HPACK

The Multi-System Archiver

Version 0.78a0

1 Sept 1992

Contents

Introduction

The HPACK Multi-System Archiver is an archiver that was written to allow the transfer of archived data to different systems. In the past archivers have traditionally been available for single systems only, for example PKZIP™ and LHARC for MSDOS, Larc for the Amiga, StuffIt™ and Compact Pro™ for the Macintosh, and tar and compress for UNIX systems (while these archivers are available on other systems, their use is not widespread). It is intended to make HPACK a more universal archiver by offering versions for the following environments:

- Apple IIgs (planned)
- Archimedes
- Atari ST
- Commodore Amiga
- Microsoft Windows (currently in development)
- Macintosh
- MSDOS
- OS/2 (16 and 32-bit versions)
- OS/2 Presentation Manager (currently in development)
- Primos (planned)
- UNIX
- VMS (currently in development)
- Any other system which people offer to port it to

In addition HPACK has built-in internationalization support (inasmuch as the system on which it is running will allow), allowing easy creation of multi-language versions. Currently Dutch, English German, and Italian versions of HPACK exist.

Using HPACK

HPACK is run with the following command:

```
HPACK command [-options] archive [filenames...][@scriptfiles...]
```

Allowed commands are:

`A`	Add files to an archive.
`X`	Extract files from an archive.
`V`	Display a directory of files inside an archive.
`P`	View files within an archive.
`T`	Test the integrity of an archive.
`D`	Delete files from an archive.
`F`	Freshen files to an archive.
`R`	Replace files in an archive.
`U`	Update files to an archive.

Allowed options are:

`-0`	Store files without attempting any compression.
`-a`	Store file attributes.
`-b`	Specify a base pathname for all files.
`-c`	Encryption options (conventional and public-key encryption).
`-d`	Directory options (Mkdir, Rmdir, Mvdir, path options etc).
`-e`	Add/use error recovery information.
`-f`	Move files into/out of an archive.
`-i`	Interactive mode (prompt for all actions).
`-k`	Overwrite existing archive.
`-l`	Security options (data authentication)
`-m`	Create a multipart archive.
`-o`	Overwrite on extraction options (All, None, Prompt, Smart).
`-r`	Recurse through subdirectories.
`-s`	Run in stealth mode.
`-u`	Unified compression mode.
`-v`	View files options (Files, Directories, All).
`-w`	Treat files as archive comments.
`-x`	Text file translate options (see below).
`-z`	Extended options (system-specific, see below).

Commands, options, and archive and file names may be given in upper or lower case (they are given in uppercase in the examples which follow merely for consistency). Options may be lumped together or may be given seperately preceded by the '-' delimiter. The option '--' may be used to indicate that no more options are present on the command line, following the standard Unix convention.

The default archive extension is '`.HPK`'. HPACK will always add this extension (provided the underlying filesystem supports it), changing any other given extension if necessary. Note that some quantum physics theories suggest that when the user is not directly observing the HPACK archive, it may cease to exist or will exist only in a vague and indeterminate state; in this case the existence of the '`.HPK`' extension cannot be guaranteed.

The filename's field may consist of zero or more filenames, defaulting to all files if no filenames are given (in other words archive all files in the current directory, or dearchive/view all files in the archive). HPACK uses UNIX-style wildcards (which are described in more detail below in the section "HPACK Wildcards"), as well as UNIX-style pathnames in place of the usual ones used by the systems' command interpreter, so that for example the path:

```
/TEMP/WORK/FILES
```

is used instead of the MSDOS equivalent:

```
\TEMP\WORK\FILES
```

or the VMS equivalent:

```
[TEMP.WORK]FILES
```

If any additional components are needed, such as drives, network nodes, or devices, these can be specified in the manner usual to the host command interpreter. For example if the path in the above example were on drive A: under MSDOS the pathname would be:

```
A:/TEMP/WORK/FILES
```

If the path in the above example were on the '`HOME`' node, on device '`DISK1`' under VMS, the pathname would be:

```
HOME::DISK1:/TEMP/WORK/FILES
```

The scriptfiles are names of files preceded with an '`@`' character. When HPACK encounters one of these it will treat the filename as a file containing a list of files to process. Script files and normal filenames may be freely mixed on the command line. See the section "HPACK Script Files" below for more information.

HPACK Wildcards

When HPACK scans any filename that has been given to it, it will check for the presence in the filename of any of the special characters:

```
* ? [ ] \
```

If one of these is detected, then the string of characters making up the filename will be treated as being a pattern for a filename, rather than the name of a specific file. These special characters have the following meanings when used in a filename string:

`*`	Matches zero or more characters
`?`	Matches any one character
`[...]`	Matches any of the enclosed range of characters '`...`' in turn. If two characters appearing in the pattern in alphabetical order are separated by a dash '`-`', then any character in the alphabetic range between these two characters will be matched. This is a more selective version of the '`?`' form.
`[^...]`	As above, but this time matches anything *not* in the enclosed range of characters '`...`'.
`\`	Treat the next character as a normal character rather than one of the special characters. This can be used to override the usual meaning of the '`*`', '`?`', '`[]`', and '`\`' characters. Note that the Atari ST, MSDOS, and OS/2 versions of HPACK use '`#`' instead of '`\`' since the command interpreter uses '`\`' in its pathnames.

The case-sensitivity when handling filenames depends on the operating system HPACK is being run under. On the Atari ST and under MSDOS and VMS, filenames are converted to uppercase, but are not case-sensitive; on the Amiga and Archimedes and under OS/2, filenames are left as is, but are also not case sensitive (so that "`FileName`" will match "`Filename`", "`FILENAME`", and "`filename`"); and under Unix, filenames are left as is, and are case sensitive, so that "`makefile`" and "`Makefile`" are treated as separate names.

These wildcards can be combined to create quite powerful expressions. For example, to match any file not beginning with an '`a`', '`b`', '`c`', or '`f`', and containing at least two characters, the required expression would be:

```
[^a-cf]?*
```

where the `[^a-cf]` would match anything but '`a`', '`b`', '`c`', or '`f`'; the '`?`' would match the second character required, and the '`*`' would match any remaining characters.

For example, if we have an archive whose contents are:

```
file1.txt  file2.txt  file3.txt  file4.txt  file5.txt
file6.txt  file7.txt  file8.txt  file9.txt  Index.txt
```

we could perform the following file matches on it:

Expression:	Matches files:				
`*`	file1.txt	file2.txt	file3.txt	file4.txt	file5.txt
	file6.txt	file7.txt	file8.txt	file9.txt	Index.txt
`F*`	file1.txt	file2.txt	file3.txt	file4.txt	file5.txt
	file6.txt	file7.txt	file8.txt	file9.txt	
`*4*`	file4.txt				
`FILE?.TXT`	file1.txt	file2.txt	file3.txt	file4.txt	file5.txt
	file6.txt	file7.txt	file8.txt	file9.txt	
`FILE[1-4].TXT`	file1.txt	file2.txt	file3.txt	file4.txt	
`FILE[^1-4].TXT`	file5.txt	file6.txt	file7.txt	file8.txt	file9.txt
`FILE[1-46].TXT`	file1.txt	file2.txt	file3.txt	file4.txt	file6.txt
`FILE[^13-5].TXT`	file2.txt	file6.txt	file7.txt	file8.txt	file9.txt

Finally, note that these wildcards may not perform quite like the standard wildcards used by the system's command interpreter. For example under GemDOS, MSDOS, and OS/2, to specify all files in a directory or archive, it is not necessary to use the usual sequence of "`*.*`". Instead simply typing "`*`" will match all the files in the archive (the DOS-like wildcards are in fact a bit of a hack — for example "`*abc.*`" will (incorrectly) match "`*.*`"). The extended wildcards act as true wildcards, so that "`FILE.*`" will not match "`FILE`" as it would under DOS, since there is a spurious "`.`" in the DOS wildcard matching. These extended wildcards can be useful when extracting files which cannot normally be handled by the command interpreter from archives, for example to extract the file "`Filename with spaces in it`" from the archive `ARCHIVE.HPK`, you would type:

```
HPACK X ARCHIVE FILENAME?WITH?SPACES?IN?IT
```

where the '?' wildcard will match the spaces. Alternatively you can just type:

```
HPACK X ARCHIVE FILENAME*
```

which will usually do the same thing with a lot less typing involved.

Another point is that these wildcards cannot be used to represent full regular expressions (the idea behind HPACK was to create an archiver, not a regular expression parser), so that, for example, '`*?`' will not work as an expression, and that they can only be used inside pathnames when the path is inside the archive (since the extended wildcards are built on top of the usual operating system ones (if they exist) the resulting system would run very slowly if they had to be simulated on an external file-system).

Using wildcards for pathnames inside archives is perfectly legal. When matching wildcards to pathnames, HPACK follows the standard convention of matching the directory contents if the directory name is given, or matching the directory itself when a wildcard matching it is given. Consider an archive with a subdirectory `DIR`, with both the archive root directory and the subdirectory containing files. Then the following wildcard matches could occur:

'`*`'	Match all files and directories in the root directory
'`D?R`'	Match the directory `DIR` (and possibly other files) in the root directory.
'`DIR`'	Match the contents of directory `DIR` (equivalent to '`DIR/*`')
'`*/*`'	Match all files and directories (equivalent to using '`*`' with the [`-r`]ecurse option).

Unlike most filesystems, HPACK has no notion of a 'current directory' within an archive, so all pathnames must be given relative to the archive root directory.

Finally, Unix users may have to either set the shell variable 'noglob' or quote the wildcard characters used to prevent them being expanded by the shell if it is required that HPACK handle wildcard expansion. One case in which this is necessary is when recursively adding files in subdirectories which match a certain filespec to an archive:

```
HPACK A -D TEST *.H
```

will cause the shell to expand the '*.H' filespec to match all the files in the current directory, and not match any files in subdirectories. In contrast:

```
set noglob
HPACK A -D TEST *.H
unset noglob
```

will cause HPACK to perform the wildcard expansion, and will correctly add files in subdirectories.

HPACK Commands

`A` Add files to an archive.

The archive given in the command-line is opened, or created if it doesn't already exist, and is given the extension '`.HPK`' if this is not explicitly specified. Then all files given in the list of filenames are added to to it.

Example: To add all files in the current directory to the archive `ARCHIVE.HPK` in the current directory:

```
HPACK A ARCHIVE
```

`X` Extract files from an archive.

All files are extracted from the archive given in the command line to the current directory. If the file already exists and no overwrite options are given (these are explained in the section "HPACK Options" below), you will be asked if you wish to overwrite it:

```
File already exists - overwrite [y/n/a]
```

'`Y`' will replace the file with the file from the archive, '`N`' will skip the extraction of this file, and '`A`' will process all files from this point as if a '`Y`' answer was given for each file. In some cases the name of the file extracted will need to be translated to allow for the naming conventions of different operating systems. In these cases the original name will be printed, followed by the name it will be extracted under.

