**《信息安全技术实验》**

**实验一 Wireshark网络数据包捕获及协议分析实验**

**一、实验目的**

1、掌握Wireshark软件的基本使用方法

2、学会网络数据包的捕获技巧

3、初步掌握针对常用网络协议进行网络分析的方法

**二、实验内容**

1、安装Wireshark软件

请在QQ群或电脑登录学习通网站进入课程资料栏目，下载最新版的Wireshark软件安装包，在自己的电脑安装Wireshark软件。在安装的过程中会提示是否安装Npcap或Winpcap和USBPcap（前者用于捕获有线和无线网卡，后者用于捕获USB网卡数据包），请选择同意安装即可。

2、运行Wireshark软件，并抓取网络数据包

打开Wireshark，在主界面选择需要抓包的接口（Interface），如果电脑装有多块网卡，则注意选择需要抓包的对应网卡。新版的Wireshark可以支持多种不同的网卡类型。

选择网卡之后即可开始捕获数据包，请点击左上角的“停止”和“开始”按钮进行控制。

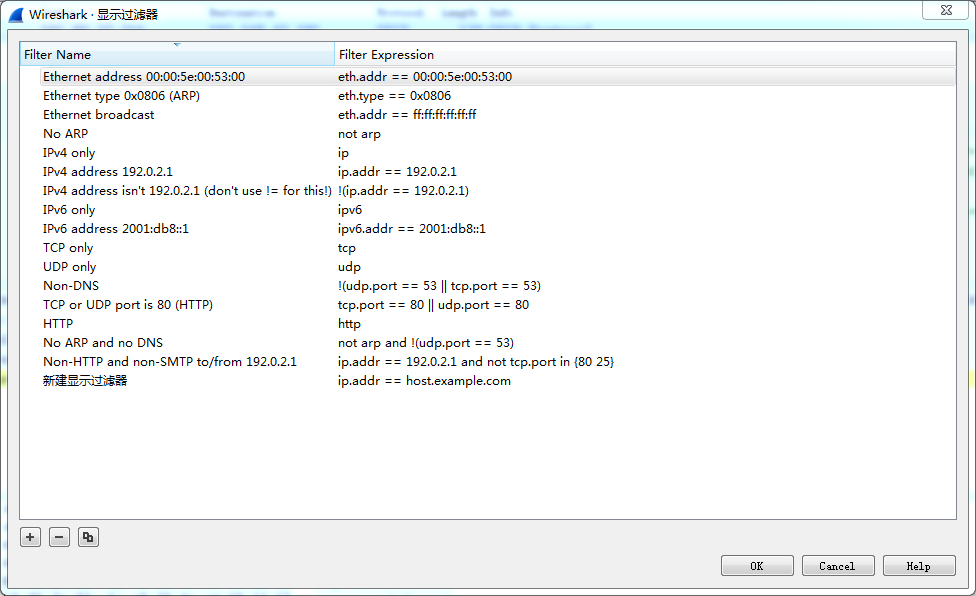
3、保存和打开捕获文件

通过菜单或者工具栏进行捕获文件的保存和打开。

4、学习使用显示过滤器

一般来说，捕获的数据包的数量会十分庞大，在一秒钟内可能捕获数十个甚至数千个数据包，如果不加分别地同时显示，则根本看不过来，也毫无用处。

我们首先学会使用显示过滤器。



Wireshark已经帮我们预设了一些常用的显示过滤器。请按照以下要求逐个完成并截图保存，截图放置在实验报告Word文档内。

（1）仅显示HTTP协议数据包；

（2）仅显示TCP协议数据包；

（3）仅显示ICMP协议数据包；

（4）仅显示DNS协议数据包；

（5）仅显示ARP协议数据包；

（6）仅显示指定IP地址发送和接收的数据包，其中IP地址为本机IP地址（使用IPv4）；

（7）仅显示指定MAC地址发送和接收的数据包，其中MAC地址为本网卡MAC地址。

5、如何减少干扰数据包

很多时候，捕获数据包经过初步过滤后仍然非常杂乱。初学者根本分不清这一大堆数据包是用来干什么的？那么，如何才能减少干扰数据包呢？

首先，请将本机后台运行的程序尽量关闭。包括所有的防火墙、杀毒软件、监控软件，还有所有的社交软件，以及所有的服务器程序（如数据库服务器）。

关闭后台程序可以使用Ctrl+Alt+Del进入系统任务管理器，然后进入“进程”页面，关闭非必要的进程。哪些是非必要的进程？大家可以百度一下。如果意外关闭了重要系统进程而导致系统故障，请重新启动系统即可。这个过程也可以让大家有机会了解一下自己电脑后台运行了什么程序。对系统安全很有帮助。（本课程暂不解释各个后台进程的作用，有兴趣的同学请自行了解）

关闭了部分进程之后，可以尝试启动Wireshark进行数据包捕获。如果在没有任何人为操作的情况下，捕获到的数据包几乎为空，则说明基本达到目标。

6、ARP协议分析

这个时候，我们眼前的Wireshark捕获数据包应该非常简明，有利于我们进一步实验。为了练习网络协议分析，我们应该主动地创造所需的数据包。

请完成以下的要求：

（1）了解ARP协议，在实验报告中回答ARP协议的作用，ARP协议的数据包格式。

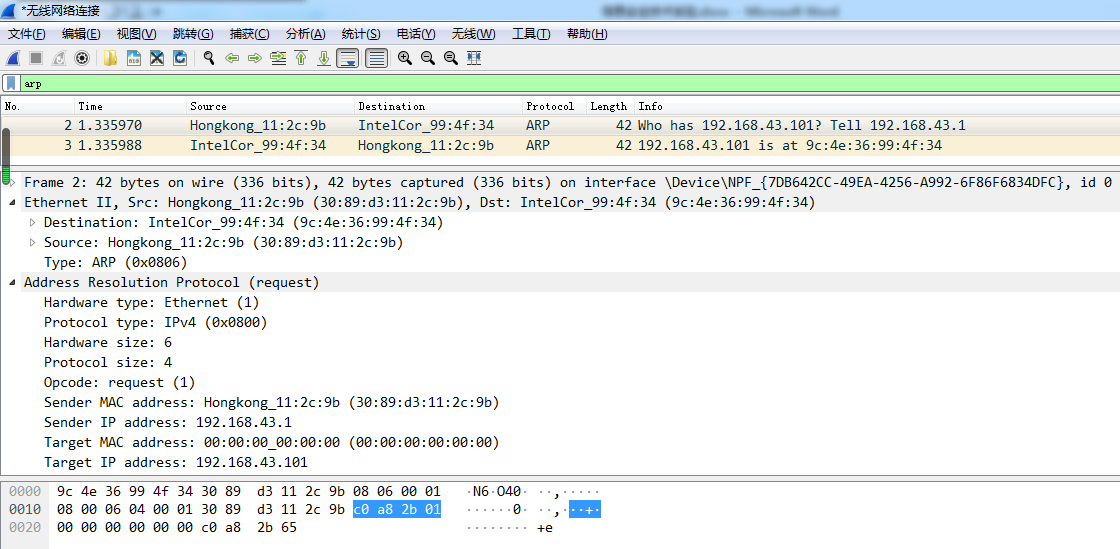
（2）在Windows命令行状态（开始—运行—cmd命令）使用命令“arp -a”，查看当前主机的ARP缓存。

