RAR 文件格式的研究

[摘要] 随着科技的发展，RAR 文件的使用已经渗透到人们生活，越来越多的工具被开发出来处理压缩文件，例如 Winrar，zip，gzip 等等，这些工具都为用户提供了良好的管理界面环境。RAR 文件中蕴藏着丰富的个人信息，发掘分析其中的有用线索是调查取证的重要手段，而其首要条件就是需要对 RAR 文件进行格式解析。本文利用 RAR 实验室提供的 Unrar 程序对 rar 数据文件进行格式解析，从而为证据信息获取提供重要手段。

[关键词] Winrar 文件　压缩文件　格式分析　加密解密　Unrar

The Research of Rar Files' Format

Abstract With the rapid development and application of computer and network，the usage of RAR files using computer more and more furious, more and more tools which come out for dealing with compressed files, such as winrar, zip, pzip etc. These tools all provide a friendly user-interface. There are rich personal information containing in RAR files. Analyzing and extracting the useable clues is very significant for case-investigation and evidence-gaining. But chiefly you have to parse the file format of RAR files. RAR lab privides the unrar functions which can be used to do the parsing work. These functions will be an important information-gaining tools.

Key Words Winrar Files Compressed-File Formats analysis Encryption-and-decryption Unrar function

目录

引言

随着科学技术水平的快速发展，越来越多的科研和工程应用部门对数据压缩和解压缩技术提出了更高的要求。RAR 作为现在最流行的数据压缩软件而备受关注，RAR 的压缩技术，密钥生成技术，加解密技术成为大家热衷研究的课题。

很多人都有过这样的经历：用 RAR 对自己的文件或文档进行有效地管理，对一些较重要的进行加密处理，可是一段时间之后需要使用时，却忘记了密码，用过各种手段之后不得不以失败告终。同样针对网络犯罪，传输经过加密后的 RAR 压缩文件，这时对 RAR 信息的取证极为重要，从中挖掘、捕获直接的犯罪信息成为调查取证的重要手段。

利用 RAR 文件进行取证，首要的任务就是要解析 RAR 文件的数据格式以及解压的方法，将经过加密的二进制文件数据还原成为课件的文本文档格式。

本文分为六个部分：

第一章 RAR 简介

第二章 RAR 压缩文件格式分析

第三章 RAR 文件解压流程

第四章 加密 RAR 文件中数据的处理

第五章 Unrar 源程序分析

第六章 总结与展望

第一章　RAR 简介

RAR 是一种专利文件格式，用于数据压缩与归档打包，开发者尤金·罗谢尔（Eugene Roshal），所以 RAR 的全名是：Roshal ARchive。首个公开版本 RAR 1.3 发布于 1993 年[1]。

Roshal 最初编写了 Dos 版本的编码和解码程序，后来被移植到很多平台，例如比较著名的 Windows 平台上的 WinRAR。Eugene Roshal 公开了解码程序的源代码，但是编码程序仍然是私有的[2]。

RAR 因为其独特的压缩算法，能够在无损数据压缩的基础上，达到很高的压缩比，同时压缩速度也不会很低[3]。因为 RAR 文件头需要占据一定空间，在数据压缩余地不大时，压缩过的文件可能比源文件要大，除此之外 RAR 文件中可能会加入冗余数据用户恢复记录，在压缩包本身受损但恢复记录够多是可以对受损压缩包进行恢复。但是 RAR 最主要的一个优点是分卷压缩，可以把文件压缩目标分割到多个文件，并且很容易从这样的分割的压缩文件中解压出源文件[4]。另外，RAR 也可以把所有文件压缩到同一个数据区以加大压缩比，代价就是解压一个单独的文件是必须解压其前面所有文件[5]。

RAR 同时也拥有成熟的加密算法，2.0 版本前加密算法未公开，2.0 后使用 AES 算法加密，在没有密码情况下目前只有暴力破解。

第二章　RAR 压缩文件格式分析

前面一章简要介绍了 RAR 的历史，本章将会从 RAR 文件的格式入手，对一个标准的 RAR 文件进行分析，深入了解 RAR 文件中的每一个块，甚至每一个字节的含义。

2.1 实例

创建 Helloworld.rar 文件，添加进文件名为 test.txt 的文本文件，该文本文件中包含以下字符串：Hello world！

在 Ultra 中加载 rar 文件：

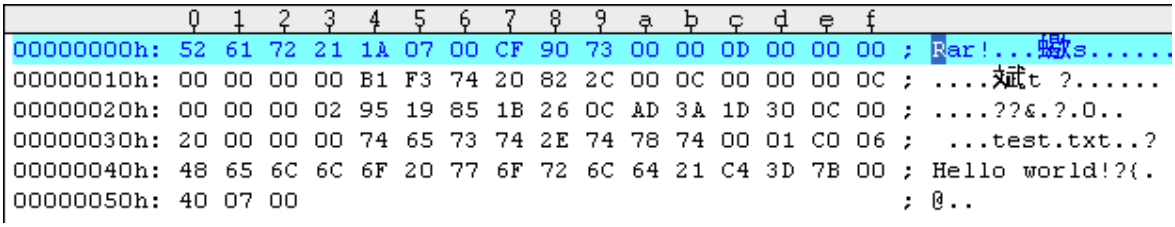


图 2-1　Helloworld.rar 文件的二进制编码

2.2 文件块结构

压缩文件由可变长度的块组成。这些块的顺序可以变化，但是第一块必须是一个在压缩文件头后的标记块[6]。

现在公开的块类型有[7]：标记块，压缩文件头块，文件头块，注释头，用户身份信息，子块和恢复记录块等。

每一块均由下列结构开始：

HEAD_CRC	2 字节	所有块或块部分的 CRC
HEAD_TYPE	1 字节	块类型
HEAD_FLAGS	2 字节	块标记
HEAD_SIZE	2 字节	块大小

