

# REPORT

## OTP, 페이스 텔 암호



과목명 :		컴퓨터보안			
담당교수 :		정준호		교수님	
제출일 :	2021	년 04	월 06	일	
공과	대학	컴퓨터공학			과
학번 :	2016112154	이름:		정동구	

# 1. OTP

## a. 소스 코드

```
std::string encryption(std::vector<std::string> plain, std::vector<char> pad)//암호화
{
    std::string result;
    int x = 0;
    for (int i = 0; i < plain.size(); i++)
    {
        for (int j = 0; j < plain[i].size(); j++)
        {
            std::string temp;
            temp = plain[i][j] ^ pad[x++];
            result += temp;
        }
    }
    return result;
}

std::string decryption(std::string encrypted_, std::vector<char> pad)//복호화
{
    std::string result;
    int x = 0;
    for (int i = 0; i < encrypted_.size(); i++)
    {
        std::string temp;
        temp = encrypted_[i] ^ pad[x++];
        result += temp;
    }

    return result;
}
```

암호화와 복호화를 위한 함수. 입력받은 평문을 패드와 xor 연산을 해서 암호화 하고 동일한 방식으로 복호화한다.

```
void createOtp(std::vector<std::string> plain, std::string out, std::vector<char>& otp_)//평문 길이만큼 pad 생성
{
    std::ofstream otp(out);//복호문 저장
    int x = 0;
    for (int i = 0; i < plain.size(); i++)
    {
        for (int j = 0; j < plain[i].size(); j++)
        {
            otp_.push_back((char)rand());
        }
    }
    for (auto it : otp_)
    {
        otp << otp_[it];
    }
}
```

패드를 생성하기 위한 함수. 입력받은 평문과 동일한 길이만큼 랜덤한 패드를 생성한다. 패드를 텍스트 파일로 출력한다.

```

int main(void)
{
    std::string input_file; //평문 txt
    std::string OTP_file; // 생성된 패드
    std::string enc_file; //암호문 txt
    std::string dec_file;
    std::vector<std::string> plain_text; //평문
    std::vector<char> otp; //otp
    std::string encrypted; //암호문
    std::string decrypted; //복호문
    srand(time(NULL));

    std::cout << "input file name : ";
    std::cin >> input_file;
    std::cout << "encrypted file name : ";
    std::cin >> enc_file;
    std::cout << "decrypted file name : ";
    std::cin >> dec_file;
    OTP_file = "otp.txt";

    std::ifstream read_file(input_file); //평문
    std::ofstream enc_file_out(enc_file); //암호문 저장
    std::ofstream dec_file_out(dec_file); //복호문 저장

    if (read_file.is_open()) //평문 읽어오기
    {
        while (!read_file.eof())
        {
            std::string s;
            std::getline(read_file, s);
            s += '\n'; //개행문자 추가
            plain_text.push_back(s);
        }
    }

    createOtp(plain_text, OTP_file, otp);
    encrypted = encryption(plain_text, otp);
    enc_file_out << encrypted;
    decrypted = decryption(encrypted, otp);
    dec_file_out << decrypted;

    return 0;
}

```

Main 함수. 텍스트 파일로 저장된 평문을 읽어온다. createOtp 를 통해 패드를 생성하고 암호화와 복호화를 한다.

## b. 결과

sample.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

이 편지는 영국에서 최초로 시작되어 일년에 한 바퀴 돌면서 받는 사람에게 행운을 주었고  
 지금은 당신에게로 옮겨진 이 편지는 4일 안에 당신 곁을 떠나야 합니다.  
 이 편지를 포함해서 7통을 행운이 필요한 사람에게 보내 주셔야 합니다.  
 복사를 해도 좋습니다.혹 미신이라 하실지 모르지만 사실입니다.  
 영국에서 HGXWCH이라는 사람은 1930년에 이 편지를 받았습니다.  
 그는 비서에게 복사해서 보내라고 했습니다.  
 며칠 뒤에 복권이 당첨되어 20억을 받았습니다.  
 어떤 이는 이 편지를 받았으나 96시간 이내 자신의 손에서 떠나야 한다는 사실을 잊었습니다.  
 그는 곧 사직되었습니다.나중에는 이 사실을 알고 7통의 편지를 보냈는데 다시 좋은 직장을 얻었습니다.  
 미국의 케네디 대통령은 이 편지를 받았지만 그냥 버렸습니다.  
 결국 9일 후 그는 암살 당했습니다.기억해 주세요.  
 이 편지를 보내면 7년의 행운이 있을 것이고 그렇지 않으면 3년의 불행이 있을 것입니다.  
 그리고 이 편지를 버리거나 낙서를 해서는 절대로 안됩니다. 7통입니다.  
 이 편지를 받은 사람은 행운이 깃들 것입니다. 힘들겠지만 좋은게 좋다고 생각하세요. 7년의 행운을 빌면서..

