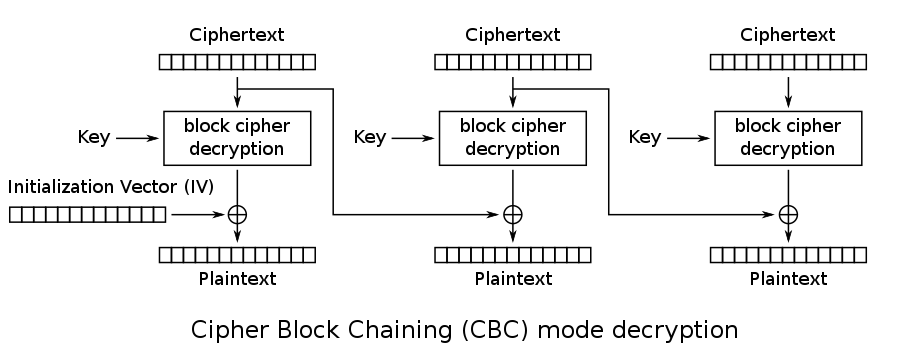
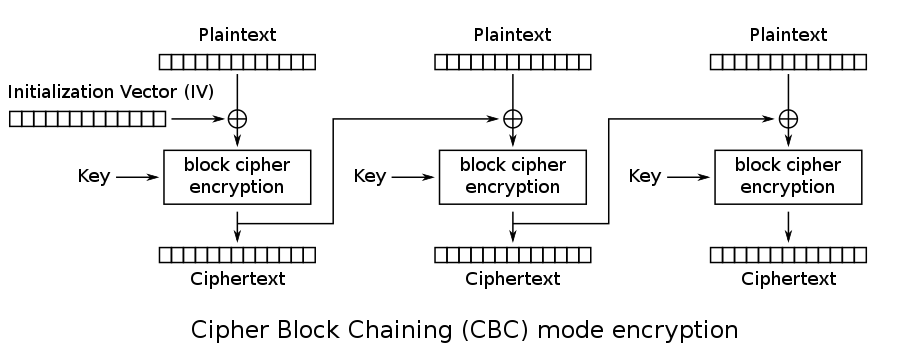
**CBC模式的DES加解密过程**

# 程序简介

整个项目主要分为三个模块： **DES加解密模块** 、 **CBC模式加解密模块** 、 **测试模块** ；

* DES加解密模块：密钥长度为8字节，明文长度为8字节，加解密前，先利用密钥生成16分子密钥用于后续的16轮轮函数的使用；对明密文转化为64为的二进制形式，然后进行 **初始置换** 、 **十六轮轮函数计算** 、 **初始逆置换** ；对于十六轮轮函数，大致流程为，将明文划分为两部分，将旧的右部作为新的左部，右部同这一轮的子密钥进行F函数操作，得到的结果与旧左部进行异或得到的结果作为新右部；对于F函数，是将32位的右部通过扩展表扩展为48位（E盒置换），然后与子密钥进行异或，最后通过S盒函数压缩成为32位，最后进行一次P盒置换即可；对于S盒函数，就是将48位分为8组，每组6位，取第一位和最后一位组合成表示的十进制数作为行，中间四位表示的十进制作为列，用S盒中的数值进行替换即可。对于解密，将十六轮轮函数F中的操作反向即可。
* CBC模式加密模块：CBC需要指定一个8位的密钥用于DES，同时需要指定一个 **初始向量IV** ，对于给定的明文，将使用0扩展到8字节的整数倍，然后对每8字节明文进行如下操作：首先明文与IV进行异或，然后进行DES加密，结果作为这一组的加密结果，然后使用新得到的密文更新IV，开始下一轮操作，最后将所有的密文合并即可。对于解密操作，同样分组后进行加密的反向操作： 利用密钥以及密文进行DES解密，解密后的结果与初始向量IV进行异或即可得到该组的明文，同时用原密文更新IV即可进行下一组的解密操作。
* 测试模块：调用CBC加解密的一个逻辑模块，选择加解密以及退出操作，给定密钥、初始向量以及明文密文即可得到最后的执行结果。