如果块标记的第一位被置 1 的话，还存在：

ADD_SIZE	4 字节	可选结构 - 增加块大小

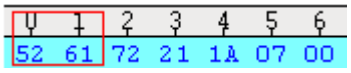
所以文件大小的计算分两种情况，当块标记 HEAD_FLAGS 首位未置 1，则总块大小就是 HEAD_SIZE，当块标记 HEAD_FLAGS 首位置 1，可选结构存在，则总块大小为

HEAD_SIZE+ ADD_SIZE[8]。

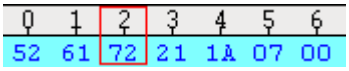
2.2.1 标记块（MARK_HEAD）

HEAD_CRC	2 字节	总是 0x6152
HEAD_TYPE	1 字节	头类型 0x72
HEAD_FLAGS	2 字节	总是 0x1a21
HEAD_SIZE	2 字节	块大小 = 0x0007,即 7 个字节

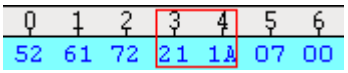
Test 文件：HEAD_CRC：



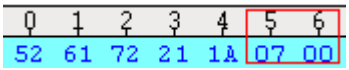
HEAD_TYPE：



HEAD_FLAGS：



HEAD_SIZE：



所以这里标记块的大小固定是 7 个字节，且是一个固定的字节序列。

2.2.2 压缩文件头（MAIN_HEAD）

HEAD_CRC	2 字节	HEAD_TYPE 到 RESERVED2 的 CRC 结构
HEAD_TYPE	1 字节	头类型:0x73
HEAD_FLAGS	2 字节	位标记:

0x0001 - 卷属性(压缩文件卷)

0x0002 - 压缩文件注释存在

RAR 3.x 使用分开的注释块，不设置这个标记。

0x0004 - 压缩文件锁定属性

0x0008 - 固实属性 (固实压缩文件)

0x0010 - 新的卷命名法则 ('volname.partN.rar')

0x0020 - 用户信息存在

RAR 3.x 不设置这个标记。

0x0040 - 恢复记录存在

0x0080 - 块头被加密

0x0100 - 第一卷(只有 RAR 3.0 及以后版本设置)

其中的其它位为内部使用保留

HEAD_SIZE　　2 字节　　压缩文件头总大小（包括压缩文件注释）

RESERVED1　　2 字节　　保留

RESERVED2　　4 字节　　保留

对于压缩文件头里的位标记，如果它的第九位被置 1，块头被加密，也就是通常所说的加密文件名，打开这样加密的 rar 文件时，需要先输入密码才能看到压缩包内的文件列表。

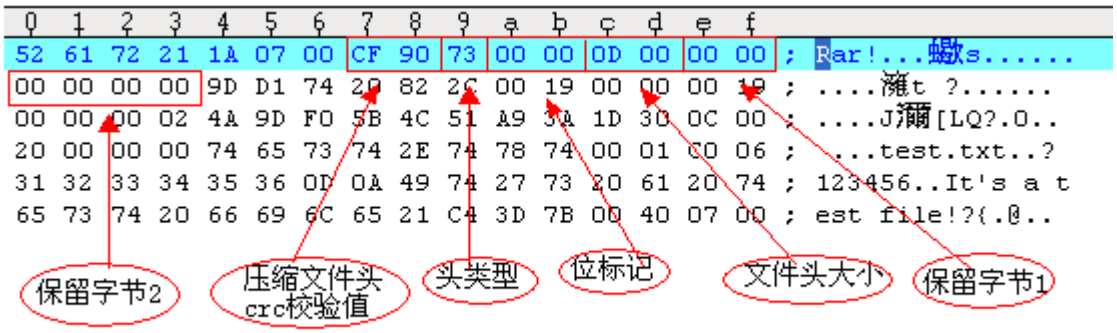


图 2-2　压缩文件中压缩文件头的格式

这里头类型是 0x73 表示是压缩文件头块，位标记为 0x0000 未有位被置 1，如果块头被加密则位标记应为 0x8000，文件头大小为 0x0D00，所以这个压缩文件头块占用 13 个字节，保留字节用 0x00 填充。

2.2.3 文件头(FILE_HEAD)

HEAD_CRC　　2 字节　　从 HEAD_TYPE 到 FILEATTR 的 CRC 结构和文件名

HEAD_TYPE　　1 字节　　头类型: 0x74

HEAD_FLAGS　　2 字节　　位标记:

0x01 - 文件在前一卷中继续

0x02 - 文件在后一卷中继续

0x04 - 文件使用密码加密

0x08 - 文件注释存在

RAR 3.x 使用分开的注释块，不设置这个标记。

0x10 - 前一文件信息被使用(固实标记)

(对于 RAR 2.0 和以后版本)

7 6 5 位(对于 RAR 2.0 和以后版本)

0 0 0 - 字典大小 64 KB

0 0 1 - 字典大小 128 KB

0 1 0 - 字典大小 256 KB

0 1 1 - 字典大小 512 KB

1 0 0 - 字典大小 1024 KB

1 0 1 - 字典大小 2048 KB

1 1 0 - 字典大小 4096 KB

1 1 1 - 文件作为字典

0x100 - HIGH_PACK_SIZE 和 HIGH_UNP_SIZE 结构存在。这些结构仅用在非常大(大于 2GB)的文档，对于小文件这些结构不存在。[9]

0x200 - FILE_NAME 包含用 0 隔开的普通的和 Unicode 编码的文件名。所以 NAME_SIZE 结构长度等于普通文件名的长度加 Unicode 编码文件名的长度再加 1。