otp.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

|Acc 락빚릿쑈2쑈 嗟?&x↑~뽕 락P적??뽕ha??뽕 쥔 ||w?~Eg 쥔0o~s{~뽕?烙[蘆n◀|t↑뽕?1n?뽕2P P~?鹹 ?第-sW 噫속 ^

enc.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

m?|↑↑궤導+?똥jH)P0??\*?!|枸)?F??←?₩通D릐-u H헨l☐?炊'齡醪F?(h탐M!책흑?±핑 <늬<?  
ず +  
r/?←?꺅9)킨??A◀까?χ?매沸湧衙F첼<?뢔채?? 꺽흥J?해꺾?話暈瘦?DdH?ER●똥?鮑???w±V\_웁?-[P?●?ᄇJr携쯤C  
i??騁송?p1L환禍甞?기뵙±/헨?? ?뵙 뵙±\$J존茁o)낭P? ^?썩#~낭f熏+갸±kW?釵??O噴?'N\_셔뵙[쟁-'N₁g?W?  
→?7笠\_?8+=+???|3[?]?"O  
?윰!?嗜 꺽9렉g 환Jsw>>샅.뒗→?r? τ!!?74{^?需v2@|b튼-?腺??썰◀@뎡웁뜨(Li甞¼4lh??>?96d塙臨喆)|>?  
嘲뵙霜窠~E긋)?·qD-)곱?궜2漆릴냐?팸+죿?툯\*鞞d?q刳刧강P꺽J?,뵙D+?=Vy犧y3吳?●蔦●칼?p헌??wlD?檣~-?28갯t?)  
①y|τ?뵙?\$?殮J4x? 쏴??xp?죿l8)rB읃5{?pP=땡?巡1\*P:Yw우急늉1有(?p數4?갯4꺽m?  
{q粱O?輕&?+榜uiτ-X'?q5헝&늉?b|\$?( 섯錦\*z?괘n|i뎡₩彰臥垓|췊[冢聻공o(룟?죿l?M밌?Fq?j\_轢e\*t~?-E넹잠뵙8뵙

dec.bat - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

이 편지는 영국에서 최초로 시작되어 일년에 한 바퀴 돌면서 받는 사람에게 행운을 주었고 지금은 당신에게로 옮겨진 이 편지는 4일 안에 당신 곁을 떠나야 합니다. 이 편지를 포함해서 7통을 행운이 필요한 사람에게 보내 주셔야 합니다. 복사를 해도 좋습니다.혹 미신이라 하실지 모르지만 사실입니다. 영국에서 HGXWCH이라는 사람은 1930년에 이 편지를 받았습니다. 그는 비서에게 복사해서 보내라고 했습니다. 며칠 뒤에 복권이 당첨되어 20억을 받았습니다. 어떤 이는 이 편지를 받았으나 96시간 이내 자신의 손에서 떠나야 한다는 사실을 잊었습니다. 그는 곧 사직되었습니다.나중해야 이 사실을 알고 7통의 편지를 보냈는데 다시 좋은 직장을 얻었습니다. 미국의 케네디 대통령은 이 편지를 받았지만 그냥 버렸습니다. 결국 9일 후 그는 암살 당했습니다.기억해 주세요. 이 편지를 보내면 7년의 행운이 있을 것이고 그렇지 않으면 3년의 불행이 있을 것입니다. 그리고 이 편지를 버리거나 낙서를 해서는 절대 안됩니다. 7통입니다. 이 편지를 받은 사람은 행운이 깃들 것입니다. 힘들겠지만 좋게 생각하세요. 7년의 행운을 빌면서..

읽어온 텍스트가 암호화 되었다가 줄바꿈 및 공백까지 완전히 동일하게 복호화 되었음을 볼 수 있다.

