = 0; i < 16; i++)
75         {
76             char c = s[i];
77             int num = c - 'd';
78             assert(num < 16 and num >= 0);
79             bits = (bits << 4) | bits64(num);
80         }
81         bits64 actual = apply_ip(bits);
82         auto [R, L] = get_LR(actual);
83
84         fout << L << R << endl;
85     }
86
87     return 0;
88 }

```

▼ Round6-KeyFinder.cpp

[Download](#)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 using bits64 = bitset<64>;
6 using bits32 = bitset<32>;
7 using bits48 = bitset<48>;
8 using bits56 = bitset<56>;
9 using bits28 = bitset<28>;
10
11 int IP[] = {
12     58, 50, 42, 34, 26, 18, 10, 2,
13     60, 52, 44, 36, 28, 20, 12, 4,
14     62, 54, 46, 38, 30, 22, 14, 6,
15     64, 56, 48, 40, 32, 24, 16, 8,
16     57, 49, 41, 33, 25, 17, 9, 1,
17     59, 51, 43, 35, 27, 19, 11, 3,
18     61, 53, 45, 37, 29, 21, 13, 5,
19     63, 55, 47, 39, 31, 23, 15, 7};
20
21

```

```

41     int IP_INV[1] = i
42     40, 8, 48, 16, 56, 24, 64, 32,
43     39, 7, 47, 15, 55, 23, 63, 31,
44     38, 6, 46, 14, 54, 22, 62, 30,
45     37, 5, 45, 13, 53, 21, 61, 29,
46     36, 4, 44, 12, 52, 20, 60, 28,
47     35, 3, 43, 11, 51, 19, 59, 27,
48     34, 2, 42, 10, 50, 18, 58, 26,
49     33, 1, 41, 9, 49, 17, 57, 25};
50
51
52     bits64 apply_ip(bits64 in, bool inv = false)
53 {
54     bits64 out;
55     for (int i = 63; i >= 0; i--)
56     {
57         int j = (inv ? IP_inv[63 - i] : IP[63 - i]) - 1;
58         out[i] = in[63 - j];
59     }
60     return out;
61 }
62
63     pair<bits32, bits32> get_LR(bits64 in)
64 {
65     bits32 L, R;
66     for (int i = 63, j = 31; i >= 32; i--, j--)
67     {
68         L[j] = in[i];
69         R[j] = in[i - 32];
70     }
71     return make_pair(L, R);
72 }
73
74     int E[] = {
75     32, 1, 2, 3, 4, 5,
76     4, 5, 6, 7, 8, 9,
77     8, 9, 10, 11, 12, 13,
78     12, 13, 14, 15, 16, 17,
79     16, 17, 18, 19, 20, 21,
80     20, 21, 22, 23, 24, 25,
81     24, 25, 26, 27, 28, 29,
82     28, 29, 30, 31, 32, 1};
83
84     bits48 apply_E(bits32 in)
85 {
86     bits48 out;
87     for (int i = 47; i >= 0; i--)
88     {
89         int j = E[47 - i] - 1;
90         out[i] = in[31 - j];
91     }
92     return out;
93 }
94
95     int S1[4][16] = {
96     14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
97     0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
98     4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
99     15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13};
100
101    int S2[4][16] = {
102     15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
103     3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
104     0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
105     13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9};
106
107    int S3[4][16] = {
108     10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
109     13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
110     13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
111     1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12};
112
113    int S4[4][16] = {
114     10, 0, 9, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
115     13, 8, 11, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
116     10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
117     3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14};
118
119    int S5[4][16] = {
120     2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
121     14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
122     4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
123     11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3};
124
125    int S6[4][16] = {
126     12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
127     10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
128     9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
129     4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13};
130
131    int S7[4][16] = {
132     4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
133     13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
134     1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
135     6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12};
136
137    int S8[4][16] = {
138     13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
139     1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
140     7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
141     2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11};
142
143    bits32 apply_S(bits48 in)
144 {
145     bits32 out;
146     int i = 47, j = 31;
147     for (auto S : {S1, S2, S3, S4, S5, S6, S7, S8})
148     {
149         int row = 0, col = 0;
150         for (int k = i - 1; k > i - 5; k--)
151         {
152             col = col * 2 + in[k];

