**网络信息安全**

**实验报告**

学号： 2220193816

姓名： 赤凯

班级： 网络2班

报告日期：2022年6月25日

实验二： 密码学实验

**【实验目的】**

1. **了解对称和非对称密码体制基本原理**
2. **掌握编程语言实现对称和非对称加解密算法**

**【实验内容】**

**使用C++ 实现RSA非对称加密算法**

**【实验环境】**

**1、Linux 5.15**

**2、gcc 12.1**

**【实验步骤】**

1. **RSA 算法原理**

**1.1 生成密钥**

**1).取任意两个素数记为p,q(****)，其乘积记为N。**

**2).根据公式 φ(N)=(p-1)(q-1)，计算N的欧拉函数φ(N)。**

**3).随机选择一个整数e，条件是1< e < φ(N)，且e与φ(N) 互质。**

**4).根据公式ed ≡ 1 (mod φ(N))，计算e对于φ(N)的模反元素d。**

**5).将N和e封装成公钥(N, e)，N和d封装成私钥(N, d)。**

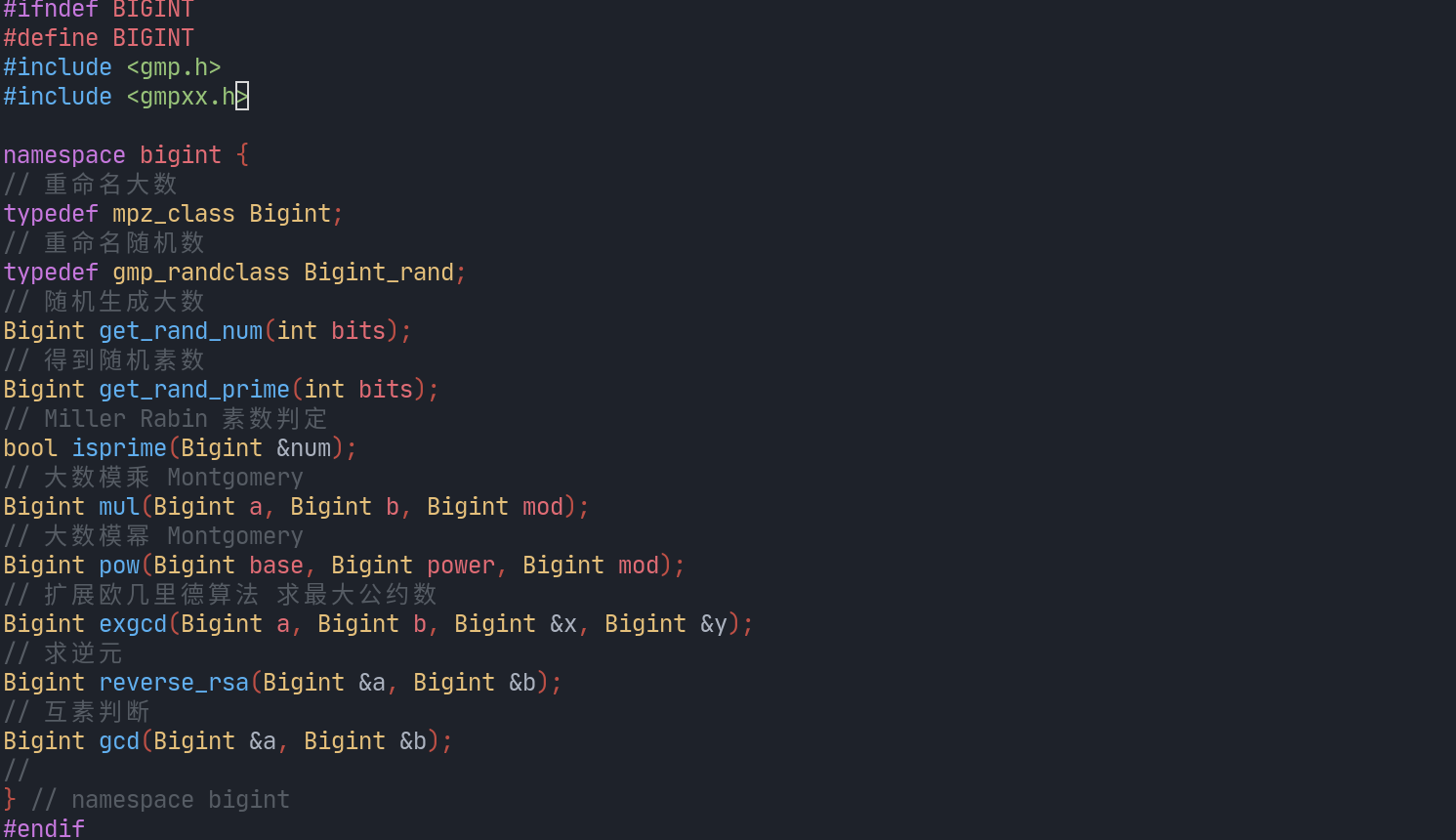
**1.2 加密与解密**

**对于明文x,用公钥（n,e)对x进行加密的过程，就是将x转换成数字，如果是字符串的话，取ASCII码值，然后通过幂取模运算计算y，也就是密文，这就是加密过程。**

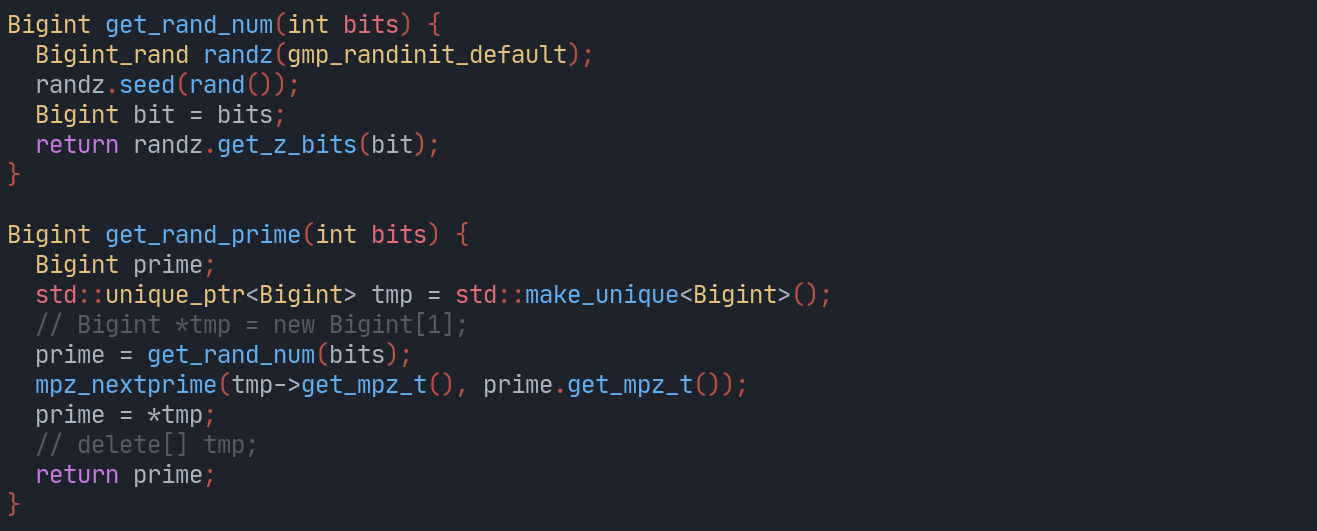
**对于密文y，用私钥（n,d)对y进行解密的过程和机密类似，同样是计算幂取模**

