**《信息安全概论》实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | 谢双骏 | | **年级** | | 2019级 |
| **学号** | 20194196 | | **专业、班级** | | 计算机科学与技术01班 |
| **实验名称** | DES加解密算法的实现 | | | | |
| **实验时间** | 2022.10.29 | **实验地点** | | 线上 | |
| **实验成绩** |  | **实验性质** | | □验证性 □设计性 □综合性 | |
| **教师评价：**  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  评语：  评价教师签名（电子签名）： | | | | | |
| 1. **实验目的** 2. 掌握DES加密算法的加解密过程。 | | | | | |
| 1. **实验环境** 2. VC或TC编程环境；主机操作系统为Windows7或Windows XP； | | | | | |
| 1. **实验原理**   要深入理解对称加密算法，掌握DES加密过程。具备一定的C语言编程技能。  DES( Data DES( Data Encryption Standard)算法，于1977年得到美国政府的正式许可，是一种用56位密钥来加密64位数据的方法。DES算法以被应用于许多需要安全加密的场合。  **(一)、密钥生成**  **1、变换密钥**  取得64位的密钥，每个第8位作为奇偶校验位。  **2、变换密钥**。  (1)、舍弃64位密钥中的奇偶校验位，根据下表（PC-1）进行密钥变换得到56位的密钥，在变换中，奇偶校验位以被舍弃。  (2)、将变换后的密钥分为两个部分，开始的28位称为C[0]，最后的28位称为D[0]。  (3)生成16个子密钥，初始I=1。  (i)、同时将C[I]、D[I]左移1位或2位，根据I值决定左移的位数。  (ii)、将C[I]D[I]作为一个整体按下表（PC-2）变换，得到48位的K[I]  (iii)、从1-2-3-1处循环执行，直到K[16]被计算完成。  **(二)、处理64位的数据**  1、取得64位的数据，如果数据长度不足64位，应该将其扩展为64位（例如补零）  2、将64位数据按下表变换（IP）  3、将变换后的数据分为两部分，开始的32位称为L[0]，最后的32位称为R[0]。  4、用16个子密钥加密数据，初始I=1。  (1)、将32位的R[I-1]按下表（E）扩展为48位的E[I-1]  (2)、异或E[I-1]和K[I]，即E[I-1] XOR K[I]  (3)、将异或后的结果分为8个6位长的部分，第1位到第6位称为B[1]，第7位到第12位称为B[2]，依此类推，第43位到第48位称为B[8]。  (4)、按S表变换所有的B[J]，初始J=1。所有在S表的值都被当作4位长度处理。  (i)将B[J]的第1位和第6位组合为一个2位长度的变量M，M作为在S[J]中的行号。  (ii)、将B[J]的第2位到第5位组合，作为一个4位长度的变量N，N作为在S[J]中的列号。  (iii)、用S[J][M][N]来取代B[J]。  (iv)、从(i)处循环执行，直到B[8]被替代完成。  (5)、将B[1]到B[8]组合，按下表（P）变换，得到P。  (6)、异或P和L[I-1]结果放在R[I]，即R[I]=P XOR L[I-1]。  (7)、L[I]=R[I-1]  (8)、从2-4-1处开始循环执行，直到K[16]被变换完成。  5、、组合变换后的R[16]L[16]（注意：R作为开始的32位），按下表（IP-1）变换得到最后的结果。 | | | | | |
| 1. **实验步骤** 2. 编程：包含的功能函数有： 3. static void des(char Out[8], char In[8], const SUBKEY\_P pskey, bool Type);//标准DES 加/解密   *//des加/解密 void* des(*char* Out[8], *char* In[8], SUBKEY\_P subkeyP, *bool* Type) {  *static bool* M[64], tmp[32], \*Li = &M[0], \*Ri = &M[32];  byteToBit(M, In, 64);  replacement(M, M, ipTable, 64);  *if* (Type) {  *for* (*int* i = 0; i < 16; ++i) {  memcpy(tmp, Ri, 32);  fFunction(Ri, (\*subkeyP)[i]);  XOR(Ri, Li, 32);  memcpy(Li, tmp, 32);  }  } *else* {  *for* (*int* i = 15; i >= 0; --i) {  memcpy(tmp, Li, 32);  fFunction(Li, (\*subkeyP)[i]);  XOR(Li, Ri, 32);  memcpy(Ri, tmp, 32);  }  }  replacement(M, M, ipFTable, 64);  bitToByte(Out, M, 64); }   1. static void setKey(const char\* Key, int len);// 设置密钥   *// 设置密钥 void* setKey(*const char* \*Key, *int* len) {  memset(desKey, 0, 8);  memcpy(desKey, Key, len > 8 ? 