如果此标记存在，单 FILE_NAME 不包含 0 字节，它意味文件使用 UTF-8 编码。[10]

0x400 - 头在文件名后包含附加的 8 位，它对于增加加密的安全性是必需的。(所谓的'Salt')。

0x800 - 版本标记。他是老文件版本，版本号作为';n'附加到文件名后。

0x1000 - 扩展时间区域存在。

0x8000 -此位总被设置，所以完整的块的大小是 HEAD_SIZE+ PACK_SIZE (如果 0x100 位被设置，再加上 HIGH_PACK_SIZE)

HEAD_SIZE	2 字节	文件头的全部大小(包含文件名和注释)
PACK_SIZE	4 字节	已压缩文件大小
UNP_SIZE	4 字节	未压缩文件大小
HOST_OS	1 字节	保存压缩文件使用的操作系统

0 - MS DOS

1 - OS/2

2 - Win32

3 - Unix

4 - Mac OS

5 - BeOS

FILE_CRC 4 字节 文件 CRC

FTIME 4 字节 MS DOS 标准格式的日期和时间

UNP_VER 1 字节 解压文件所需要最低 RAR 版本

版本编码方法 10 * 主版本 + 副版本。

METHOD 1 字节 压缩方式

0x30 - 存储

0x31 - 最快压缩

0x32 - 快速压缩

0x33 - 标准压缩

0x34 - 较好压缩

0x35 - 最好压缩

NAME_SIZE 2 字节 文件名大小

ATTR 4 字节 文件属性

HIGH_PACK_SIZE 4 字节

压缩文件大小 64 位值的高 4 字节。可选值，只有 HEAD_FLAGS 中的 0x100 位被设置才存在。[11]

HIGH_UNP_SIZE 4 字节

未压缩文件大小 64 位值的高 4 字节。可选值，只有 HEAD_FLAGS 中的 0x100 位被设置才存在。

FILE_NAME 文件名 - NAME_SIZE 字节大小字符串

SALT[12] 8 字节 如果 (HEAD_FLAGS & 0x400) != 0 则存在

EXT_TIME 可变大小 如果 (HEAD_FLAGS & 0x1000) != 0 则存在

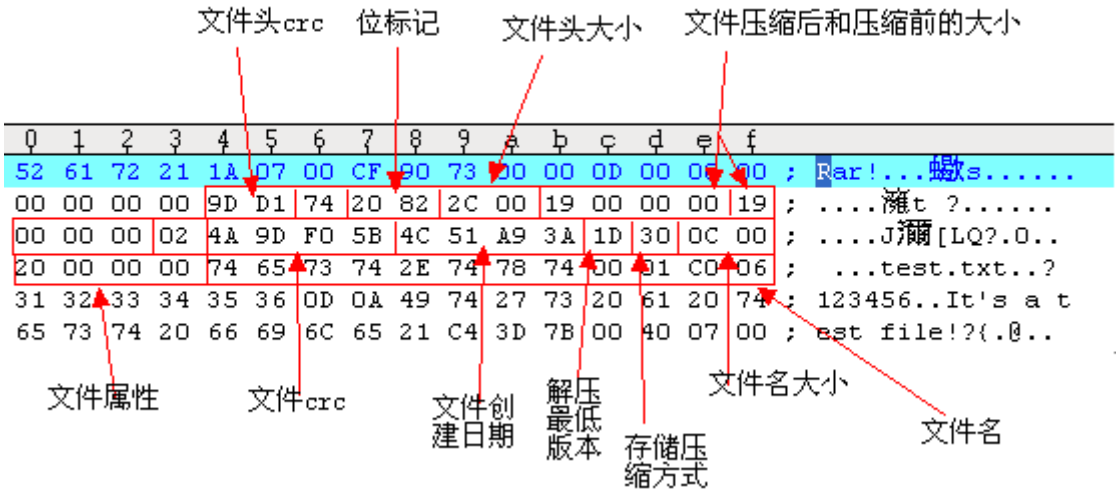


图 2-3 压缩文件中文件头的格式[13]

在这个块中，存在两个 crc 值，一个是文件头块中从位标记到文件名这 42 个字节的校验，后一个则是压缩包中所含文件的 crc 校验，解压时，会计算解压后生成文件的 crc 值，如果等于这里的 crc，则解压完成，如果不同，则报错中断。

2.2.4 结尾块

HEAD_CRC	2 字节	从 HEAD_TYPE 到 HEAD_SIZE 的 crc 校验值
HEAD_TYPE	1 字节	头类型 0x7B
HEAD_FLAGS	2 字节	位标记
HEAD_SIZE	2 字节	注释头大小

与标记块类似的是，结尾块也是一个固定字节串的块，依次是 0xC4 3D 7B 00 40 07 00。

2.2.5 旧风格的块类型

除以上格式块以外，还存在一些旧风格的块类型，不过在新的版本中已经不存在了。

注释头块：

HEAD_CRC	2 字节	从 HEAD_TYPE 到 COMM_CRC 的 crc 校验值
HEAD_TYPE	1 字节	头类型 0x75
HEAD_FLAGS	2 字节	位标记
HEAD_SIZE	2 字节	注释头大小
UNP_SIZE	2 字节	未压缩注释大小
UNP_VER	1 字节	提取注释的 RAR 最低版本

