**《网络空间安全概论》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | |  | | **年级** | |  |
| **学号** | |  | | **专业、班级** | |  |
| **实验名称** | **实验一 加密解密算法** | | | | | |
| **实验时间** |  | | **实验地点** | |  | |
| **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）： | | | | | | |
| 一、实验目的   * 掌握频度分析法原理和Feistel加解密原理。 | | | | | | |
| 二、实验项目内容   1. 使用频度分析法解密以下文本，并给出替换表：   UZ QSO VUOHXMOPV GPOZPEVSG ZWSZ OPFPESX UDBMETSX AIZ VUEPHZ HMDZSHZO WSFP APPD TSVP QUZW YMXUZUHSX EPYEPOPDZSZUFPO MB ZWP FUPZ HMDJ UD TMOHMQ   1. 编程实现Feistel加密解密以下文本：   CQUINFORMATIONSECURITYEXP | | | | | | |
| 三、实验设计  **1.频度分析法：**  对于任何一种特定的书面语言，不同的字母或字母组合出现的频率都各不相同。因此，只要使用这种语言书写足够长的文本，那么将呈现出大致相同的特征字母分布规律。对于英语，其特征字母分布规律如下表所示：    可以看出，不少字母出现的频率近乎相等，但也有极少数字母出现的频率有较大差异。因此，为了分析方便密文信息，常将英文字母表按字母出现的频率大小分类如下：  --------------------------  极高频 E  次高频 T A O I N S H R  中等频 D L  低频 C U M W F G Y P B  甚低频 V K J X Q Z  --------------------------  上述单字母频率统计并没有反映出英文双字母和多字母的特征。在双字母中统计出频率最大的30对字母按频率大小排列依次为：  th he in er an re ed on es st en at to nt ha nd ou ea ng as or ti is et it ar te se hi of  类似的，可以统计得到三字母的出现频率。根据Beker在1982年统计的结果(样本总数100 360)得到频率最大的20组三字母按频率大小排列为：  the ing and her ere ent tha nth was eth for dth hat she ionhis sth ers ver  特别地，the出现的频率几乎为ing的3倍。  因此，对于使用替换密码加密的英语文本，破译者可以使用程序统计密文中各种字符（串）的出现频率，并结合上述特征字母的分布规律进行猜测，尝试将特定的密文字符替换为猜测的明文字符并检查替换后的文本是否表达出一定语义。重复上述过程，直至破译密文，即得到所有密文字符的替换表。  **2.Feistel网络：**  在密码学中，费斯妥密码（Feistel cipher）是用于构造分组密码的对称结构，通常也称为费斯妥网络（Feistel network）。大部分分组密码都使用了该加密方案。  Feistel网络将首先每个明文分组拆分为L0和R0，然后进行n次循环处理，最后将加密后的Ln和Rn重新组合起来生成密文分组。n次循环的结构完全相同，并且每次循环都使用上一循环产生的Li-1和Ri-1和密钥K产生的子密钥Ki作为本轮迭代的输入。一般而言，根据密钥K生成的子密钥Ki与K并不相同，且每次循环所使用的子密钥Ki间也互不相同。循环过程中所使用的F函数一般为非线性映射函数。同时，由于异或操作的特殊性，F函数并不要求是可逆的，因为无论是加密过程还是解密过程，F操作都是正操作，均未用到F操作的逆操作。Feistel网络的第i轮循环如下图所示：    一个典型的分组长度为64，循环次数为16的Feistel网络结构如下图所示：    **3.DES数据加密标准：**  DES数据加密标准是美国于1977年在全球范围内公开的第一个 现代单钥加密算法。DES是一种分组密码，在设计上，DES采用了Feistel结构。其输入的明文长度为64bit，密钥长度为 56bit，输出的密文长度为64bit。其算法结构如下图所示：    其中，初始置换IP和初始置换逆变换IP^(-1)分别体现了分组密码“扩散”和“混淆”的设计思想，主要目的是打乱明文中的各位。    16轮迭代变换的算法结构如下图所示：    其中，压缩运算S是算法的唯一非线性部件,因此它的密码强度决定了整个算法的安全强度，提供了密码算法所必须的混乱作用。P置换的目的则是提供雪崩效应，使明文或密钥的微小变动可以引起密文的大幅度变化。  子密钥Ki的生成算法结构如下图所示： | | | | | | |
| 四、实验过程或算法  **1.频度分析法：**  void solve()  {  int n=3;  string s=" UZ QSO VUOHXMOPV GPOZPEVSG ZWSZ OPFPESX \  UDBMETSX AIZ VUEPHZ HMDZSHZO WSFP APPD TSVP QUZW YMXUZUHSX \  EPYEPOPDZSZUFPO MB ZWP FUPZ HMDJ UD TMOHMQ";  vector<map<string,int>>mp(n+1);  vi cnt(n+1);  int m=s.size();  for(int i=1;i<=n;i++)  for(int l=1,r=i;r<m;l++,r++)  {  auto x=s.substr(l,i);  if(x.find(' ')==x.npos)  {  mp[i][x]++;**//mp[i][x]统计的是长度为i的只含可打印字符的子段x在密文中出现的次数**  cnt[i]++;**//cnt[i]统计的是长度为i的子段数**  }  }  vector<vector<pair<string,db>>>fq(n+1);  for(int i=1;i<=n;i++)  {  for(auto [ch,x]:mp[i])  fq[i].pb({ch,db(x)\*100/cnt[i]});**//计算各种子段的出现频率**  sort(fq[i].begin(),fq[i].end(),[&](auto&x,auto&y){  return x.sec>y.sec;**//根据出现频率从大到小排序**  });  cout<<"|S| = "<<i<<":\n";  for(auto [ch,x]:fq[i])  cout<<ch<<' '<<x<<"%\n";  cout<<'\n';  }    map<char,char>rp={**//替换密码表，通过不断调整该表并观察下方打印程序的输出结果来判断替换规则的正确性**  {'Z','t'},{'W','h'},{'P','e'},{'S','a'},  {'F','v'},{'U','i'},{'Q','w'},{'O','s'},  {'D','n'},{'A','b'},{'I','u'},{'E','r'},  {'X','l'},{'Y','p'},{'M','o'},{'H','c'},  {'V','d'},{'T','m'},{'G','y'},{'B','f'},  {'J','g'},  };  for(int i=1;i<m;i++)  if(rp.count(s[i]))  cout<<rp[s[i]];**//输出文本,对于替换密码表中出现的替换输出明文文本,否则输出密文文本**  else cout<<s[i];  }  **2.Feistel网络的Des实现：**  **//IP置换**  const vi 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  };  **//IP逆置换**  const vi inv={  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  };  **//扩展变换E**  const vi e={  32, 5,  4, 9,  8, 13,  12, 17,  16, 21,  20, 25,  24, 29,  28, 1  };  **//子密钥PC-1置换**  const vi pc1={  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  };  **//子密钥PC-2置换**  const vi pc2={  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  };  **//压缩置换S**  const vvi sbox={  {  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, 5, 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, 4, 10, 15, 6, 8, 0, 5, 9, 12,  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,  1, 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**  const vi 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  };  typedef bitset<32> b32;**//位图,支持位运算的01串,可以和01字符串简单的相互转换**  typedef bitset<48> b48;  typedef bitset<64> b64;  typedef vector<b64> vb64;  typedef vector<string> vstr;  typedef pair<int,string> pis;  int toInt(string s)**//将01字符串转换为整数**  {  int m=s.size(),res=0;  for(int i=0;i<m;i++)  res=(res<<1)+s[i]-'0';  return res;  }  string to01String(char x)**//将英文字符转换为8位01字符串**  {  string res;  for (int i=7;i>=0;i--)  {  int p=(x>>i)&1;  res.pb(p+'0');  }  return res;  }  string to01String(int x)**//将整数变换为4位01字符串**  {  string res;  for(int i=0;i<4;i++)  {  res+=(x&1)+'0';  x>>=1;  }  reverse(res.begin(),res.end());  return res;  }  pis to01String(string s)**//将明文转换为64位01字符串分组**  {  string res;  for(auto ch:s)  res+=to01String(ch);  int n=res.size(),m=(n+63)/64,k=m\*64-n;//分组数m向上取整  if(k)  res=string(k,'0')+res;//不足64位的高位填充0  return {m,res};  }  char toChar(string s)**//将8位01字符串转换为英文字符**  {  char res=0;  for(int i=0;i<8;i++)  res=(res<<1)+s[i]-'0';  return res;  }  string toString(string s)**//将64位01字符串转换问明文段**  {  string res;  for(int i=0;i<64;i+=8)  res+=toChar(s.substr(i,8));  return res;  }  vstr grouping(string s)**//明文分组**  {  auto [m,p]=to01String(s);**//将明文转换为若干组64位01字符串,每8位代表一个英文字符**  vstr res;  for(int i=1,j=0;i<=m;i++,j+=64)  res.pb(p.substr(j,64));  return res;  }  const string key=to01String("Arknight").sec;**//64位密钥**  string PC1(string s)**//PC-1子密钥生成置换**  {  string res;  for(int i=0;i<56;i++)  res+=s[pc1[i]-1];  return res;  }  string PC2(string s)**//PC-2子密钥生成置换**  {  string res;  for(int i=0;i<48;i++)  res+=s[pc2[i]-1];  return res;  }  string K(string c,string d,int r)**//子密钥生成算法**  {  string ci=c.substr(r)+c.substr(0,r);**//第r轮循环左移r位**  string di=d.substr(r)+d.substr(0,r);  return PC2(ci+di);  }  string IP(string s)**//IP置换**  {  string res;  res.resize(64);  for(int i=0;i<64;i++)  res[i]=s[ip[i]-1];  return res;  }  string invIP(string