## 2. DES(페이스텔 암호)

### a. 소스코드

```
DES.h* x DES.cpp* main.cpp
DES
2  #include <iostream>
3  #include <string>
4  #include <vector>
5  #include <iomanip>
6
7  enum MODE {
8      ENCRYPTION, DECRYPTION
9  };
10
11 class DES {
12 private:
13     std::string plaintext;
14     std::string ciphertext;
15     std::string keystring;
16     std::vector<int> key;
17     std::vector<int> keys[17];
18     std::vector<int> keyPermutationTable;
19     std::vector<int> rounds;
20     std::vector<int> compressionPermutationTable;
21     std::vector<int> binaryText;
22     std::vector<int> IP;//Initial Permutaion
23     std::vector<int> IPinverse;
24     std::vector<int> left, right, rightOrigin;
25     std::vector<int> EP;//ExpansionPermutaion
26     std::vector<int> s[9]);//s-box
27     std::vector<int> Pbox;//Permutaion
28
29     std::vector<int> string2hex(const std::string& str);//string to hex
30     std::string hex2string(const std::vector<int>& hex);//hex to string
31
32     std::vector<int> initialKeySelection(const std::vector<int>& key);//64->56
33     void leftShift(std::vector<int>& key, int s);//split to 28bit - shift left
34     std::vector<int> keyPermutationAndCompression();//create 48bit round key
35     std::vector<int> initialPermutation(const std::vector<int>& text);
36     void splitLeftRight(const std::vector<int>& text);
37     std::vector<int> expansionPermutation(const std::vector<int>& right);
38     std::vector<int> XOR_Key(const std::vector<int>& right, int round);
39     std::vector<int> S_box_substitution(const std::vector<int>& right);
40     std::vector<int> PboxPermutation(const std::vector<int>& right);
41     std::vector<int> XOR_left(const std::vector<int>& left, const std::vector<int>& right);
42     void nextLeft();
43     std::vector<int> lastPermutation(const std::vector<int>& text);
44
45     void keyGeneration();
46 public:
47     DES(std::string text, std::string key, MODE mode);
48
49     void encryption();
50
51     void decryption();
52 }
```

DES class 를 정의 해 놓은 헤더 이다.

```

DES::DES(std::string text, std::string key, MODE mode) { ... }

void DES::encryption()
{
    std::cout << "Plain Text : [" << plaintext << "]" << "\n";
    std::cout << "Key : [" << keystring << "]" << "\n";
    binaryText = string2hex(plaintext);
    key = string2hex(keystring);
    key = initialKeySelection(key);
    keyGeneration();
    binaryText = initialPermutation(binaryText);
    splitLeftRight(binaryText);
    std::cout << "round left right key\n";
    for (int round = 1; round <= 16; round++)
    {
        right = expansionPermutation(right); //F function
        right = XOR_Key(right, round); //F function
        right = S_box_substitution(right); //F function
        right = PboxPermutation(right); //F function
        right = XOR_left(left, right);
        nextLeft();
        if (round == 16)
        {
            std::vector<int> tmp;
            tmp = left;
            left = right;
            right = tmp;
        }
        std::cout << std::setw(5) << round << " [" << hex2string(left) << " [" << hex2string(right) << " [" << hex2string(keys[round]) << "]" << "\n";
    }
    binaryText = lastPermutation(binaryText); //IP inverse
    ciphertext = hex2string(binaryText);
    std::cout << "Cipher Text : [" << ciphertext << "]" << "\n";
}

```

생성자에서는 plain text 와 key 를 입력받고 암호화 알고리즘에 사용되는 각종 테이블을 생성한다.

암호화에서는 des 알고리즘의 과정을 거친다. 우선 입력받은 plaintext 와 key 를 16 진수 형태로 변경한다. 이후 키 스케줄러를 실행한다. InitialKeySelection(key)를 통해 64 비트 키 데이터에서 8 비트의 패리티 체크 비트를 제거하고 선택치환을 한다. keyGeneration 에서 28 비트씩 양분하고 라운드 별 지정된 수 만큼 좌측으로 시프트 한 후 선택치환을 하여 48 비트의 라운드 키를 생성한다.

initialPermutation 을 통해 plaintext 를 전치한다. 이때 암호화는 일어나지 않는다.

splitLeftRigth 를 이용하여 plaintext 를 32 비트씩 나눈다.

F 함수에 해당하는 부분은 위 이미지에 주석으로 나타내었다. expansionPermutation 을 통해 확장순열을 사용하여 48bit 의 배열로 확장전치한다. 이후 XOR\_Key 로 해당 라운드에 맞는 key 와 우측 부분이 xor 연산을 한다.