1. **RSA 算法实现步骤**
   1. **使用大数库**

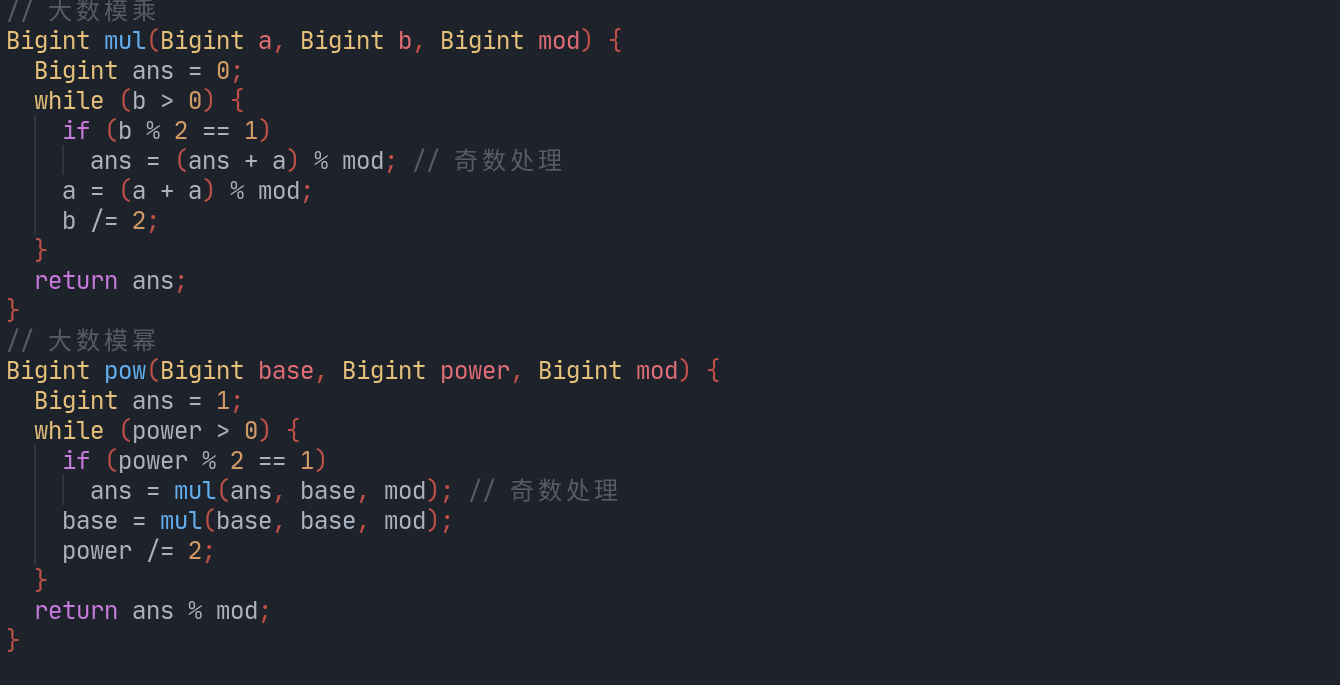
**选择使用开源大数库 GNU MP,用来生成随机大数。其他功能函数没有使用，自己实现。**

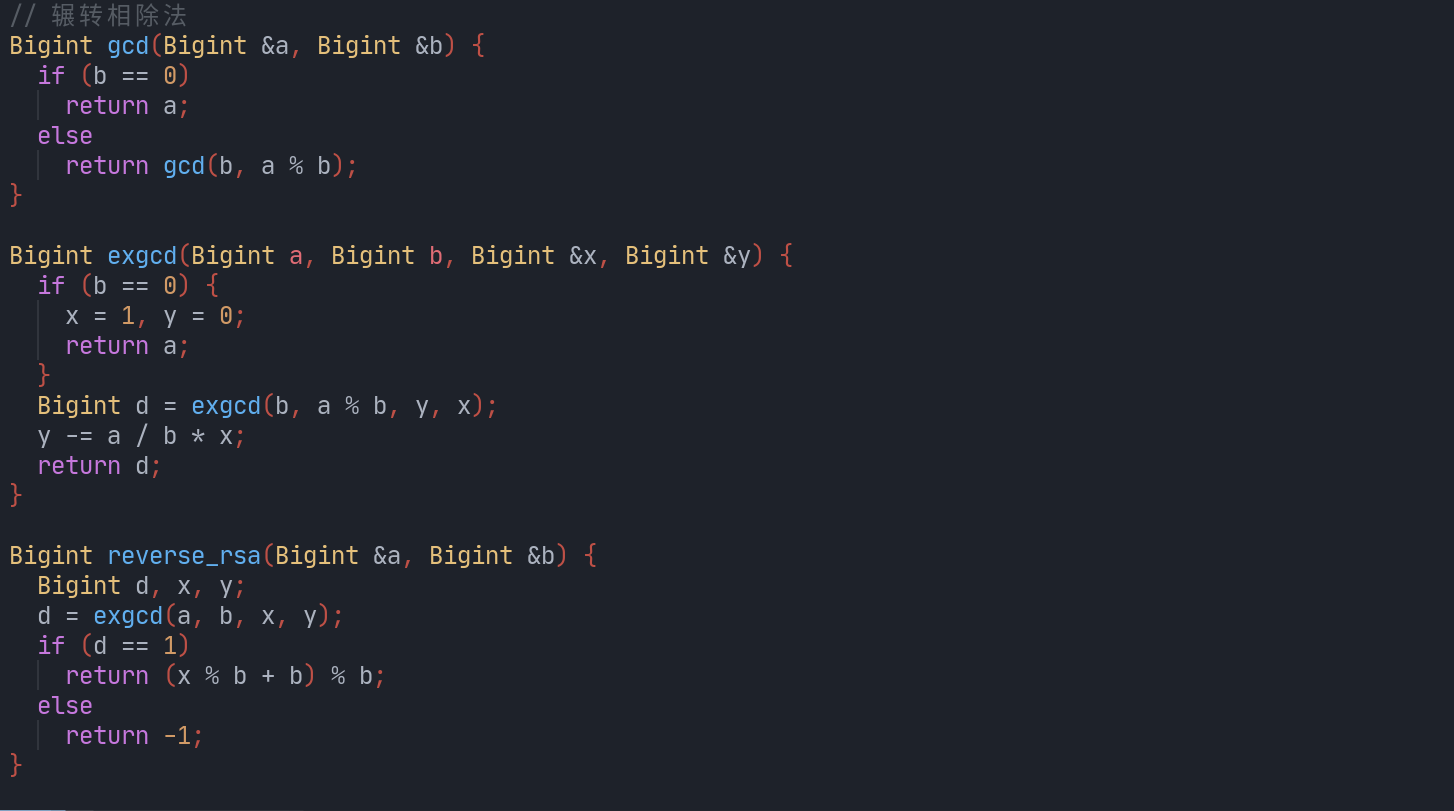
****

**生成随机大数和大素数，可以通过参数设置位数，使用大数库实现:**

****

**大数模乘和模幂，将乘法转换为加法，防止溢出:**

****

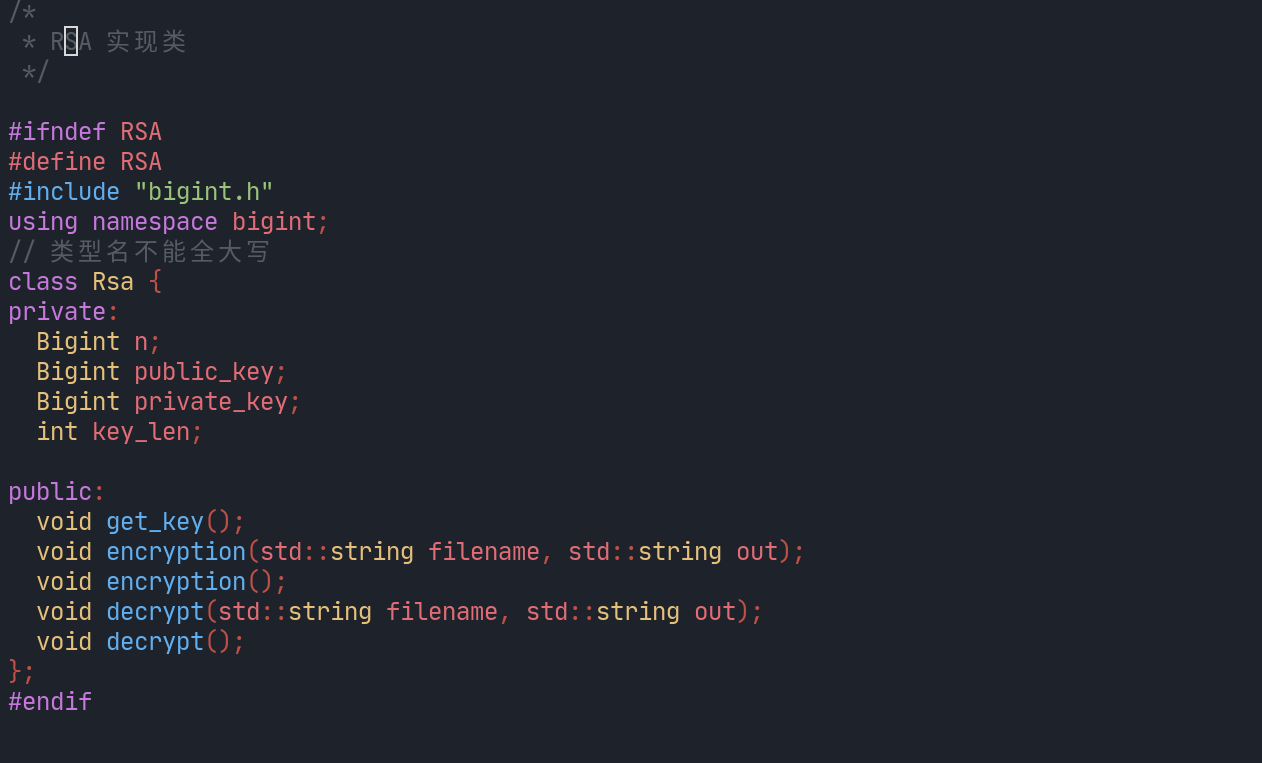
**辗转相除法和扩展欧几里德算法求逆元:  
**

**Miller\_Rabin素数判断:**

****

* 1. **RSA 类**

**RSA 类的签名:**

****

**get\_key() 函数，按照RSA原理调用实现的大数函数，先随机求得P,Q,进一步得到公共模数N和R。重点是求公钥E，需要满足 1. 1< E < R, 2. E 和 O 互素， 3. E 是素数。进行大量计算，这里通过多线程并行加速:**