```

```

132     }
133     row = in[i] * 2 + in[i - 5];
134
135     bitset<4> num = S[row][col];
136     for (int k = j; k >= j - 3; k--)
137     {
138         out[k] = num[3 - (j - k)];
139     }
140     i -= 6;
141     j -= 4;
142 }
143 return out;
144 }
145
146 int P[] = {
147     16, 7, 20, 21, 29, 12, 28, 17,
148     1, 15, 23, 26, 5, 18, 31, 10,
149     2, 8, 24, 14, 32, 27, 3, 9,
150     19, 13, 30, 6, 22, 11, 4, 25};
151
152 int P_inv[32];
153
154 void calc_P_inv()
155 {
156     for (int i = 0; i < 32; i++)
157     {
158         int to = P[i] - 1;
159         P_inv[to] = i + 1;
160     }
161 }
162
163 bits32 apply_P(bits32 in, bool inv = false)
164 {
165     bits32 out;
166     for (int i = 31; i >= 0; i--)
167     {
168         int j = (inv ? P_inv[31 - i] : P[31 - i]) - 1;
169         out[i] = in[31 - j];
170     }
171     return out;
172 }
173
174 using bits4 = bitset<4>;
175 using bits6 = bitset<6>;
176
177 bits4 calc(bits6 in, int i)
178 {
179     bits4 out;
180     int cur = 0;
181     for (auto S : {S1, S2, S3, S4, S5, S6, S7, S8})
182     {
183         if (i == cur)
184         {
185             int row = 0, col = 0;
186             for (int k = 4; k > 0; k--)
187             {
188                 col = col * 2 + in[k];
189             }
190             row = in[5] * 2 + in[0];
191
192             bitset<4> num = S[row][col];
193             for (int k = 3; k >= 0; k--)
194             {
195                 out[k] = num[k];
196             }
197             return out;
198         }
199         cur = cur + 1;
200     }
201     assert(false);
202     return out;
203 }
204 #define endl '\n'
205
206 int main()
207 {
208     ifstream fin("gen_out_bits.txt");
209     calc_P_inv();
210
211     int N = 1e5;
212
213     unordered_map<bitset<6>, int> maybe[8];
214
215     for (int _ = 0; _ < N; _++)
216     {
217         bitset<32> out1L, out1R, out2L, out2R;
218         fin >> out1L >> out1R >> out2L >> out2R;
219
220         // outputs after expansion in the last round
221         bitset<48> out1E = apply_E(out1L);
222         bitset<48> out2E = apply_E(out2L);
223         bitset<48> inxorS = (out1E ^ out2E);
224
225         // cout << inxorS << endl;
226
227         bitset<32> xorL = bitset<32>(0x04000000);
228         bitset<32> xorP = (out1R ^ out2R ^ xorL);
229         bitset<32> outxorS = apply_P(xorP, true);
230
231         for (int i = 0; i < 8; i++)
232         {
233             int st = i * 6;
234             for (int j = 0; j < (1 << 6); j++)
235             {
236                 bitset<6> in1 = j;
237                 bitset<6> in2 = 0;
238                 for (int k = st; k - st < 6; k++)
239                 {
240                     in2[5 - k + st] = (in1[5 - k + st] ^ inxorS[47 - k]);
241                 }
242                 bitset<4> out1 = calc(in1, i);
243
244                 if (out1 != maybe[i][j])
245                 {
246                     cout << "Error at index " << i << ", " << j << endl;
247                     cout << "Expected: " << out1 << endl;
248                     cout << "Actual: " << maybe[i][j] << endl;
249                     cout << "Input: " << in1 << endl;
250                     cout << "Input2: " << in2 << endl;
251                     cout << "InxorS: " << inxorS << endl;
252                     cout << "OutxorS: " << outxorS << endl;
253                     cout << "Out1: " << out1 << endl;
254                     cout << "Out2: " << out2 << endl;
255                     cout << "Out1E: " << out1E << endl;
256                     cout << "Out2E: " << out2E << endl;
257                     cout << "XorL: " << xorL << endl;
258                     cout << "XorP: " << xorP << endl;
259                     cout << "XorS: " << inxorS << endl;
260                     cout << "OutxorS: " << outxorS << endl;
261                     cout << endl;
262                 }
263             }
264         }
265     }
266 }
```

```

243         bitset<4> out2 = calc(in2, i);
244         bitset<4> got = (out1 ^ out2);
245         bool ok = 1;
246         for (int k = i * 4; k - i * 4 < 4; k++)
247         {
248             ok &= (outxorS[31 - k] == got[3 - k + i * 4]);
249         }
250         if (ok)
251         {
252             bitset<6> cand;
253             for (int k = st; k - st < 6; k++)
254             {
255                 cand[5 - k + st] = in1[5 - k + st] ^ out1E[47 - k];
256             }
257             maybe[i][cand]++;
258         }
259     }
260 }
261 }
262 }
263
bitset<48> key = 0;
265
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;
        }
    }
280
sort(cnts.begin(), cnts.end());
reverse(cnts.begin(), cnts.end());
283
key = (key << 6);
for (int j = 0; j < 6; j++)
{
    key[j] = ans[j];
}
289
cout << i + 1 << ' ' << ans << ' ' << cnts[0] << ' ' << cnts[1] << endl;
291 }
292
293 cout << key << endl;
294
295 return 0;
296 }
297

