SEM - VII - 2022-23
CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 6

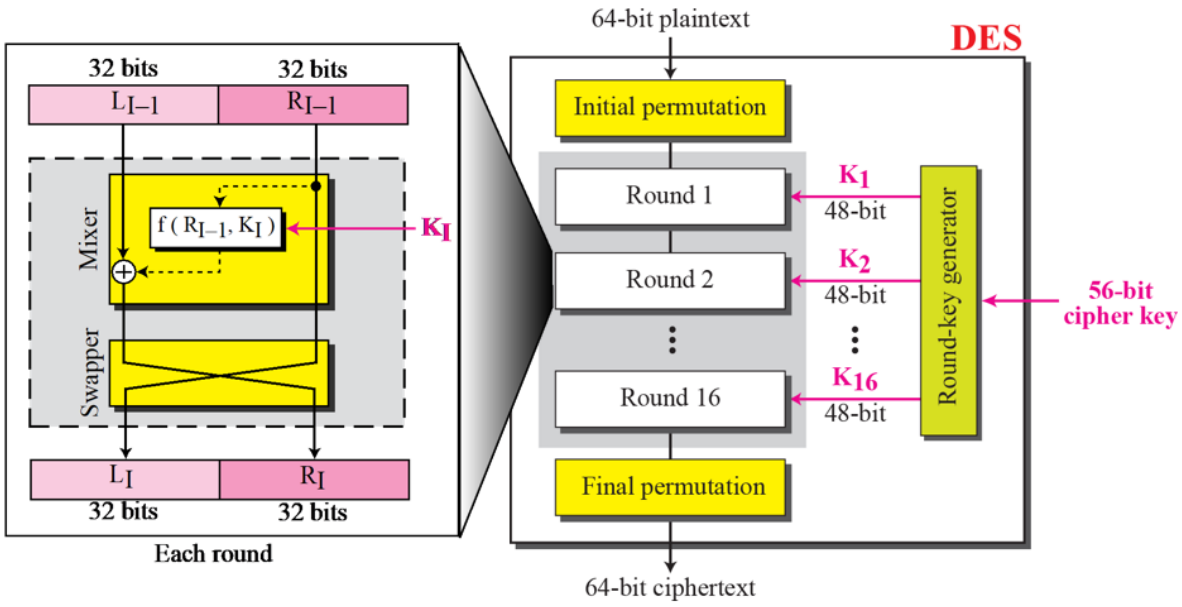
DES