****

****

****

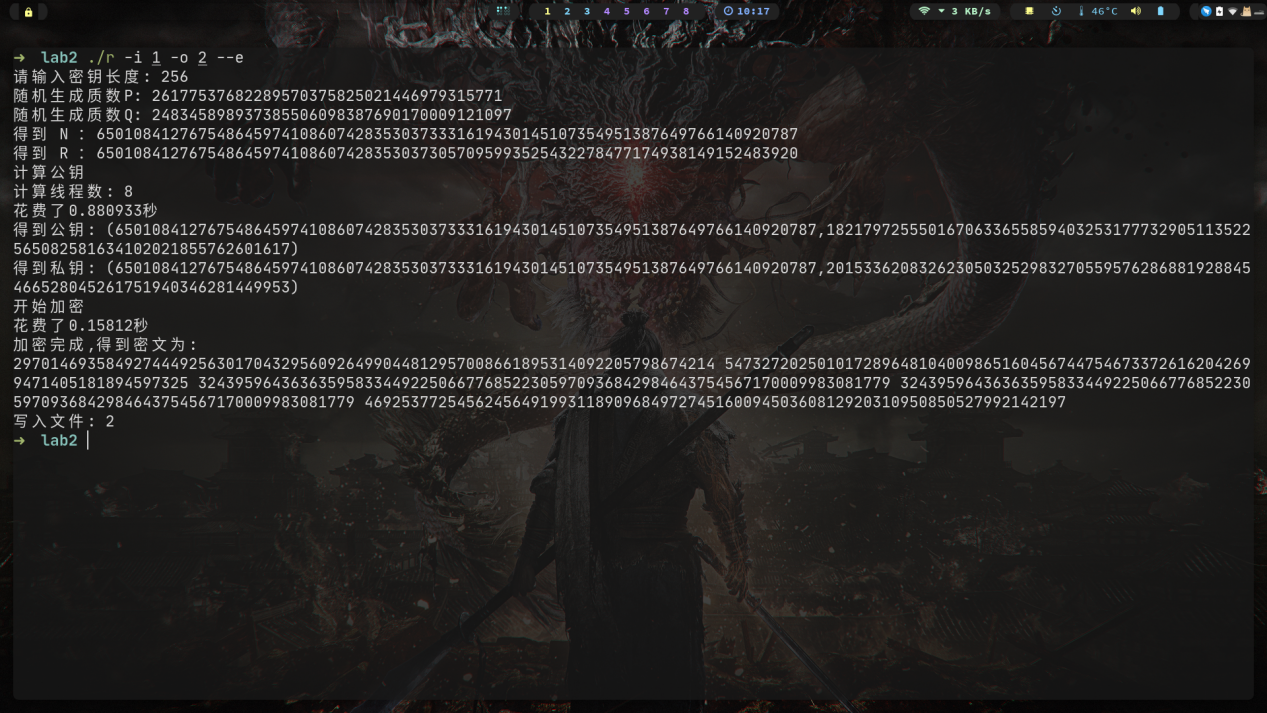
**加密过程就是将文件内容读入，按照每个字符进行模乘，再将得到的数据写入输出文件**