（3）使用命令“arp -d”实现对ARP缓存信息的删除，然后再用“arp -a”命令查看一下，就会发现主机内的ARP缓存信息已经清空了。如果使用清空ARP缓存命令之后发现ARP缓存仍然存在，则说明主机的后台仍然有进程在发送数据包，清空ARP缓存之后再次填入ARP缓存，请回到第5步骤继续关闭系统进程。

（4）启动Wireshark数据包捕获，并且设置显示过滤器为“仅显示ARP协议数据包”。

（5）任意使用网络的操作（如使用ping命令、打开浏览器网页等等）。

（6）停止Wireshark数据包捕获，这时应该可以看见若干个ARP数据包。在实验报告提交截图，示例如下：



（7）在实验报告回答这个数据包的内容分析。包括Ethernet II区域和Address Resolution Protocol区域内的每一个字段的含义是什么。

每位同学的实验报告必须先回答各自的主机IP地址和MAC地址是什么（可通过ipconfig /all命令查看），之后所有的实验截图显示的本机IP地址或MAC地址均必须相同，不得抄袭别人的报告。

（8）你刚才分析的ARP数据包的作用是什么？询问的是那台主机的IP地址？请你检查一下自己的网络配置，看看被查询的这台主机位于哪里？

6、ICMP协议和IP协议分析

在分析ARP协议之后，我们再分析一个比较简单的网络协议ICMP协议，顺道还可以分析一下IP协议。

请完成以下的要求：

（1）了解ICMP协议，在实验报告中回答ICMP协议的作用，ICMP协议的数据包格式。

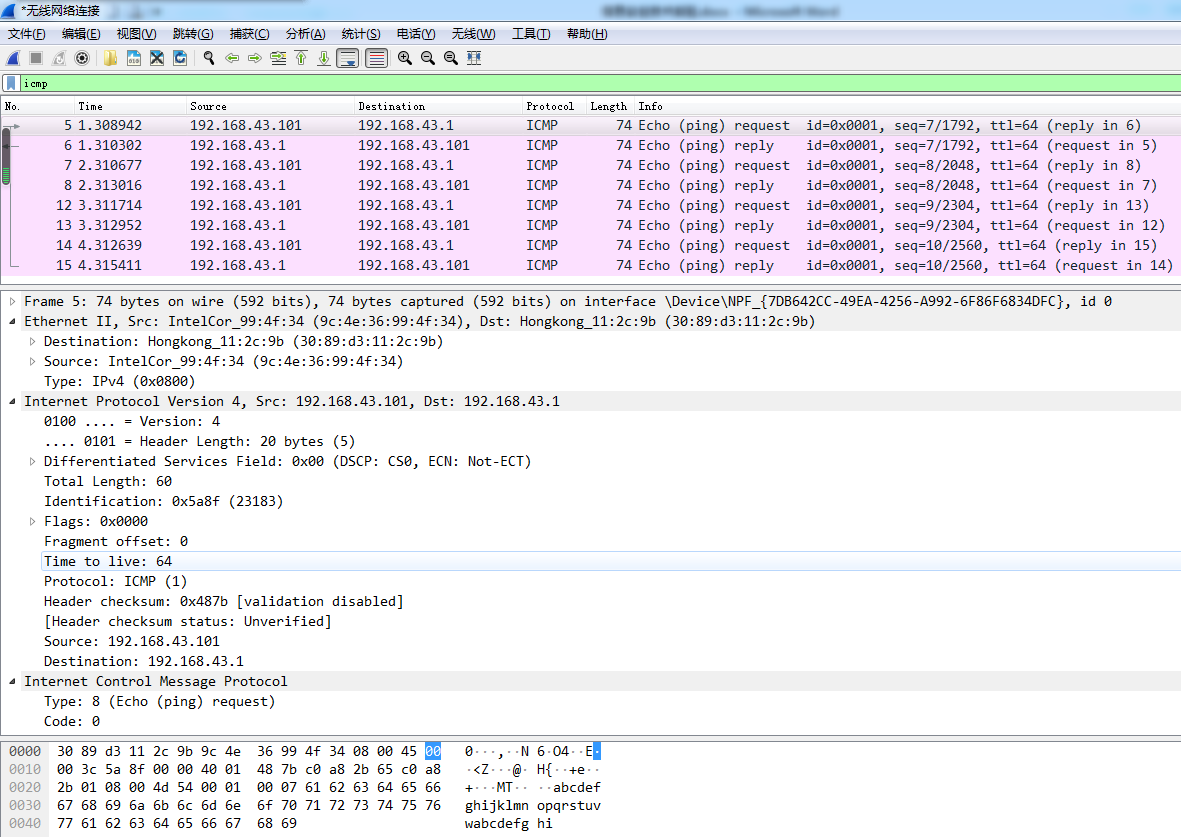
（2）了解IP协议，在实验报告中回答IP协议的作用，IP协议的数据包格式。

（3）启动Wireshark数据包捕获，并且设置显示过滤器为“仅显示ICMP协议数据包”。

（4）使用ping命令，对象为你所在网络的网关IP。

（5）停止Wireshark数据包捕获，这时应该可以看见若干个ICMP数据包。在实验报告

提交截图，示例如下：



（6）在实验报告回答这个数据包的内容分析。包括Ethernet II区域、Internet Protocol区域和Internet Control Message Protocol区域内的每一个字段的含义是什么。

注意：显示的本机IP地址或MAC地址均必须与之前的实验相同，不得抄袭别人的报告。

（7）再做一个实验。再次清空本机ARP缓存，然后开启Wireshark捕获，设置过滤器为“显示ARP协议和ICMP协议数据包”。接下来在命令行输入ping命令对象为本网的网关IP。观察捕获到的数据一共有多少个？他们之间有什么关系？注意时间前后次序。