On Apple systems data is divided into two sections in a file, a *data fork* and a *resource fork*. Under non-Apple operating systems only the data fork will be extracted from an archive: the resource fork (which only Apple systems can make any use of) will be skipped.

Example: To extract all files in `ARCHIVE.HPK` not beginning with an '`A`' to the current directory:

```
HPACK X ARCHIVE [^A]*
```

`V` Display a directory of files inside an archive.

If no view options are given (these are explained in the section "HPACK Options" below), a listing in the following format is produced:

```
HPACK - The multi-system archiver Version 0.78a0 (shareware version)
For Amiga, Archimedes, Atari ST, Macintosh, MSDOS, OS/2, and UNIX
Copyright (c) Peter Gutmann 1989 - 1992.  Release date: 1 Sept 1992

Archive is 'ARCHIVE.HPK'

System Length  Size  Ratio Date     Time   Name
------ ------  ----- ----- ----     ----   ----
 MSDOS  18537   6217  66%  04/05/90 11:32:48  README.TXT
 Atari   8301   3169  74%  27/12/90 21:02:39  LZSS.EXE
 UNIX   30061   9127  70%  24/11/89 16:52:57  compr.method.txt
 OS/2    8481   3282  61%  22/12/90 08:58:52  Comp.test
 Mac        0      0   0%  01/07/91 18:20:20  Fast LZ Decoder using B..
------ ------ ------ -----                     ----
        65380  21795  67%                     Total of 5 files

Done
```

The archive used is displayed, followed by a list of all the files within the archive. The fields for each file inside the archive are as follows:

System:	The system the file was archived under.
Length:	The length of the file when unarchived.
Size:	The size of the compressed file within the archive.
Ratio:	The ratio of the compressed file to the uncompressed file.
Date:	The creation date of the file within the archive.
Time:	The creation time of the file within the archive.
Name:	The name of the file within the archive. If the name is too long to fit on the screen then a '..' is appended to indicate this. In addition files may have a single character prepended to them: an asterisk '*' before the name indicates that this file has been encrypted, a dash '-' indicates that it includes authentication information, and a hash '#' indicates that it is both encrypted and includes authentication information. Note that the name within the archive may not be the same as the name given to the unarchived file since translation may be necessary to allow for the naming conventions of different operating systems. Note also that in the example above the file originating on the Macintosh has a length of zero bytes. This does not necessarily mean the total data size is zero bytes, since files originating on Apple systems have a second block of data called a resource fork which contains code and other program resources. These resources are only useful on Apple systems and are not shown as part of the total file size on other systems.

Finally the total length of all files within the archive, uncompressed and compressed, the overall compression ratio for the archive, and the number of files within the archive, is given. If any filespecs are given, only files which match those filespecs will be displayed. This is useful when you only want to extract certain files and would like to see how much space they will use.

If more than one archive is viewed, a grand total of the size and number of files in all the archives viewed is printed. The type of information printed may be controlled by using the view options (which are explained in the section "HPACK Options" below). The default is to print all files and directories in the root directory of an archive. If the root directory contains no files or directories matching the given filenames and/or wildcards, the error message "`No matching files in archive`" will be printed. To recursively display the contents of all subdirectories, either the [`-r`]ecurse subdirectories option or the wildcard string '`*/*`' should be used.

Example: To view the contents of any archives in the current directory

```
HPACK V *
```

P View files within an archive.

This option will output the contents of one or more files within an archive, with a prompt for more between files. The output can be redirected in the standard manner, for example:

```
HPACK P DATA CONTENTS.DOC | LPR
```

will send the file `CONTENTS.DOC` from within the archive `DATA.HPK` to the printer, and:

```
HPACK P DATA DESCRIPTION.DOC | MORE
```

will display the file `DESCRIPTION.DOC` with page breaks on your screen. Note that trying to display non-text files in this manner can be interesting, if not terribly productive.

Example: To display the file `README.TXT` from within the archive `ARCHIVE.HPK`:

```
HPACK P ARCHIVE README.TXT
```

T Test the integrity of an archive.

HPACK will display the usual messages about extracting data as it unpacks and tests the data, and if it thinks the data has been corrupted it will issue the following warning message:

```
Warning: File may be corrupted
```

Otherwise HPACK will display:

```
File tested OK
```

If, when all files have been tested, there were corrupted files, HPACK will print the total number of corrupted files:

```
Warning: n files were corrupt
```

where *n* is the number of corrupted files.

Note that this test is also performed automatically on extracting a file from an archive.

Example: To test the integrity of all MSDOS executable files in the archive `ARCHIVE.HPK`:

```
HPACK T ARCHIVE *.COM *.EXE
```

`D` Delete files from an archive.

All files given in the command-line are deleted from the given archive. If all files are moved out of an archive (leaving an empty archive), HPACK will delete the archive as well.

Example: To delete all Pascal program files (a commendable move) from the archive `ARCHIVE.HPK`:

```
HPACK D ARCHIVE *PAS
```

`F` Freshen files in an archive. The dates of all specified files in the archive are compared with the dates of the same files on disk. If the version on disk is more recent than the stored version, the stored version is replaced by the version on disk. If all files in the archive are uptodate, HPACK will display the message:

```
Archive is uptodate
```

and exit.

Example: To freshen all files in the archive `ARCHIVE.HPK`:

```
HPACK F ARCHIVE.HPK
```

`R` Replace files in an archive.

All given files in the archive are replaced by their counterparts on disk.

Example: To replace all the files which begin with the letters `A-G` in the archive `ARCHIVE.HPK` with their couterparts on disk:

```
HPACK R ARCHIVE [A-G]*
```

`U` Update files to an archive.

All specified files not already in the archive are added to the archive, and all files already in the archive are replaced if the version on disk is more recent than the version already in the archive.

Example: To update the archive `ARCHIVE.HPK` with all files from the directory `B:/DATA/JUNK`:

```
HPACK U ARCHIVE B:/DATA/JUNK
```

Note that the last four options need to create temporary work files on disk. Thus it is necessary to have at least twice as much free disk space to work with as the total length of the archive. All other options use only the archive file, so it is possible to use them with minimal free disk space.

HPACK Options

`-0` Store files without attempting any compression.

This option can be used to quickly add incompressible files to an archive or when breaking up a large file to move to another system via a multidisk archive when space isn't at a premium.

`-a` Store file attributes.

Many operating systems support two levels of files, those classed as 'normal', and those classed as 'hidden', 'invisible', 'system files', and so on (for example the Macintosh, MSDOS, and the Atari ST all have these file types). By default HPACK will only add normal files to an archive. However using the [`-a`]ttributes flag allows archiving of files with special attributes as well as full restoration of attributes when archives are extracted. The [`-a`]ttributes flag is also necessary under some operating systems to store extra information pertaining to a file, for example access control information, file/directory type information, icons, and so on.

HPACK will attempt to translate the attributes of files from different systems into equivalent file attributes for the local system. This has varying degrees of effectiveness: Many of the Apple IIgs, Atari ST, Macintosh, MSDOS, and OS/2 attributes are equivalent so a direct translation is possible; the Amiga, the Apple IIgs, the Archimedes, Unix, and VMS also have some of the read/write attributes in common, and have a rough equivalent of a read-only file attribute in these attributes; and QNX 2.x/3.x has no equivalent file attributes.

If the [`-d`]irectory path store option is used HPACK will also store directory attributes, and set the directory attributes to the stored values when creating the directories on extraction.

Example: To extract all files in the root directory of the archive `ARCHIVE.HPK`, restoring all possible attribute information:

```
HPACK X -F ARCHIVE
```

`-b` Specify a base pathname for files.

This option is followed by the pathname which HPACK will use as a base path for all files to be added to or extracted from an archive. This option is very convenient when adding a number of files in the same directory to an archive or when extracting an archive to a directory which is not the same as the current directory. It is also useful for handling archives which contain subdirectories. To add files to a subdirectory within HPACK, the base pathname is used to specify the path to the subdirectory, with the actual file pathname given being the path inside HPACK in which the file will be stored. In this case the argument given with the [`-b`]ase path option is the part of the pathname outside the archive, and the file pathname is the part of the pathname inside the archive.

Example: To extract all files in the archive `ARCHIVE.HPK` to the directory `D:/OUTPUT/DATA`:

```
HPACK X -BD:/OUTPUT/DATA ARCHIVE
```

Example: To add the files `FILE1`, `FILE2`, `TEXT`, and `PROGRAM.EXE`, all in the directory `D:/JUNK/DATA`, to the archive `ARCHIVE.HPK`:

```
HPACK A -BD:/JUNK/DATA ARCHIVE FILE[1-2] TEXT PROGRAM.EXE
```

Example: To add the files in `C:/JUNK/DATA/FILES` to the archive `ARCHIVE.HPK` inside the archive directory `DATA/FILES`:

```
HPACK A -BC:/JUNK ARCHIVE DATA/FILES/*
```

`-c` Encryption options (conventional and public-key encryption).

Encrypt/decrypt archive data using a selection of public and conventional-key encryption algorithms. Using this option with conventional-key encryption will prompt for a passphrase before any files are added, followed by a request to retype the passphrase for security if the data is being encrypted. This passphrase, which is not echoed to the screen, should be a minimum of eight and a maximum of eighty characters long, and may contain any combination of upper and lowercase letters, numbers, spaces, punctuation symbols, and control characters. If public-key encryption is used, the userID of the recipient of the data is specified as part of the encryption command. Although HPACK itself will support non-ASCII text strings, the userID's for public keys must be in ASCII format for compatibility with other programs. Be warned that forgetting or losing a conventional or public-key encryption key will present you with a fairly substantial exercise in cryptography.

By default HPACK will encrypt entire archives (rather than just the files in them), meaning that not even the archive directory can be read by someone who does not have the decryption password.

The encryption options are divided into two classes, those employing conventional-key encryption and those employing public-key encryption:

`-c` Use conventional-key encryption to encrypt the entire archive, the same as [-`c`]rypt [`a`]ll.

`-ci` Encrypt individual files using conventional-key encryption. The use of this option is not generally recommended as it is not quite as secure as the standard [`-c`]rypt [`a`]ll option. It is however useful when only a few files need to be encrypted, the rest being subject to constant change which makes encrypting them impractical.

`-ca` Encrypt entire archive. This will encrypt not only the files themselves but all additional data associated with them (attributes, icons, and so on), as well as the archive directory information. The only remaining accessible data is a small amount of archive identification information needed to allow HPACK to process the archive.

`-cs` Prompt for second password to access archive data. This allows the archive directory to be encrypted with one password, and files to be encrypted with another password. This option is useful if it is desirable to give a group of users access to the archive directory but not to the files themselves, since the contents of the archive can be made available with the first password, but a second password is required to access the files themselves.