# 程序的流程图

# 程序代码

## DES.h

DES的头文件，给出所有接口函数的定义等。

#ifndef DES\_H  
#define DES\_H  
  
// DES加解密的头文件  
  
void DES\_SetKey(const char\* key); // 设置密钥key，并初始化出16个子密钥  
void DES\_GetSubkey(); // 根据密钥得到16个子密钥，密钥由61位通过PC\_1表压缩置换变为56位（即去除校验位），然后分块，循环左移，通过PC\_2表压缩置换得到16个子密钥  
void RotaleLeft(bool \*A, const int len, const int loop); // 对数组A进行循环左移loop次  
void Func\_F(bool \*R, const bool \*Subkey\_i); // F轮函数，将32位的数据扩展、与子密钥加密、S盒函数压缩置换得到32位结果  
void Func\_S(const bool \*IN, bool \*OUT); // S盒函数，将48位的数据8位一组分成6组，并取首位两位，以及中间四位作为行列查询S盒表S\_Box压缩为4位  
void Func\_Xor(bool \*A, const bool \*B, const int len); // 数组逐位异或函数，将结果保存到A中  
  
void Transform(const bool \*IN, bool \*OUT, int len, const char \*T\_Table); // 压缩置换函数，根据表 T\_Table 将输入数据转化压缩  
  
void ByteToBit(const char \*IN, bool \*OUT, int len, const int base); // 将字节转化为位  
void BitToByte(const bool \*IN, int len, char \*OUT, const int base); // 将位转化为字节  
void BitToHex(const bool \*IN, int len, char \*OUT); // 将二进制位转化为十六进制  
void ByteToHex(const char\* IN, int len, char \*OUT); // 将字节转化成十六进制  
void HexToByte(const char\* IN, int len, char\* OUT); // 将字节形式的十六进制字符串转为字节  
  
enum {ENCRYPT,DECRYPT};// ENCRYPT:加密,DECRYPT:解密 // DES加解密，默认为True加密  
void DES(char \*key, char \*p, bool mode); // DES 入口函数，key为密钥，p为加解密后的明文或密文，由参数mode决定  
void DES(char \*key, bool \*plaintext, bool mode);  
void DES\_SETKEYFLAG();  
  
#endif

## DES.cpp

DES的实现文件。

#include"DES.h"  
#include<ctype.h>  
#include<cstring>  
  
#include<bits/stdc++.h>  
using namespace std;  
  
/\*  
\* 子密钥的形成  
\* 密钥key的长度为8字节字符串，转为64位的二进制  
\* 利用PC\_1压缩置换，去除每8位一组中的最后一位，得到56位密钥  
\* 将key分成两部分，然后循环左移，左移的位数为loop，合并后进行PC\_2压缩置换即可，重复16次  
\*/  
bool KEY[64]; // 密钥为8个字节64位  
bool SubKey[16][48]; // 子密钥为16个48位  
  
// PC\_1 压缩置换表，将原始64位密钥转为56位  
const char PC\_1[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  
};  
// PC\_2 压缩置换表，将56位密钥转为48位  
const char PC\_2[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  
};  
//左移位数表   
const char loop[16] = {  
 1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1  
};  
  
// 设置密钥，并获得子密钥  
void DES\_SetKey(const char\* key){  
 ByteToBit(key, KEY, 64, 8);  
 DES\_GetSubkey();  
 return;  
}  
  
// 获得子密钥  
void DES\_GetSubkey(){  
 memset(SubKey, 0, sizeof(SubKey));  
 bool key[56];  
 Transform(KEY, key, 56, PC\_1);  
 bool \*L = &key[0], \*R = &key[28];  
 for(int i = 0; i < 16; ++i){  
 RotaleLeft(L, 28, loop[i]);  
 RotaleLeft(R, 28, loop[i]);  
 Transform(key, SubKey[i], 48, PC\_2);  
 }  
 return;  
}  
  
// 循环左移函数，对长为len的二进制串A，循环左移loop长度  
void RotaleLeft(bool \*A, const int len, const int loop){  
 bool tmp[256];  
 memcpy(tmp, A, loop);  
 memcpy(A, A + loop, len - loop);  
 memcpy(A + len - loop, tmp, loop);  
 return;  
}  
//E位选择表(扩展置换表)，将32位的数据扩展为48位：4位一组，每组最右边一位填充到下一组的最左边一位，每组最左边一位填充到上一组的最右一位  
const char E[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  
};  
//P换位表(单纯换位表)，将S盒后32位的数据换位  
const char P[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  
};  
  