再次在命令行输入ping命令对象为本网的网关IP。观察接下来捕获到的数据包又有多少个？两次ping命令，有没有发现少了一些数据包？请说明。

（8）请回答，为什么捕获ICMP数据包会出现IP数据包？ICMP协议和IP协议是什么关系。

7、TCP协议和HTTP协议分析

在分析ICMP和IP协议之后，我们接下来分析比较复杂的网络协议TCP议，顺道看看HTTP协议。

请完成以下的要求：

（1）了解TCP协议，在实验报告中回答TCP协议的作用，TCP协议的数据包格式，TCP协议的三次握手过程和四次断开过程。

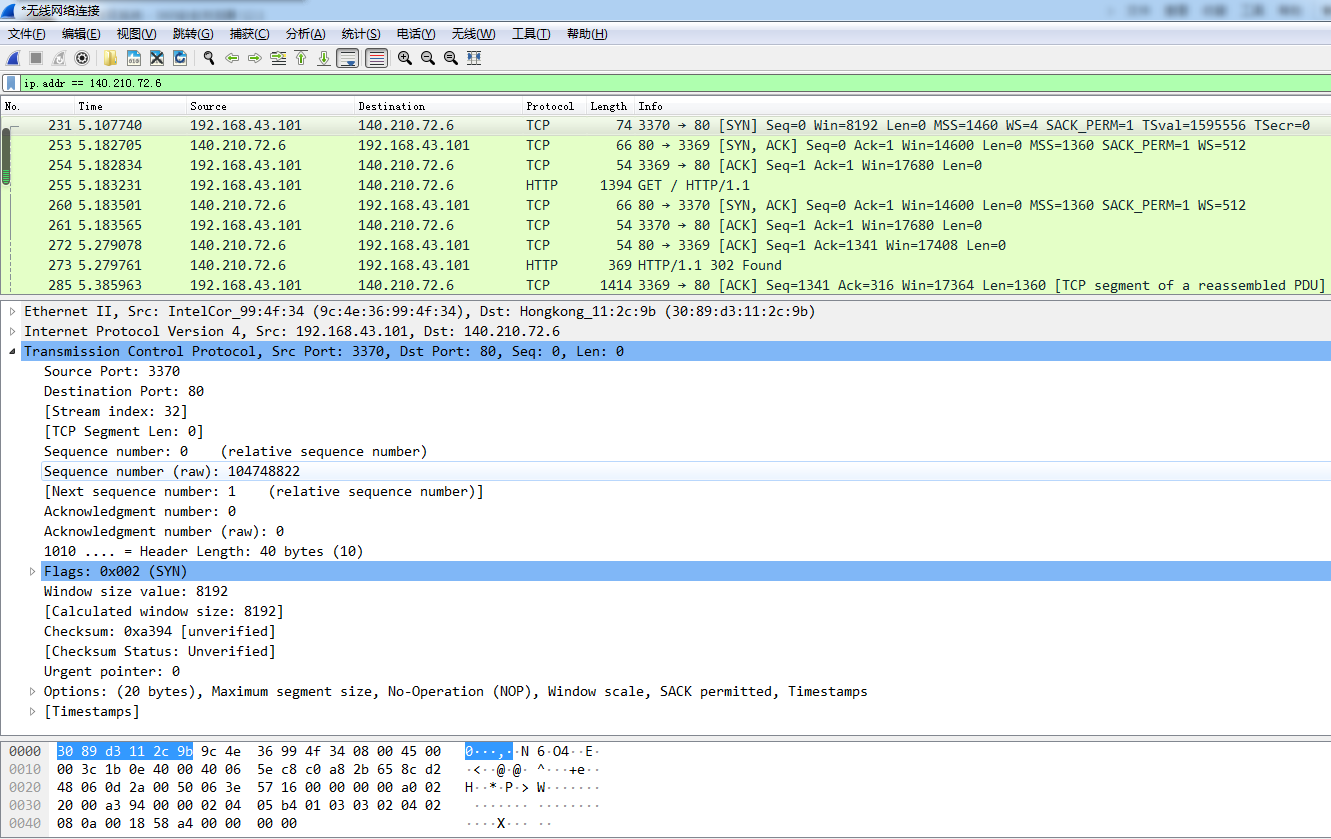
（2）了解HTTTP协议，在实验报告中回答HTTP协议的作用，HTTP的头部格式，HTTP的请求代码有哪些？HTTP的应答代码有哪些？

（3）打开浏览器

（4）启动Wireshark数据包捕获，并且设置显示过滤器为“仅显示TCP协议数据包”，或者按照IP地址过滤，过滤IP为140.210.72.6。

（5）在浏览器地址栏中输入网址访问http://www.chaoxing.com

（6）停止Wireshark数据包捕获，这时应该可以看见若干个TCP和HTTP数据包。在实验报告提交截图，示例如下：

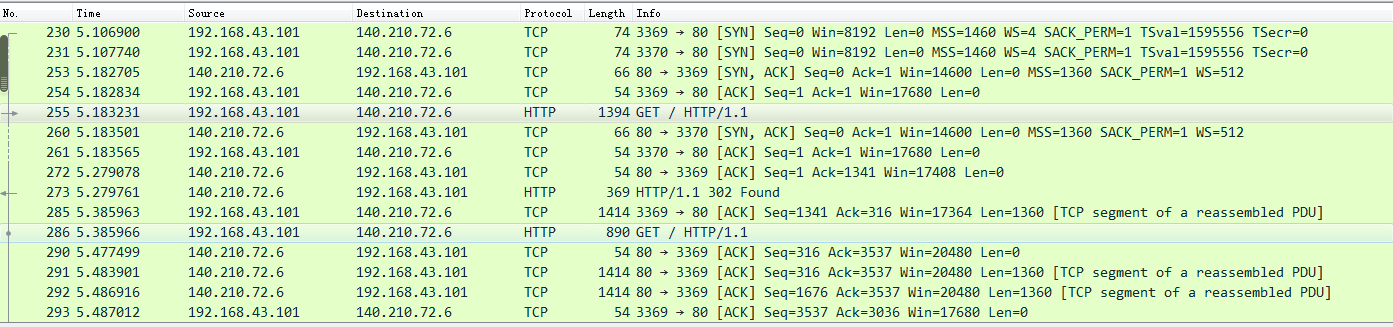


（7）请找出TCP的三次握手数据包。提示：通常是连续或相隔不远的三个TCP数据包，并且在Info信息里面可以看到分别有[SYN]、[SYN,ACK]和[ACK]字样。用红色标记圈出TCP的三次握手。

（8）请回答这三个TCP数据包的Transmission Control Protocol区域内每个字段的意义是什么？对比这三个包有哪几个字段是不同的？他们分别代表了什么？特别提示：Sequence number和Acknowledgment number这两个字段值是相对值，而Sequence number (raw)和Acknowledgment number (raw)这两个字段值是真实值。他们有什么关系？