S\_box\_substitution 에서는 xor 연산된 48 비트를 6 비트 길이의 8 토막으로 나눈 후 치환을 한다. 6 비트 중 첫번째와 마지막 비트가 합쳐져서 행번호를 카르키고 2,3,4 번째 비트는 열 번호를 가르킨다. sbbox 표를 이용하여 각 4 비트의 이진수로 바꾸고 연결하여 32 비트로 만든 뒤 PboxPermutation 으로 전치를 한다.

이렇게 생성된 결과를 좌측 32 비트와 xor 연산을 하여 한 라운드를 마치고 총 16 라운드를 한다.

16 라운드가 끝난 이후에 lastPermutation 을 통해 마지막 전치를 한다. 해당 전치는 initialPermutation 의 inverse 이다.

```

void DES::decryption()
{
    std::cout << "Cipher Text : [" << ciphertext << "]"<<endl;
    std::cout << "    Key       : [" << keystring << "]"<<endl;
    binaryText = string2hex(ciphertext);
    key = string2hex(keystring);
    key = initialKeySelection(key);
    keyGeneration();
    binaryText = initialPermutation(binaryText);
    splitLeftRight(binaryText);
    std::cout << "round    left        right        key"<<endl;
    for (int round = 16; round >= 1; round--)
    {
        right = expansionPermutation(right);
        right = XOR_Key(right, round);
        right = S_box_substitution(right);
        right = PboxPermutation(right);
        right = XOR_left(left, right);
        nextLeft();
        if (round == 1)
        {
            std::vector<int> tmp;
            tmp = left;
            left = right;
            right = tmp;
        }
        std::cout << std::setw(5) << 17 - round << "    [" << hex2string(left) << "]" << hex2string(right) << "]" << hex2string(keys[round]) << "]"<<endl;
    }
    binaryText = lastPermutation(binaryText);
    plaintext = hex2string(binaryText);
    std::cout << "Plain Text : [" << plaintext << "]"<<endl;
}

```

Decryption 은 encryption 과 동일하되 라운드를 역으로 수행하면 된다.

```

#pragma once
#include "DES.h"

int main()
{
    std::cout << "=====ENCRYPTION=====<<endl;
    DES des1("0123456789ABCDEF", "85EB13540F0AB406", MODE::ENCRYPTION);
    des1.encryption();
    std::cout << "=====DECRYPTION=====<<endl;
    DES des2("E5A951F59B31600D", "85EB13540F0AB406", MODE::DECRYPTION);
    des2.decryption();
}

```

Main 함수. DES 객체를 생성하여 암호화와 복호화를 진행한다.

DES class 의 나머지 멤버 함수들은 소감 이후에 전체 코드를 첨부하였습니다.

## b. 결과

```

=====ENCRYPTION=====
Plain Text : [0123456789ABCDEF]
Key       : [85E813540FOAB405]
round    left      right      key
  1 [FOAAF0AA] [085BE33B] [C842107B226C]
  2 [085BE33B] [80F7FA3B] [35602242C637]
  3 [80F7FA3B] [1A760BC4] [A205801F2D8C]
  4 [1A760BC4] [916A4BE6] [580215A851D1]
  5 [916A4BE6] [A266EC34] [05901843E227]
  6 [A266EC34] [51ECAEF3] [0600E6F60D88]
  7 [51ECAEF3] [45E5012A] [BA480088135F]
  8 [45E5012A] [0F3679E5] [08232857F2A0]
  9 [0F3679E5] [3AF0D85A] [23A0C408EA4F]
 10 [3AF0D85A] [EB5D0481] [18448276D494]
 11 [EB5D0481] [B56D060C] [7001388905EB]
 12 [B56D060C] [08F5AF71] [8480058EFA01]
 13 [08F5AF71] [8FB6E536] [030A16724774]
 14 [8FB6E536] [3867C7CF] [2C30A099898A]
 15 [3867C7CF] [9B6B1210] [920468C47611]
 16 [CD3C093F] [9B6B1210] [0088D3B998C1]
Cipher Text : [E5A951F59B3160C0]
=====DECRYPTION=====
Cipher Text : [E5A951F59B3160C0]
Key       : [85E813540FOAB405]
round    left      right      key
  1 [9B6B1210] [3867C7CF] [0088D3B998C1]
  2 [3867C7CF] [8FB6E536] [920468C47611]
  3 [8FB6E536] [08F5AF71] [2C30A099898A]
  4 [08F5AF71] [B56D060C] [030A16724774]
  5 [B56D060C] [EB5D0481] [8480058EFA01]
  6 [EB5D0481] [3AF0D85A] [7001388905EB]
  7 [3AF0D85A] [0F3679E5] [18448276D494]
  8 [0F3679E5] [45E5012A] [23A0C408EA4F]
  9 [45E5012A] [51ECAEF3] [08232857F2A0]
 10 [51ECAEF3] [A266EC34] [BA480088135F]
 11 [A266EC34] [916A4BE6] [0600E6F60D88]
 12 [916A4BE6] [1A760BC4] [05901843E227]
 13 [1A760BC4] [80F7FA3B] [580215A851D1]
 14 [80F7FA3B] [085BE33B] [A205801F2D8C]
 15 [085BE33B] [FOAAF0AA] [35602242C637]
 16 [CC00CCFF] [FOAAF0AA] [C842107B226C]
Plain Text : [0123456789ABCDEF]