`-cpi`*`userID`*
: Encrypt individual files using the public key which matches the given userID. The public-key encryption equivalent of the [`-c`]rypt option. This option is not recommended if more than a small number of files are present in an archive due to the amount of time needed to perform each public-key decryption calculation.

`-cpa`*`userID`*
: Encrypt the entire archive using the public key which matches the given userID. The public-key encryption equivalent of the [`-c`]rypt [`a`]ll option.

`-cps`*`userID`*
: Encrypt the archive with a secondary public key which matches the given userID. The public-key encryption equivalent of the [`-c`]rypt [`s`]econdary option.

When processing an encrypted archive, all that is necessary is to tell HPACK to handle encrypted data by specifying the [`-c`]rypt option. HPACK will determine the encryption type and prompt for passwords as necessary as it processes the archive. If the archive is public-key encrypted and the recipients secret key is protected by encryption, HPACK will prompt:

```
Please enter password for secret key (8..80 characters)
```

If an incorrect password is given, HPACK will warn:

```
Password is incorrect.
```

and allow the password to be reentered. Up to three attempts at the password are allowed before HPACK gives up.

When searching for the key corresponding to a given userID, HPACK will perform a case-insensitive match of the given userID against any part of the keys userID. This means that only a partial userID need be given on the command line, and HPACK will use the first key for which the partial userID matches. This makes specifying the userID easier since the entire ID need not be given, but also means care should be taken if there are several similar userID's (all of which may match a certain userID fragment) in a collection of keys.

If there is a chance that the userID is ambiguous, the key can also be specified by its keyID using the standard C programming language format for hexadecimal values namely a prefix of '`0x`' and then the value itself. The keyID is the 6-digit hexadecimal value displayed for the key. As with userID's, HPACK will perform a case-insensitive match against any part of the full keyID.

HPACK will also allow the public-key encryption of data for multiple recipients, meaning that a single encrypted archive can be sent to an arbitrarily large number of recipients. This allows archives to be distributed to working groups or via mailing lists without necessitating a seperate encrypted archive for each recipient. Note that there is a slightly increased risk involved in this process since the chain of intended recipients is only as strong as its weakest link — only one of the private keys needs to be compromised to render the encrypted data insecure.

Multiple recipients are currently specified as a comma-seperated list of standard userID's or keyID's. There are moves afoot to add a mailing-list capability to the public keyring format, which will be fully supported by HPACK if it eventuates.

Example: To encrypt the entire archive `ARCHIVE.HPK`, leaving only a small amount of identification information readable:

```
HPACK A -C ARCHIVE
```

Example: To encrypt all files to be added to the archive ARCHIVE.HPK:

```
HPACK A -CI ARCHIVE
```

Example: To encrypt the entire archive ARCHIVE.HPK with the archive directory readable by "TheMunsters" but the archive data itself only readable by "UncleFester" or "CousinItt":

```
HPACK A -CPATheMunsters -CPSUncleFester,CousinItt ARCHIVE
```

Example: To encrypt the file BASEBALL.BAT in the archive ARCHIVE.HPK with the key corresponding to the hexadecimal keyID A72F3B, taking advantage of the fact that HPACK will match any fragment of the ID:

```
HPACK A -CPI0x2F3 ARCHIVE BASEBALL.BAT
```

Note that the public-key decryption process can take a long time, especially when it must be performed multiple times (for example when the [-c]rypt [p]ublic-key [i]ndividual files option is used). HPACK will display the message:

```
One moment...
```

whenever it performs the calculations involved in public-key decryption. This process can often take significantly longer than one moment — on slower machines HPACK may appear to have ground to a halt as it performs the public-key decryption operation. Unless there is a special need for it, the use of conventional-key encryption is recommended.

One advantage of public-key encryption is that the key management is automatic — there is no need to enter passwords to perform the encryption and decryption operations. When passwords must be entered manually for conventional-key encryption, there is a chance that an incorrect password will be entered by mistake. If this happens for archive directories, HPACK will warn:

```
Warning: Archive directory corrupted, probably due to
         incorrect password. Continue (y/n)
```

Since the password was incorrect, the archive directory information has been decrypted incorrectly. The 'No' option should be selected and the password reentered.

If this happens for archived files, the files will not be extracted properly:

```
Warning: File may be corrupted
```

Again, the password should be reentered when HPACK is re-run.

-d Directory options.

This hideously complicated command has a large number of suboptions. The basic command creates directories inside the archive corresponding to the directories containing the file(s) being archived, in effect allowing you to archive entire directory trees.

HPACK will add directories traversed only if there are files contained in them. To add all directories regardless of whether they contain files or not, use the [-d]irectories [a]ll option (see below).

Example: To add all files in all directories, along with the directories themselves, in and below the current directory to the archive ARCHIVE.HPK:

```
HPACK A -RD ARCHIVE *
```

The extended directory options are are as follows:
<Note: Some of these aren't implemented yet: Feedback on whether this way of doing things is a good idea or not would be appreciated>.

-dm Mkdir: Create the given directories.
<Not yet implemented — if necessary this can be faked by using 'hpack a -da <path> <non-matching filespec>' where path is the new directory to add and the non-matching filespec ensures that only the new directory, but none of the files in it, are added>

-dr Rmdir: Delete the given directories.
<Not yet implemented>

-dv Mvdir: Move the given directory into the second given directory. Warning: Through injudicious use of this command it is possible to create circular directory references, or to cut off entire directory trees. Do not try to move directories into subdirectories of themselves! (Or should HPACK check for this??)
<Not yet implemented>

-da Store/extract all directories scanned, even if they contain no files. This option is useful for storing entire filesystems inside archives and restoring entire directory trees from archives.

-dn Do not create the directory inside the archive if it doesn't already exist. If an attempt is made to add a file to a nonexistent directory, HPACK will exit with the error message:

```
Path not found
```

-dc Create only immediate containing directory for a group of files, not the entire directory path. This is mainly for use by GUI versions of HPACK to allow extraction of individual directories, folders, or drawers (depending on what the host operating system calls them) of files.

Example: To delete the directory DATA/JUNK inside the archive ARCHIVE.HPK:

```
HPACK A -DR ARCHIVE DATA/JUNK
```

Note the use of the [A]dd command as a dummy command: In this case the main command is ignored and only the directory option is used (ICK!).

Example: To move all files and directories in and below DATA/JUNK in the archive ARCHIVE.HPK into the directory STUFF:

```
HPACK A -DV ARCHIVE DATA/JUNK STUFF
```

Example: To add all files and directories (even empty ones) in and below the current directory to the archive ARCHIVE.HPK:

```
HPACK A -RPDA ARCHIVE
```

-e Add/use error recovery information.

This option allows the adding of error-recovery information to the archive when adding files, or lets HPACK make use of error recovery information present in the archive. When adding files to an archive, HPACK will write extra information with the file data which may be used to recover the data if the main archive directory is seriously damaged. This recovery information will increase the amount of data stored for each file by about 20-25 bytes. Note that although HPACK will add this extra error recovery information, the current version will not make use of it.

Example: To add all text files in the directory `A:/DATA` to archive `ARCHIVE.HPK`, adding error recovery information for each file:

```
HPACK A -E ARCHIVE A:/DATA/*.TXT
```

Example: To extract the same files from the archive, making use of the error recovery information:

```
HPACK X -E ARCHIVE *.TXT
```

-f Force file move into/out of an archive.

This option can be used with the `[A]`dd, `[F]`reshen, `[R]`eplace, `[U]`pdate, and `[X]`tract commands to move the files instead of merely copying the data into or out of an archive. Note that when the move option is used in conjunction with encryption, HPACK will go to extreme lengths to destroy any traces of the file which is being moved. Caution is recommended when using move with encryption as forgetting the password or using the wrong public key will lead to the permanent loss of the encrypted data.

If all files are moved out of an archive (leaving an empty archive, HPACK will delete the archive as well.

Example: To move all text files from the archive `ARCHIVE.HPK` into the current directory:

```
HPACK X -J ARCHIVE *.TXT
```

-i Interactive mode (prompt for all actions).

This option works for all commands except `[V]`iew files (for which it would be pointless). Before the file is processed, HPACK will ask whether you wish to process this file; answering `'Y'` will handle the file, `'N'` will skip the file, and `'A'` will process all files from this point (in other words it will assume a `'Y'` answer for each file from this point).

Example: To add all files in the current directory to the archive `ARCHIVE.HPK`, prompting for each file before adding it:

```
HPACK A -I ARCHIVE *
```

`-k` Overwrite existing archive.

This only works with the `[A]`dd command. Normally, using the `[A]`dd command will add any new files to the end of an existing archive. Using the `[-k]`ill switch will erase the old archive and create a new one.

Example: To create the archive `ARCHIVE.HPK` on drive A:, overwriting it if it already exists, and add all files in the current directory to it:

```
HPACK A -K A:ARCHIVE
```

`-l` Security options (data authentication).

This option allows the encapsulation of either entire archives or individual files inside a secure envelope which it is computationally infeasible to break. This option allows anyone to later determine that the data has been both untampered with, and genuinely came from the source (corresponding to the userID) from which it claims to have originated.

When searching for the key corresponding to a given userID, HPACK will perform a case-insensitive match of the given userID against any part of the keys userID. This means that only a partial userID need be given on the command line, and HPACK will use the first key for which the partial userID matches. This makes specifying the userID easier since the entire ID need not be given, but also means care should be taken if there are several similar userID's (all of which may match a certain userID fragment) in a collection of keys.

If there is a chance that the userID is ambiguous, the key can also be specified by its keyID using the standard C programming language format for hexadecimal values namely a prefix of '`0x`' and then the value itself. The keyID is the 6-digit hexadecimal value displayed for the key. As with userID's, HPACK will perform a case-insensitive match against any part of the full keyID.

If the secret key is protected by encryption, HPACK will prompt:

```
Please enter password for secret key (8..80 characters)
```

If an incorrect password is given, HPACK will warn:

```
Password is incorrect.
```

and allow the password to be reentered. Up to three attempts at the password are allowed before HPACK gives up.

The security options are as follows:

`-l`*`userID`*
: Secure the entire archive with security information from a given userID.

`-li`*`userID`*
: Secure individual files rather than the archive as a whole, with security information for a given userID. The use of this option is not generally recommended since the generation of the security information for each file can consume a considerable amount of time, and since only the files themselves are secured it leaves the file attributes and directory information open to modification. It is however useful when only a few files need to be secured, the rest being subject to constant change which makes securing them impractical.

Example: To create the archive `ARCHIVE.HPK`, add all files in the current directory to it, and secure it on behalf of your cat:

```
HPACK A -LTHECAT ARCHIVE
```

`-m` Create a multipart archive.

Normally when HPACK runs out of disk space it will exit with an error message. When this option is used HPACK will instead prompt for another disk and continue the archive on the new disk. In this manner it is possible to spread an archive which would be too large for a single disk over several disks. Due to their rather special nature, multipart archives do not support the [`D`]elete, [`F`]reshen, [`R`]eplace, or [`U`]pdate commands (since, for example, deleting a file from the middle of an archive which stretches over 15 disks would be quite time-consuming). All other operations are supported however.