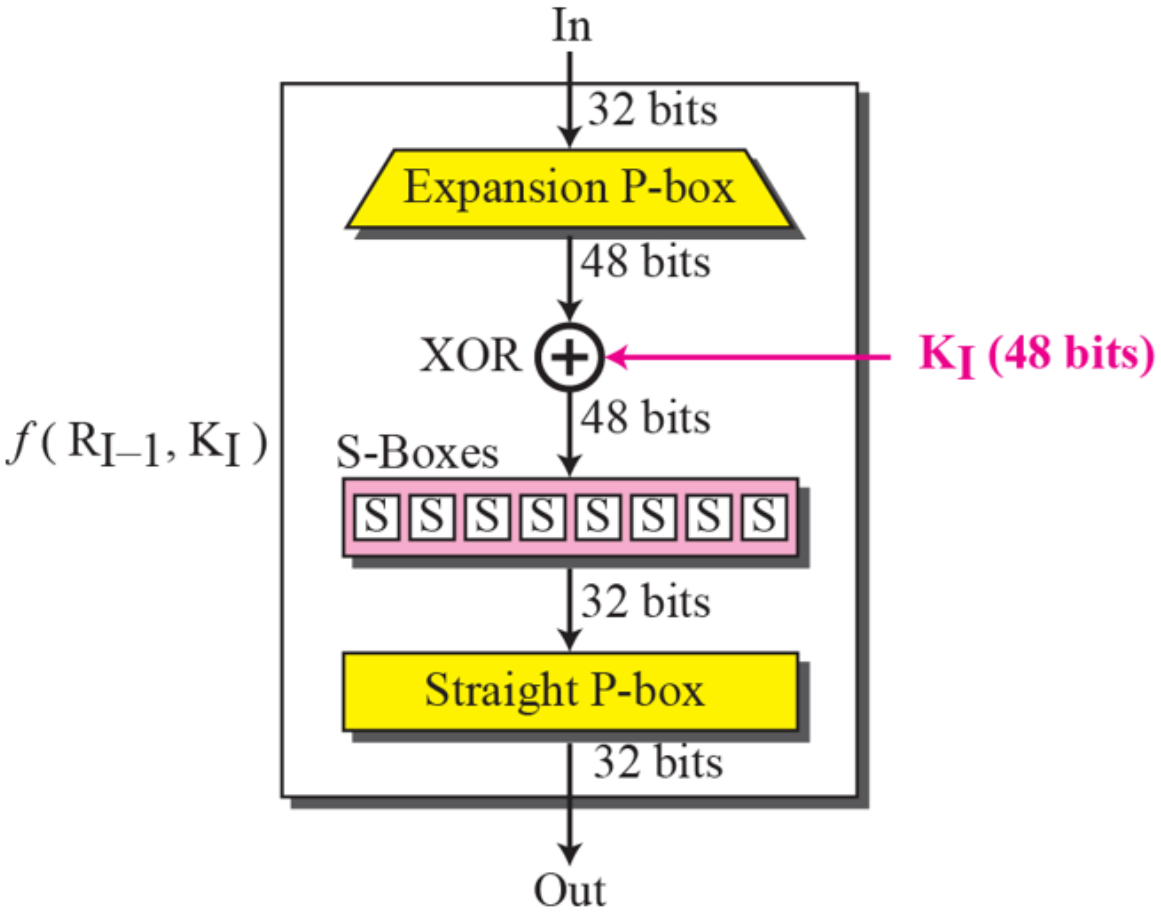
DES - Data Encryption Standard

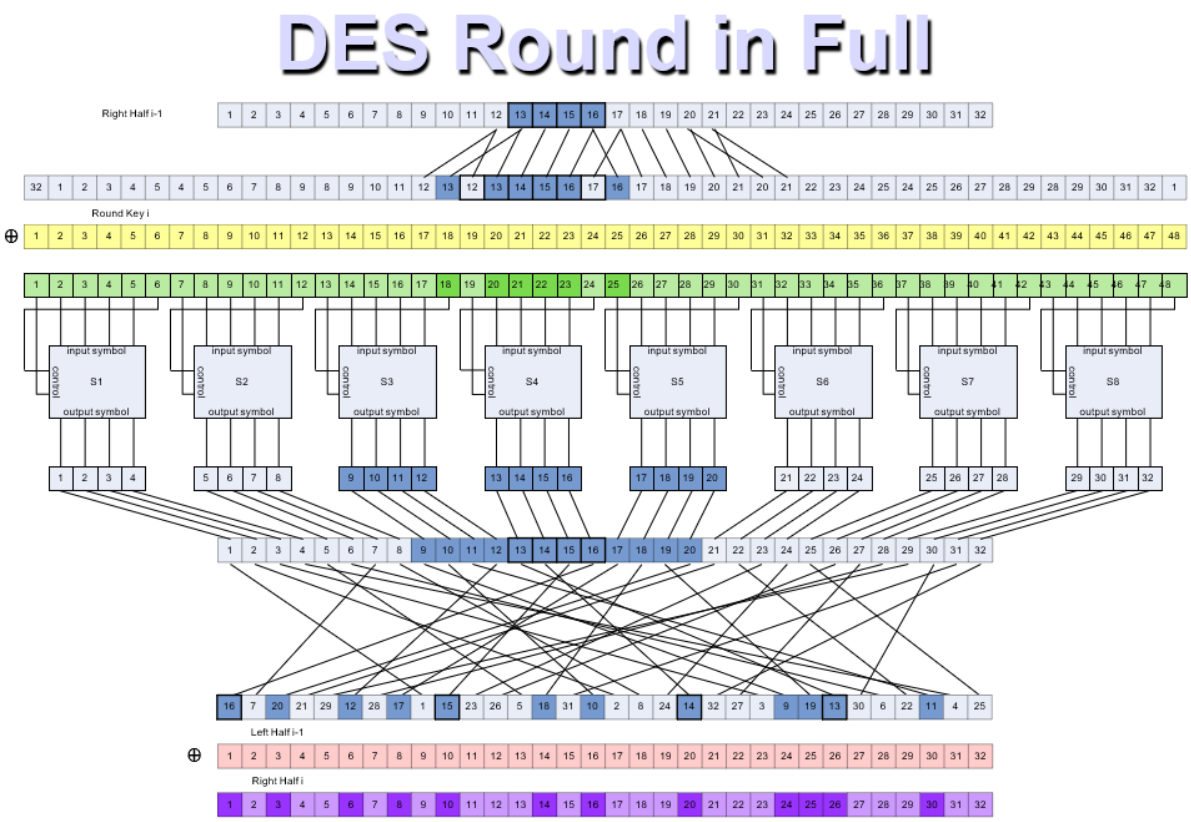
Theory:

DES is a block cipher and encrypts data in blocks of size of **64 bits** each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is **56 bits**.

- In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function.
- The initial permutation is performed on plain text.
- Next, the initial permutation (IP) produces two halves of the permuted block; saying Left Plain Text (LPT) and Right Plain Text (RPT).
- Now each LPT and RPT go through 16 rounds of the encryption process.
- In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
- The result of this process produces 64-bit ciphertext.


64-bit plaintext
DES
32 bits
32 bits
L_{I-1}
R_{I-1}
Mixer
$f(R_{I-1}, K_I)$
K_I
Swapper
L_I
R_I
32 bits
32 bits
Each round
Initial permutation
Round 1
Round 2
Round 16
Final permutation
K_1
48-bit
K_2
48-bit
K_{16}
48-bit
Round-key generator
56-bit cipher key
64-bit ciphertext


In
32 bits
Expansion P-box
48 bits
XOR
K_I (48 bits)
48 bits
$f(R_{I-1}, K_I)$
S-Boxes
S S S S S S S S
32 bits
Straight P-box
32 bits
Out



Code:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
string hex2bin(string s)
{
// hexadecimal to binary conversion
    map<char, string> mp;
    mp['0'] = "0000";
    mp['1'] = "0001";
    mp['2'] = "0010";
    mp['3'] = "0011";
    mp['4'] = "0100";
    mp['5'] = "0101";
    mp['6'] = "0110";
    mp['7'] = "0111";
    mp['8'] = "1000";
    mp['9'] = "1001";
```

```
    mp['A'] = "1010";
    mp['B'] = "1011";
    mp['C'] = "1100";
    mp['D'] = "1101";
    mp['E'] = "1110";
    mp['F'] = "1111";
    string bin = "";
    for (int i = 0; i < s.size(); i++) {
        bin += mp[s[i]];
    }
    return bin;
}
string bin2hex(string s)
{
// binary to hexadecimal conversion
    map<string, string> mp;
    mp["0000"] = "0";
    mp["0001"] = "1";
    mp["0010"] = "2";
    mp["0011"] = "3";
    mp["0100"] = "4";
    mp["0101"] = "5";
    mp["0110"] = "6";
    mp["0111"] = "7";
    mp["1000"] = "8";
    mp["1001"] = "9";
    mp["1010"] = "A";
    mp["1011"] = "B";
    mp["1100"] = "C";
    mp["1101"] = "D";
    mp["1110"] = "E";
    mp["1111"] = "F";
    string hex = "";
    for (int i = 0; i < s.length(); i += 4) {
        string ch = "";
        ch += s[i];
        ch += s[i + 1];
        ch += s[i + 2];
        ch += s[i + 3];
        hex += mp[ch];
```