s)**//IP逆置换**  {  string res;  res.resize(64);  for(int i=0;i<64;i++)  res[i]=s[inv[i]-1];  return res;  }  string E(string s)**//扩展变换E**  {  **//与密钥无关的纯位移置换**  **//32位分为8组: 每组4位,经E置换后,变成每组6位**  string res;  for(int i=0,j=0;i<32;i+=4,j+=2)  res+=s[e[j]-1]+s.substr(i,4)+s[e[j+1]-1];  return res;  }  string S(string s)**//压缩变换S**  {  **//共8个S盒,每盒输入6位,输出4位**  string res;  for(int i=0,j=0;j<8;i+=6,j++)  {  string x=s.substr(i,6);**//6位输入B1B2B3B4B5B6**  int nx=toInt(x.substr(0,1)+x.substr(5,1));**//B1B6为S盒行坐标**  int ny=toInt(x.substr(1,4));**//B2B3B4B5为S盒列坐标**  res+=to01String(sbox[j][nx\*16+ny]);  }  return res;  }  string P(string s)**//P置换**  {  string res;  for(int i=0;i<32;i++)  res+=s[p[i]-1];  return res;  }  void F(string&l,string&r,string&c,string&d,int i)**//F函数**  {  string ri=(b48(E(r))^b48(K(c,d,i))).to\_string();**//将扩展运算E后的48位Ri与48位子密钥Ki按位异或**  ri=P(S(ri));**//将48位Ri压缩变换S为32位,并进行P置换**  string p=l;  l=r;//Li = Ri-1  r=(b32(p)^b32(ri)).to\_string();**//将Ri与Li-1按位异或**  }  string encode(string s)**//分组加密**  {  string lr=IP(s);**//IP置换**  string l=lr.substr(0,32),r=lr.substr(32,32);  string cd=PC1(key);  string c=cd.substr(0,28),d=cd.substr(28,28);  for(int i=1;i<=16;i++)  F(l,r,c,d,i);**//16轮变换**  return r+l;  }  string decode(string s)**//分组解密**  {  string l=s.substr(0,32),r=s.substr(32,32);  string cd=PC1(key);  string c=cd.substr(0,28),d=cd.substr(28,28);  for(int i=16;i;i--)  F(l,r,c,d,i);**//16轮变换**  return invIP(r+l);**//IP逆置换**  }  void solve()  {  string plainText="CQUINFORMATIONSECURITYEXP";  cout<<"PlainText: "+plainText+"\n\nEncoding...\n\n";    vstr plainGroups=grouping(plainText);**//对明文进行分组**  int n=plainGroups.size();  vstr cipherGroups;  for(int i=0;i<n;i++)  {  string s=plainGroups[i],es=encode(s);**//对明文分组进行加密**  cout<<"Group "<<i<<":\nPlainText: "+s+"\nCipherText: "+es<<'\n';  cipherGroups.pb(es);  }  cout<<"\nDecoding...\n\n";  string ans;  for(int i=0;i<n;i++)  {  string s=cipherGroups[i],ds=decode(s);**//对密文分组进行解密**  cout<<"Group "<<i<<":\nCipherText: "+s+"\nPlainText: "+ds<<'\n';  ans+=toString(ds);**//破译后的明文**  }  int m=ans.size();  for(int i=0;i<=m;i++)  if(isprint(ans[i]))**//去除分组过程中填充的不可打印字符**  {  cout<<"\nPlainText: "+ans.substr(i)<<'\n';  break;  }  } | | | | | | |
| 五、实验过程中遇到的问题及解决情况  **问题1：**  密文字符的频率分析结果与英文字母的特征分布没有出现明显匹配。  **原因和解决办法：**  密文段文本量太小，不具有显著的统计学意义。只能通过试错法不断尝试。  **问题2：**  密文文本的产生源未知，无法结合其产生背景，如历史环境等综合判断。  **原因和解决办法：**  暴力检索可能的替换策略并反推原文背景。例如，密文段FUPZ HMDJ的替换结果为viet cong，不属于常用的英文单词。但经查验可以解释为“越共，越南南方民族解放阵线”，则可以尝试采纳该替换并反推原文背景与越南战争相关。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  **1.频度分析法：**  频度分析法的结果（部分）如下图所示，其中三字母的频度分析表基本不含有有效信息：    据此可以猜测ZW代表th，结合文本中的ZWP可知P为e。重复上述过程可得替换密码如下图所示：    破译得到的明文为：  It was disclosed yesterday that several informal but direct contacts have been made with political representatives of the viet cong in moscow.（据昨天披露，已与莫斯科的越南共产党政治代表进行了几次非正式但直接的接触。）  **2.Feistel网络：**  使用本次实验实现的Feistel网络对密文加密的过程如下图所示。明文一共被分为4组，每组64位。可以看到，分组解密过程和分组加密过程是一一对应的，最终的解密结果也与明文保持一致。（密钥为“Arknight”的64位01串表示，各置换矩阵采用的是课程中所教授的经典设计，不再赘述） | | | | | | |