```

Right 가 다음 라운드의 left 가 됨을 확인 할 수 있다.

암호문을 복호화 했을 때 최초의 평문이 나오는 것으로 암호화와 복호화가 정상적으로 일어남을 알 수 있다.



### 3. 소감

OTP 의 알고리즘은 단순해서 완성하는 것이 많이 어렵지 않았다. 일찍 끝났기에 조금 긴 평문을 암호화 해보고 싶어서 텍스트를 읽어와 OTP 를 통해 암호화 하고 복호화 하는 것 까지 구현을 해 보았다. OTP 알고리즘을 구현하는 것 보다 파일 입출력하고 처리하는 것을 구현하는데 시간이 훨씬 오래 걸린 것 같다. 내가 생각하는 기능을 하는 프로그램을 구현하는 능력을 많이 길러야겠다는 생각이 들었다.

DES 의 경우 처음 교재만 보았을 때에는 상세한 과정을 전부 알기 힘들어서 인터넷에서 여러 사람이 올려 놓은 정보들을 찾아보고 간신히 이해하고 만들어 낼 수 있었다. 페이스텔이 진행되는 방식 자체는 어렵지 않게 이해가 되었는데 중간에 F 함수나 라운드 키를 생성하는 방법 등이 어려웠다. 생각보다 해당 부분을 자세하게 알려주지도 않아서 여러 사람이 올린 글들을 취합하고 나서야 완전히 이해하게 되었다.한편으로는 이 암호 알고리즘을 고안한 사람과, 그걸 분해 할 방법을 알아낸 사람들이 굉장히 대단하다고 생각되었다. 후에 트리플 DES 나 AES 도 직접 내 힘으로 구현해보고 싶다.

점점 컴퓨터 보안 과목이 어렵게 느껴지고 있어서 앞으로 더 많은 시간을 투자해야겠구나라는 생각이 드는 과제였다.

## 4. 소스코드 첨부

DES.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>
enum MODE {
    ENCRYPTION, DECRYPTION
};
class DES {
private:
    std::string plaintext;
    std::string ciphertext;
    std::string keystring;
    std::vector<int> key;
    std::vector<int> keys[17];
    std::vector<int> keyPermutationTable;
    std::vector<int> rounds;
    std::vector<int> compressionPermutationTable;
    std::vector<int> binaryText;
    std::vector<int> IP; //Initial Permutation
    std::vector<int> IPinverse;
    std::vector<int> left, right, rightOrigin;
    std::vector<int> EP; //ExpansionPermutation
    std::vector<int> s[9]; //s-box
    std::vector<int> Pbox; //Permutation
    std::vector<int> string2hex(const std::string& str); //string to hex
    std::string hex2string(const std::vector<int>& hex); //hex to string
    std::vector<int> initialKeySelection(const std::vector<int>& key); //64->56
    void leftShift(std::vector<int>& key, int s); //split to 28bit - shift left
    std::vector<int> keyPermutationAndCompression(); //create 48bit round key
    std::vector<int> initialPermutation(const std::vector<int>& text);
    void splitLeftRight(const std::vector<int>& text);
    std::vector<int> expansionPermutation(const std::vector<int>& right);
    std::vector<int> XOR_Key(const std::vector<int>& right, int round);
    std::vector<int> S_box_substitution(const std::vector<int>& right);
    std::vector<int> PboxPermutation(const std::vector<int>& right);
    std::vector<int> XOR_left(const std::vector<int>& left, const std::vector<int>& right);
    void nextLeft();
    std::vector<int> lastPermutation(const std::vector<int>& text);
    void keyGeneration();
public:
    DES(std::string text, std::string key, MODE mode);
    void encryption();
    void decryption();
};
```

## DES.cpp