```

▼ KeyFinder.cpp

[Download](#)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 using bits64 = bitset<64>;
6 using bits32 = bitset<32>;
7 using bits48 = bitset<48>;
8 using bits56 = bitset<56>;
9 using bits28 = bitset<28>;
10
11 int IP[] = {
12     58, 50, 42, 34, 26, 18, 10, 2,
13     60, 52, 44, 36, 28, 20, 12, 4,
14     62, 54, 46, 38, 30, 22, 14, 6,
15     64, 56, 48, 40, 32, 24, 16, 8,
16     57, 49, 41, 33, 25, 17, 9, 1,
17     59, 51, 43, 35, 27, 19, 11, 3,
18     61, 53, 45, 37, 29, 21, 13, 5,
19     63, 55, 47, 39, 31, 23, 15, 7};
20
21 int IP_inv[] = {
22     40, 8, 48, 16, 56, 24, 64, 32,
23     39, 7, 47, 15, 55, 23, 63, 31,
24     38, 6, 46, 14, 54, 22, 62, 30,
25     37, 5, 45, 13, 53, 21, 61, 29,
26     36, 4, 44, 12, 52, 20, 60, 28,
27     35, 3, 43, 11, 51, 19, 59, 27,
28     34, 2, 42, 10, 50, 18, 58, 26,
29     33, 1, 41, 9, 49, 17, 57, 25};
30
31 bits64 apply_ip(bits64 in, bool inv = false)
32 {
33     bits64 out;
34     for (int i = 63; i >= 0; i--)
35     {
36         int j = (inv ? IP_inv[63 - i] : IP[63 - i]) - 1;
37         out[i] = in[63 - j];
38     }
39     return out;
40 }
41
42 pair<bits32, bits32> get_LR(bits64 in)
43 {
44     bits32 L, R;
45     for (int i = 63, j = 31; i >= 32; i--, j--)
46     {
47         L[j] = in[i];
48         R[j] = in[i - 32];
49     }
50     return make_pair(L, R);
51 }
52
53 int E[] = {

```