// S盒[48->32]，将48位的数据每6位一组分成8组，每一组取第一位和最后一位的十进制数作为该组S盒对应的行数，中间四位对应的十进制数为该组S盒对应的列数，最后将十进制数转为4位二进制  
const char S[8][4][16] = {  
 // S1   
 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,  
 //S2  
 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,  
 //S3  
 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,  
 //S4  
 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,  
 //S5  
 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,  
 //S6  
 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,  
 //S7  
 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,  
 //S8  
 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  
};  
  
  
// 十六轮轮函数中的F函数，完成对右部的扩展、与子密钥异或、S盒压缩为32位以及P盒置换等操作  
void Func\_F(bool \*R, const bool \*Subkey\_i){  
 bool R\_ex[48]; // 将32位压缩置换后的48位数据  
 Transform(R, R\_ex, 48, E); // E盒置换  
 Func\_Xor(R\_ex, Subkey\_i, 48);  
 Func\_S(R\_ex, R); // 通过S盒函数将48位变为32位  
 Transform(R, R, 32, P); // P盒置换  
 return;  
}  
  
// S盒函数，对48位长的二进制串，  
// 分为8组，每组6位，取第一位以及最后一位表示的十进制作为行数，  
// 取中间四位表示的十进制串作为列数，获得S盒中的值  
void Func\_S(const bool \*IN, bool \*OUT){  
 int col, row;  
 for(int i = 0, j = 0, k = 0; i < 48; i += 6, ++j, k += 4){  
 row = (IN[i] << 1) + IN[i + 5];  
 col = (IN[i + 1] << 3) + (IN[i + 2] << 2) + (IN[i + 3] << 1) + IN[i + 4];  
 ByteToBit(&S[j][row][col], OUT + k, 4, 4);  
 }  
 return;  
}  
  
// 对两个数组进行异或，结果保存到前一个数组中  
void Func\_Xor(bool \*A, const bool \*B, const int len){  
 for(int i = 0; i < len; ++i){  
 A[i] ^= B[i];  
 }  
}  
  
// 压缩置换函数，将输入的数组按照给定的置换表进行压缩置换  
void Transform(const bool \*IN, bool \*OUT, int len, const char \*T\_Table){  
 bool tmp[256];  
 for(int i = 0; i < len; ++i){  
 tmp[i] = IN[T\_Table[i] - 1];  
 }  
 memcpy(OUT, tmp, len);  
}  
  
void ByteToBit(const char \*IN, bool \*OUT, int len, const int base){  
 for(int i = 0; i < len; ++i){  
 OUT[i] = (IN[i >> 3] >> (base - 1 - (i & 7))) & 1;  
 }  
 return;  
}  
  
void BitToByte(const bool \*IN, int len, char \*OUT, const int base){  
 memset(OUT, 0, (len + 7) / 8);  
 for(int i = 0; i < len; ++i){  
 OUT[i >> 3] |= (IN[i] << (base - 1 - (i & 7)));  
 }  
 return;  
}  
  
void BitToHex(const bool \*IN, int len, char \*OUT){  
 // char hex[] = "0123456789abcdefg";  
 char hex[] = "0123456789ABCDEFG";  
 for(int i = 0; i < len; i += 4){  
 OUT[i >> 2] = hex[(IN[i] << 3) + (IN[i + 1] << 2) + (IN[i + 2] << 1) + IN[i + 3]];  
 }  
 return;  
}  
void ByteToHex(const char\* IN, int len, char \*OUT){  
 bool bits[1024];  
 ByteToBit(IN, bits, len << 3, 8);  
 BitToHex(bits, len << 3, OUT);  
 return;  
}  
inline char HexToByte(const char ch){  
 if(ch >= '0' && ch <= '9')return (char)(ch & 0x00000000f);  
 else return (char)(ch - 'A' + 10);  
}  
void HexToByte(const char\* IN, int len, char\* OUT){  
 for(int i = 0; i < len; i += 2){  
 OUT[i / 2] = (HexToByte(IN[i]) << 4) | HexToByte(IN[i + 1]);  
 }  
 return;  
}  
  