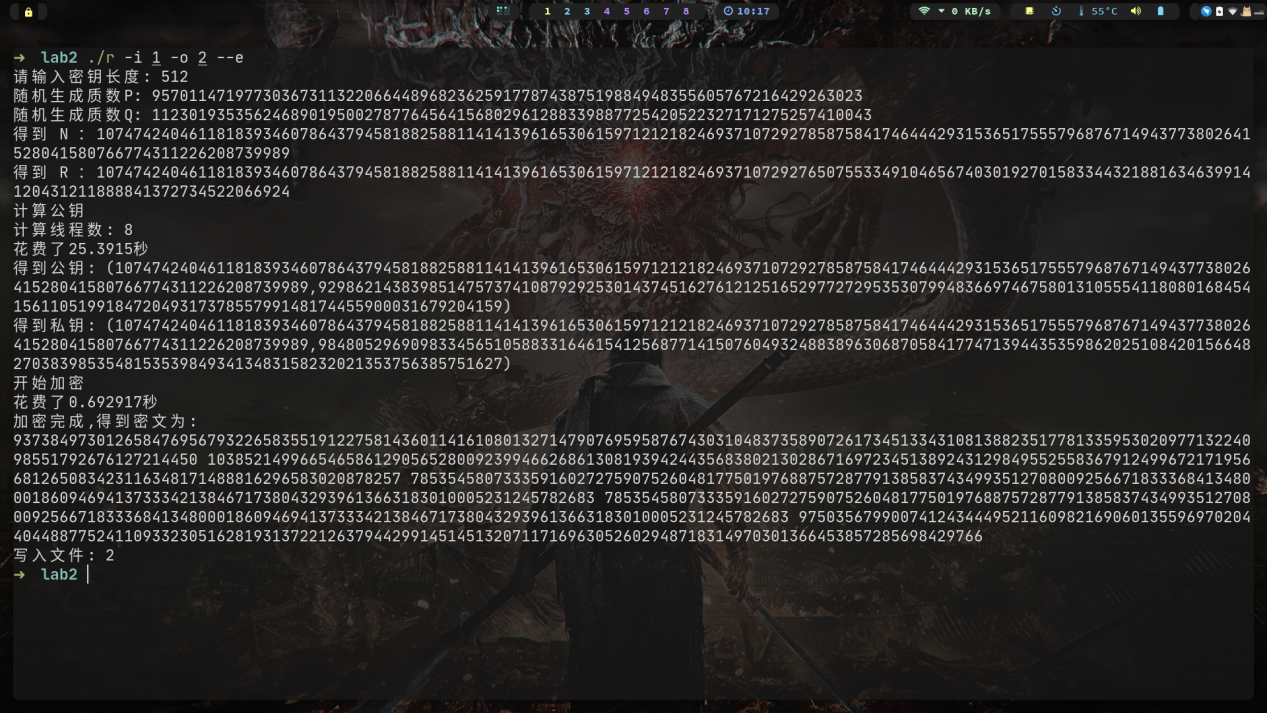
****

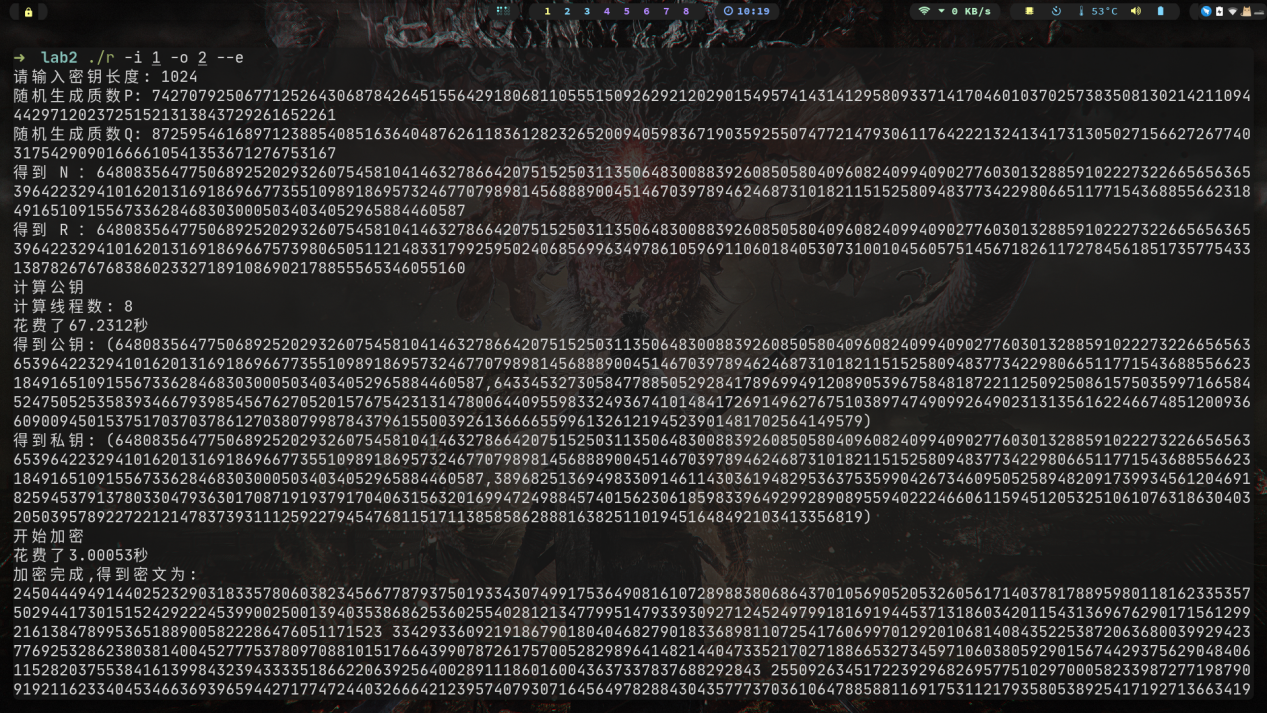
**解密过程类似。**

1. **不同长度密钥效率对比**

**下面我们对同一明文不同位数密钥的加密效率进行对比。**

****

****

****

**可以看到，对于同一明文，生成密钥的速度随着位数的增加而减缓，256位密钥和1024位密钥速度相差近60倍。同样破解难度也随之提高，现在最低的安全位数是2048位。**

**【结果分析】**

**RSA 算法在生成密钥过程中随着密钥位数的增加，花费时间也更多。通常用来加密传输中的对称加密的密钥，而不是直接用来加密文件。**

**【个人总结&心得体会】**

**通过此次实验，我进一步加深了对RSA算法的理解。自己实现算法，明白了RSA算法的瓶颈和使用场景，对我今后的工作和学习有很大的帮助。**

**源代码:**

**bigint.cc:**

#include "bigint.h"

#include <gmp.h>

#include <memory>

namespace bigint {

Bigint get\_rand\_num(int bits) {

Bigint\_rand randz(gmp\_randinit\_default);

randz.seed(rand());

Bigint bit = bits;

return randz.get\_z\_bits(bit);

}

Bigint get\_rand\_prime(int bits) {

Bigint prime;

std::unique\_ptr<Bigint> tmp = std::make\_unique<Bigint>();

// Bigint \*tmp = new Bigint[1];

prime = get\_rand\_num(bits);

mpz\_nextprime(tmp->get\_mpz\_t(), prime.get\_mpz\_t());

prime = \*tmp;

// delete[] tmp;

return prime;

}

// 大数模乘

Bigint mul(Bigint a, Bigint b, Bigint mod) {

Bigint ans = 0;

while (b > 0) {

if (b % 2 == 1)

ans = (ans + a) % mod; // 奇数处理

a = (a + a) % mod;

b /= 2;

}

return ans;

}

// 大数模幂

Bigint pow(Bigint base, Bigint power, Bigint mod) {

Bigint ans = 1;

while (power > 0) {

if (power % 2 == 1)

ans = mul(ans, base, mod); // 奇数处理

base = mul(base, base, mod);

power /= 2;

}

return ans % mod;

}

// 辗转相除法

Bigint gcd(Bigint &a, Bigint &b) {

if (b == 0)

return a;

else

return gcd(b, a % b);

}

Bigint exgcd(Bigint a, Bigint b, Bigint &x, Bigint &y) {

if (b == 0) {

x = 1, y = 0;

return a;

}

Bigint d = exgcd(b, a % b, y, x);

y -= a / b \* x;

return d;

}

Bigint reverse\_rsa(Bigint &a, Bigint &b) {

Bigint d, x, y;

d = exgcd(a, b, x, y);

if (d == 1)

return (x % b + b) % b;

else

return -1;

}

bool isprime(Bigint &num) {

if (num == 2)

return true; // 2为质数保留

if (num % 2 != 1 || num < 2)

return false; // 筛掉偶数和小于2的数

Bigint s = 0, t = num - 1;

while (t % 2 != 1) { // 当t为偶数时，继续分解, s递增

s++;

t /= 2;

}

for (int i = 0; i < 10;

i++) { // 二次探测定理, 进行十次探测, 使方程a^(num-1)=1(mod

// num)成立的解有仅有a=1或者a=num-1

Bigint\_rand randz(gmp\_randinit\_default);

Bigint a = randz.get\_z\_range(num - 1); // 随机整数a，取(1, num-1)

Bigint x = pow(a, t, num); // x为二次探测的解

for (int j = 0; j < s; j++) { // x^s=a^(num-1)

Bigint test = mul(x, x, num);

if (test == 1 && x != 1 && x != num - 1)

return false; // 若平方取模结果为1，但x不是1或者num-1，则num不是质数

x = test;

}

if (x != 1)

return false; // 费马小定理

}

return true;

}

} // namespace bigint

**RSA.cc：**

#include "rsa.h"

#include "bigint.h"

#include <atomic>

#include <chrono>

#include <fstream>

#include <iostream>

#include <mutex>

#include <sstream>

#include <thread>

#include <vector>

void Rsa::get\_key() {

// 素数长度为key的一半

std::cout << "请输入密钥长度: ";

std::cin >> key\_len;

int prime\_len = key\_len / 2;

// 得到P,Q

Bigint P = get\_rand\_prime(prime\_len);

Bigint Q = get\_rand\_prime(prime\_len);

// 计算 N 和 欧拉函数 O

std::cout << "随机生成质数P: " << P << std::endl;

std::cout << "随机生成质数Q: " << Q << std::endl;

Bigint N = P \* Q;

n = N;

Bigint O = (P - 1) \* (Q - 1);

std::cout << "得到 N ：" << N << std::endl;

std::cout << "得到 R ：" << O << std::endl;

// 并行计算 计算公钥

bool sign(true);

std::mutex mu;

// 使用8个线程并行

std::vector<std::thread> threads;

std::cout << "计算公钥" << std::endl;

unsigned concurrent\_count = std::thread::hardware\_concurrency(); //

std::cout << "计算线程数: " << concurrent\_count << std::endl;

auto start = std::chrono::system\_clock::now();

for (int i = 0; i < concurrent\_count; i++) {

threads.push\_back(std::thread([&sign, O, this, &mu]() {

while (sign) {

Bigint E = get\_rand\_num(key\_len);

if (1 < E && E < O && isprime(E) && gcd(E, O) == 1) {

mu.lock();

sign = false;

public\_key = E;

mu.unlock();

}

}

}));

}

for (auto &i : threads) {

i.join();

}

auto end = std::chrono::system\_clock::now();

auto duration =

std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "花费了"

<< double(duration.count()) \*

std::chrono::microseconds::period::num /

std::chrono::microseconds::period::den

<< "秒" << std::endl;

/\*

Bigint E;

auto start = std::chrono::system\_clock::now();

while (sign) {

E = get\_rand\_num(key\_len);

if (1 < E && E < O && isprime(E) && gcd(E, O) == 1)

break;

}

auto end = std::chrono::system\_clock::now();

auto duration =

std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "花费了"

<< double(duration.count()) \*

std::chrono::microseconds::period::num /

std::chrono::microseconds::period::den

<< "秒" << std::endl;

public\_key = E;

\*/

// 计算私钥

private\_key = reverse\_rsa(public\_key, O);

// 打印出来

std::cout << "得到公钥: (" << N.get\_str() << "," << public\_key.get\_str()

<< ")" << std::endl;

std::cout << "得到私钥: (" << N.get\_str() << "," << private\_key.get\_str()

<< ")" << std::endl;

}

void String\_Split(const std::string &str, const char split,

std::vector<std::string> &res) {

if (str == "")

return;

// 在字符串末尾也加入分隔符，方便截取最后一段

std::string strextend = str; // +split;

size\_t pos = strextend.find(split);

// 若找不到内容则字符串搜索函数返回 npos

while (pos != strextend.npos) {

std::string temp = strextend.substr(0, pos);

res.push\_back(temp);

// 去掉已分割的字符串,在剩下的字符串中进行分割

strextend = strextend.substr(pos + 1, strextend.size());

pos = strextend.find(split);

}

}

void Rsa::encryption(std::string filename, std::string out) {

get\_key();

// 读取文件到string

std::ifstream f(filename);

std::stringstream s;

s << f.rdbuf();

f.close();

//加密

std::cout << "开始加密" << std::endl;

std::string ans;

auto start = std::chrono::system\_clock::now();

for (auto i : s.str()) {

Bigint b = static\_cast<unsigned char>(i);

Bigint a = pow(b, public\_key, n);

ans += a.get\_str() + " ";

}

auto end = std::chrono::system\_clock::now();

auto duration =

std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "花费了"