8 : len);  setSubKey(&subKey, &desKey[0]); }   1. static void setSubKey(SUBKEY\_P pskey, const char Key[8]);// 设置子密钥   *// 设置子密钥 void* setSubKey(SUBKEY\_P subkeyP, *const char* Key[8]) {  *static bool* K[64], \*L = &K[0], \*R = &K[28];  byteToBit(K, Key, 64);  replacement(K, K, pc1, 56);  *for* (*int* i = 0; i < 16; ++i) {  moveLeft(L, 28, moveTimes[i]);  moveLeft(R, 28, moveTimes[i]);  replacement((\*subkeyP)[i], K, pc2, 48);  } }   1. static void fFunction(bool In[32], const bool Ki[48]);// f 函数完成扩展置换、S-盒代替和P 盒置换   *// f函数封装扩展置换/S盒代替/P盒置换 void* fFunction(*bool* In[32], *const bool* Ki[48]) {  *static bool* MR[48];  replacement(MR, In, eTable, 48);  XOR(MR, Ki, 48);  sCompress(In, MR);  replacement(In, In, pTable, 32); }   1. static void S\_Compress(bool Out[32], const bool In[48]);// S-盒代替函数   *//s盒变换 void* sCompress(*bool* Out[32], *const bool* In[48]) {  *for* (*char* i = 0, j, k; i < 8; ++i, In += 6, Out += 4) {  j = *char*((In[0] << 1) + In[5]);  k = *char*((In[1] << 3) + (In[2] << 2) + (In[3] << 1) + In[4]);  byteToBit(Out, &sBox[i][j][k], 4);  } }   1. static void replacement(bool \*Out, bool \*In, const char \*Table, int len);// 变换函数   *// 置换函数 void* replacement(*bool* \*Out, *const bool* \*In, *const int* \*Table, *int* len) {  *char* temp[256];  *for* (*int* i = 0; i < len; ++i)  temp[i] = *char*(In[Table[i] - 1]);  memcpy(Out, temp, len); }   1. static void XOR(bool \*InA, const bool \*InB, int len);//异或函数   *//异或函数 void* XOR(*bool* \*InA, *const bool* \*InB, *int* len) {  *for* (*int* i = 0; i < len; ++i)  InA[i] ^= InB[i]; }   1. static void moveLeft(bool \*In, int len, int loop);// 循环左移函数   *// 左移函数 void* moveLeft(*bool* \*In, *int* len, *int* loop) {  *bool* out[len];  *for* (*int* i = 0; i < len; ++i) {  out[i] = In[(loop + i) % len];  }  \*In = out[0]; }   1. static void byteToBit(bool \*Out, const char \*In, int bits);// 字节组转换成位组函数   *//字节到二进制 void* byteToBit(*bool* \*Out, *const char* \*In, *int* bits) {  *for* (*int* i = 0; i < bits; ++i) {  Out[i] = (In[i / 8] >> (i & 7)) & 1;  } }   1. static void bitToByt(char \*Out, const bool \*In, int bits);// 位组转换成字节组函数   *// 二进制到字节 void* bitToByte(*char* \*Out, *const bool* \*In, *int* bits) {  memset(Out, 0, bits >> 3);  *for* (*int* i = 0; i < bits; ++i)  Out[i >> 3] = *char*(Out[i >> 3] | In[i] << (i & 7)); }   1. 主函数   *int* main() {  *char* key[] = "kno3\_xsj", buf[1024];  *char* text[] = "I am Shuangjun Xie, my school number is 20194196. "  "This is the information security exp 2.";  memset(buf, 0, *sizeof*(buf));  strcpy(buf, text);  cout << "\nData:" << endl;  cout << buf << endl;  cout << "Length：" << *sizeof*(text) << endl;  cout << "\nKey:" << endl;  cout << key << endl;  cout << "Key length：" << *sizeof*(key) << endl;  callDes(buf, buf, *sizeof*(text), *sizeof*(key), key, *true*);  cout << "\nData after encode:" << endl;  puts(buf);  callDes(buf, buf, *sizeof*(text), *sizeof*(key), key, *false*);  cout << "\nData after decode:" << endl;  cout << buf << endl; }   1. 调试 2. 运行结果 3. 存盘 | | | | | |
| 1. **实验结果**   QQ截图20221030135608 | | | | | |
| 1. **实验思考题** 2. DES算法主要有哪几部分？可以画出流程   Des算法主要有两个部分，一个是计算16个子密钥，还有一个是根据子密钥加密解密，流程图如下：   1. 得到子密钥   4AAE1F13A844AAC627598DE35D6F64CB   1. 加密解密模块   4660BD0FB10B4B73CBC55412C1CDC576 | | | | | |