Example: To fully back up hard drive C: onto drive A:

```
HPACK A -KMARDA A:ARCHIVE C:/*
```

This will create the archive `ARCHIVE.HPK` on one or more disks in drive A: containing the complete contents of the drive C: (the options used are [`-k`]ill existing archive, [`-m`]ultipart archive, store file and directory [`-a`]ttributes, [`-r`]ecurse through all subdirectories, store [`-d`]irectories, [`a`]ll of them even empty ones). Once each disk has been completely filled, HPACK will prompt:

```
Please insert the next disk and press a key
```

followed by the message:

```
Continuing...
```

as it continues the archive on the next disk inserted. Note the use of the [`-k`]ill existing archive option to overwrite any existing archive of the same name which may already exist on the disk — if HPACK finds an archive of the given name already on the disk and the [`-k`]ill option is not specified, it will not, by default, overwrite it but will exit with an error message.

Multipart archives have a minimum size of around 500 bytes (roughly the size of a disk sector for many disk formats — storing archive parts of less than 500-odd bytes would be pointless). If a section of a multi-part archive is less than approximately 500 bytes HPACK will skip it and move it to the next (hopefully less full) disk after printing the following warning:

```
Warning: Archive section too short, skipping...
```

If a multipart archive is small enough to fit onto a single disk, HPACK will store the archive as a standard archive instead of a multipart one.

Example: To view the files in the previously created multipart archive:

```
HPACK V A:ARCHIVE
```

HPACK will automatically determine whether the archive is a multipart archive or not so the [-m]ultipart command is only necessary when creating the archive. Since HPACK stores its directory information at the end of the archive, only the last disk or disks of the archive must be read to obtain the archive directory. Initially HPACK will prompt:

```
Please insert the disk containing the last part of this archive and
    press a key.
```

If the wrong disk is inserted, HPACK will prompt:

```
This is part n of a multipart archive.
Please insert the disk containing the last part of this archive and
    press a key.
```

where *n* is the part number of this section of the archive.

Example: To extract the single file `LETTER.TXT` from the previously created archive:

```
HPACK X A:ARCHIVE LETTER.TXT
```

HPACK will then prompt for the disk which contains the file `LETTER.TXT`:

```
Please insert the disk containing part n of this archive and
    press a key.
```

where *n* is the part number of the archive section which contains `LETTER.TXT`.

Example: To fully restore the contents of a hard drive from the previously created archive:

```
HPACK X -ARDA A:ARCHIVE
```

HPACK will prompt for the last part of the archive as usual and then for each disk in turn as it extracts the files from the archive.

-o Overwrite on extraction options.

Normally when HPACK tries to extract a file which already exists, it will prompt for whether the existing file should be overwritten or not. With this switch it is possible to specify a default action to be taken. There are four possibilities:

- `-oa` Automatically overwrite [A]ll existing files on extraction.
- `-on` Automatically overwrite [N]one of the existing files on extraction.
- `-os` [S]mart overwrite. HPACK will change the extension of the file to be extracted to "`000`", and try to extract this file. If a file of this name already exists, the extension will be changed to "`001`" and so on until it is possible to extract the file. This option is very useful for files that have been archived on a system which allows filenames which are longer or more complex than those allowed by the local system and which due to the filename being truncated or translated end up with identical names. Note that the term "extension" can mean different things to different operating systems — HPACK will do its best to use the local equivalent.
- `-op` [P]rompt for new filename. HPACK will ask for a new filename and try to use that name, repeating until a non-conflicting filename is given.

Example: To extract all files from the archive ARCHIVE.HPK to the current directory, skipping any files that already exist:

```
HPACK X -ON ARCHIVE
```

`-r` Recurse through subdirectories.

HPACK will step through all subdirectories of the current directory, and add all files with names matching those given on the command line.

Example: To add all files in all directories in and below the current directory to the archive `ARCHIVE.HPK`

```
HPACK A -R ARCHIVE *
```

`-s` Run in stealth mode.

All messages except warnings and error messages are suppressed. HPACK will automatically turn on stealth mode if it detects it is running in the background on operating systems which support background operation.

Example: To add all files in the current directory to all archives in the current directory, without printing the usual progress reports to the screen:

```
HPACK A -S *
```

`-u` Unified compression mode.

In this mode HPACK will attempt to achieve increased compression by using unified compression across all files to be added. This is especially effective when many generally similar files or many small files are being added to an archive. The disadvantage of unified compression is that the archive cannot be updated or changed later, and that when extracting individual files there is a slight speed penalty as intervening files are skipped. Unified compression is ideal for storing data like Usenet news articles, collections of icons, and program source code, in which cases significant compression gains are usually seen.

Example: To all all files in `/USR/SPOOL/NEWS/COMP/COMPRESSION` to the archive `COMPRESS.HPK` using the unified compression mode:

```
HPACK A -U COMPRESS /USR/SPOOL/NEWS/COMP/COMPRESSION/*
```

`-v` View options.

Normally when the `[V]`iew command is used, HPACK will print all the directories and files in an archive. With this options it is possible to specify which parts of the archive are to be viewed. There are three possible options:

`-vf` Display only matching files.

`-vd` Display only matching directories.

`-vs` Sort files. Normally files are displayed in the order in which they are stored within the archive. Using this option will sort the files by name before displaying them.

Example: To print all the files, but not any directories or subdirectories, in the archive ARCHIVE.HPK:

```
HPACK V -VF ARCHIVE.HPK
```

-w Treat files as archive comments.

This makes the [A]dd, [D]elete, [F]reshen, [R]eplace, and [U]pdate commands work for archive comments instead of normal files. Archive comments are files which are displayed when the [V]iew archive command is used, and may contain text describing the contents of the archive, ANSI extended characters, graphics, digitised sound, even full motion video with 16-bit stereo sound if desired. Most CLI versions of HPACK will currently only handle the displaying of plain text, although some will handle ANSI-text type archive comments as well. Comments can be added, deleted, replaced, and so on just like normal files. Subdirectories can contain their own comments, and each directory can contain multiple comment files. In addition comment filenames are not treated as normal archive filenames, so a directory can contain a comment file and a normal archived file of the same name. Possible archive comment types are:

-w Plain text comment. This can be entered in free-form since HPACK will automatically word wrap the text to fit the screen size. Plain text comments may also contain formatting commands which control the way the text is displayed. For more information on formatting the text comments see the section "HPACK Archive Comments" below.

-wa ANSI text comment. This type of comment can contain the extended 8-bit character set used by IBM PC's, as well as ANSI escape codes. No reformatting of any type is done on ANSI comments. Note that the plain text comment type is preferred, since many systems cannot display the extended ANSI character set or interpret ANSI escape codes. Also note that if an ANSI comment is stored as plain text HPACK will quietly delete all extended characters and ANSI escape codes when it displays the comment, to make it conform to the 7-bit ASCII character set.

-wg GIF™ format graphics comment. Display of this comment format is generally only suported on systems running graphics-based user interfaces.

-wj JPEG format graphics comment. Display of this comment format is generally only supported on systems running graphics-based user interfaces.

-wm MPEG format graphics comment. Display of this comment format is generally only supported on high-end systems running graphics-based user interfaces.

Example: To view all comment-type files in the archive ARCHIVE.HPK as files instead of displaying them as comments:

```
HPACK V -W ARCHIVE.HPK
```

`-x` Text file translate options.

Different systems store text files in different formats, for example on the Amiga, the Archimedes, and under Unix, these are stored in ASCII format with a linefeed at the end of each line; on the Atari ST and under MSDOS and OS/2, there are carriage return/linefeed pairs at the end of each line; and on the Macintosh, there are carriage returns at the end of each line. Some systems store text in a non-ASCII format altogether, for example IBM systems which use the EBCDIC character set, and Prime systems which use their own bizarre text encoding technique. Using this option it is possible to specify translation of different character systems and end-of-line markers to the one used by the local system. In most cases the `[-x]`late option will suffice, however it is possible to override the translation using the following options:

`-x` Smart translate. HPACK will attempt to translate all files it recognises as text files from the textfile format used on the system the file was archived on to the textfile format used on the system the file is being extracted on. In most cases this option will be the only one necessary.

`-xr` Treat carriage return (ASCII 13) as end-of-line marker. The Macintosh stores text this way.

`-xl` Treat linefeed (ASCII 10) as end-of-line marker. Text on the Amiga, the Archimedes, and UNIX systems is stored in this manner.

`-xc` Treat carriage return/linefeed pairs as end-of-line marker. The Atari ST, MSDOS and OS/2 store text files like this.

`-xx`*`nn`* Treat the one- or two-character hexadecimal value *n* or *nn* as the character to use as the end-of-line delimiter. For example when moving textfiles from a QNX 2.x/3.x system (which uses the RS character, ASCII 30 or `1E` hexadecimal), the option would be `-xx1e`.

`-xe` Translate the character set from EBCDIC to ASCII. Text on IBM systems is stored this way.

`-xp` Translate the character encoding from that used on Prime systems to ASCII.

`-xa` Translate from ASCII to ASCII.

The textfile translation works by first translating entire character sets (either ASCII, Prime ASCII or EBCDIC) if necessary, and then translating the end-of-line characters, depending on the options specified.

The end-of-line translation options are only available where they would make sense (for example the ability to translate linefeed to linefeed isn't particularly useful). The possible translation options are shown below, with LF being the linefeed character, CR being carriage return, and CRLF being carriage return/linefeed pairs:

To From	Atari ST MSDOS OS/2	Amiga Archimedes Unix	Macintosh
Atari ST MSDOS OS/2		`-xc` CRLF -> LF	`-xc` CRLF -> CR
Amiga Archimedes Unix	`-xl` LF -> CRLF		`-xl` LF -> CR
Macintosh	`-xr` CR -> CRLF	`-xr` CR->CRLF	

Example: To extract all files with the extension `.TXT` from the archive `ARCHIVE.HPK`, translating linefeed characters into whatever end-of-line character the local system uses:

```
HPACK X -XL ARCHIVE *.TXT
```

Example: To extract all files with the extension `.TXT` from the archive `ARCHIVE.HPK`, translating the files from EBCDIC to ASCII, and translating all carriage return characters to the end-of-line character used by the local system:

```
HPACK X -XEXR ARCHIVE *.TXT
```

`-z` Extended options.

These are highly system-specific and in general will only be present on one particular version of HPACK. They either support the storing/extraction of system-specific information in archives, or the special handling of data which has been archived on another system. The recognised options are:

`-zinvert`

Archimedes version only. ADFS doesn't support extensions to filenames since dots are used as directory delimiters, which can lead to problems when moving, for example, source code files ending with the traditional `.c` and `.h` to the Archimedes. Many programs allow a workaround where the file `foo.c` is accessed as `c.foo`. This option allows this form of inversion of the filename/directory structure so that `*.c` and `*.h` would be extracted as `c.*` and `h.*`. This command usually invokes an intense sense of disbelief in non-Archimedes owners.