<< double(duration.count()) \*

std::chrono::microseconds::period::num /

std::chrono::microseconds::period::den

<< "秒" << std::endl;

std::cout << "加密完成,得到密文为: " << std::endl;

std::cout << ans << std::endl;

std::cout << "写入文件: " << out << std::endl;

std::ofstream file;

file.open(out);

file << ans;

file.close();

}

void Rsa::decrypt(std::string filename, std::string out) {

// 读取文件到string

std::cout << "请输入N: ";

std::cin >> n;

std::cout << "请输入private\_key: ";

std::cin >> private\_key;

std::ifstream f(filename);

std::stringstream s;

s << f.rdbuf();

f.close();

// 密文拆分

std::vector<std::string> ciphertext\_string;

std::vector<Bigint> ciphertext\_gz;

String\_Split(s.str(), ' ', ciphertext\_string);

int length = ciphertext\_string.size(); // 字符个数

// cout << "字符个数为: " << length << endl; //test

// 类型转换string to mpz\_class

for (int i = 0; i < length; i++) {

Bigint temp;

temp.set\_str(ciphertext\_string[i], 0);

// cout << temp << endl;

ciphertext\_gz.push\_back(temp);

// cout << ciphertext\_gz[i] << endl; //test

}

// 解密

std::string DecryptedText;

std::cout << "解密中...\n";

auto start = std::chrono::system\_clock::now();

for (int i = 0; i < length; i++) {

Bigint c = ciphertext\_gz[i]; // 提取密文

Bigint m = pow(c, private\_key, n); // 解密

DecryptedText.append(1, static\_cast<unsigned char>(m.get\_ui()));

// cout << "目前已解密明文为:\n" << DecryptedText << endl; //test

}

auto end = std::chrono::system\_clock::now();

auto duration =

std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "花费了"

<< double(duration.count()) \*

std::chrono::microseconds::period::num /

std::chrono::microseconds::period::den

<< "秒" << std::endl;

std::cout << "解密后明文: \n";

std::cout << DecryptedText;

std::cout << "写入文件: " << out << std::endl;

std::ofstream file;

file.open(out);

file << DecryptedText;

file.close();

}

**main.cc:**

#include "cmdline.h"

#include "rsa.h"

#include <string>

int main(int argc, char \*argv[]) {

// 解析参数

cmdline::parser a;

a.add<std::string>("input", 'i', "inputfile", true, "");

a.add<std::string>("output", 'o', "outputfile", true, "");

a.add("e", '\0', "encrypt");

a.add("d", '\0', "decrypt");

a.parse\_check(argc, argv);

if (!a.exist("e") && !a.exist("d")) {

std::cout << "choose mode: enc or dec" << std::endl;

std::cout << a.usage();

return 0;

}

std::string in = a.get<std::string>("input");

std::string out = a.get<std::string>("output");

// 加密

if (a.exist("e")) {

Rsa d;

d.encryption(in, out);

} else if (a.exist("d")) {

Rsa d;

d.decrypt(in, out);

} else {

std::cout << a.usage() << std::endl;

}

return 0;

}

实验二： 密码学实验(2)