```
    }
    return hex;
}

string permute(string k, int* arr, int n)
{
    string per = "";
    for (int i = 0; i < n; i++) {
        per += k[arr[i] - 1];
    }
    return per;
}

string shift_left(string k, int shifts)
{
    string s = "";
    for (int i = 0; i < shifts; i++) {
        for (int j = 1; j < 28; j++) {
            s += k[j];
        }
        s += k[0];
        k = s;
        s = "";
    }
    return k;
}

string xor_(string a, string b)
{
    string ans = "";
    for (int i = 0; i < a.size(); i++) {
        if (a[i] == b[i]) {
            ans += "0";
        }
        else {
            ans += "1";
        }
    }
    return ans;
}
```

```
string encrypt(string pt, vector<string> rkb, vector<string> rk)
{
// Hexadecimal to binary
    pt = hex2bin(pt);

// Initial Permutation Table
    int initial_perm[64] = { 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
                           };
// Initial Permutation
    pt = permute(pt, initial_perm, 64);
    cout << "After initial permutation: " << bin2hex(pt) << endl;

// Splitting
    string left = pt.substr(0, 32);
    string right = pt.substr(32, 32);
    cout << "After splitting: L0=" << bin2hex(left)
         << " R0=" << bin2hex(right) << endl;

// Expansion D-box Table
    int exp_d[48] = { 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
                    };

// S-box Table
    int s[8][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
```

```
        },
        {   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
        },

        {   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
        },
        {   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
        },
        {   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
        },
        {   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
        },
        {   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
        },
        {   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
        }
    };

// Straight Permutation Table
```

```
    int per[32] = { 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
                  };

    cout << endl;
    for (int i = 0; i < 16; i++) {
        // Expansion D-box
        string right_expanded = permute(right, exp_d, 48);

        // XOR RoundKey[i] and right_expanded
        string x = xor_(rkb[i], right_expanded);

        // S-boxes
        string op = "";
        for (int i = 0; i < 8; i++) {
            int row = 2 * int(x[i * 6] - '0') + int(x[i * 6 + 5] - '0');
            int col = 8 * int(x[i * 6 + 1] - '0') + 4 * int(x[i * 6 + 2] -
'0') + 2 * int(x[i * 6 + 3] - '0') + int(x[i * 6 + 4] - '0');
            int val = s[i][row][col];
            op += char(val / 8 + '0');
            val = val % 8;
            op += char(val / 4 + '0');
            val = val % 4;
            op += char(val / 2 + '0');
            val = val % 2;
            op += char(val + '0');
        }
        // Straight D-box
        op = permute(op, per, 32);

        // XOR left and op
        x = xor_(op, left);

        left = x;
```

```
        // Swapper
        if (i != 15) {
            swap(left, right);
        }
        cout << "Round " << i + 1 << " " << bin2hex(left) << " "
             << bin2hex(right) << " " << rk[i] << endl;
    }

// Combination
    string combine = left + right;

// Final Permutation Table
    int final_perm[64] = { 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
                         };

// Final Permutation
    string cipher = bin2hex(permute(combine, final_perm, 64));
    return cipher;
}
int main()
{
// pt is plain text
    string pt, key;
    /*cout<<"Enter plain text(in hexadecimal): ";
    cin>>pt;
    cout<<"Enter key(in hexadecimal): ";
    cin>>key;*/

    pt = "123456ABCD1325F7";
    key = "1CBB0918273CCDD1";
// Key Generation
```

```
// Hex to binary
    key = hex2bin(key);

// Parity bit drop table
    int keyp[56] = { 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
                   };

// getting 56 bit key from 64 bit using the parity bits
    key = permute(key, keyp, 56); // key without parity