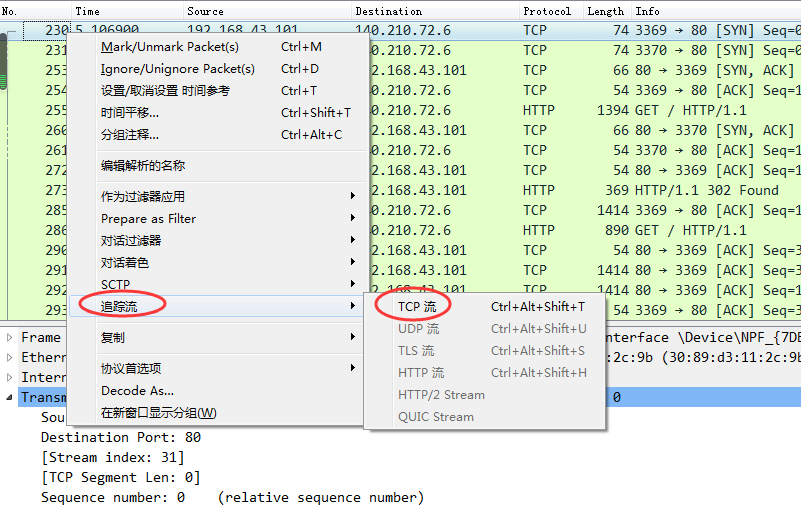
（9）紧接着TCP三次握手建立TCP连接之后的HTTP数据包通常就是我们的网页请求。请回答这个数据包中Hypertext Transfer Protocol区域每一个字段的含义。

（10）能力好的同学可以进一步分析HTTP协议的过程。示例：

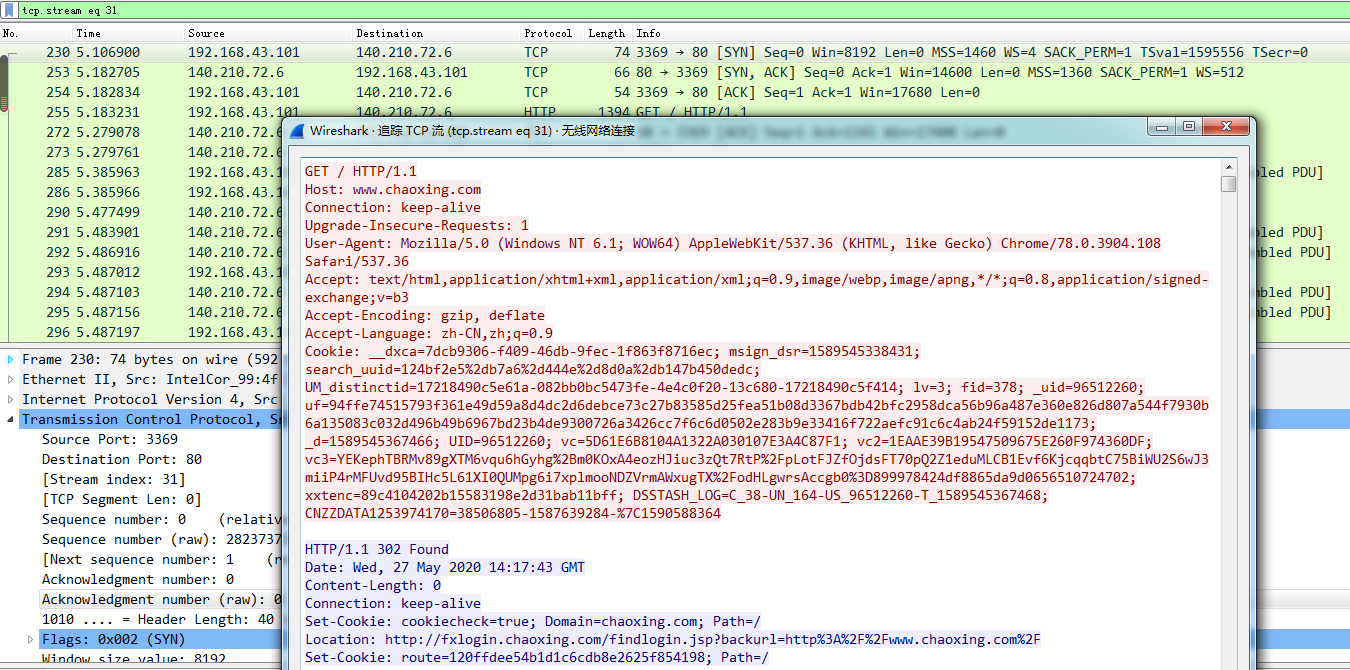


以上的例子中，客户端（浏览器）使用3369端口与Web服务器的80端口建立了一个TCP连接，我们称之为Socket A。之后，客户端又用3370端口与Web服务器的80端口建立了另一个TCP连接，我们称之为Socket B。这两个Socket之间并没有关联，是各自独立的。第230、253、254号数据包是Socket A的三次握手；第231、260、261号数据包是Socket B的三次握手。（**客户端的端口号是随机的，但服务器的端口号80是固定的**）

为了看清楚各个Socket干了什么事情，不至于搞混，可以使用“TCP流跟踪”功能。



例如，右键点击第一条TCP数据包，跟踪这个TCP流，可得：



这时候，可以很清晰地观看这个TCP连接建立之后发生的所有数据传送内容。红色代表发送的数据，蓝色代表接收的数据。

到这里，相信同学们已经能够比较熟练地使用WIreshark进行数据包捕获和网络协议分析了。请有兴趣的同学继续分析刚才访问[www.chaoxing.com](http://www.chaoxing.com)这个网页所捕获的所有数据包。

此外，目前很多网站采用了HTTPS协议取代HTTP协议，HTTPS协议是使用了加密和认证功能的，是一种比HTTP协议安全性高很多的网络协议。由于目前我们的课程暂未讲到SSL、TLS和HTTPS协议，大家可以先自行了解。

**实验二 对称加密算法基础实验**

**一、实验目的**

掌握对称加密算法中替代操作、置换操作的原理。

**二、实验平台**

本次实验建议使用Visual C++ 6.0作为实验编程平台，如果个人认为其他编程语言平台更熟悉的话，可以在征得老师同意之后使用。

**三、实验内容**

1、替代操作（S盒子）

我们在课堂中学习的替代加密操作明文为26个英文字母。实际上，加密的明文应为任意二进制数据。通常，替代操作可以使用如下的替换表（也称为替代密码表）：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 后4bit  前4bit | 0 | 1 | 2 | 3 | … | 14 | 15 |
| 0 | **91** | **190** | **36** | **211** |  | **81** | **55** |
| 1 | **126** | **0** | **45** | **34** |  | **6** | **72** |
| 2 | **89** | **65** | **39** | **18** |  | **20** | **132** |
| 3 | **27** | **40** | **90** | **8** |  | **233** | **62** |
| … |  |  |  |  |  |  |  |
| 14 | **176** | **60** | **83** | **48** |  | **63** | **36** |
| 15 | **49** | **155** | **29** | **80** |  | **92** | **209** |