```

54     32, 1, 2, 3, 4, 5,
55     4, 5, 6, 7, 8, 9,
56     8, 9, 10, 11, 12, 13,
57     12, 13, 14, 15, 16, 17,
58     16, 17, 18, 19, 20, 21,
59     20, 21, 22, 23, 24, 25,
60     24, 25, 26, 27, 28, 29,
61     28, 29, 30, 31, 32, 1};
62
63 bits48 apply_E(bits32 in)
64 {
65     bits48 out;
66     for (int i = 47; i >= 0; i--)
67     {
68         int j = E[47 - i] - 1;
69         out[i] = in[31 - j];
70     }
71     return out;
72 }
73
74 int S1[4][16] = {
75     14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
76     0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
77     4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
78     15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13};
79
80 int S2[4][16] = {
81     15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
82     3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
83     0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
84     13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9};
85
86 int S3[4][16] = {
87     10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
88     13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
89     13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
90     1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12};
91
92 int S4[4][16] = {
93     7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
94     13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
95     10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
96     3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14};
97
98 int S5[4][16] = {
99     2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
100    14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
101    4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
102    11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3};
103
104 int S6[4][16] = {
105    12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
106    10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
107    9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
108    4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13};
109
110 int S7[4][16] = {
111    4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
112    13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
113    1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
114    6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12};
115
116 int S8[4][16] = {
117    13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
118    1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
119    7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
120    2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11};
121
122 bits32 apply_S(bits48 in)
123 {
124     bits32 out;
125     int i = 47, j = 31;
126     for (auto S : {S1, S2, S3, S4, S5, S6, S7, S8})
127     {
128         int row = 0, col = 0;
129         for (int k = i - 1; k > i - 5; k--)
130         {
131             col = col * 2 + in[k];
132         }
133         row = in[i] * 2 + in[i - 5];
134
135         bitset<4> num = S[row][col];
136         for (int k = j; k >= j - 3; k--)
137         {
138             out[k] = num[3 - (j - k)];
139         }
140         i -= 6;
141         j -= 4;
142     }
143     return out;
144 }
145
146 int P[] = {
147    16, 7, 20, 21, 29, 12, 28, 17,
148    1, 15, 23, 26, 5, 18, 31, 10,
149    2, 8, 24, 14, 32, 27, 3, 9,
150    19, 13, 30, 6, 22, 11, 4, 25};
151
152 int P_inv[32];
153
154 void calc_P_inv()
155 {
156     for (int i = 0; i < 32; i++)
157     {
158         int to = P[i] - 1;
159         P_inv[to] = i + 1;
160     }
161 }
162
163 bits32 apply_P(bits32 in, bool inv = false)
164 {

```

```

165     bits32 out;
166     for (int i = 31; i >= 0; i--)
167     {
168         int j = (inv ? P_inv[31 - i] : P[31 - i]) - 1;
169         out[i] = in[31 - j];
170     }
171     return out;
172 }
173
174 using vec64 = array<int, 64>;
175 using vec32 = array<int, 32>;
176 using vec48 = array<int, 48>;
177 using vec56 = array<int, 56>;
178 using vec28 = array<int, 28>;
179
180 int PC_1[] = {
181     57, 49, 41, 33, 25, 17, 9,
182     1, 58, 50, 42, 34, 26, 18,
183     10, 2, 59, 51, 43, 35, 27,
184     19, 11, 3, 60, 52, 44, 36,
185     63, 55, 47, 39, 31, 23, 15,
186     7, 62, 54, 46, 38, 30, 22,
187     14, 6, 61, 53, 45, 37, 29,
188     21, 13, 5, 28, 20, 12, 4};
189
190 unsigned short shifts[] = {
191     1, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};
192
193 int PC_2[] = {
194     14, 17, 11, 24, 1, 5,
195     3, 28, 15, 6, 21, 10,
196     23, 19, 12, 4, 26, 8,
197     16, 7, 27, 28, 13, 2,
198     41, 52, 31, 37, 47, 55,
199     30, 40, 51, 45, 33, 48,
200     44, 49, 39, 56, 34, 53,
201     46, 42, 50, 36, 29, 32};
202
203 vector<vec48> keys(16);
204
205 void getKEYS()
206 {
207     vec56 key;
208     for (int i = 55; i >= 0; i--)
209     {
210         key[i] = 56 - i;
211     }
212
213     vec28 C, D;
214     for (int i = 55, j = 27; i >= 28; i--, j--)
215     {
216         C[j] = key[i];
217         D[j] = key[i - 28];
218     }
219
220     for (int r = 0; r < 16; r++)
221     {
222         vec28 rC, rD;
223         int s = shifts[r];
224         for (int i = 27; i >= 0; i--)
225         {
226             int to = (i + s) % 28;
227             rC[to] = C[i];
228             rD[to] = D[i];
229         }
230         C = rC, D = rD;
231
232         for (int i = 47; i >= 0; i--)
233         {
234             int j = PC_2[47 - i] - 1;
235             j = 55 - j;
236             if (j > 27)
237             {
238                 keys[r][i] = C[j - 28];
239             }
240             else
241             {
242                 keys[r][i] = D[j];
243             }
244         }
245     }
246 }
247
248 void print(vec48 key)
249 {
250     for (int i = 47; i >= 0; i -= 6)
251     {
252         int num = 0;
253         for (int j = i; j >= i - 5; j--)
254         {
255             num = num * 2 + key[j];
256         }
257         cout << setw(2) << setfill('0') << hex << num;
258     }
259     cout << " ";
260 }
261
262 bits64 join(bits32 L, bits32 R)
263 {
264     bits64 out;
265     for (int i = 31; i >= 0; i--)
266     {
267         out[i + 32] = L[i];
268         out[i] = R[i];
269     }
270     return out;
271 }
272
273 bits64 get(string s)
274 {
275     assert(s.length() == 16);

```