// 初始置换表，将数据置换  
const char IP[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  
};  
// 初始逆置换表，  
const char IP\_inv[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  
};   
bool KEYFLAG = true;  
void DES\_SETKEYFLAG(){  
 KEYFLAG = true;  
}  
// DES主要接口，对给定密钥以及数据和方式进行一定的操作，结果保存到双向变量p中  
void DES(char \*key, char \*p, bool mode){  
 if(KEYFLAG)DES\_SetKey(key);KEYFLAG = false;  
 bool plaintext[64];  
 ByteToBit(p, plaintext, 64, 8); // 将字节数据变为位数据  
 for(int i = 0; i < 64; ++i){  
 if(i % 8 == 0)cout << " ";cout << plaintext[i];  
 }  
 DES(key, plaintext, mode);   
 BitToByte(plaintext, 64, p, 8); // 位转字节   
}  
  
void DES(char \*key, bool \*plaintext, bool mode){  
 if(KEYFLAG)DES\_SetKey(key);KEYFLAG = false;  
 bool \*L = (plaintext), \*R = (plaintext + 32), tmp[32];  
 Transform(plaintext, plaintext, 64, IP); // 初始置换  
 if(mode == ENCRYPT){  
 // 进行16轮轮函数  
 for(int i = 0; i < 16; ++i){  
 memcpy(tmp, R, 32); // 保存未处理的右部分  
 Func\_F(R, SubKey[i]); // 对右部和子密钥进行F函数操作  
 Func\_Xor(R, L, 32); // 左部与右部异或，结果保存到右部  
 memcpy(L, tmp, 32); // 将原来的右部作为左部  
 }  
 }  
 else{  
 // 与加密过程相反即可解密  
 for(int i = 15; i >= 0; --i){  
 memcpy(tmp, L, 32);  
 Func\_F(L, SubKey[i]);  
 Func\_Xor(L, R, 32);  
 memcpy(R, tmp, 32);  
 }  
 }  
 Transform(plaintext, plaintext, 64, IP\_inv); // 一次逆置换   
 return;  
}

## CBC.h

CBC的主要接口的头文件。

#ifndef CBC\_H  
#define CBC\_H  
  
#include"DES.h"  
// CBC 加解密头文件  
  
void CBC(char\* key, char\* in, int len\_in, bool mode, char\* out, int\* len\_out); // CBC 加解密入口  
void CBC\_SetIV(const bool \*iv); // 设置IV向量  
  
#endif

## CBC.cpp

CBC模式的主要实现文件：

#include"CBC.h"  
#include"DES.h"  
#include<cstring>  
  
  
bool IV[64];  
bool plaintext[64];  
bool ciphertext[64];  
char Hex[32];  
void CBC(char\* key, char\* in, int len\_in, bool mode, char\* out, int\* len\_out){  
 \*len\_out = len\_in = (len\_in + 7) / 8 \* 8;  
 // cout << len\_in << endl;  
 if(mode == ENCRYPT){  
 for(int i = 0; i < len\_in; i += 8){  
 // cout << (in + i) << endl;  
 ByteToBit(in + i, plaintext, 64, 8);  
 // for(int i = 0; i < 64; ++i)cout << plaintext[i];cout << endl;  
 // for(int i = 0; i < 64; ++i)cout << IV[i];cout << endl;  
 Func\_Xor(plaintext, IV, 64);  
 // for(int i = 0; i < 64; ++i)cout << plaintext[i];cout << endl;cout << endl;  
 DES(key, plaintext, mode);  
 BitToByte(plaintext, 64, out + i, 8);  
 // for(int j = 0; j < 8; ++j)std::cout << (int)out[8 \* i + j];std::cout << std::endl;  
 memcpy(IV, plaintext, 64);  
 }  
 }  
 else if(mode == DECRYPT){  
 for(int i = 0; i < len\_in; i += 8){  
 ByteToBit(in + i, ciphertext, 64, 8);  
 memcpy(plaintext, ciphertext, 64);  
 DES(key, plaintext, mode);  
 Func\_Xor(plaintext, IV, 64);  
 memcpy(IV, ciphertext, 64);  
 memcpy(out + (i << 6), plaintext, 64);  
 // for(int j = 0; j < 64; ++j)std::cout << plaintext[j];std::cout << std::endl;  
 BitToByte(plaintext, 64, out + i, 8);  
 }  
 }  
 return;  
}  
  
void CBC\_SetIV(const bool \*iv){  
 memcpy(IV, iv, 64);  
}

## main.cpp

测试接口文件，整个项目的入口：

#include"CBC.h"  
#include<bits/stdc++.h>  
using namespace std;  
  
  
void test(){  
 char key[9];  
 bool k[64];  
  
 char plaintext[1024];  
 char type;  
  
 char IV[9];  
 bool iv[64];  
  
 char clipertext[1024];  
 char clipertext\_hex[2048];  
 do{  
 cout << "Chose (E)NCRYPT, (D)ECRYPT or E(X)it: ";  
 cin >> type;  
 if(type == 'X')break;  
  