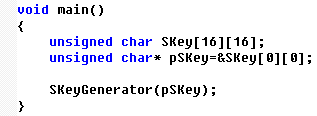
图2-1 替代操作加密密码表

假设我们的明文和密文都是一个字节（长度为8bit）。在密码表中用0~255表示一个8bit的无符号数。密码表中的列代表明文的前4bit，行代表明文的后4bit，而表格的内容则为密文（8bit）。加密的时候，根据明文的内容，就可以查表得到密文。那么，我们如何生成一张密码表呢？

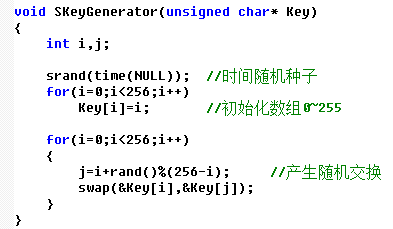
（1）使用C++语言编写替代密码表生成函数。其函数原型为：

void SKeyGenerator(unsigned char\* Key);

为了使用和观看方便，可以定义密码表为一个16×16的二维数组SKey[16][16]。但在实际计算的时候，可将这个二维数组看作一维数组处理。



下面给出函数的参考代码：



关于随机数的产生及用法，各位同学可以自行查阅相关资料。

（2）使用C++语言编写替代加密函数。其函数原型为：

void SEncrytion(unsigned char\* PlainText, unsigned char\* CipherText, unsigned char\* Key);

具体的代码请同学们自行完成，下同。

（3）使用C++语言编写替代解密函数。其函数原型为：

void SDecrytion(unsigned char\* PlainText, unsigned char\* CipherText, unsigned char\* Key);

解密的时候，解密密码表是否与加密密码表相同？如果不同的话，如何从加密密码表生成解密密码表？

具体的代码请同学们自行完成。

2、置换操作（P盒子）

置换操作也称为移位操作，可以使用如下的移位表（也称为移位密码表）：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0~7 | 0 | **5** | **19** | **29** | **2** | **12** | **24** | **11** | **18** |
| 8~15 | 1 | **8** | **10** | **14** | **16** | **31** | **0** | **30** | **27** |
| 16~23 | 2 | **21** | **23** | **4** | **9** | **22** | **29** | **25** | **7** |
| 24~31 | 3 | **15** | **1** | **20** | **26** | **6** | **17** | **3** | **13** |

图2-2 移位操作加密密码表

假设明文和密文为4字节的二进制数据（32bit），从最低位（第0bit）开始，查表可得密文第0bit的内容为明文第5bit的内容；密文第1bit的内容为明文第19bit的内容；以此类推。即可得到密文数据。

（1）使用C++语言编写移位密码表生成函数。其函数原型为：

void PKeyGenerator(unsigned char\* Key);

生成这张表的方法与替代操作类似。先初始化一个数组PKey[32]，值为0~31。然后采用随机数函数rand()，不断对这个数组的任意两个元素交换。具体的代码请同学们自行完成。

（2）使用C++语言编写移位加密函数。其函数原型为：

void PEncrytion(unsigned int\* PlainText, unsigned int\* CipherText, unsigned char\* Key);

具体的代码请同学们自行完成。提示，这里需要用到位操作，提取出指定位置的bit。请同学们留意。

（3）使用C++语言编写替代解密函数。其函数原型为：

void SDecrytion(unsigned int\* PlainText, unsigned int\* CipherText, unsigned char\* Key);

解密的时候，解密密码表是否与加密密码表相同？如果不同的话，如何从加密密码表生成解密密码表？具体的代码请同学们自行完成。

**三、实验报告要求**

1、在报告中必须给出编程的核心代码，并对代码进行中文注释

2、运行程序，截图给出你生成的密码表、加密的执行例子（输入明文=？，得到密文=？）、解密的执行例子（输入密文=？，得到明文=？）。没有硬性规定必须有界面，但推荐同学们设计一个简单的输入/输出界面。也可以单纯使用DOS界面，但必须用printf语句调整格式，要求基本清晰美观。

**实验三 DES加密算法实验**

**一、实验目的**

掌握DES对称加密算法中密钥生成、加密和解密操作。

**二、实验平台**

本次实验建议使用Visual C++ 6.0作为实验编程平台，如果个人认为其他编程语言平台更熟悉的话，可以在征得老师同意之后使用。

**三、实验内容**

在实验二对称加密算法的基础上，本次实验将实现一个完整的DES算法。

1、算法概况

* DES算法的明文和密文都是8字节（64bit）的二进制数据。
* 主密钥长度为7字节（56bit）。由于DES算法在1977年提出，当时主要通信模式为异步串行通信，每7bit数据要加上1bit奇偶校验位。因此，56bit的主密钥在传送的时候实际传送64bit，其中第8、16、24、32、40、48、56和64bit并不包含密钥信息。（具体请详见子密钥的生成）
* 56bit的主密钥经过密钥产生器产生16个48bit的子密钥。
* 中间经过16轮乘积变换。

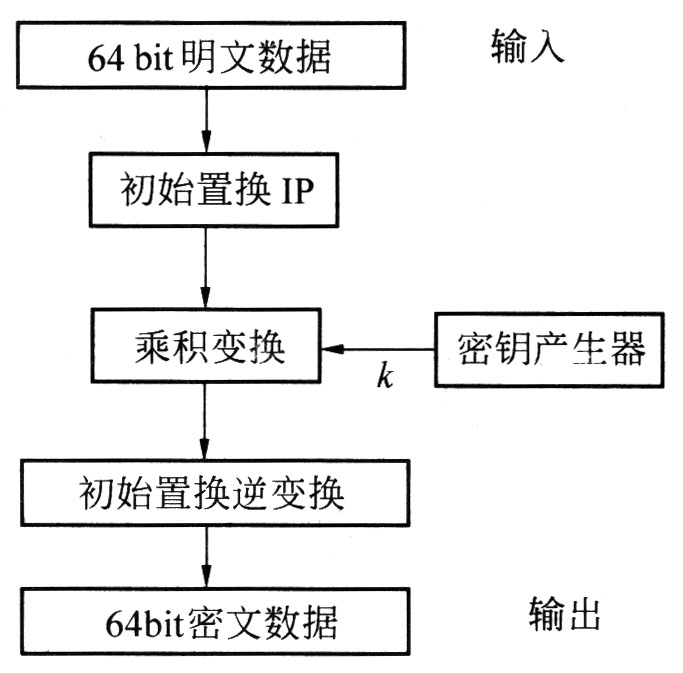


图3-1 DES算法框图

2、子密钥的生成

由于整个DES算法要进行16轮的迭代（乘积变换），如果每一轮所使用的key都完全相同的话，显然会降低算法的安全性。因此，我们的目标是，如何从一个56bit的主密钥产生16个不相同的子密钥？