**【实验目的】**

1. **了解对称和非对称密码体制基本原理**

**2. 掌握编程语言实现对称和非对称加解密算法**

**【实验内容】**

**使用C++ 实现 DES对称加密算法**

**【实验环境】**

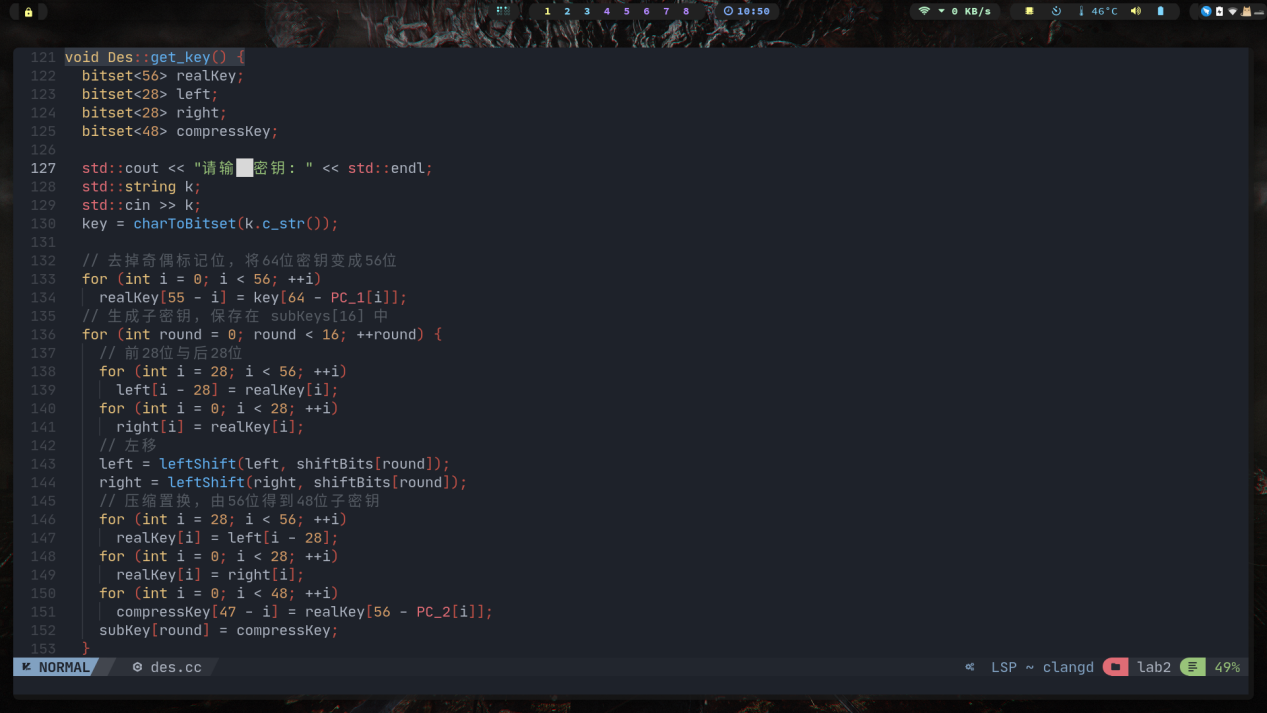
**1、Linux 5.15**

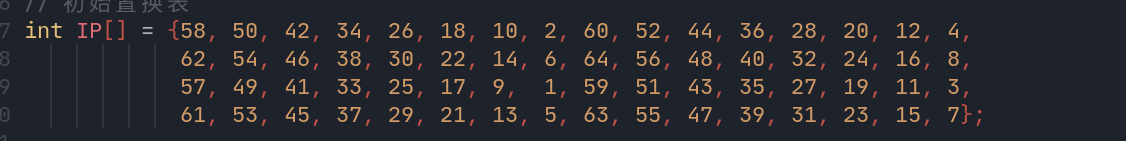
**2、gcc 12.1**

**【实验步骤】**

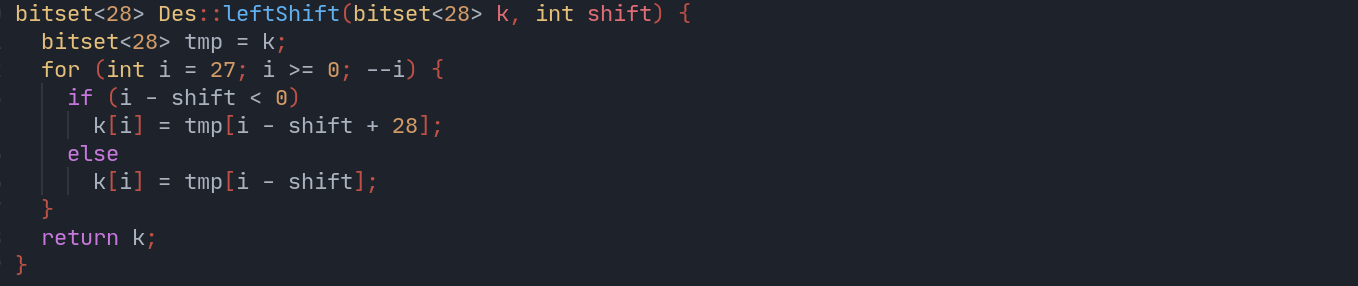
1. **获得密钥**

**首先将输入的密钥转化为64位二进制，再通过转换表将64位中的奇偶检验位去除，转换为56位。**

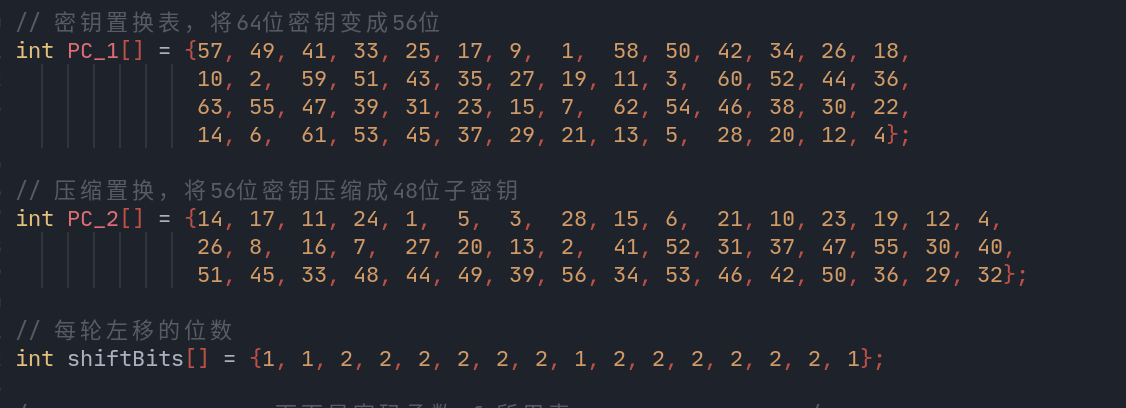
**DES使用一个56位的初始密钥，但是这里提供的是一个64位的值，这是因为在硬件实现中每8位可以用于奇偶校验，在软件实现中多出的位只是简单的忽略掉。要获得一个56位的密钥，可以执照表IP的方式执行密钥转换。按照从左往右从上往下的方式看，表格中每个位置P包含初始密钥中位在转换后的密钥中所占的位置。比如，初始密钥中的第57位就是转换后的密钥中的第1位，而初始密钥中的第49位则变成转换后的密钥中的第2位，以此类推...。（数据位的计数顺序按照从左到右从1开始的） **

****

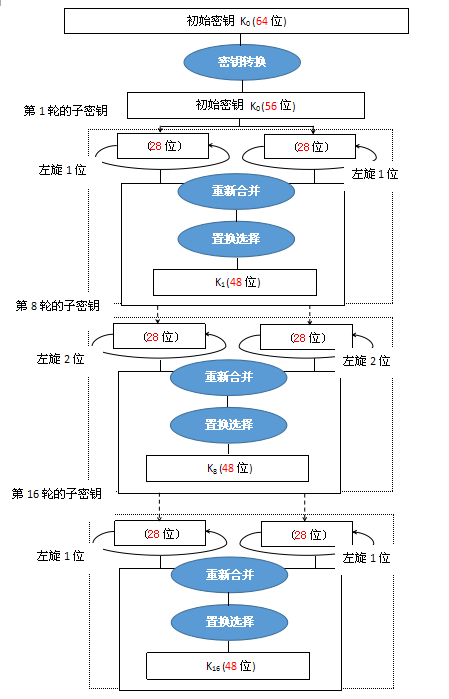
**将密钥转换为56位后，接下来计算子密钥。首先，将56位的密钥分为两个28位的组。然后，针对每个子密钥，根据子密钥的序列值（也就是16个子密钥中的第几个）旋转这两组值（旋转的位数见表shiftBits），然后重新合并。之后，再按照表PC\_2所示对重组后的密钥进行置换，使56位的子密钥缩小为48位（注意表PC\_2只有48位，丢弃了8位）。这个排列过程就称为置换选择。**

****

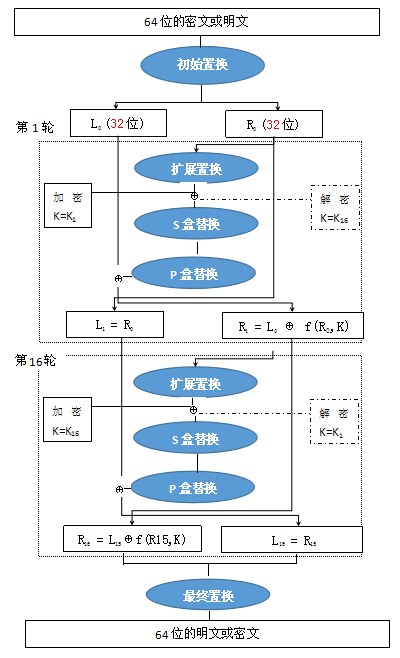
****

****

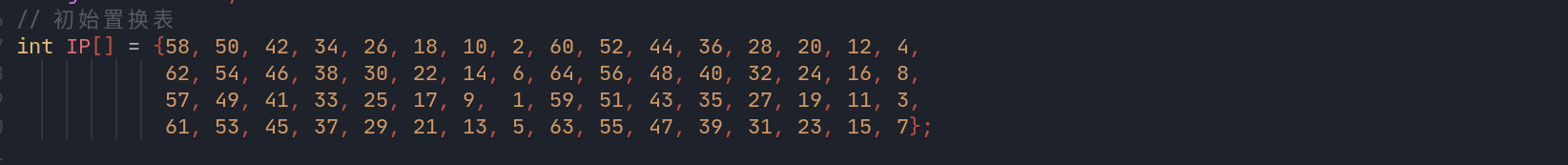
**针对16个子密钥，每个子密钥重复一次该过程。这里的目的是保证将初始密钥中的不同位在每一轮排列后应用于加密的数据上。**