Incidentally, non-Archimedes owners may wonder what the Archimedes uses in place of a '`.`' in filenames. It uses a '`/`', of course.

<Not yet implemented — will be implemented when HPACK-internal mkdir() and mv() calls exist>

`-zlower`

Amiga, Archimedes, OS/2, Macintosh, and Unix versions only. Force all file and directory names to lowercase. Some systems store file and directory names in uppercase only. Using this option all names will be converted to lowercase before any operations (such as [`V`]iew archive, [`X`]tract from archive, and so on), are performed on them. When processing Atari ST, MSDOS, and some OS/2 archives, the use of the [`-zlower`] option is recommended.

`-znoumask`
: Unix version only. Normally the setting of the umask environment variable affects the attribute bits of any files and directories HPACK creates. This option overrides the umask setting and uses the attributes stored within the archive. Since these attributes can be set to allow outsiders read/write access to files and directories which would normally be off-limits to them, this option should be used with care.

`-zrsx` VMS version only. Normally VMS files can have 39 characters of filename and 39 characters of extension, or type. This can lead to strange-looking filenames when they are truncated from operating systems which allow longer or more flexible names. Using this option truncates filenames to an RSX-11 compatible format, which results in more traditional-looking names with 9 characters of filename and three characters of file type, and directory names with 9 characters of filename.

`-zs` MSDOS version only. Check all files for `[s]`afe extraction. MSDOS has a serious problem in that when a file with the same name as a device driver is extracted (for example '`CON`', '`COM1`', or '`LPT1`') it will force the contents of that file into the device driver. The damage can be minor for devices like '`CLOCK$`' (it will at a minimum mangle the system date and time, perhaps scramble the CMOS ram, or cause the system to hang — even the changing of system dates can cause problems on a system running a computer bulletin board which relies on correct timestamps), all the way through to very serious for devices like '`SMARTAAR`' (it may corrupt the disk cache and therefore corrupt the drives being cached).

The `[s]`afe option will check each file before extracting it, and if it corresponds to a device driver will print the warning:

```
File corresponds to device driver - skipping
```

and move on to the next file. The only way in which HPACK will allow a file of this type to be extracted is by using the `[-o]`verwrite `[p]`rompt option, in which you will be prompted for a new filename to extract under. Even `[-o]`verwrite `[s]`mart is unsafe since the peculiar handling of device drivers by DOS makes the automatic substitution of a new filename very difficult.

Filenames which can cause these problems are virtually unheard-of — they would have to be created deliberately by a malicious user, in which case there are few limits on the potential damage they can cause.

`-ztype`*`type-association`*
: Archimedes, Apple IIgs, and Macintosh versions only. Some operating systems store file type information for each file. HPACK will, when extracting files, try to determine the type information for each file and set it correctly. However in some cases no type information can be determined, or the determined information may be incorrect. This option can be used to set type information for a file, or to override HPACK's internal type-determining rules.

Type information is given as a type-association, associating a file extension with whatever type information the OS requires, and is given in the form:

```
.<extension>=<type info>{,<type info>}
```

For example to associate the Macintosh ThinkC source file type with the extension "`.c`", the type argument would be "`TEXT,KAHL`", corresponding to the ThinkC file type and creator type.

Example: To associate the "`.c`" extension on the Macintosh with the ThinkC source file type:

```
-ztype.c=TEXT,KAHL
```

Example: To associate the "`.gif`" extension on the Archimedes with the GIF file type:

```
-ztype.gif=693
```

The types given above are actually already part of HPACK's default rule set for type associations, but can be overridden with the use of the [`-ztype`] option if desired.

HPACK Script Files

Whenever HPACK encounters a filename beginning with an '@' character it treats the rest of the filename as a file containing a list of files to process combined with commands which control the operation of HPACK. Script files have the following format:

- Leading spaces/tabs (whitespace) are ignored.
- Lines with a '#' as the first non-whitespace character followed by at least one whitespace character are treated as comment lines. For compatibility with future versions of HPACK which will include scripting there should always be at least one whitespace character after the '#'. If HPACK finds a non-whitespace character following a '#', it will try and interpret it as a script command. If it cannot interpret the command, it will warn:

  ```
  Warning: Unknown script command
  ```

- All other lines are treated as filenames for HPACK to process.
- Lines may be terminated by either linefeeds, carriage returns, or carriage return/linefeed pairs, HPACK isn't particular.

A sample script is:

```
# Sample script file

# Save latest work
/users/peterg/hpack/src/*.c
/users/peterg/hpack/src/*.h
/users/peterg/hpack/src/asm/*.asm

# Save correspondence
/usr/spool/mail/p*
```

Note that the script files themselves are not subject to being overridden by the base path specified with the [-b]ase path option (see "HPACK Options" above). However filenames given within the script file will be overridden by the base path.

Example: To freshen the archive `ARCHIVE.HPK` with all the files in the files matching `SCRIPT*` as well as `MAIL.TXT` and `COMPMAIL.TXT`:

```
HPACK F ARCHIVE MAIL.TXT @SCRIPT* COMPMAIL.TXT
```

HPACK Archive Integrity

HPACK checks the integrity of files stored within an archive by encoding a continuous checksum as part of each file when it is archived, and decoding it as the file is extracted. As soon as an error is encountered, HPACK will skip over the rest of the file and move on to the next one. If the file checksums differ then it may be reasonably assumed that there is an error in the extracted data. This is important: The entire physical universe, including HPACK itself, may one day collapse back into an infinitely small space. Should another universe subsequently re-emerge, the integrity of HPACK archives in that universe cannot be guaranteed.

HPACK also checksums the archive directory. If an error is found in the directory information, the message:

```
Warning: Archive directory corrupted. Continue (y/n)
```

will be displayed. Hitting 'N' at this point will abort any attempts to process the archive, hitting 'Y' will process the (damaged) archive. HPACK will attempt some error recovery in this case (for example files and directories which seem to be in impossible directories will be moved into the root directory), and some files may be able to be recovered. If error recovery information (specified with the [-e]rror recovery option (see "HPACK Options" above)) is present, chances of recovering data from an archive with a corrupted directory are greatly enhanced.

HPACK Archive/Data Authentication

Data authentication facilities in an archiver should provide the following features:

1. Sender authenticity. The data could only have come from the source which it is supposed to have come from.

2. Data integrity measures. Any attempt to tamper with the data should be recognised and reported by the authentication system.

3. Non-repudiation of origin. The originator of the data cannot later disclaim responsibility for it.

HPACK includes provisions for authenticating archived data by adding a unique digital signature to either entire archives or individual files within an archive. This works exactly like a normal signature on a piece of paper, proving that the sender was the true originator of the file or archive. Forgery of a digital signature of this sort is computationally infeasible, and once the data has been signed the sender cannot later disavow his or her signature. In addition a cryptographic checksum of the data or archive is made and included as part of the signature to allow detection of any attempts to tamper with the data. Like forging a signature, defeating the checksum is computationally infeasible.

When the data in an archive with authentication information present is extracted or tested, an authentication check is performed for the entire archive before it is processed. Similarly, an authentication check is performed on files before they are extracted if there is authentication information present.

In order for the authenticity information generation and checking to be possible, two key files are required. To secure archives, the file `SECRING.PGP`, which contains the secret key of the person securing the archive, is required. To check secured archives, the file `PUBRING.PGP`, which contains the public key of the person or organization who signed the file or archive is needed. These key files are generated and managed by version 2.0 or higher of Philip Zimmermanns excellent PGP encryption package. If you don't have a copy of PGP, it is recommended you obtain one. PGP is freely available on many archive sites, computer bulletin boards, and other systems. PGP is also required to handle any key management, and for authentication of the keys themselves. The PGP documentation and other portions of this document is recommended reading for those wanting more information on the background of PGP and the security techniques used by PGP and HPACK.

If PEM (Privacy-Enhanced Mail) ever becomes a reality, HPACK will support PEM/PKCS format keys as well as PGP ones. This will allow data to be encrypted and decrypted inside the US with PEM keys and outside the US with PGP keys, allowing, for the first time, the easy transfer of public-key encrypted data to and from the US.

Before HPACK can use it, the secret keyring `SECRING.PGP` must have a small amount of extra information added to it. Details on this are given in the section "Using PGP Secret Keyrings" below.

Like PGP, HPACK will either search for the key files in the current directory or use the environment variable `PGPPATH` to look for the file. The `PGPPATH` variable should contain the path to the `PUBRING.PGP` and `SECRING.PGP` files, so for example if they were in the directory `/BIN/PGP`, then `PGPPATH` should be set with:

```
   set PGPPATH=/bin/pgp                    (MSDOS, OS/2)
```

or

```
setenv PGPPATH ~/bin/pgp             (Unix csh)
```

or

```
PGPPATH=bin/pgp; export PGPPATH      (Unix sh)
```

If the `PGPPATH` directory exists and the key files are present in both the current and the `PGPPATH` directories, the keyring in the `PGPPATH` directory will be searched first, and if no matching key is found the local keyring will be checked. This means that by default the master keyring is always used, overriding any local keys if necessary.

If the data in the archive, or the archive itself, is to be checked, the authentication check will be performed using the key found in `PUBRING.PGP`. If the authentication check succeeds, the message:

```
Good signature from name.
Signature made on date.
```

will be displayed, with *name* being the name of the person or organization who made the signature, and *date* being the date the signature was made (on some systems the signature time may be off by a few hours if the system doesn't adjust for different time zones properly). If the check fails, the message:

```
Warning: Bad signature, doesn't match file contents.
Bad signature from name.
Signature made on date.
```

will be printed. If the archive is a multipart archive (for which the authentication check can take some time since an archive stretching over several disks needs to be processed), the message:

```
Verify multipart archive authenticity [y/n]
```

is displayed. '`Y`' will check the authenticity of the archive, '`N`' will skip the authenticity check and continue with processing the archive.

If the data in the archive, or the archive itself, is to be secured, the authentication information will be generated using the key found in `SECRING.PGP`.

The signature scheme used is the RSA public key cryptosystem. This scheme involves the manipulation of very large numbers, which can be quite time-consuming on slow systems, where the signature generation and checking can take several minutes (for this reason the use of authentication for entire archives rather than individual files in an archive is recommended).

Unlike encryption software, authentication code is not export-restricted from the US. The U.S. Code of Federal Regulations, Title 22 which includes sub-chapter M, "International Traffic in Arms Regulations" (ITAR), makes an exception for software that can only be used for authentication purposes and cannot not be used for general-purpose encipherment and decipherment, as is the case for the data authentication code in HPACK.