```

276     bits64 ans = 0;
277     for (int i = 0; i < 16; i++)
278     {
279         int foo = s[i] - 'd';
280         ans = (ans << 4) | bits64(foo);
281     }
282     return ans;
283 }
284
285 vector<bits48> rkeys(16);
286 void getKEYS2(vector<int> key)
287 {
288     for (int r = 0; r < 16; r++)
289     {
290         for (int i = 47; i >= 0; i--)
291         {
292             int j = keys[r][i] - 1;
293             rkeys[r][i] = key[55 - j];
294         }
295     }
296 }
297
298 int main()
299 {
300     getKEYS();
301     bits48 key_r6 = bits48("1011011100110101011010011011110110000101111110");
302
303     vector<int> key(56, -1);
304     for (int i = 47, cur = 0; i >= 0; i -= 6, cur++)
305     {
306         if (cur == 2 || cur == 3)
307             continue;
308         for (int j = i; j >= i - 5; j--)
309         {
310             int k = keys[5][j] - 1;
311             key[55 - k] = key_r6[j];
312         }
313     }
314
315     auto doDES = [&](bits64 in, bool inv = false) {
316         bits64 LR = apply_ip(in);
317         auto [L, R] = get_LR(LR);
318         for (int r = 0; r < 6; r++)
319         {
320             auto ER = apply_E(R);
321             ER ^= (inv ? rkeys[5 - r] : rkeys[r]);
322             auto SR = apply_S(ER);
323             auto PR = apply_P(SR);
324             auto nL = R, nR = L ^ PR;
325             L = nL, R = nR;
326         }
327         auto O = join(R, L);
328         auto out = apply_ip(O, true);
329         return out;
330     };
331
332     vector<int> pos;
333     int cnt = 0;
334     for (int i = 55; i >= 0; i--)
335     {
336         if (key[i] == -1)
337             pos.push_back(i);
338         cnt += (key[i] == -1);
339     }
340
341     calc_P_inv();
342
343     bits64 in = get("qosppgjaphiporfs");
344     bits64 out = get("jispdnkkprngkqnii");
345
346     for (int i = 0; i < (1 << cnt); i++)
347     {
348         for (int j = 0; j < cnt; j++)
349         {
350             if ((i & (1 << j)) != 0)
351             {
352                 key[pos[j]] = 1;
353             }
354             else
355                 key[pos[j]] = 0;
356         }
357         getKEYS2(key);
358         if (doDES(in) == out)
359         {
360             cout << "The key is : " << endl;
361             for (int k = 55; k >= 0; k--)
362             {
363                 cout << key[k];
364             }
365             cout << endl;
366             break;
367         }
368     }
369
370     bits64 in21 = get("gdpgdkhhjsgnlmk");
371     bits64 in22 = get("kjogidkodkfphilf");
372
373     for (auto in2 : {in21, in22})
374     {
375         auto out2 = doDES(in2, true);
376         for (int i = 63; i >= 0; i -= 8)
377         {
378             int num = 0;
379             for (int j = i; j >= i - 7; j--)
380             {
381                 num = num * 2 + out2[j];
382             }
383             cout << char(num);
384         }
385     }
386     cout << endl;

```

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



Select a question.

Group M