METHOD 1 字节 压缩方法

COMM_CRC 2 字节 注释 CRC

COMMENT 注释正文

额外信息块：

HEAD_CRC 2 字节 块 CRC

HEAD_TYPE 1 字节 头类型 0x76

HEAD_FLAGS 2 字节 位标记

HEAD_SIZE 2 字节 总块大小

INFO 额外信息正文

字块[14]：

在压缩文件中任意文件头块后面都可以附加一个字块。这个字块依赖于它前面的这个主块。当更新时新版本的 RAR 压缩包可能会删除或者移动这个字块。

字块包含下面几个部分：

HEAD_CRC 2 字节 块 crc

HEAD_TYPE 1 字节 头类型: 0x77

HEAD_FLAGS 2 字节 位标记

HEAD_SIZE 2 字节 总块大小

DATA_SIZE 4 字节 总数据块大小

SUB_TYPE 2 字节 子块类型

RESERVED 1 字节 保留字段，必须为 0

其余字段 由 SUB_TYPE 决定其余字段类型

以 SUB_TYPE 为 0x100 为例，0x100 定义子块类型为扩展属性类型，一般用于压缩一些文件属性信息较详细的文件。

字段中可以包括以下格式:

HEAD_CRC 2 字节 块 CRC

HEAD_TYPE 1 字节 头类型: 0x77

HEAD_FLAGS 2 字节 位标记

HEAD_SIZE 2 字节 总块大小

DATA_SIZE 4 字节 总数据大小

SUB_TYPE 2 字节 0x100 //定义子块为扩展属性类型

[15]

RESERVED	1 字节	全 0	//以上为子块中固定格式
UNP_SIZE	4 字节	未压缩扩展属性大小	//以下为扩展属性附加字段
UNP_VER	1 字节	RAR 版本信息	
METHOD	1 字节	压缩方法	
EA_CRC	4 字节	扩展属性 CRC	

第三章　RAR 文件解压流程

上一章节分析了 RAR 文件的格式，本章要从 RAR 文件的解压开始，进一步分析一般情况下是如何处理 RAR 文件的，详细了解文件解压处理的流程。

3.1 压缩文件处理步骤

文件的处理过程可以简要分成下面几步，其中省略了压缩包的打开和关闭过程和内存分配的步骤。RAR 文件具体步骤如下：

1. 读取和检查标记块

一般情况下就是需要读取文件首个 7 字节，检查是否与固定情况相同，如果相同则表明这是一个 RAR 文件。

2. 读取压缩文件头

这里读取紧接下来的 7 字节，首先检查第三个字节，即块类型是否为 0x73,其次检查位标记的两个字节，特别需要注意的是 0x0008 位和 0x0080 位。如果 0x08 位为 1，则压缩包使用固实压缩方法处理。

固实压缩包是用一种特殊压缩方式压缩的 RAR 压缩包，它把压缩包中的所有文件当成一个连续数据流来看待。固实压缩只被 RAR 格式的压缩包支持，ZIP 压缩包不支持。使用固实压缩可以明显提高压缩比，特别是在添加大量的小文件时。

如果 0x80 位为 1 的话，则表示从下一个块开始所有数据均被加密处理，如果需要解压，或者需要了解任何有关压缩文件的信息都需要进行数据恢复处理。

3. 读取（先跳过 HEAD_SIZE-sizeof(MAIN_HEAD) 字节）

这里需要跳过压缩文件头，将指针指向下一个块，也就是文件头块的开始位置，然后读取紧接下来的 7 字节。

4. 如果发现压缩文件结尾则压缩文件处理终止，否则读取 7 字节到结构 HEAD_CRC，HEAD_TYPE，HEAD_FLAGS，HEAD_SIZE 中。

接下来的这几个步骤构成了一个循环体，所以需要设置一个检验条件，因为每个压缩包的最后一个块都是结尾块，而且是一个固定的字符串，这个时候只需要比较字符串就可以判断是否压缩文件终止。

5. 检查 HEAD_TYPE

if HEAD_TYPE==0x74

读取文件头 (开始的 7 字节必须读取)

读取或跳过 HEAD_SIZE-sizeof(FILE_HEAD) 字节

if (HEAD_FLAGS & 0x100)

读取或跳过 HIGH_PACK_SIZE*0x100000000+PACK_SIZE 字节

else

读取或跳过 PACK_SIZE 字节

else

读取 corresponding HEAD_TYPE 块:

读取 HEAD_SIZE-7 字节

if (HEAD_FLAGS & 0x8000)