HPACK Archive/Data Encryption

HPACK allows data stored in an archive to be encrypted with a variety of public or conventional-key encryption algorithms. Encryption of either entire archives or individual files is possible, as well as the use of multiple keys to encrypt different sections of an archive (though the use of public-key encryption is preferred for this since its inherent automatic key management greatly simplifies handling the encrypted data).

When chosing an encryption key for a conventional-key algorithm, the following guidelines should be observed:

- HPACK allows keys of up to 80 characters in length. These keys can contain letters, numbers, spaces, and punctuation. This fact should be made use of to the fullest, with preferred passwords being entire phrases rather than individual words. There exist programs designed to allow high-speed password cracking of conventional-key encryption algorithms which can, in a matter of hours (sometimes minutes, even seconds in the case of very weak algorithms), attempt to use the contents of an average large dictionary as sample passwords. Most passwords composed of single words can be broken with ease in this manner, even in the case of algorithms like the MDC one which is used by HPACK, which has been specially designed to be resistant to this form of attack (doing a brute-force search of all 8-letter passwords (the minimum length allowed by HPACK) assuming a worst-case situation in which the password contains lowercase letters only, can be accomplished in 662 years on a fast CPU (Sparcstation IPX) if the attacker knows the contents of the encrypted file(s) in advance — or about 8 months on a network of 1000 of these machines. Using an intelligent dictionary-based cracking program will reduce this time significantly).

- Simple modifications to passwords should not be trusted. Capitalizing some letters, spelling the word backwards, adding one or two digits to the end, and so on, only present a slightly more difficult challenge to the average password-cracker than plain unadorned passwords.

- Probably the most difficult passwords to crack in this manner are ones comprising phrases or sentences, since instead of searching a small body of text like the contents of a dictionary, the cracker must search a much larger corpus of data, namely all possible phrases in the language being used. Needless to say, the use of common phrases should be avoided, since these will be an obvious target for crackers.

As is the case when using public-key encryption for data/archive authentication, the secret keyring `SECRING.PGP` will need to have a small quantity of extra information added to it, for which details are given in the section "Using PGP Secret Keyrings" below.

Authentication/Encryption Details

HPACK allows files to be encrypted using a variety of either public or conventional cryptosystems. At the moment only the RSA public-key cryptosystem is used for data authentication, and the MDC conventional-key and RSA public-key cryptosystems are used for file or archive encryption. The RSA key format used in HPACK is compatible with version 2.0 or higher of Philip Zimmermann's excellent PGP encryption package.

In public-key encryption, each user choses a pair of keys, a public key (which as the name suggests is made available publicly), and a private key which only the user knows. This allows a complete stranger to use the public key to encrypt a message to the user which only the user can decrypt, unlike conventional encryption which requires that the encryption key be first somehow communicated to the user.

HPACK uses the RSA Data Security Inc. MD5 message digest algorithm to generate a unique 'fingerprint' of the data to be authenticated. This fingerprint consists of a cryptographically strong 128-bit one-way hash value which it is computationally infeasible to invert. This is a bit like a CRC, but unlike a CRC it is *very* difficult for an attacker to bypass: Not only is the MD5 function specifically designed for this purpose (which the CRC function is not), but it is also generally agreed that a message digest of this sort should be a minimum of 128 bits long: A 32-bit CRC will take around 65,000 attempts to defeat using a so-called 'birthday attack', a 128-bit message digest will take in the order of 20,000,000,000,000,000,000 attempts to break (in fact CRC's are even easier to defeat than that — it is a very simple matter to generate a message with an arbitrary CRC value, making CRC's worthless for detecting deliberate manipulation of data). MD5 has so far resisted attack, although a much-reduced form of MD5 was broken at Eurocrypt '92 in 2^{13} attempts with a chosen plaintext attack using differential cryptanalysis.

This message digest is then signed using the RSA public-key encryption algorithm, with the option of using a commercial-grade (512 bits or 155 digits) or military-grade (1024 bits or 310 digits) key (HPACK will in fact use keys of any length up to 1200 bits or 360 digits, the two values given above are simply the default key lengths used by PGP 2.0). The exact size of the key to use depends on how long the secret must be kept secure, and how large an amount of resources your opponent is prepared to commit to breaking (factoring) the key. In "Public-Key Cryptography, State of the Art and Future Directions", a report from a workshop involving the world's leading experts in the field, the authors state:

> "For most applications a modulus size of 1024 bits for RSA should achieve a sufficient level of security for 'tactical' secrets for the next ten years. This is for long term secrecy purposes, for short term authenticity purposes 512 bits might suffice in this century".

RSA Data Security's frequently-asked-question list contains the statement:

> "Rivest estimates that a 512-bit modulus, currently the most common modulus length, can be factored with an $8.2 million effort today, less in the future. Those with extremely valuable data (or large potential damage from digital forgery) should use a modulus of, say 700 or 800 bits in length. A certifying authority should use a modulus of 1000 bits or more, because the validity of many other key pairs depends on the security of one central key."

The $8.2 million figure is actually somewhat optimistic, in reality it should be somewhat more difficult than that. For an example from real life, in 1977 RSA Data Security published a 129-digit (or 430-bit) number which is a product of two primes, and offered a $100 prize to the first person to factor it. In spite of fifteen years of work on it, noone has done so (at least not publicly).

Finally, the US government allows export of RSA encryption code provided the key size is limited to less than 512 bits. You are left to draw your own conclusions from this fact.

The encryption routine used by HPACK employs the MDC algorithm run in cipher feedback mode, a 128-bit block cipher derived from the MD5 message digest algorithm with a key size of up to 2048 bits (though HPACK limits this to 80 ASCII characters or around 560 bits of effective key information). This algorithm has been designed to make encryption relatively fast and brute-force attacks slow and painful. Once the initial password management is done, en/decryption proceeds at a fairly rapid pace, however if the password is changed constantly (as it would be for a brute-force attack) a lot of time is spent in password management after each change. MDC has so far not been seriouly attacked, and it is not known whether generalizing the MD5 attack to MDC is possible.

Due to the layout of an HPACK archive all encrypted data blocks begin with raw compressed data, greatly reducing the chances of a so-called known plaintext attack (in which the attacker knows, or can guess, some of the unencrypted data). HPACK overwrites any sensitive data in memory once the encryption/decryption/authentication process has completed, and contains extensive error trapping and handling to ensure that even if a serious error were to occur, the program stack and data areas would be wiped on exit.

The code for the encryption and authentication routines used in HPACK is freely available in source form (see the next section, "Verification of HPACK's Encryption/Authentication"). In this way it is possible to compile the code and independantly verify that HPACK is indeed using the correct algorithms and encryption/authentication techniques, and thus eliminate any fears of trapdoors hidden in the code/encrypted data.

Verification of HPACK's Encryption/ Authentication

The encryption and authentication code used by HPACK can be freely examined by anyone wishing to reassure themselves of its security. The MD5 message digest algorithm is described in the following source:

Internet RFC 1321, "The MD5 Message-Digest Algorithm", Internet Activities Board, 1992. Obtainable from `ftp.nisc.sri.com` in the `rfc` directory.

The mode of operation of the MDC cipher is described in the following federal and international standards:

National Bureau of Standards FIPS publication 81: "DES Modes of Operation", December 1980.

ISO/IEC 10116:1991 "Information technology — Modes of operations for an n-bit block cipher algorithm", 1991.

ISO 10126-2:1991 "Banking — Procedures for message encipherment (wholesale) — Part 2", 1991

The RSA algorithm is described in many texts, among them:

Brassard, G. "Modern Cryptology", Lecture Notes in Computer Science vol.325, 1988.

Rivest, R., Shamir, A., and Adleman, L. "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM* **Vol.21**, **No.2** (February 1978).

RSA Data Security Inc. "PKCS #1: RSA Encryption Standard", June 1991, Version 1.4.

It is also a part of the following standards:

AS 2805.6.5.3 "Electronic Funds Transfer — Key Management", 1990.

ISO 9796:1988 "Information technology, security techniques — Digital signature scheme giving message recovery", 1988

RSA Data Security Inc. "PKCS #1: RSA Encryption Standard", June 1991, Version 1.4.

In all cases the above algorithms can be derived from the relevant standards, and all code used in HPACK checked against official implementations for accuracy.

An Analysis of HPACK Encryption Security

Much has been made recently of the dangerously insecure encryption methods used by many archivers. HPACK goes to great lengths to try and make breaking of its encryption system as difficult as possible. An analysis of possible weak points in the encryption is given below:

Encryption of individual files:

Slow and reasonably safe. Since a different 64-bit initialisation vector is used to rekey the MDC algorithm for each file, performing a brute-force attack on a collection of files would involve rekeying the algorithm for each file being attacked. Very difficult to attack unless the plaintext is known.

Encryption of the entire archive:

Faster than encrypting individual files, and more secure for the data portion of the archive since only the start of the first file is available to be attacked. However the encryption of the directory information provides partially-known plaintext in the form of the directory headers. The information content is probably enough to allow at least a preliminary form of automatic checking.

Encryption of the entire archive with a secondary password:

About the same speed as encrypting the entire archive, and the most secure of the encryption schemes. Even if the encryption for the directory is broken, the main data is still protected by a second password, and again only the start of the first file is available to be attacked.

There are two possible methods of attack, either a known-plaintext attack on the archive data, or a partially-known-plaintext attack on the directory data. If the uncompressed, unencrypted contents of the archive are known, HPACK can be used to compress them without encryption and provide plaintext which can be used in a brute-force or dictionary-based attack. Similary, the archive directory contains a relatively fixed format which can be parsed as part of a brute-force attack to narrow down the search range considerably.

Using public-key encryption offers more security against dictionary-based cracking programs since the hybrid method employed by HPACK uses a 128-bit binary MDC key encrypted with an RSA public key. Breaking this system would entail a brute-force search on the entire 128-bit key space, a total of 1.7 x 10^{38} keys assuming a match is found after half the keys have been scanned.

Using PGP Secret Keyrings

Like PGP, HPACK can store its secret keys in an encrypted format to protect them from prying eyes. However the encryption algorithm HPACK uses is somewhat different to the one use in PGP due to patent restrictions. This means that encrypted PGP keyrings need to have a small amount of extra information added to them before HPACK can use them. The HPACK distribution includes a utility called KEYCVT which performs this task. KEYCVT is run as follows:

```
KEYCVT <input keyfile> <output keyfile>
```

The input and output keyfiles are the PGP secret keyring file, and in normal usage would be the same file. KEYCVT will either create a new file to copy the extra information to, or will append it to the existing key file.