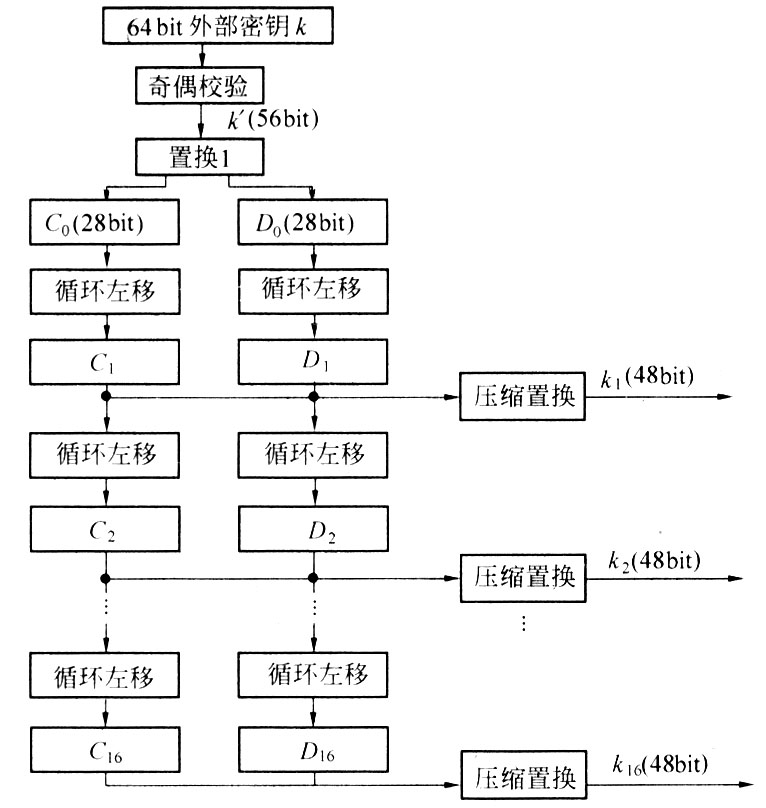


图3-2 DES算法字密钥产生流程图

DES算法字密钥产生流程图如上图所示。我们直接使用去掉奇偶检验之后的56bit主密钥即可。

（1）置换1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0~7 | 0 | **49** | **42** | **35** | **28** | **21** | **14** | **7** | **0** |
| 8~15 | 1 | **50** | **43** | **36** | **29** | **22** | **15** | **8** | **1** |
| 16~23 | 2 | **51** | **44** | **37** | **30** | **23** | **16** | **9** | **2** |
| 24~27 | 3 | **52** | **45** | **38** | **31** |  |  |  |  |

图3-3(a) C0寄存器置换1（移位操作密码表）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0~7 | 0 | **55** | **48** | **41** | **34** | **27** | **20** | **13** | **6** |
| 8~15 | 1 | **54** | **47** | **40** | **33** | **26** | **19** | **12** | **5** |
| 16~23 | 2 | **53** | **46** | **39** | **32** | **25** | **18** | **11** | **4** |
| 24~27 | 3 | **24** | **17** | **10** | **3** |  |  |  |  |

图3-3(b) D0寄存器置换1（移位操作密码表）

C0和D0寄存器长度均为28bit，可使用unsigned int型32bit变量，取其低位28bit使用，抛弃最高4bit。

特别提醒：置换1只需要做一次操作，从主密钥得到两个28bit的部分，分别称为C0和D0寄存器。

（2）循环左移

每一轮寄存器Ci+1，Di+1的内容分别是寄存器对Ci，Di的内容循环左移1位或者2位得到的。

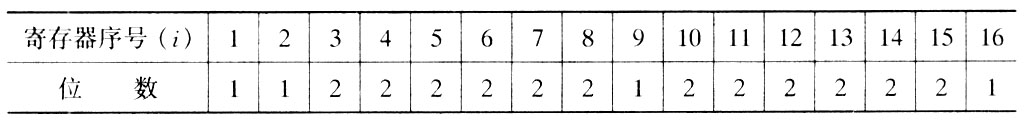


图3-4 循环左移位数

即C1，D1的内容分别是对C0，D0的内容循环左移1位得到；C2，D2的内容分别是对C1，D1的内容循环左移1位得到；C3，D3的内容分别是对C2，D2的内容循环左移2位得到；以此类推。

特别提醒：由于我们使用的是32bit的变量，请大家留意28bit的循环左移（第0bit🡪第1bit、第1bit🡪第2bit……第27bit🡪第0bit），最后第27bit要特殊处理。

（3）压缩置换

将每一轮的Ci和Di (i=1…16)寄存器的内容（合共56bit）送入图3-5所示的压缩置换表，就可以得到16个子密钥。之所以称之为压缩置换，是因为这个置换操作（移位操作）是56bit的输入，48bit的输出，抛弃了8bit的数据。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0~7 | 0 | **13** | **16** | **10** | **23** | **0** | **4** | **2** | **27** |
| 8~15 | 1 | **14** | **5** | **20** | **9** | **22** | **18** | **11** | **3** |
| 16~23 | 2 | **25** | **7** | **15** | **6** | **26** | **19** | **12** | **1** |
| 24~31 | 3 | **40** | **51** | **30** | **36** | **46** | **54** | **29** | **39** |
| 32~39 | 4 | **50** | **44** | **32** | **47** | **43** | **48** | **38** | **55** |
| 40~47 | 5 | **33** | **52** | **45** | **41** | **49** | **35** | **28** | **31** |

图3-5 压缩置换表（压缩移位密码表）

上图中第0~27bit位置来自于C寄存器，第28~54bit来自于D寄存器。你能看出来抛弃了哪几个bit？答案是抛弃了第8、17、21、24和第34、37、42、53bit。

特别提醒：压缩置换表只有一个，但由于每轮输入的Ci和Di内容是不一样的，因此得到的16个子密钥ki也是不一样的。

3、加密过程

（1）初始置换

把明文块的64bit按下表进行置换，并分成左右两半供后面的加密迭代使用。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0~7 | 0 | **57** | **49** | **41** | **33** | **25** | **17** | **9** | **1** |
| 8~15 | 1 | **59** | **51** | **43** | **35** | **27** | **19** | **11** | **3** |
| 16~23 | 2 | **61** | **53** | **45** | **37** | **29** | **21** | **13** | **5** |
| 24~31 | 3 | **63** | **55** | **47** | **39** | **31** | **23** | **15** | **7** |
| 32~39 | 4 | **56** | **48** | **40** | **31** | **24** | **16** | **8** | **0** |
| 40~47 | 5 | **58** | **50** | **42** | **33** | **26** | **18** | **10** | **2** |
| 48~55 | 6 | **60** | **52** | **44** | **35** | **28** | **20** | **12** | **4** |
| 56~63 | 7 | **62** | **54** | **46** | **37** | **30** | **22** | **14** | **6** |