```
#pragma once
#include "DES.h"
DES::DES(std::string text, std::string key, MODE mode)
{
    if (mode == MODE::ENCRYPTION)
        plaintext = text;
    else if (mode == MODE::DECRYPTION)
        ciphertext = text;
    keystring = key;
    keyPermutationTable =
    { 0,
      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
    };
    rounds =
    { 0,
      1,1,2,2,2,2,2,2,1,2,2,2,2,2,1
    };
    compressionPermutationTable =
    { 0,
      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
    };
    IP =
    { 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,
      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
    };
    IPinverse =
    { 0,
      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
    };
    EP =
    { 0,
      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[1] =
{ 0,
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
};
s[2] =
{ 0,
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
};
s[3] =
{ 0,
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
};
s[4] =
{ 0,
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
};
s[5] =
{ 0,
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
};
s[6] =
{ 0,
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
};
s[7] =
{ 0,
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
};
s[8] =
{ 0,
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
};
Pbox =
{ 0,
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
};
}
void DES::encryption()
{
    std::cout << "Plain Text : [" << plaintext << "]\n";
    std::cout << "    Key      : [" << keystring << "]\n";
    binaryText = string2hex(plaintext);
    key = string2hex(keystring);
    key = initialKeySelection(key);
    keyGeneration();
    binaryText = initialPermutation(binaryText);
    splitLeftRight(binaryText);
    std::cout << "round    left            right            key\n";
    for (int round = 1; round <= 16; round++)
    {
        right = expansionPermutation(right);//F function
        right = XOR_Key(right, round);//F function
        right = S_box_substitution(right);//F function
        right = PboxPermutation(right);//F function
        right = XOR_left(left, right);
        nextLeft();
        if (round == 16)
        {
            std::vector<int> tmp;
            tmp = left;
            left = right;
            right = tmp;
        }
        std::cout << std::setw(5) << round << " [" << hex2string(left) << " ] [" <<
hex2string(right) << " ] [" << hex2string(keys[round]) << "]\n";
    }
    binaryText = lastPermutation(binaryText);//IP inverse
    ciphertext = hex2string(binaryText);
    std::cout << "Cipher Text : [" << ciphertext << "]\n";
}
void DES::decryption()
{
    std::cout << "Cipher Text : [" << ciphertext << "]\n";
    std::cout << "    Key      : [" << keystring << "]\n";
    binaryText = string2hex(ciphertext);
    key = string2hex(keystring);
    key = initialKeySelection(key);
    keyGeneration();
    binaryText = initialPermutation(binaryText);
    splitLeftRight(binaryText);
    std::cout << "round    left            right            key\n";
    for (int round = 16; round >= 1; round--)
    {
        right = expansionPermutation(right);
        right = XOR_Key(right, round);
        right = S_box_substitution(right);
        right = PboxPermutation(right);
        right = XOR_left(left, right);
        nextLeft();
        if (round == 1)
        {
            std::vector<int> tmp;
            tmp = left;
            left = right;
            right = tmp;
        }
        std::cout << std::setw(5) << 17 - round << " [" << hex2string(left) << " ] [" <<
hex2string(right) << " ] [" << hex2string(keys[round]) << "]\n";
    }
}

```