The source keyring may contain one or more secret keys. For each key in the file, KEYCVT will display its key information and ask:

```
Add information for this key so HPACK can use it (y/n)
```

A "No" response will cause KEYCVT to print:

```
Skipping key...
```

and move on to the next key. A "Yes" response will cause KEYCVT to try to add the extra information needed by HPACK. If the key is encrypted, KEYCVT will ask for the decryption password:

```
Please enter password for this private key
Password:
```

If the incorrect password is entered, KEYCVT will warn:

```
Incorrect checksum, possibly due to incorrect password
```

and allow it to be reentered. Up to three attempts at the password are allowed before KEYCVT gives up. When the key has been read in, KEYCVT will add the extra information encrypted with the same password as was used for the decryption. This is the password which HPACK, like PGP, will use to decrypt the secret key.

Finally, when KEYCVT has finished processing the key file, it will ask:

```
Finished processing keys. Add new key information to output keyring (y/n)
```

A "Yes" response will add the new key information to the output key file. If no keys were converted or a "No" response is given, KEYCVT will display:

```
No keys converted, output file left unchanged.
```

and exit.

Archive Comments

To make plain text archive comments portable across multiple operating systems it is recommended that they not be hardcoded for some fixed screen size (such as 80 x 24) but instead use HPACK's built-in formatting commands as part of the text. HPACK will then format the text of the comment according to these commands when displaying it. The formatting commands available are:

Text type control:

`.no`	normal text	`.iv`	inverse text
`.ul`	underline text	`.fl`	flashing text
`.bo`	boldface text	`.it`	italics

Text display control:

`.ce`	center following text	`.fi`	justify following text
`.nf`	stop justifying text		

Miscellaneous control options:

`.bp`	page break (new page)	`.br`	line break
`.in xx`	left margin at xx	`.rm xx`	right margin at (max - xx)
`.ti`	temporary indent	`.ew`	end woof

Note that some of the text types may not be available on some systems. By default HPACK will wrap words at the end of the screen. Justification of the output text (the right margins of the text being made even) can be turned on with the `.fi` command and off with the `.nf` command. When a `.nf` command is encountered, the current text is printed before the `.nf` command takes effect. This is known as a break, and can be explicitly forced using the `.br` command. Page breaks can be forced with the `.bp` command. The left and right margins are controlled by the `.in` and `.rm` commands respectively. The `.ti` command can be used to produce a temporary indent which sets the indent position for one line only. This can be used to produce hanging indents:

```
.in 5
.ti -5
```

will produce output of the form:

```
The .in, .rm, and .ti commands can also take relative values
     instead of absolute ones; thus they can be used to specify a
     change in the current value.  This is illustrated in the above
     example, where the left margin is set to 5, and a temporary
     indent of 5 places to the left of the margin is set for the
     next line.  If the value is given merely as is, it is treated
     as an absolute value; if it is given as +value or -value it is
     treated as a change in the existing value instead of an
     absolute setting.
```

An example of text with format control codes is:

```
.ce
This is a sample of formatted text
.br
.ti 2
.rm 5
.fi
```

```
This sample of text shows how to use HPACK's build-in formatting
commands.  Since the text is not preformatted but is reformatted by
HPACK according to the width of the screen the text is being
displayed on, there are no problems with lines going off the
screen, text only covering half the screen, or the system being
unable to display half the character set in use.
.br
.br
.ce
--------
```

After formatting for an 80-column screen this comes out as:

```
                    This is a sample of formatted text

  This sample of text shows how to use HPACK's build-in  formatting
commands.  Since the text is not preformatted but is reformatted by
HPACK  according  to  the  width  of  the  screen the text is being
displayed on, there are  no  problems  with  lines  going  off  the
screen,  text  only  covering  half the screen, or the system being
unable to display half the character set in use.

                                --------
```

After formatting for a 40-column screen this comes out as:

```
 This is a sample of formatted
             text

  This  sample  of  text  shows
how  to  use  HPACK's  build-in
formatting commands.  Since the
text is  not  preformatted  but
is    reformatted    by    HPACK
according to the width  of  the
screen   the   text   is  being
displayed  on,  there  are   no
problems  with  lines going off
the screen, text only  covering
half  the screen, or the system
being unable  to  display  half
the character set in use.

            --------
```

HPACK Return Codes

If an error occurs during archive processing, HPACK will return one of four sets of return values. These are:

0	No error
1-99	Severe error
100-199	Error
201-255	Warning

The error types can thus be quickly grouped into one of three classes and handled appropriately. The actual values of these error return values are:

Type	Value	Description
`EXIT_OK`	0	No error
`EXIT_INT_ERR`	1	Internal error
`EXIT_NO_MEM`	2	Out of memory
`EXIT_NO_DISK`	3	Out of disk space
`EXIT_NO_ARCH`	4	Can't open archive file
`EXIT_NO_SCRIPT`	5	Can't open script file
`EXIT_NO_PATH`	6	Can't find path
`EXIT_NO_BASE`	7	Can't access base directory
`EXIT_NO_MKDIR`	8	Can't create directory
`EXIT_BREAK`	9	User interrupt
`EXIT_FILE_ERR`	10	Unknown file error
`EXIT_DIR_CORRUPT`	11	Archive directory corrupted
`EXIT_LONG_PATH`	100	Path too long
`EXIT_NO_OVERRIDE`	101	Can't override base path
`EXIT_NEST`	102	Directories nested too deeply
`EXIT_SCRIPT_ERR`	103	Error(s) in script file
`EXIT_NOT_ARCH`	104	Not an HPACK archive
`EXIT_BAD_KEYFILE`	105	Bad keyfile
`EXIT_NOTHING`	106	Nothing to do
`EXIT_COMMAND`	107	Unknown command
`EXIT_OPTION`	108	Unknown option
`EXIT_PASS`	200	Differing or incorrect password(s)
`EXIT_CHANGE`	201	Bad attempt to change archive (eg attempt to change a block-encrypted archive)
`EXIT_NOLONG`	202	Long arg.format not supported
`EXIT_BADWILD`	203	Bad wildcard format or inappropriate use of wildcards
`EXIT_SECURITY`	204	General security/encryption error
`EXIT_NOCRYPT`	205	Cannot process encrypted archive

Under VMS return values are handled in a somewhat different manner. HPACK will return the error code as above in the facility number field, as well as a status of success, warning, error, or severe error corresponding to the above categories of error.

For more details on the error types, see the section "HPACK Error Messages" below.

General Archive Information

HPACK uses a central directory structure contained at the end of the archive, with the option of including directory information headers with each archived file for improved error recovery. All data stored in HPACK archives is left unchanged — there is no need for any truncation of filenames, translation of file attributes, or omission of OS-dependant information from a file in order to archive it. All information about a file (such as attributes, file datestamps, attached comments, icons, graphics information, security information, user-defined file attributes, and so on), is stored with the file and is translated on extraction if necessary. HPACK includes a fairly complete set of internal filesystem management routines which handle operations such as creating directories, adding and deleting files to/from directories, and so on. Filenames and directory names of any length and containing any characters are supported, as are special operating system-dependant files such as links, volume labels/ID's, and so on. All versions of HPACK will handle all the extra information which can be added to a file, but due to differences between various operating systems some of the information may not be used on some systems (for example MSDOS, a somewhat complex bootstrap loader used on many 80x86 systems, will ignore most of the extra information it finds in an archive).

Availability of HPACK for Other Systems

HPACK is currently available in Amiga, Archimedes, Atari ST, MSDOS, OS/2 (16 and 32 bit), Unix, and Windows versions, with other ports becoming available as access to the relevant hardware and software permits. If anyone wants a version for their particular system, they are welcome to try to port it across. The code consists of around 500K of mostly ANSI C code with some low-level system I/O thrown in, and a knowledge of assembly language being useful on low-end systems to speed up a few of the core compression and encryption routines. Anyone interested in porting HPACK to any system is urged to contact the author...

All code contained in HPACK, with the exception of the RSA encryption/decryption routines and the MD5 message digest routines, is entirely the result of original research and is not patented, nor do I have any intention of ever patenting it. The only code in HPACK which falls under any sort of restrictions is the RSA code. MD5 has been placed in the public domain by its authors. Since HPACK originates from outside the US, and since I don't believe in crippleware, there is no "restricted" or "crippled" version — full encryption and authentication capabilities are available to everyone.

Authentication of HPACK Executables

There have been several occasions in the past when fake versions of new archivers have been distributed. Sometimes these fake release are even wrapped up in a nice-looking "security envelope" guaranteeing their authenticity. Since the source code is freely available, it's all too easy for anyone to create a fake version of HPACK in which the encryption or authentication code has been compromised. In order to avoid any problems in this respect, the distributed HPACK executables are accompanied by a digital signature (see the section "HPACK Archive/Data Authentication" above) which can be used to verify that it is indeed an official version. Unlike the schemes used by other archivers, it is computationally infeasible to compromise this method of authentication.

In order to check the authenticity of the particular version of HPACK, you will need the PGP encryption package (see the section "HPACK Archive/Data Authentication" above), and my public key, which is included in the standard PGP distribution and also in the HPACK distribution. My key is signed by Philip Zimmerman, the inventor of PGP, and several members of the PGP development team. First, you should check my key for authenticity:

```
pgp -kc "Peter Gutmann"
```

When it performs the key check, PGP should display the following signatures:

```
Type bits/keyID     Date     User ID
pub  1024/997D47 1992/08/02  Peter Gutmann <pgut1@cs.aukuni.ac.nz>
sig!      E722D9 1992/11/26    Branko Lankester <lankeste@fwi.uva.nl>
sig!      997D47 1992/10/11    Peter Gutmann <pgut1@cs.aukuni.ac.nz>
sig!      7C02F9 1992/09/07    Felipe Rodriquez <nonsenso@utopia.hacktic.nl>
sig!      1336F5 1992/09/05    Harry Bush <Harry@castle.riga.lv>
sig!      67F70B 1992/09/02    Philip R. Zimmermann <prz@sage.cgd.ucar.edu>
```

Version 2.1 and up of PGP can, in addition, calculate a key fingerprint for a key. This can be calculated with:

```
pgp -kvc "Peter Gutmann"
```

PGP should display the following:

```
pub  1024/997D47 1992/08/02 Peter Gutmann <pgut1@cs.aukuni.ac.nz>
          Key fingerprint = 7C 6D 81 DF F2 62 0F 4A  67 0E 86 50 99 7E A6 B1
```

If the keyID or key fingerprint for my key differs from the one shown above or the signatures don't check out, then the key is a probably a fake and should *not be trusted.* Assuming the key is in order, the authenticity of the program itself can be checked with

```
pgp <program name> hpack.sig
```

where `hpack.sig` is the digital signature included with the program as distributed. For example to check the authenticity of the Archimedes HPACK executable type

```
pgp hpack hpack.sig
```

When it performs the check, PGP should display

```
Good signature from user "Peter Gutmann <pgut1@cs.aukuni.ac.nz>".
Signature made <date of signature>
```

If PGP reports a bad signature then the executable should *not be trusted.* A new, hopefully untouched, version can be obtained from any archive site, BBS, or system which carries the standard HPACK distribution, or you can arrange to get it directly from the author.