****

1. **加解密过程**

****

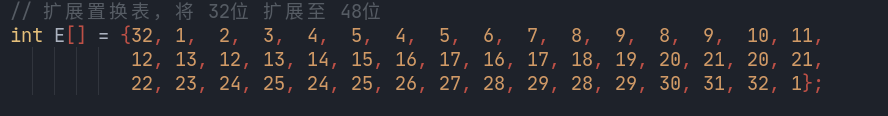
**从表IP所示的方式置换64位的数据块开始，该置换过程称为初始置换。**

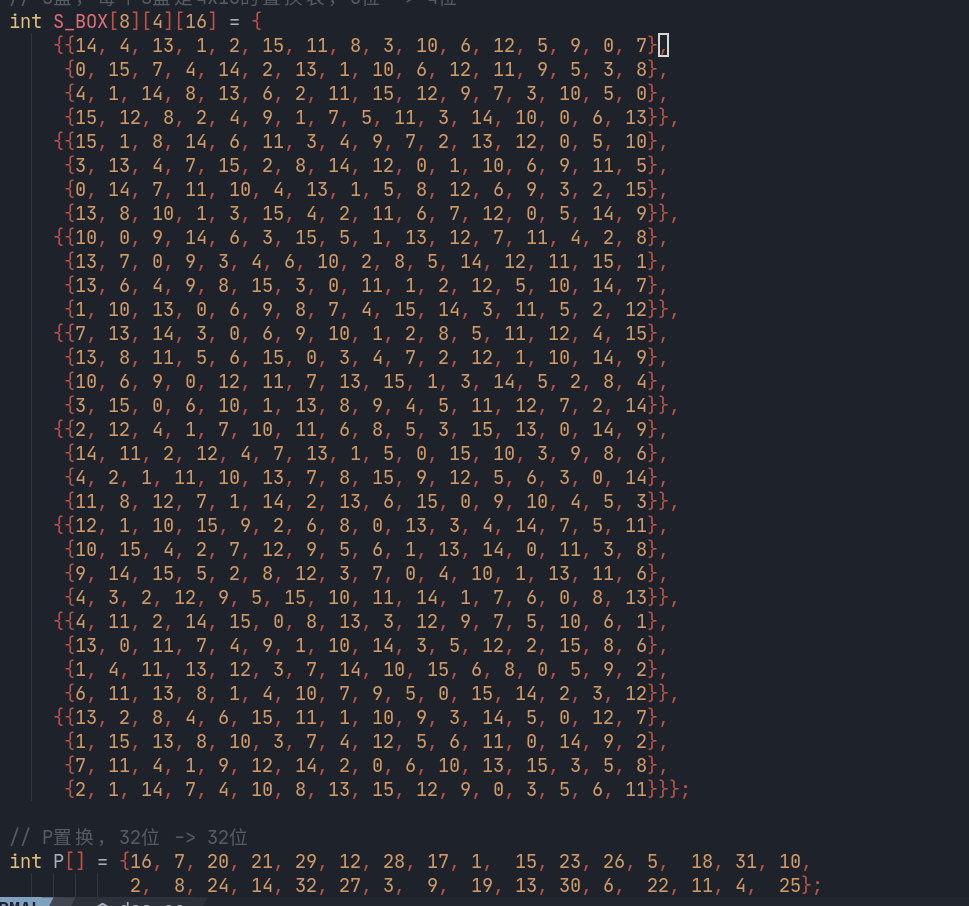
****

**然后将64位数据分为L和R各占32位。再进行16轮迭代计算，这里的f是费斯妥函数。其每次对半块（32位）进行操作，并包括四个步骤：**

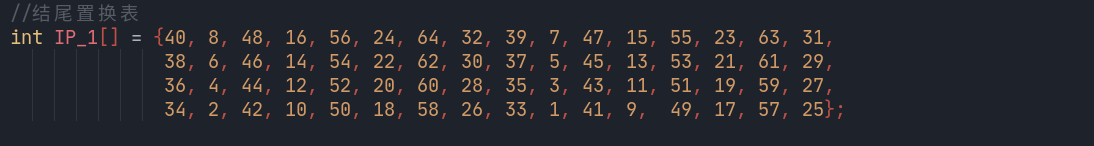
1. **扩张———用[扩张置换](https://zh.wikipedia.org/wiki/DES%E8%A1%A5%E5%85%85%E6%9D%90%E6%96%99" \l "%E6%89%A9%E5%BC%A0%E5%87%BD%E6%95%B0_(E%E5%87%BD%E6%95%B0)" \o "DES补充材料)（图中的E）将32位的半块扩展到48位，其输出包括8个6位的块，每块包含4位对应的输入位，加上两个邻接的块中紧邻的位。**
2. **与密钥混合———用异或操作将扩张的结果和一个子密钥进行混合。16个48位的子密钥—每个用于一个回次的F变换—是利用[密钥调度](https://zh.wikipedia.org/w/index.php?title=%E5%AF%86%E9%92%A5%E8%B0%83%E5%BA%A6&action=edit&redlink=1" \o "密钥调度（页面不存在）)从主密钥生成的。**
3. **S盒———在与子密钥混合之后，块被分成8个6位的块，然后使用“[S盒](https://zh.wikipedia.org/wiki/S%E7%9B%92" \o "S盒)”，或称“置换盒”进行处理。8个S盒的每一个都使用以[查找表](https://zh.wikipedia.org/wiki/%E6%9F%A5%E6%89%BE%E8%A1%A8" \o "查找表)方式提供的非线性的变换将它的6个输入位变成4个输出位。S盒提供了DES的核心安全性—如果没有S盒，密码会是线性的，很容易破解。**
4. **置换———最后，S盒的32个输出位利用固定的[置换](https://zh.wikipedia.org/wiki/%E7%BD%AE%E6%8D%A2" \o "置换)，“[P置换](https://zh.wikipedia.org/wiki/DES%E8%A1%A5%E5%85%85%E6%9D%90%E6%96%99" \l "P%E7%BD%AE%E6%8D%A2" \o "DES补充材料)”进行重组。这个设计是为了将每个S盒的4位输出在下一回次的扩张后，使用4个不同的S盒进行处理。**

****

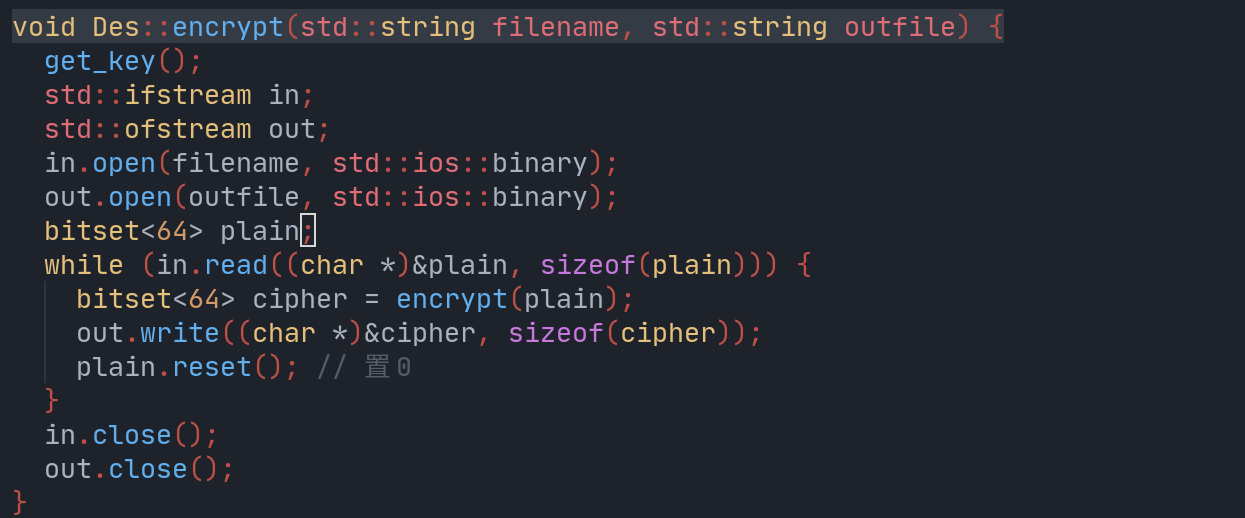
****

****

**最后，将得到的L和R 合并为RL,再按照IP\_1表进行一次尾置换得到密文。**

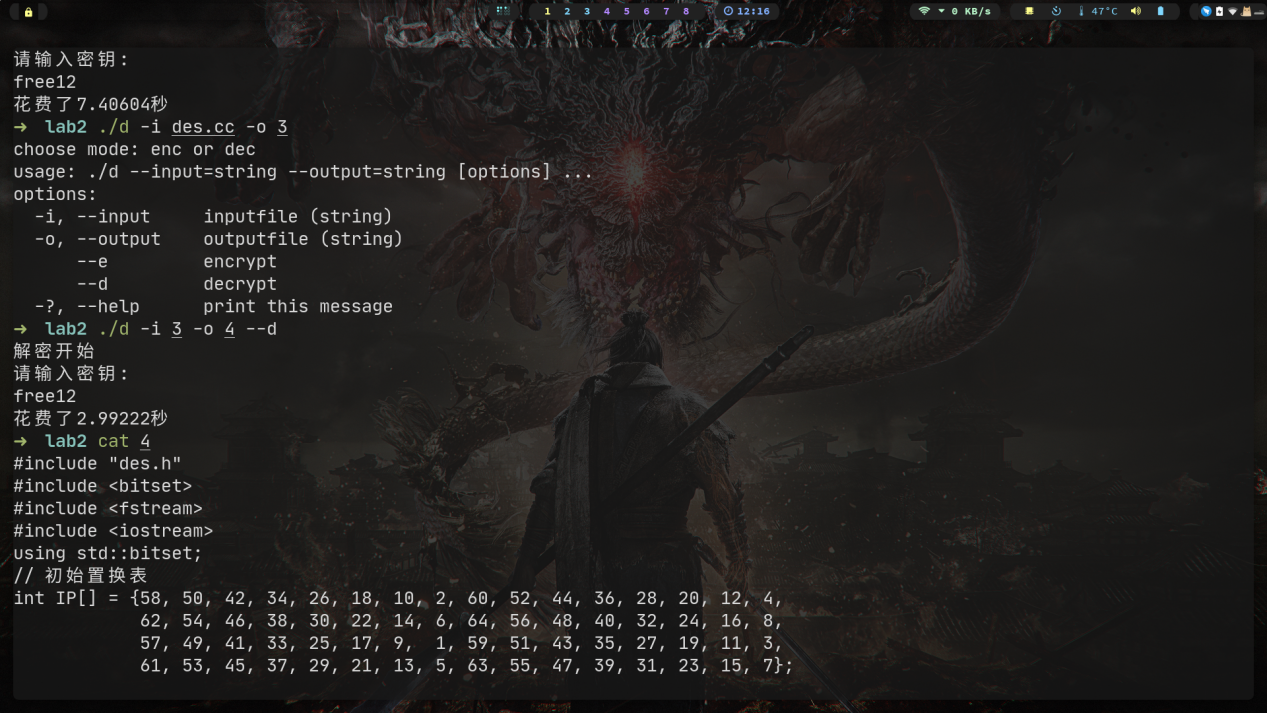
****

**加密函数从明文文件一次读取64位进行上述处理，得到密文，写入密文文件。解密过程类似。**

****

**【结果分析】**

**程序运行结果如下，可以看到与RSA相比，加密文件速度很快。**

****

**通常使用混合加密，通过非对称算法加密对称算法的密钥，使用对称算法加密文件。**

**【个人总结&心得体会】**

**对称算法的数学原理比非对称加密要复杂的多。通过此次实验我明白了，数学是加密算法的基础，不同加密算法有不同的应用场景。并不是所有场景使用非对称加密才是安全的。**

**源代码:**

**DES.h**

#ifndef DES

#define DES

#include <bitset>

#include <string>

class Des {

private:

std::bitset<64> key; // 64位密钥

std::bitset<48> subKey[16]; // 存放16轮子密钥

public:

std::bitset<64> charToBitset(const char s[8]);

std::bitset<32> f(std::bitset<32> R, std::bitset<48> k);

std::bitset<28> leftShift(std::bitset<28> k, int shift);

void get\_key();

void encrypt(std::string filename, std::string outfile);

void decrypt(std::string filename, std::string outfile);

std::bitset<64> encrypt(std::bitset<64> &plain);

std::bitset<64> decrypt(std::bitset<64> &cipher);