读取 ADD_SIZE 字节

6. goto 步骤 4

3.2 压缩文件处理流程图

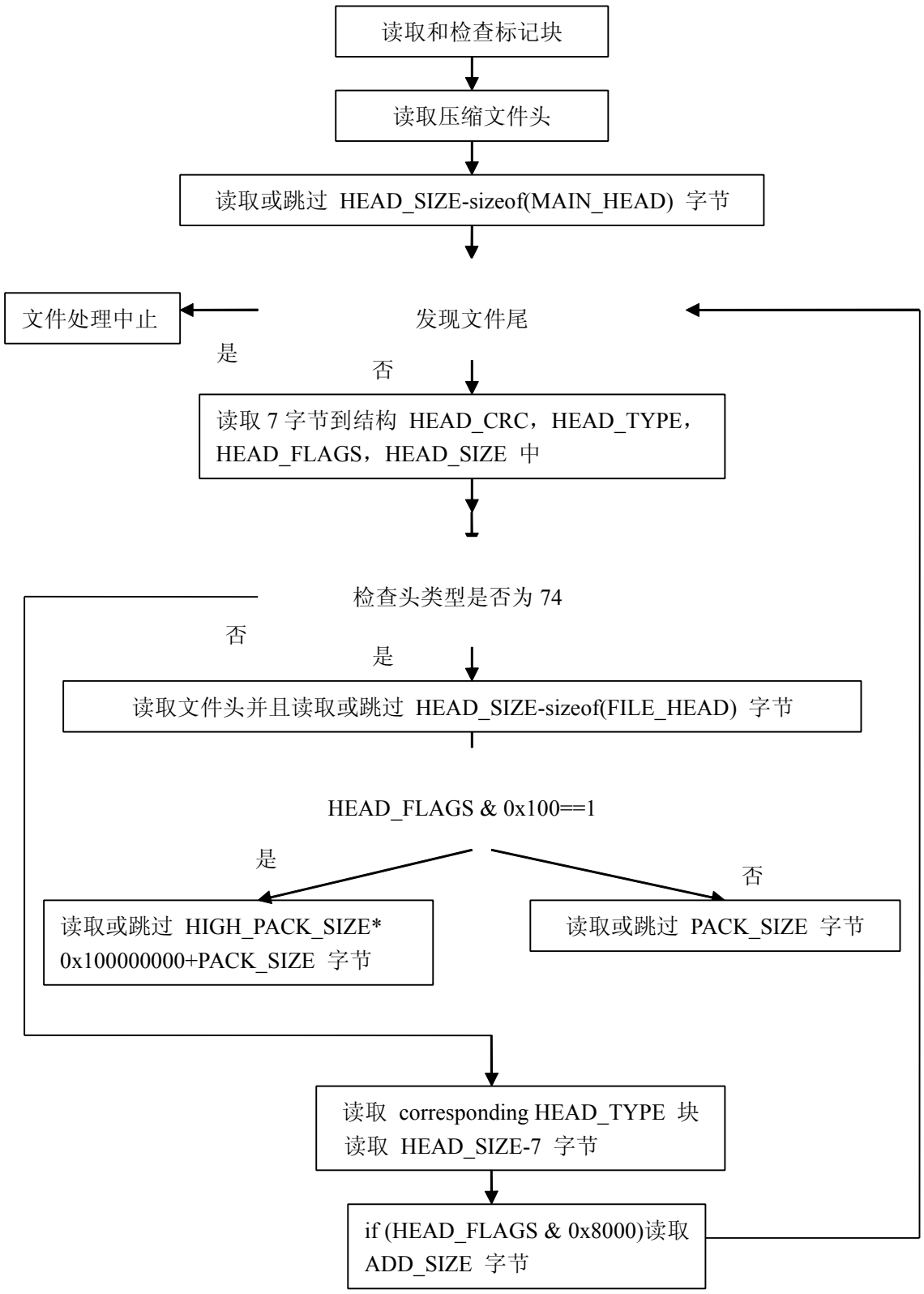


图 3-1 压缩文件处理流程图

第四章　加密 RAR 文件数据的处理

从第一章的分析我们知道，当压缩文件头的位标记的 0x0080 位置 1，或者文件头块的第 0x0004 位置 1，则 RAR 文件被加密，那么说明需要密码才能解压，所以需要先用密码将加密文件中的数据恢复成原始压缩数据，才能用上面所述步骤解压。

4.1 密钥的生成

将明文的密码与 Salt 一起，通过 HASH 算法，生成两个 16 字节的密钥。生成的 2 个 16 字节中，其中一个是 KEY，也就是 AES 算法的参数，另一个 16 字节为 initVector，用于处理加密数据的第一个 16 字节。

4.2 源数据的恢复

加密 RAR 文件中的数据是采用 AES 算法的 rijndael 标准应用，由于 AES 算法是对称的，所以解密的过程，是加密过程的逆运算。但解密时 AES 算法过程与加密所用的不一样，这是是因为解密过程中由 KEY 生成的子密钥表不一样。仍然需要我们将密码输入，与 salt 一起生成两个 16 字节密钥，KEY 和 initVector。然后才能用这两个参数处理下面的源数据恢复的操作处理。

```
packblock[0]=AES1(encryptBlock[0])                    //（KEY为AES的密钥）
packblock[0]=packblock[0]^initVector
for i=1  to   块数量-1
      packblock[i]=AES1(encryptBlock[i])              //（KEY为AES的密钥）
      packblock[i]=packblock[i]^encryptBlock[i-1]
next
//packblock表示压缩完的每16字节数据
//encryptBlock表示加密完的每16字节数据
//initVector即为密码Hash运算后生成的第二个16字节
```

第五章　Unrar 源程序分析

Unrar 源程序是专门配套用来解压 RAR 文件的程序，以 Linux 版本为例，可以通过 gcc 编译和 link 连接后生成可执行文件，生成的可执行文件可以用来解压任意 RAR 文件。

Unrar 源程序包中包括 58 个头文件和 58 个 C++文件，以及一个 makefile 编译文件，这是 RAR 实验室首度公开的源码程序。

5.1 典型函数分析

Unrar 中常见的函数大概分外围处理函数，文件头处理函数，数据处理函数和其余处理函数。

5.1.1 RAR 外围处理函数

==========

HANDLE PASCAL **RAROpenArchive**(struct RAROpenArchiveData *ArchiveData)

==========

用途

打开 RAR 压缩包，并且为压缩分配内存空间

参数

ArchiveData：　　指针指向 RAROpenArchiveData 结构体

结构体 RAROpenArchiveData：

```
struct RAROpenArchiveData
{
  char *ArcName;
  UINT OpenMode;
  UINT OpenResult;
  char *CmtBuf;
  UINT CmtBufSize;
  UINT CmtSize;
  UINT CmtState;
};
```

ArcName　　//压缩包文件名

输入参数，指向压缩包文件名，文件名必须是一个以 0 结尾的字符串

OpenMode　　//打开方式

输入参数

OpenMode 有以下几种打开方式

方式一：RAR_OM_LIST（只读取文件头）

这种打开方式只会读取压缩包中文件头字段信息

方式二：RAR_OM_EXTRACT

这种打开方式会解压出压缩包中所有文件并且进行 CRC 检验

RAR_OM_LIST_INCSPLIT

这种打开方式同样只会读取压缩包中的文件头字段信息。但是如果在这个模式下打开压缩包，之后运行函数 RARReadHeader[Ex]会返回所有文件的头信息，而在 RAR_OM_LIST 的模式下这些头信息会自动忽略。所以以 RAR_OM_LIST_INCSPLIT 模式处理 RAR 时，会得到一系列的文件头，这些文件头信息中间会用“file continued from previous volume”这样的字样隔开。