图3-6 初始置换表（初始移位密码表）

将得到的64bit数据分成两部分，高32bit称为左半部L，低32bit称为右半部R。

以下将要执行16轮乘积变换操作，每一轮的操作流程如图3-7所示。

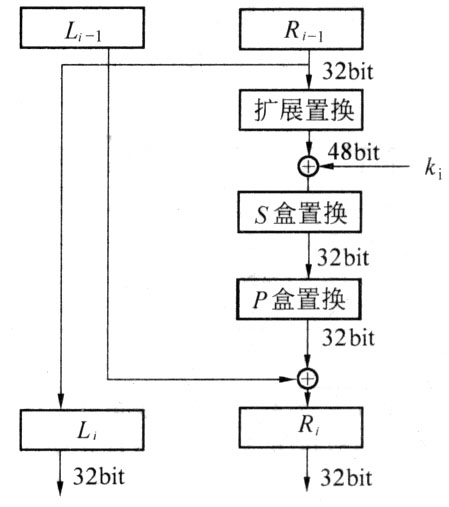


图3-7 乘积变换流程图

（2）扩展置换

在每一轮变换中（共16轮），对上一轮结果的右半部分32bit扩展为48bit。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0~7 | 0 | **31** | **0** | **1** | **2** | **3** | **4** | **3** | **4** |
| 8~15 | 1 | **5** | **6** | **7** | **8** | **7** | **8** | **9** | **10** |
| 16~23 | 2 | **11** | **12** | **11** | **12** | **13** | **14** | **15** | **16** |
| 24~31 | 3 | **15** | **16** | **17** | **18** | **19** | **20** | **19** | **20** |
| 32~39 | 4 | **21** | **22** | **23** | **24** | **23** | **24** | **25** | **26** |
| 40~47 | 5 | **27** | **28** | **27** | **28** | **29** | **30** | **31** | **0** |

图3-8 扩展置换表（扩展移位密码表）

注意红色字体位置为重复出现的bit。

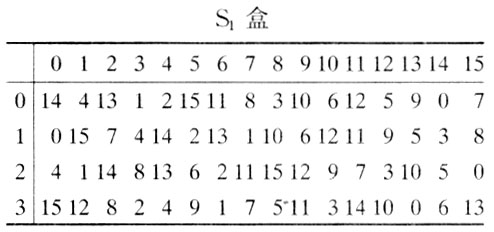
特别提醒：扩展置换表只有一个，每一轮均使用同一个置换表，但由于输入Ri是不同的，因此每一轮的结果都是不同的。

（3）48bit异或运算

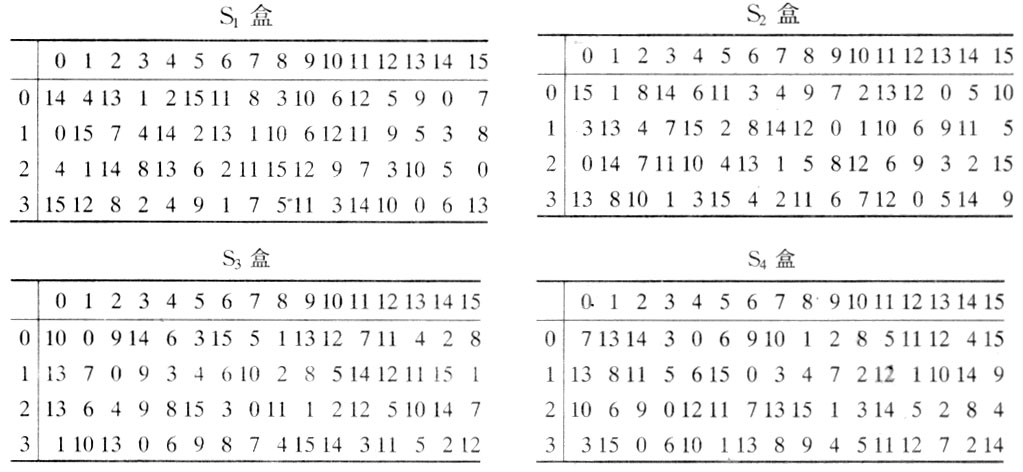
将上一步的结果48bit数据与第i轮的48bit子密钥Ki进行按位异或运算，得到的结果也为48bit。

（4）压缩S盒操作（压缩替换操作）

将上一步的48bit数据分成8组，每组6bit，并行送入8个S盒。每个S盒为6bit输入，4bit输出。



以S1盒为例，输入的6bit数据为“110010”。取第1位和第6位“10”对应表中的行，取中间4位“1001”对应表中的列。因此，输出为12，即“1100”。



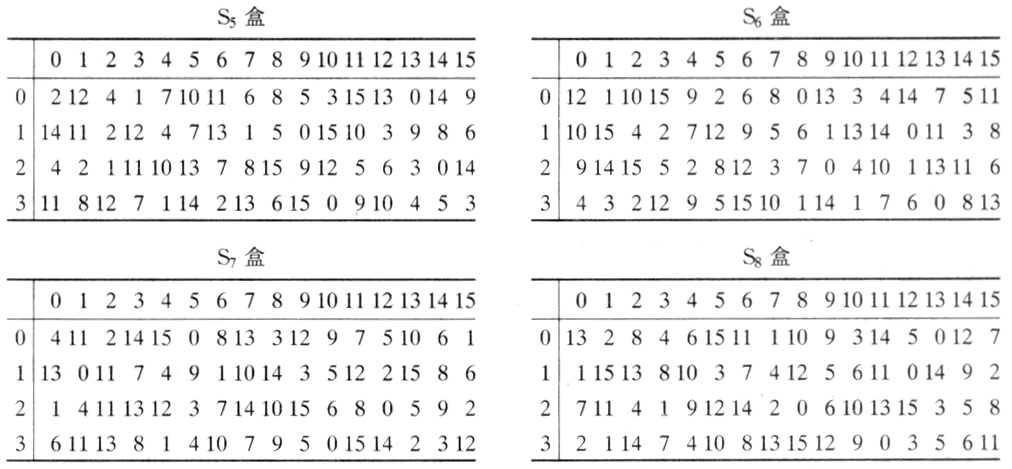


图3-9 压缩S盒子（压缩替换密码表）

（5）P盒操作

把上一步S盒子的32bit输出数据，按照图3-10的P盒子（移位密码表）重排一遍顺序。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0~7 | 0 | **15** | **6** | **19** | **20** | **28** | **11** | **27** | **16** |
| 8~15 | 1 | **0** | **14** | **22** | **25** | **4** | **17** | **30** | **9** |
| 16~23 | 2 | **1** | **7** | **23** | **13** | **31** | **26** | **2** | **8** |
| 24~31 | 3 | **18** | **12** | **29** | **5** | **21** | **10** | **3** | **24** |