};

#endif

**DES.cc**

#include "des.h"

#include <bitset>

#include <fstream>

#include <iostream>

using std::bitset;

// 初始置换表

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};

//结尾置换表

int IP\_1[] = {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};

/\*------------------下面是生成密钥所用表-----------------\*/

// 密钥置换表，将64位密钥变成56位

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};

// 压缩置换，将56位密钥压缩成48位子密钥

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};

// 每轮左移的位数

int shiftBits[] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

/\*------------------下面是密码函数 f 所用表-----------------\*/

// 扩展置换表，将 32位 扩展至 48位

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};

// S盒，每个S盒是4x16的置换表，6位 -> 4位

int S\_BOX[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}}};

// P置换，32位 -> 32位

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};

// 费斯妥函数（F函数）

bitset<32> Des::f(bitset<32> R, bitset<48> k) {

bitset<48> expandR;

// 第一步：扩展置换，32 -> 48

for (int i = 0; i < 48; ++i)

expandR[47 - i] = R[32 - E[i]];

// 第二步：异或

expandR = expandR ^ k;

// 第三步：查找S\_BOX置换表

bitset<32> output;

int x = 0;

for (int i = 0; i < 48; i = i + 6) {

int row = expandR[47 - i] \* 2 + expandR[47 - i - 5];

int col = expandR[47 - i - 1] \* 8 + expandR[47 - i - 2] \* 4 +

expandR[47 - i - 3] \* 2 + expandR[47 - i - 4];

int num = S\_BOX[i / 6][row][col];

bitset<4> binary(num);

output[31 - x] = binary[3];

output[31 - x - 1] = binary[2];

output[31 - x - 2] = binary[1];

output[31 - x - 3] = binary[0];

x += 4;

}

// 第四步：P-置换，32 -> 32

bitset<32> tmp = output;

for (int i = 0; i < 32; ++i)

output[31 - i] = tmp[32 - P[i]];

return output;

}

bitset<28> Des::leftShift(bitset<28> k, int shift) {

bitset<28> tmp = k;

for (int i = 27; i >= 0; --i) {

if (i - shift < 0)

k[i] = tmp[i - shift + 28];

else

k[i] = tmp[i - shift];

}

return k;

}

void Des::get\_key() {

bitset<56> realKey;

bitset<28> left;

bitset<28> right;

bitset<48> compressKey;

std::cout << "请输入密钥: " << std::endl;

std::string k;

std::cin >> k;

key = charToBitset(k.c\_str());

// 去掉奇偶标记位，将64位密钥变成56位

for (int i = 0; i < 56; ++i)

realKey[55 - i] = key[64 - PC\_1[i]];

// 生成子密钥，保存在 subKeys[16] 中

for (int round = 0; round < 16; ++round) {

// 前28位与后28位

for (int i = 28; i < 56; ++i)

left[i - 28] = realKey[i];

for (int i = 0; i < 28; ++i)

right[i] = realKey[i];

// 左移

left = leftShift(left, shiftBits[round]);

right = leftShift(right, shiftBits[round]);

// 压缩置换，由56位得到48位子密钥

for (int i = 28; i < 56; ++i)

realKey[i] = left[i - 28];

for (int i = 0; i < 28; ++i)

realKey[i] = right[i];

for (int i = 0; i < 48; ++i)

compressKey[47 - i] = realKey[56 - PC\_2[i]];

subKey[round] = compressKey;

}

}

bitset<64> Des::charToBitset(const char s[8]) {

bitset<64> bits;

for (int i = 0; i < 8; ++i)

for (int j = 0; j < 8; ++j)

bits[i \* 8 + j] = ((s[i] >> j) & 1);

return bits;

}

void Des::encrypt(std::string filename, std::string outfile) {

get\_key();

std::ifstream in;

std::ofstream out;

in.open(filename, std::ios::binary);

out.open(outfile, std::ios::binary);

bitset<64> plain;

while (in.read((char \*)&plain, sizeof(plain))) {

bitset<64> cipher = encrypt(plain);

out.write((char \*)&cipher, sizeof(cipher));

plain.reset(); // 置0

}

in.close();

out.close();

}

void Des::decrypt(std::string filename, std::string outfile) {

get\_key();

std::ifstream in;

std::ofstream out;

in.open(filename, std::ios::binary);

out.open(outfile, std::ios::binary);

bitset<64> plain;

while (in.read((char \*)&plain, sizeof(plain))) {

bitset<64> temp = decrypt(plain);

out.write((char \*)&temp, sizeof(temp));

plain.reset(); // 置0

}

in.close();

out.close();

}

bitset<64> Des::encrypt(bitset<64> &plain) {

bitset<64> cipher;

bitset<64> currentBits;

bitset<32> left;

bitset<32> right;

bitset<32> newLeft;

// 第一步：初始置换IP

for (int i = 0; i < 64; ++i)

currentBits[63 - i] = plain[64 - IP[i]];

// 第二步：获取 Li 和 Ri

for (int i = 32; i < 64; ++i)

left[i - 32] = currentBits[i];

for (int i = 0; i < 32; ++i)

right[i] = currentBits[i];

// 第三步：共16轮迭代

for (int round = 0; round < 16; ++round) {

newLeft = right;

right = left ^ f(right, subKey[round]);

left = newLeft;

}

// 第四步：合并L16和R16，注意合并为 R16L16

for (int i = 0; i < 32; ++i)

cipher[i] = left[i];

for (int i = 32; i < 64; ++i)

cipher[i] = right[i - 32];

// 第五步：结尾置换IP-1

currentBits = cipher;

for (int i = 0; i < 64; ++i)

cipher[63 - i] = currentBits[64 - IP\_1[i]];

// 返回密文

return cipher;

}

bitset<64> Des::decrypt(bitset<64> &cipher) {

bitset<64> plain;

bitset<64> currentBits;

bitset<32> left;

bitset<32> right;

bitset<32> newLeft;

// 第一步：初始置换IP

for (int i = 0; i < 64; ++i)

currentBits[63 - i] = cipher[64 - IP[i]];

// 第二步：获取 Li 和 Ri

for (int i = 32; i < 64; ++i)

left[i - 32] = currentBits[i];

for (int i = 0; i < 32; ++i)

right[i] = currentBits[i];

// 第三步：共16轮迭代（子密钥逆序应用）

for (int round = 0; round < 16; ++round) {

newLeft = right;

right = left ^ f(right, subKey[15 - round]);

left = newLeft;

}

// 第四步：合并L16和R16，注意合并为 R16L16

for (int i = 0; i < 32; ++i)

plain[i] = left[i];

for (int i = 32; i < 64; ++i)

plain[i] = right[i - 32];

// 第五步：结尾置换IP-1

currentBits = plain;

for (int i = 0; i < 64; ++i)

plain[63 - i] = currentBits[64 - IP\_1[i]];

// 返回明文

return plain;

}

**main.cc**

#include "cmdline.h"

#include "des.h"

#include <chrono>

#include <string>

int main(int argc, char \*argv[]) {

// 解析参数

cmdline::parser a;

a.add<std::string>("input", 'i', "inputfile", true, "");

a.add<std::string>("output", 'o', "outputfile", true, "");

a.add("e", '\0', "encrypt");

a.add("d", '\0', "decrypt");

a.parse\_check(argc, argv);

if (!a.exist("e") && !a.exist("d")) {

std::cout << "choose mode: enc or dec" << std::endl;

std::cout << a.usage();

return 0;

}

std::string in = a.get<std::string>("input");

std::string out = a.get<std::string>("output");

// 加密

if (a.exist("e")) {

std::cout << "加密开始" << std::endl;

Des d;

auto start = std::chrono::system\_clock::now();

d.encrypt(in, out);

auto end = std::chrono::system\_clock::now();

auto duration =

std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "花费了"

<< double(duration.count()) \*

std::chrono::microseconds::period::num /

std::chrono::microseconds::period::den

<< "秒" << std::endl;

} else if (a.exist("d")) {

std::cout << "解密开始" << std::endl;

Des d;

auto start = std::chrono::system\_clock::now();

d.decrypt(in, out);

auto end = std::chrono::system\_clock::now();

auto duration =

std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

std::cout << "花费了"

<< double(duration.count()) \*

std::chrono::microseconds::period::num /

std::chrono::microseconds::period::den

<< "秒" << std::endl;

} else {

std::cout << a.usage() << std::endl;

}

return 0;

}