Credits

Many people have helped to get HPACK where it is today, among them (in no particular order):

Ports to different operating systems

- Stuart Woolford did the Unix port.
- John Burnell did the OS/2 port.
- Jason Williams and Edouard Poor helped with the Archimedes port.
- Nick Little and Peter McGavin helped with the Amiga port.
- Murray Moffatt did the Atari ST port.

Translations

- Peter Sowa did the German translation.
- Peter de Vocht did the Dutch translation.
- Arrigo Triulzi did the Italian translation.
- Peter Gutmann had a go at the Klingon translation.

Contacting the Author

You can contact me in any of the following ways:

By calling me on +64 9 426-5097 within reasonable hours (10am-12pm NZT).

By Usenet email to either `pgut1@cs.aukuni.ac.nz` (preferred), or `p.gutmann@\ cs.aukuni.ac.nz`, `peterg@kcbbs.gen.nz`, or `peter@nacjack.gen.nz` (In rough order of preference)

By Fidonet email to `Peter` at `3:772/90.0`. However this is probably about as reliable a link as using carrier mackerel across the Sahara.

By snail mail to:

24 Durness Place
Orewa
North Auckland
NEW ZEALAND

By throwing a bottle in the ocean (slightly less reliable than the Fidonet link).

By implanting subliminal messages in the music I listen to.

HPACK Error Messages

The error messages given by HPACK are the following:

`No matching files on disk`

The command you have given resulted in no changes being made to the archive(s) specified, due to the files you specified not being found on the disk.

`No matching files in archive`

The command you have given resulted in no changes being made to the archive(s) specified. There can be several reasons for this, among them being that there were no matching files in the archive to replace/update, or that there were no matching files in the root directory of the archive (HPACK will not, by default, check subdirectories: To handle subdirectories you must either give the full path within the archive, or use the [`-r`]ecurse subdirectories option).

`No matching archives found`

The command you have given resulted in no changes being made to the archive(s) specified, as they could not be found on the disk.

`Out of disk space`

There is not enough room on the current disk to finish archiving/unarchiving a file.

`Unknown command`

You have given HPACK a command which it doesn't understand. Check the section "HPACK Commands" for valid HPACK commands.

`Unknown option`

You have given HPACK an option which it doesn't understand. Check the section "HPACK Options" for valid HPACK options.

`Unknown overwrite option`

You have specified an unknown option for the [`-o`]verwrite switch. This error may also be caused by forgetting to specify an option for the switch, in which case HPACK will treat the character following the `-o` as the option.

`Unknown view option`

You have specified an unknown option for the [`-v`]iew options switch. This error may also be caused by forgetting to specify an option for the switch, in which case HPACK will treat the character following the `-v` as the option.

`Unknown directory option`

You have specified an unknown option for the [`-d`]irectory options switch. This error may also be caused by forgetting to specify an option for the switch, in which case HPACK will treat the character following the `-d` as the option.

`Bad wildcard format`

You have used an incorrect format when using wildcards in filenames.

`Wildcard expression too complex`

You have used a very complex wildcard sequence, and HPACK doesn't have enough room to process it fully. Try simplifying the expression, or break it up into separate simpler expressions.

`Cannot use wildcards in external pathname`

You have used wildcards in a pathname outside the archive. Since HPACK's extended wildcards are built on top of the usual operating system ones (if they exist) the resulting system would run very slowly if they had to be simulated on an external filesystem.

`Not an HPACK archive`

The supposed archive file does not appear to be an archive created by HPACK.

`Cannot open archive file`

HPACK cannot open the archive file.

`Cannot open data file - skipping`

HPACK cannot open the data file to dearchive data to.

`Cannot open temp file`

HPACK cannot open the temporary work file it needs for the [`D`]elete, [`F`]reshen, [`R`]eplace, or [`U`]pdate commands.

`Cannot open script file - skipping`

HPACK cannot open the script file containing the list of files to process and HPACK control commands.

`Cannot access base directory`

The base directory you have specified does not exist.

`Cannot create directory`

HPACK cannot create a directory to unarchive files into.

`Cannot override base path`

This error message arises when using the [`-b`]ase path option and either a drive specifier is given in the pathname of one of the files to process or the base path includes an actual path and the file pathname is specified as being off the root directory. Since the base path is prepended to the pathname of the file, the pathname cannot contain a drive specifier or absolute directory reference in it.

`Unknown archiving method - skipping`

The version of HPACK you are using is unfamiliar with the archiving method used for this file, and is skipping to the next file.

`Data is encrypted - skipping`

The data in this file is encrypted, and no means of decryption has been supplied by the user; therefore HPACK cannot process it and will move on to the next file.

`Cannot process encrypted archive`

The entire archive (rather than just the data in it) is encrypted, and HPACK cannot decrypt it to read it. This is either because no decryption password has been supplied, or because the archive is encrypted with an encryption method for which no decryption information is available.

`Cannot change deleted archive`

You have used the [`-k`]ill original archive option along with the [`R`]eplace, [`F`]reshen, or [`U`]pdate options.

`Cannot change multipart archive`

You have asked HPACK to [`F`]reshen, [`U`]pdate, [`R`]eplace, or [`D`]elete files in a multipart archive, which would involve rewriting an archive spread across multiple disks.

`Cannot change unit-compressed archive`

If an archive is compressed as one unit, it cannot later be changed in any way. Unit-compressed archives trade off compressed size for flexibility in updating them.

`Cannot change encrypted archive`

You have asked HPACK to [`A`]dd, [`F`]reshen, [`U`]pdate, [`R`]eplace, [`D`]elete, or [`X`]tract with move, files in an encrypted archive. Files or data in an encrypted archive cannot be changed without decrypting and re-encrypting the entire archive.

`Cannot change unencrypted archive`

You have asked HPACK to make changes to an unencrypted archive, using block-encryption. HPACK will not be retroactively encrypt the archive, however the files can be individually encrypted as they are added to the archive.

`Cannot use both conventional and public-key encryption`

You have specified that data be both conventional and public-key encrypted. Data can be encrypted with one of the two methods, but not with both at once.

`Passwords not the same`

When encrypting data, HPACK will ask you to retype the password for security. If the second password doesn't match the first password, this error message is issued.

`Password must be between 8 and 80 characters long`

The passphrase you have entered is either less than eight characters (making it dangerously short), or more than 80 characters (making it excessively long).

`Bad keyfile`

The keyfile containing the keys used for decrypting/authenticating data is corrupted.

`Missing userID for encryption/authentication`

You have forgotten to specify a userID when adding authentication information to an archive, or when using public-key encryption, or haven't given a primary userID for public-key encryption when using a secondary userID.

`Cannot find secret key for userID`

The secret key needed to add authentication information to data for the given userID cannot be found, and the authentication information cannot be added.

`Cannot find public key for userID`

The public key needed to encrypt data for the given userID cannot be found, and the encryption can't be carried out.

`Cannot find secret key - cannot decrypt data`

The secret key needed to decrypt the data cannot be found, and the data cannot be decrypted.

`Cannot find public key - cannot perform authentication check`

The public key for the authentication information attached to the data or archive can't be found, and thus the data authentication check can't be carried out.

`Authentication data corrupted - cannot perform authentication check`

The information used for authenticating the data in an archive has been corrupted.

`Cannot read random seed file`

The random seed file needed for public-key encryption cannot be read. If it has been destroyed, a new one should be created using PGP.

`Archive directory corrupted`

The archive directory information has been corrupted. Depending on how bad the corruption is, HPACK will either exit or prompt the user for whether it should try to continue, attempting to fix any errors in the directory as it processes it.

`Out of memory`

There is not enough memory available for HPACK to continue.

`Path too long`

You have used more than the maximum allowable number of characters (the limit set by the host OS) in a pathname.

`Path not found`

You have asked HPACK to add a file to a nonexistent directory inside an archive.

`Bad character in filename`

Input lines in script files are given a basic check for being valid filenames. If blatantly incorrect characters are discovered in a line of input this warning will be printed.

`Errors detected in script file`

Errors (either illegal characters or too-long pathnames) have been detected in the script file.

`Maximum level of errors detected in script file`

More than 10 errors have been detected in the script file. This error message generally results from erroneously specifying a non-script file as a script file.

`Too many levels of directory nesting`

You have tried to go down through more than 15 levels of subdirectories using the [`-r`]ecurse subdirectories option.

`Stopped at user request`

The user or some other program interrupted HPACK during the archiving process (for example by pressing the 'Break' key or by sending it a termination signal).

`Truncated EOF padding`

In some implementations of the Xmodem and Ymodem transfer protocols, CP/M end-of-file characters (Ctrl-Z's) are appended to the end of the transmitted file. When HPACK detects these spurious characters on the end of an archive, it will truncate the archive to its correct size and issue this warning message.

`Archive section too short, skipping...`

If part of a multipart archive is less than approximately 500 bytes long, HPACK will not attempt to create the archive but will move the data to the next, hopefully less full, disk.

`File error`

A miscellaneous type of file error occurred during the archiving process (for example some sort of network error). The message itself will if possible contain more detail about the precise type of error.

`Internal error`

Some form of internal error occurred in HPACK. Contact the author with details on how the error was brought about.

`Long argument format not supported in this version`

Some CLI versions of HPACK, notably the Amiga, Archimedes, Mac-CLI, MSDOS, OS/2, UNIX, and VMS ones, support a long argument format signalled by the `-z` switch, which is used to handle OS-specific options. The version you are using doesn't support this option.

Disclaimer

This program is guaranteed to perform as claimed, excluding any delays caused or enhanced by war, civil commotion, or rioting, whether declared, spontaneous, reprehensible, or justified; undue pressure to perform, from whatsoever source; mal de mer, mal de pays, mal de siecle, mal de code, mal de machine, or any force majeure not pretofore invoked.

The program warranty is void in case of nuclear war, whether caused by the program or not.

Trademarks

Amiga is a trademark of Commodore Business Machines.
Archimedes is a trademark of Acorn Computers.
Atari ST is a trademark of Atari Computers.
GIF is a trademark of Compuserve Inc.
IBM is a trademark of International Business Machines Corp.
Macintosh is a trademark of Apple Computer Inc.
MSDOS is a trademark of Microsoft Inc.
OS/2 is a trademark of International Business Machines Corp.
PKZIP is a trademark of PKWare Inc.
Prime and Primos are a trademarks of Prime Computer Corp.
QNX is a trademark of Quantum Software.
Rolemaster is a trademark of Iron Crown Enterprises Inc.
StuffIt is a trademark of Raymond Lau and Aladdin Systems.
ThinkC is a trademark of Symantec Corporation.
UNIX is both a trademark and a footnote of AT&T.
Windoze is not a trademark of Microsoft Corp. Windows is, but it's a somewhat less accurate description of its performance.
These trademark lists are trademarks of too many lawyers and too few people with common sense being involved.

Index

"Good compression isn't a matter of life and death — it's far more important than that"