OpenResult　　//结果的返回信息

输出参数

一个 RAR 文件的打开可能出现以下几种结果：

```
0                   //打开成功
ERAR_NO_MEMORY      //内存空间不足
ERAR_BAD_DATA       //头文件被破坏
ERAR_BAD_ARCHIVE    //RAR文件无效
ERAR_UNKNOWN_FORMAT //压缩包文件头加密方法未知
ERAR_EOPEN          //文件打开出错
```

CmtBuf

输入参数，指针指向压缩包注释信息的缓冲区，注释信息最大 64Kb，同样也必须以 0 结尾，如果注释大于分配的缓冲区大小，剩余信息就会被截断。如果 CmtBuf 被设置成 0 的话，就不需要提取注释信息。

CmtBufSize

输入参数，指出压缩包注释信息缓冲区的大小

CmtSize

输出参数，实际读出的注释信息的大小，不能超过分配的空间大小

CmtState

输出参数，CmtState 的状态有以下几种情况：

```
0                       //注释不存在
1                       //注释完整读取
ERAR_NO_MEMORY          //内存不足
ERAR_BAD_DATA           //注释被损坏
ERAR_UNKNOWN_FORMAT     //注释格式未知
ERAR_SMALL_BUF          //缓冲区过小，不能完整读取
```

==========

HANDLE PASCAL **RAROpenArchiveEx**(struct RAROpenArchiveDataEx *ArchiveData)

==========

用途：

具体用途与 RAROpenArchive 相似，但是 RAROpenArchiveDataEx 能够允许使用 Unicode 统一编码的文件名，并且能够返回压缩包标志位信息。

以下列出参数 RAROpenArchiveDataEx 结构体的信息，其余字段信息和函数返回值的具体描述省略，可以参考 RAROpenArchive。

```
struct RAROpenArchiveDataEx
  {
    char          *ArcName;
    wchar_t       *ArcNameW;
    unsigned int OpenMode;
    unsigned int OpenResult;
    char          *CmtBuf;
    unsigned int CmtBufSize;
    unsigned int CmtSize;
    unsigned int CmtState;
    unsigned int Flags;
    unsigned int Reserved[32];
  };
```

==========

int PASCAL **RARCloseArchive**(HANDLE hArcData)

==========

用途：

关闭 RAR 压缩包，并且释放所有分配的内存。一般在对压缩包的所有处理完成之后运行，也可是压缩包处理过程出现错误被迫中止。

参数：

hArcData

这个参数是运行函数 RAROpenArchive 获取的数据。

返回值

```
0                       //关闭成功
ERAR_ECLOSE             //关闭出错
```

5.1.2 文件头处理函数

==========

int PASCAL **RARReadHeader**(HANDLE hArcData, struct RARHeaderData *HeaderData)

==========

用途

读取压缩文件头

参数

hArcData

这个参数是运行函数 RAROpenArchive 获取的数据。

HeaderData

指针指向 RARHeaderData 结构体：

```
struct RARHeaderData
  {
    char ArcName[260];
    char FileName[260];
    UINT Flags;
    UINT PackSize;
    UINT UnpSize;
    UINT HostOS;
    UINT FileCRC;
    UINT FileTime;
    UINT UnpVer;
    UINT Method;
    char *CmtBuf;
    UINT CmtBufSize;
    UINT CmtSize;
    UINT CmtState;
  };
```

结构体字段的参数具体描述：

ArcName

输出参数，输出 RAR 文件名，此文件名必须是以 0 为结尾的字符串。

FileName

输出参数，输出以 OEM (DOS)编码的文件名，也必须是以 0 为结尾的字符串。

Flags

输出参数，文件头块的位标记

PackSize

输出参数，标明文件压缩后的大小或者如果文件被分卷则表示每一分卷的大小。

UnpSize

输出参数，文件未压缩是的大小

HostOS

输出参数，保存压缩文件使用的操作系统

FileCRC

输出参数，未压缩文件的 crc，如果文件被分卷，则之后最后一个分卷才含有正确的 crc 值，并且只能用 RAR_OM_LIST_INCSPLIT 模式才能获取。

FileTime

输出参数，MS DOS 标准格式的时间和日期

UnpVer

输出参数，解压文件所需要的最低 RAR 版本

Method

输出参数，压缩方式

FileAttr

输出参数，文件属性

CmtBuf

输入参数，注释缓冲区

CmtBufSize

输入参数，注释缓冲区大小

CmtSize

输出参数，读进缓冲区的注释大小

CmtState

输出参数

Return values

```
0                       //读取文件头成功
ERAR_END_ARCHIVE        //压缩包结束
ERAR_BAD_DATA           //文件头损坏
```

==========

int PASCAL **RARReadHeaderEx**(HANDLE hArcData,struct RARHeaderDataEx *HeaderData)

==========

用途

与函数 RARReadHeader 类似

```
struct RARHeaderDataEx
    {
      char          ArcName[1024];
      wchar_t       ArcNameW[1024];
      char          FileName[1024];
      wchar_t       FileNameW[1024];
      unsigned int Flags;
      unsigned int PackSize;
      unsigned int UnpSize;
      unsigned int UnpSizeHigh;
      unsigned int HostOS;
      unsigned int FileCRC;
      unsigned int FileTime;
      unsigned int UnpVer;
      unsigned int Method;
      unsigned int FileAttr;
      char          *CmtBuf;
      unsigned int CmtBufSize;
      unsigned int CmtSize;
      unsigned int CmtState;
      unsigned int Reserved[1024];
    };
```