// Number of bit shifts
    int shift_table[16] = { 1, 1, 2, 2,
                            2, 2, 2, 2,
                            1, 2, 2, 2,
                            2, 2, 2, 1
                          };

// Key- Compression Table
    int key_comp[48] = { 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
                       };

// Splitting
    string left = key.substr(0, 28);
    string right = key.substr(28, 28);

    vector<string> rkb; // rkb for RoundKeys in binary
```

```
    vector<string> rk; // rk for RoundKeys in hexadecimal
    for (int i = 0; i < 16; i++) {
        // Shifting
        left = shift_left(left, shift_table[i]);
        right = shift_left(right, shift_table[i]);

        // Combining
        string combine = left + right;

        // Key Compression
        string RoundKey = permute(combine, key_comp, 48);

        rkb.push_back(RoundKey);
        rk.push_back(bin2hex(RoundKey));
    }

    cout << "Plain text:"<<pt<<"\n";
    cout << "\nEncryption:\n\n";
    string cipher = encrypt(pt, rkb, rk);
    cout << "\nCipher Text: " << cipher << endl;

    cout << "\nDecryption\n\n";
    reverse(rkb.begin(), rkb.end());
    reverse(rk.begin(), rk.end());
    string text = encrypt(cipher, rkb, rk);
    cout << "\nPlain Text: " << text << endl;
}
```

Output:

outputf.in

```
Plain text:123456ABCD1325F7

Encryption:

After initial permutation: 94A7D6F898CA18AD
After splitting: L0=94A7D6F8 R0=98CA18AD

Round 1 98CA18AD E98FF09F 1944C074D6A8
Round 2 E98FF09F 4CDE422D 4528581AFC5C
Round 3 4CDE422D CB006489 06ECA069D5B4
Round 4 CB006489 D7D55F5E DA2D02896CAB
Round 5 D7D55F5E 12E9DECD 68A609EE5A15
Round 6 12E9DECD 9520EBB2 41940E9343FE
Round 7 9520EBB2 C4354D7F 6088D2959B81
Round 8 C4354D7F D82AC081 34E822D22675
Round 9 D82AC081 EEB57875 849B44F1D0C7
Round 10 EEB57875 170A32C8 02724706A6AF
Round 11 170A32C8 5EC71E48 295560BE3DC5
Round 12 5EC71E48 335A2EC9 C041E92AC3F3
Round 13 335A2EC9 17E7FA69 91C31157ED03
Round 14 17E7FA69 FDE5C7B9 051B83EE0558
Round 15 FDE5C7B9 5434F99B 3330C5C9F34E
Round 16 1E362632 5434F99B 081C1D4D8F69

Cipher Text: 0A57F44AFB3D880A
```

```
Decryption

After initial permutation: 1E3626325434F99B
After splitting: L0=1E362632 R0=5434F99B

Round 1 5434F99B FDE5C7B9 081C1D4D8F69
Round 2 FDE5C7B9 17E7FA69 3330C5C9F34E
Round 3 17E7FA69 335A2EC9 051B83EE0558
Round 4 335A2EC9 5EC71E48 91C31157ED03
Round 5 5EC71E48 170A32C8 C041E92AC3F3
Round 6 170A32C8 EEB57875 295560BE3DC5
Round 7 EEB57875 D82AC081 02724706A6AF
Round 8 D82AC081 C4354D7F 849B44F1D0C7
Round 9 C4354D7F 9520EBB2 34E822D22675
Round 10 9520EBB2 12E9DECD 6088D2959B81
Round 11 12E9DECD D7D55F5E 41940E9343FE
Round 12 D7D55F5E CB006489 68A609EE5A15
Round 13 CB006489 4CDE422D DA2D02896CAB
Round 14 4CDE422D E98FF09F 06ECA069D5B4
Round 15 E98FF09F 98CA18AD 4528581AFC5C
Round 16 94A7D6F8 98CA18AD 1944C074D6A8

Plain Text: 123456ABCD1325F7
```