 cout << "Input DES's key(8 bytes): ";  
 cin >> key;  
 ByteToBit(key, k, 64, 8);  
 cout << "key: ";  
 for(int i = 0; i < 64; ++i)cout << k[i]; cout << endl;  
  
 cout << "Input CBC'IV(8 bytes): ";  
 cin >> IV;  
 ByteToBit(IV, iv, 64, 8);  
 CBC\_SetIV(iv);  
 cout << "IV: ";  
 for(int i = 0; i < 64; ++i)cout << iv[i];cout << endl;  
  
 if(type == 'E'){  
 cout << "Input Plaintext: ";  
 cin >> plaintext;  
 int c\_len = strlen(plaintext);  
 CBC(key, plaintext, c\_len, ENCRYPT, clipertext, &c\_len);  
 ByteToHex(clipertext, c\_len, clipertext\_hex);  
 cout << "clipertext\_hex: ";  
 c\_len <<= 1;  
 for(int i = 0; i < c\_len; ++i)cout << clipertext\_hex[i];cout << endl << endl;  
 }  
 else{  
 cout << "Input Clipertext(Hex): ";  
 cin >> clipertext\_hex;  
 int p\_len = strlen(clipertext\_hex);  
 HexToByte(clipertext\_hex, p\_len, clipertext);  
 p\_len >>= 1;  
 CBC(key, clipertext, p\_len, DECRYPT, plaintext, &p\_len);  
 cout << "plaintext: ";  
 for(int i = 0; i < p\_len; ++i)cout << plaintext[i];cout << endl << endl;  
 }  
 }while(true);  
}  
int main(){  
   
 // freopen("in.in", "r", stdin);  
 test();  
 // key: aaabbbcc  
 // iv: abcdefgh  
 // plain: abcdefghijklmnopqrstuvwxyz  
  
   
 return 0;  
}

# 运行结果

一个简单的运行示例如下：

PS G:\Backup\CollegeProjectBackup\ExperimentalReport\信息   
安全导论\CBC模式的DES加解密> g++ .\main.cpp .\DES.cpp .\CBC.cpp -o main.exe  
PS G:\Backup\CollegeProjectBackup\ExperimentalReport\信息   
安全导论\CBC模式的DES加解密> .\main.exe  
Chose (E)NCRYPT, (D)ECRYPT or E(X)it: E  
Input DES's key(8 bytes): aaabbbcc  
key: 0110000101100001011000010110001001100010011000100110001101100011  
Input CBC'IV(8 bytes): abcdefgh  
IV: 0110000101100010011000110110010001100101011001100110011101101000  
Input Plaintext: abcdefghijklmnopqrstuvwxyz  
clipertext\_hex: 676F1B793FB6C36508D4BC91DAF34B1AA6537535FC0F6D7D56D2EF263EE3C6BA  
  
Chose (E)NCRYPT, (D)ECRYPT or E(X)it: D  
Input DES's key(8 bytes): aaabbbcc  
key: 0110000101100001011000010110001001100010011000100110001101100011  
Input CBC'IV(8 bytes): abcdefgh  
IV: 0110000101100010011000110110010001100101011001100110011101101000  
Input Clipertext(Hex): 676F1B793FB6C36508D4BC91DAF34B1AA6537535FC0F6D7D56D2EF263EE3C6BA  
plaintext: abcdefghijklmnopqrstuvwxyz  
  
Chose (E)NCRYPT, (D)ECRYPT or E(X)it: X

可以看出当给定密钥、明文以及初始向量后，程序将运行CBC模式的DES加密代码，最后生成加密后的Hex编码的加密文本，当进行解密并输入何使的参数后，将完成解密功能。

# 参考内容：

* [关于AES加解密中CBC模式的IV初始化向量的安全性问题](https://www.jianshu.com/p/45848dd484a9)
* [Block cipher mode of operation](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation)
* [Data Encryption Standard](https://en.wikipedia.org/wiki/Data_Encryption_Standard)
* [DES supplementary material](https://en.wikipedia.org/wiki/DES_supplementary_material)
* [DES.cpp](https://github.com/yangxt225/DES/blob/master/des.cpp)

# 其他

[github](https://github.com/31415926535x/CollegeProjectBackup/tree/master/ExperimentalReport/信息安全导论/CBC模式的DES加解密) [blog](https://blog.csdn.net/pi31415926535x/article/details/106748634)