```

    }
    binaryText = lastPermutation(binaryText);
    plaintext = hex2string(binaryText);
    std::cout << "Plain Text : [" << plaintext << "]\n";
}
std::vector<int> DES::string2hex(const std::string& str)
{
    std::vector<int> hexcode;
    hexcode.resize(1 + str.size() * 4);
    for (int i = 0; i < str.size(); i++)
    {
        switch (str[i])
        {
            case '0':
                hexcode[4 * i + 1] = 0; hexcode[4 * i + 2] = 0; hexcode[4 * i + 3] = 0;
hexcode[4 * i + 4] = 0;
                break;
            case '1':
                hexcode[4 * i + 1] = 0; hexcode[4 * i + 2] = 0; hexcode[4 * i + 3] = 0;
hexcode[4 * i + 4] = 1;
                break;
            case '2':
                hexcode[4 * i + 1] = 0; hexcode[4 * i + 2] = 0; hexcode[4 * i + 3] = 1;
hexcode[4 * i + 4] = 0;
                break;
            case '3':
                hexcode[4 * i + 1] = 0; hexcode[4 * i + 2] = 0; hexcode[4 * i + 3] = 1;
hexcode[4 * i + 4] = 1;
                break;
            case '4':
                hexcode[4 * i + 1] = 0; hexcode[4 * i + 2] = 1; hexcode[4 * i + 3] = 0;
hexcode[4 * i + 4] = 0;
                break;
            case '5':
                hexcode[4 * i + 1] = 0; hexcode[4 * i + 2] = 1; hexcode[4 * i + 3] = 0;
hexcode[4 * i + 4] = 1;
                break;
            case '6':
                hexcode[4 * i + 1] = 0; hexcode[4 * i + 2] = 1; hexcode[4 * i + 3] = 1;
hexcode[4 * i + 4] = 0;
                break;
            case '7':
                hexcode[4 * i + 1] = 0; hexcode[4 * i + 2] = 1; hexcode[4 * i + 3] = 1;
hexcode[4 * i + 4] = 1;
                break;
            case '8':
                hexcode[4 * i + 1] = 1; hexcode[4 * i + 2] = 0; hexcode[4 * i + 3] = 0;
hexcode[4 * i + 4] = 0;
                break;
            case '9':
                hexcode[4 * i + 1] = 1; hexcode[4 * i + 2] = 0; hexcode[4 * i + 3] = 0;
hexcode[4 * i + 4] = 1;
                break;
            case 'A':
                hexcode[4 * i + 1] = 1; hexcode[4 * i + 2] = 0; hexcode[4 * i + 3] = 1;
hexcode[4 * i + 4] = 0;
                break;
            case 'B':
                hexcode[4 * i + 1] = 1; hexcode[4 * i + 2] = 0; hexcode[4 * i + 3] = 1;
hexcode[4 * i + 4] = 1;
                break;
            case 'C':
                hexcode[4 * i + 1] = 1; hexcode[4 * i + 2] = 1; hexcode[4 * i + 3] = 0;
hexcode[4 * i + 4] = 0;

```

```

        break;
    case 'D':
        hexcode[4 * i + 1] = 1; hexcode[4 * i + 2] = 1; hexcode[4 * i + 3] = 0;
hexcode[4 * i + 4] = 1;
        break;
    case 'E':
        hexcode[4 * i + 1] = 1; hexcode[4 * i + 2] = 1; hexcode[4 * i + 3] = 1;
hexcode[4 * i + 4] = 0;
        break;
    case 'F':
        hexcode[4 * i + 1] = 1; hexcode[4 * i + 2] = 1; hexcode[4 * i + 3] = 1;
hexcode[4 * i + 4] = 1;
        break;
    }
}
return hexcode;
}
std::string DES::hex2string(const std::vector<int>& hex)
{
    std::string str;
    for (int i = 0; i < (hex.size() - 1) / 4; i++)
    {
        int s = 8 * hex[4 * i + 1] + 4 * hex[4 * i + 2] + 2 * hex[4 * i + 3] + hex[4 * i +
4];
        switch (s)
        {
            case 0:
                str += '0';
                break;
            case 1:
                str += '1';
                break;
            case 2:
                str += '2';
                break;
            case 3:
                str += '3';
                break;
            case 4:
                str += '4';
                break;
            case 5:
                str += '5';
                break;
            case 6:
                str += '6';
                break;
            case 7:
                str += '7';
                break;
            case 8:
                str += '8';
                break;
            case 9:
                str += '9';
                break;
            case 10:
                str += 'A';
                break;
            case 11:
                str += 'B';
                break;
            case 12:
                str += 'C';

```