5.1.3 RAR 文件数据处理函数

==========

int PASCAL **RARProcessFile**(HANDLE hArcData,

int Operation,

char *DestPath,

char *DestName)

==========

用途

从当前位置移动到下一个文件的位置，并且从压缩包中解压出当前文件，如果是在 RAR_OM_LIST 模式下运行的话，则不解压直接移动到下一个位置。

参数

hArcData

这个参数是运行函数 RAROpenArchive 获取的数据。

Operation

文件操作

有以下三种情况：

第一种：RAR_SKIP

直接移动到压缩包中的下一个文件处，但是如果是在 RAR_OM_LIST 模式下的话，或者压缩包被固实处理，则当前文件还是会被处理，并且速度会慢于正常情况下的处理。

第二种：RAR_TEST

测试当前文件，移动到下一个文件处。同样如果是在 RAR_OM_LIST 模式下的话操作将等同于 RAR_SKIP。

第三种：RAR_EXTRACT

加压当前文件，移动到下一个文件处。同样如果是在 RAR_OM_LIST 模式下的话操作将等同于 RAR_SKIP。

DestPath

指针指向文件解压后要放到的路径，必须是一个以 0 为结尾的字符串。如果此参数被设置成 NULL，则在当前目录下解压缩。只有 DestName 是 NULL 是此参数才有意义。

DestName

指针指向一个包含全路径和名称的字符，这里路径为解压路径，名称为希望另存的名称，如果为 NULL，则使用默认的名称。如果 DestName 被定义了，则会自动更改压缩包的路径和名称设置。

DestPath 和 DestName 都必须用 OEM 编码。如果需要的话，可以调用 CharToOem 函数将文本转化为 OEM 编码。

Return values

```
0                       //成功
ERAR_BAD_DATA           //文件CRC出错
ERAR_BAD_ARCHIVE        //分卷无效
ERAR_UNKNOWN_FORMAT     //未知格式
ERAR_EOPEN              //打开分卷出错
ERAR_ECREATE            //文件新建出错
ERAR_ECLOSE             //文件关闭出错
ERAR_EREAD              //读取出错
ERAR_EWRITE             //写入出错
```

注：如果中途需要中止解压，可以直接输入 return -1。

==========

int PASCAL **RARProcessFileW**(HANDLE hArcData,

int Operation,

wchar_t *DestPath,

wchar_t *DestName)

==========

用途：

RARProcessFile 的代码版本，功能与 RARProcessFile 类似

5.1.4 其余处理函数

==========

void PASCAL **RARSetCallback** (HANDLE hArcData,

int PASCAL (*CallbackProc) (UINT msg,LPARAM UserData,

LPARAM P1,LPARAM P2) , LPARAM UserData);

==========

用途

设置一个用户定义的回调函数来处理 Unrar 活动。

参数

hArcData

这个参数是运行函数 RAROpenArchive 获取的数据。

CallbackProc

指针指向用户定义的回调函数

这个回调函数可以有下面几个参数

Msg　　处理事件的类型。

UserData　　用户定义的传递给 RARSetCallback 的值

P1 and P2　　事件依赖的参数

可以处理的事件类型有：

UCM_CHANGEVOLUME　　改变分卷，参数不同是有以下两种处理方式：

P1　　指向下一卷中以 0 为结尾的名称

P2　　该函数调用模式有：

```
RAR_VOL_ASK          //被调用分卷不存在。这个函数会促使用户重新设置
                  //参数值或者用return -1中止操作。这个函数还能够
                  //指定一个新的卷名，把他的地址指定为P1参数。
RAR_VOL_NOTIFY       //被调用分卷成功打开，并提示不允许对分卷名进行
                  //修改操作。这个函数同样要求用户重新设置参数值
                  //或者用return -1中止操作。
```

UCM_PROCESSDATA　　处理解压后的数据，他可以用来处理文件虽然被解压了却没有出现在磁盘上的情况。提示用户修改参数值或者 return -1 中止操作。

P1　　可以用来指向解压后产生的数据，但是不能修改。

P2　　解压后的数据大小，可以用来检验它是否超过最高字典大小（RAR 3.8 为 4MB）

UCM_NEEDPASSWORD　　当文件名被加密时，提示必须输入密码才能查看文件名。甚至在处理加密了压缩包而未加密文件名的 RAR 文件是，它可以用来代替函数 RARSetPassword。

P1　　指针指向用来存储密码的缓冲区，如果要查看文件名，这个缓冲区中的必须存在一个可行的密码

P2　　密码缓冲区的大小。

UserData

用户传递给回调函数的数据

Unrar 中的任何其他函数都不能调用 callback 函数。

==========

void PASCAL **RARSetPassword**(HANDLE hArcData, char *Password);

==========

用途

设置密码来解密文件

参数

hArcData

这个参数是运行函数 RAROpenArchive 获取的数据。

Password

必须指向以 0 为结尾的密码字符串。

==========

void PASCAL **RARGetDllVersion**();

==========

用途

返回 API 版本信息

5.2 调用 Unrar 解压缩的代码

```
#include "UnRAR.h"
#ifdef _UNICODE
#define _ArcName ArcNameW
#define STR_RARProcessFile TEXT("RARProcessFileW")
#else
#define _ArcName ArcName
#define STR_RARProcessFile TEXT("RARProcessFile")
#endif
typedef HANDLE (PASCAL *PRAROpenArchiveEx)(struct RAROpenArchiveDataEx
*ArchiveData);
typedef INT (PASCAL *PRARReadHeader)(HANDLE hArc, struct RARHeaderData
*HeaderData);
typedef INT (PASCAL *PRARProcessFile)(HANDLE hArc, INT iOperation, PTSTR
pwzDestPath, PTSTR pwzDestName);
typedef INT (PASCAL *PRARCloseArchive)(HANDLE hArc);
```