图3-10 P盒子（移位密码表）

（6）32bit异或运算

将上一步的结果32bit数据与32bit的第i-1轮的左半部Li-1进行按位异或运算，得到的结果也为32bit。

（7）左右交换

将上一步的32bit的异或结果作为下一轮的右半部Ri。将上一轮的右半部Ri-1直接作为下一轮的左半部Li。

**到此为止，一轮的乘积变换就做完了，请按照上述第（2）~（7）六个步骤重复做16轮。**

特别提醒：每一轮当中，图3-8、3-9、3-10这三个步骤的密码表都是不变的，但是每一轮加入异或的Ki是不同的。请注意！

（8）终末置换

将上面第16轮的结果（64bit）送入图3-11的终末置换（终末移位密码表）中，就可以得到64bit的密文。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0~7 | 0 | **39** | **7** | **47** | **15** | **55** | **23** | **63** | **31** |
| 8~15 | 1 | **38** | **6** | **46** | **14** | **54** | **22** | **62** | **30** |
| 16~23 | 2 | **37** | **5** | **45** | **13** | **53** | **21** | **61** | **29** |
| 24~31 | 3 | **36** | **4** | **44** | **12** | **52** | **20** | **60** | **28** |
| 32~39 | 4 | **35** | **3** | **43** | **11** | **51** | **19** | **59** | **27** |
| 40~47 | 5 | **34** | **2** | **42** | **10** | **50** | **18** | **58** | **26** |
| 48~55 | 6 | **33** | **1** | **41** | **9** | **49** | **17** | **57** | **25** |
| 56~63 | 7 | **32** | **0** | **40** | **8** | **48** | **16** | **56** | **24** |

图3-11 终末置换表（终末移位密码表）

4、以上给出了DES算法加密的过程，解密的过程为逆向操作过程，请同学们自行完成。特别提醒，异或运算的逆向运算仍然是异或运算。

5、加密测试。以上算法过程给出的所有密码表均为固定的密码表，是否意味着密钥就是固定的？？其实，密钥当然不是固定的，DES算法里面的密钥长度是56bit，因此实验加密测试的时候，请随机生成一个56bit的数[0, 256-1]作为密钥即可。请留意，解密密钥与加密密钥相同。

四、实验报告要求

本次实验的难度较大，不要求所有步骤全部完成，也可以只完成加密。请在实验报告中叙述自己完成的情况。即按照自己的程度完成，完成多少记录多少。

实验四 简化的RSA加密算法实验

**一、实验目的**

 掌握消息RSA密钥生成和加密算法的原理。

**二、实验环境**

本次实验建议使用Visual C++ 6.0作为实验编程平台，如果个人认为其他编程语言平台更熟悉的话，可以在征得老师同意之后使用。

**三、实验步骤**

与对称加密算法中仅有一个密钥不同，RSA算法的密钥是成对的两个数值，称之为公钥/私钥对。而且，公钥/私钥对也不是随机生成的两个数值，他们之间必须符合某种特定的关系。

目前网络使用的RSA算法的公钥/私钥长度可达1024bit，甚至2048bit。为简化计算起见，本次实验将减小密钥的长度，降低算法的复杂性。

1、公钥/私钥对的生成算法

生成算法步骤如下：

（1）选择两个不同的素数（质数）*p*、*q*；

（2）计算它们的乘积*n*=*p*×*q*；

（3）计算欧拉函数*Ф*(*n*)=(*p*-1)(*q*-1)；

（4）选择与*Ф*(*n*)互素（互质），并且小于*Ф*(*n*)的整数*e*；

（5）计算*d*，使得*d*×*e* mod *Ф*(*n*) = 1。

结果的密钥分别为{*e*,*n*}和{*d*,*n*}。

2、具体的密钥生成步骤

（1）编程搜索两个1000以内的不同的素数（质数）*p*和*q*。搜索算法可采用最简单的穷举法，即如果正整数*i*（*i*>2,且*i*为奇数）不能被[2,]之间的任何整数所整除，则*i*为素数。（判断素数的算法请自行翻看C++入门教材）

为了达到*p*和*q*的随机性，请同学们先将搜索出来的素数存放在一个数组中，然后生成两个随机数抽取其中的两个不同的素数。注意随机数要加上时间随机种子。

（2）计算两个数值：*n*=*p*×*q*和*phi*=(*p*-1)×(*q*-1)。

（3）选择一个与*phi*互素（互质），并且小于*phi*的整数*e*。为了保证结果的随机性，请同学们随机生成一个[2,*phi*-1]之间的正整数*e*，然后判断这个正整数是否与*phi*互素，直到找到一个与*phi*互素的正整数*e*为止。（判断互素的算法请自行翻看C++入门教材）

（4）令*k*=1,2,3,… 搜索直到使得(*phi*×*k*+1)能被*e*整除，计算。

到此为止，我们计算出公钥PubliceKey（*e*，*n*）和私钥PrivateKey（*d*，*n*）。注意，无论是公钥或者是私钥都具有两个数值，推荐使用结构体变量来存储。

3、加密和解密

RSA加密和解密公式分别为：

加密：*C*=*Pe* mod *n*

解密：*P*=*Cd* mod *n*

其中的幂指数运算速度慢，可采用下面的公式进行迭代计算：

*C*=*Pe* mod *n*=((...((*P*\**P* mod *n*)\**P* mod *n*)\**P* mod *n*)... ...)\**P* mod *n*

特别提示：请注意循环的次数，应该有*e*个*P*相乘，那么究竟需要循环多少次呢？

随机选择选择一个[0, *n*-1]之间的正整数作为明文*P*，加密得到密文*C*，然后解密得到明文*P*，验证解密得到的明文*P*是否与原来的明文*P*相等。

特别提示：请注意运算过程的各个变量的范围，是否出现运算溢出？

**四、实验报告要求**

1、给出生成密钥（公钥/私钥对）、加密和解密的程序代码，要求生成的密钥具有随机性。

2、生成一对公钥和私钥，给出运算结果截图。

3、随机选择3个[0, *n*-1]之间的正整数，用刚才生成的公钥/私钥对验证加密和解密的正确性，给出运算结果截图。

运算结果要带有提示性字句，截图界面可读。