```

        break;
    case 13:
        str += 'D';
        break;
    case 14:
        str += 'E';
        break;
    case 15:
        str += 'F';
        break;
    }
}
return str;
}
std::vector<int> DES::initialKeySelection(const std::vector<int>& key)
{
    std::vector<int> selectedKey(57);
    for (int i = 1; i <= 56; i++)
        selectedKey[i] = key[keyPermutationTable[i]];
    return selectedKey;
}
void DES::leftShift(std::vector<int>& key, int s)
{
    std::vector<int> temp1, temp2;
    for (int i = 1; i <= 56; i++)
    {
        if (i <= 28)
        {
            if (i <= s)
                temp1.push_back(key[i]);
            key[i] = i + s <= 28 ? key[i + s] : temp1[i + s - 29];
        }
        else
        {
            if (i - 28 <= s)
                temp2.push_back(key[i]);
            key[i] = i + s <= 56 ? key[i + s] : temp2[i + s - 57];
        }
    }
}
std::vector<int> DES::keyPermutationAndCompression()
{
    std::vector<int> compressedKey(49);
    for (int i = 1; i <= 48; i++)
        compressedKey[i] = key[compressionPermutationTable[i]];
    return compressedKey;
}
std::vector<int> DES::initialPermutation(const std::vector<int>& text)
{
    std::vector<int> IPtext(65);
    for (int i = 1; i <= 64; i++)
        IPtext[i] = text[IP[i]];
    return IPtext;
}
void DES::splitLeftRight(const std::vector<int>& text)
{
    left.resize(33); right.resize(33);
    for (int i = 1; i <= 32; i++)
        left[i] = text[i];
    for (int i = 33; i <= 64; i++)
        right[i - 32] = text[i];
    rightOrigin = right;
}
std::vector<int> DES::expansionPermutation(const std::vector<int>& right)

```



```

{
    std::vector<int> expandedRight(49);
    for (int i = 1; i <= 48; i++)
        expandedRight[i] = right[EP[i]];
    return expandedRight;
}
std::vector<int> DES::XOR_Key(const std::vector<int>& right, int round)
{
    std::vector<int> newRight(49);
    for (int i = 1; i <= 48; i++)
        newRight[i] = right[i] ^ keys[round][i];
    return newRight;
}
std::vector<int> DES::S_box_substitution(const std::vector<int>& right)
{
    std::vector<int> newRight(1);
    for (int i = 1; i <= 8; i++)
    {
        int from = (i - 1) * 6 + 1, to = i * 6;
        int row = 2 * right[from] + right[to], column = 8 * right[from + 1] + 4 *
right[from + 2] + 2 * right[from + 3] + right[from + 4];
        int idx = 16 * row + column + 1;
        int num = s[i][idx];
        int a1 = num / 8; num -= a1 * 8;
        int a2 = num / 4; num -= a2 * 4;
        int a3 = num / 2; num -= a3 * 2;
        int a4 = num;
        newRight.push_back(a1);
        newRight.push_back(a2);
        newRight.push_back(a3);
        newRight.push_back(a4);
    }
    return newRight;
}
std::vector<int> DES::PboxPermutation(const std::vector<int>& right)
{
    std::vector<int> newRight(33);
    for (int i = 1; i <= 32; i++)
        newRight[i] = right[Pbox[i]];
    return newRight;
}
std::vector<int> DES::XOR_left(const std::vector<int>& left, const std::vector<int>& right)
{
    std::vector<int> newRight(33);
    for (int i = 1; i <= 32; i++)
    {
        newRight[i] = left[i] ^ right[i];
    }
    return newRight;
}
void DES::nextLeft()
{
    left = rightOrigin;
    rightOrigin = right;
}
std::vector<int> DES::lastPermutation(const std::vector<int>& text)
{
    std::vector<int> compound(1);
    std::vector<int> newText(1);
    for (int i = 1; i <= 32; i++)
        compound.push_back(left[i]);
    for (int i = 1; i <= 32; i++)
        compound.push_back(right[i]);
    for (int i = 1; i <= 64; i++)

```

```

        newText.push_back(compound[IPinverse[i]]);
    return newText;
}
void DES::keyGeneration()
{
    for (int i = 1; i <= 16; i++)
    {
        leftShift(key, rounds[i]);
        keys[i] = keyPermutationAndCompression();
    }
}

```

Main.cpp

```

#pragma once
#include "DES.h"
int main()
{
    std::cout << "=====ENCRYPTION=====\\n";
    DES des1("0123456789ABCDEF", "85E813540F0AB405", MODE::ENCRYPTION);
    des1.encryption();
    std::cout << "=====DECRYPTION=====\\n";
    DES des2("E5A951F59B3160C0", "85E813540F0AB405", MODE::DECRYPTION);
    des2.decryption();
}

```