以上定义了程序中所有用到的数据类型。

```
HRESULT RARX(PTSTR ptzSrcFile, PTSTR ptzDstDir)
{
   HMODULE hLib = LoadLibrary(TEXT("UnRAR"));
   if (hLib == NULL)
   {
      return ERROR_DLL_NOT_FOUND;
   }

   PRAROpenArchiveEx pRAROpenArchiveEx = (PRAROpenArchiveEx)
GetProcAddress(hLib, TEXT("RAROpenArchiveEx"));
   PRARReadHeader pRARReadHeader = (PRARReadHeader) GetProcAddress(hLib,
TEXT("RARReadHeader"));
   PRARProcessFile pRARProcessFile = (PRARProcessFile) GetProcAddress(hLib,
STR_RARProcessFile);
   PRARCloseArchive pRARCloseArchive = (PRARCloseArchive) GetProcAddress(hLib,
TEXT("RARCloseArchive"));
   if (!pRAROpenArchiveEx || !pRARReadHeader || !pRARProcessFile || !
pRARCloseArchive)
   {
      FreeLibrary(hLib);
      return E_NOINTERFACE;
   }
   struct RAROpenArchiveDataEx od = {0};
   od._ArcName = ptzSrcFile;
   od.CmtBufSize = 16384;
   od.CmtBuf = (PSTR) UMemAlloc(od.CmtBufSize);
   if (od.CmtBuf == NULL)
   {
     FreeLibrary(hLib);
     return E_OUTOFMEMORY;
   }

   od.OpenMode = RAR_OM_EXTRACT;
   HANDLE hArc = pRAROpenArchiveEx(&od);
   if (od.OpenResult == S_OK)
   {
      struct RARHeaderData hd = {0};
      while ((od.OpenResult = pRARReadHeader(hArc, &hd)) == S_OK)
      {
         od.OpenResult = pRARProcessFile(hArc, RAR_EXTRACT, ptzDstDir, NULL);
         if (od.OpenResult != S_OK)
         {
            break;
         }
      }
   }

   pRARCloseArchive(hArc);
   UMemFree(od.CmtBuf);
   FreeLibrary(hLib);
   return od.OpenResult;
}
```

这一段代码是程序的主体，但只是解压中一个最基本情况的应用，会用到独特的库文件 Unrar.Lib。

总结与展望

RAR 文件解析研究范围十分广泛。在网络发展初期，由于带宽有限，人们不断地寻找可以讲数据压缩同时又能完全恢复的技术，于是 RAR 文件压缩技术就有了初步的发展。随着信息技术的发展，数据量成指数级增长，所传送的数据量大大增加，超过了网络带宽所能承受的极限：同时，由于大量的文档和数据需要存储和备份，也给数据的存储带来极大地压力。因此有必要 RAR 格式信息以及解压的具体内容，以促进压缩技术的研究。

本文从压缩文件的格式分析入手，对压缩包中的块结构分别进行解析，进而讨论了压缩文件一般处理情况和加密情况下的处理方法，加密情况下需要先使用密码进行源数据的恢复，最后分析了解压程序 Unrar 源码中一些常见函数的详细解析。

本文的研究还有许多不足和值得改进之处，以后的研究方向可以更深入的从以下方面考虑：

1、 研究更多可能存在的 RAR 格式。

2、 分析研究整个解压过程的细节。

3、 定位加密，解密模块，为将来有效地破解提供有用的信息。

参考文献

[1]Langdon G/Rissanes J. The format of RAR files[J]. IEEE Trans On Comm,1981,29(6):858-867.

[2]陆军/刘大昕. RAR 常数级压缩方法中随机文件字节频率统计研究[J]. 微电子学与计算机,2007,24(9):49-51.

[3]Ziv J/Lempel A. A universal algorithm for sequential data compression[J]. IEEE Transactions on Information Theory,1977,23(3):337-343.

[4] RARlab. WinRAR Version History. 2005:Available at http://www.rararchiver.com/WinRARVersion.

[5]李世畅/杨朝军/陶洋. Linux 嵌入式系统的优化[J]. 重庆邮电学报,2008,14(4):61-64.

[6]王平. LZW 无损压缩算法的实现与研究[J]. 计算机工程,2007,28(8):98-99.

[7]RARlab. Unrarsrc-3.8.5.tar.gz. 2008:Available at http://www.rarlab.com/.

[8]王虹. 文件系统的管理方法[J]. 河南科学,2007,21(3):375-378.

[9] WinZip Computing,Inc. What's New in WinZip 9.0. 2005:Available at http://www.winzip.com/

[10] 陈浩. 压缩工具[M]. 上海:上海科学技术出版公司,2008.

[11] RARlab:WinRAR Archiver.2005:RAR—What's New in the Latest Version. Available at http:// www.rarlab.com/rarnew.htm.

[12] 尤霞光. 信息技术基础[M]. 北京:中国电力出版社,2007.

[13] Gary S.-W.Yeo/Raphael C.-W.Phan. On the security of the WinRAR encryption feature[J]. Int. J.Inf. Secur,2006,5(2):115-123.

[14] 洪锦魁. WinZip 8.X 压缩解压缩[M]. 北京:海洋出版社,2001.

[15] 朱明方/朱峰. 压缩软件精华工具集[M]. 北京:清华大学出版社,1996.

[16] 胡鸣. Windows 网络编程技术[M]. 北京:科学出版社,2008.