

# Re: Using Python To Create An Encrypted Container

---

*Source:* <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2006-04/msg02567.html>

---

- *From:* John Hunter <[jdhunter@xxxxxxxxxxxxxxxxxxxxxxxx](mailto:jdhunter@xxxxxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Mon, 17 Apr 2006 08:54:56 -0500
- 

"Michael" == Michael  
Sperlle  
<[sperlle@xxxxxxxx](mailto:sperlle@xxxxxxxx)>  
writes:

Michael> Is it possible? Bestcrypt can supposedly be set up on  
Michael> linux, but it seems to need changes to the kernel before  
Michael> it can be installed, and I have no intention of going  
Michael> through whatever hell that would cause.

Michael> If I could create a large file that could be encrypted,  
Michael> and maybe add files to it by appending them and putting  
Michael> in some kind of delimiter between files, maybe a homemade  
Michael> version of truecrypt could be constructed.

One problem in using a large file which is encrypted is that a single byte error can destroy all your data (see DEDICATION below). I wrote a little python module called hashtar that works like tar, but encrypts every file independently in a flattened dir structure with filename hiding. It stores the file permissions, dir structure, and ownership in an encrypted header of each encrypted file after first padding some random data at the top to reduce the risk of known plaintext attacks.

Here is some example usage

```
> hashtar.py -cvf numeric.htar numeric
Password:
Confirm:
numeric/__init__.py -> numeric.htar/1a/1a9f48d439144d1fa33b186fa49a6b63
numeric/contiguous_demo.py -> numeric.htar/8a/8a7757bf6f4a20e6904173f7c597eb45
numeric/diff_dmo.py -> numeric.htar/0c/0cea827761aef0ccfc55a869dd2aeb38
numeric/fileio.py -> numeric.htar/3e/3e50f59a1d2d87307c585212fb84be6a
numeric/find -> numeric.htar/b1/b1070de08f4ea531f10abdc58cfe8edc
...snip

> find numeric.htar|head
```

## Re: Using Python To Create An Encrypted Container

```
numeric.htar
numeric.htar/1a
numeric.htar/1a/1a9f48d439144d1fa33b186fa49a6b63
numeric.htar/8a
numeric.htar/8a/8a7757bf6f4a20e6904173f7c597eb45
numeric.htar/8a/8a4343ba60feda855fbaf8132e9b5a6b
numeric.htar/8a/8a72457096828c8d509ece6520e49d0b
numeric.htar/0c
numeric.htar/0c/0cea827761aef0ccfc55a869dd2aeb38
numeric.htar/3e
```

```
> hashtar.py -tvf numeric.htar
Password:
131 numeric/__init__.py
594 numeric/contiguous_demo.py
26944 numeric/Quant.pyc
209 numeric/extensions/test/rpm_build.sh
230 numeric/diff_dmo.py
439 numeric/fileio.py
```

It works across platforms (win32, OSX and linux tested), so files encrypted on one will decrypt on others.

All the code lives in a single file (hashtar.py) included below.

See also the WARNING below (hint — I am not a cryptographer)

```
#!/usr/bin/env python
"""
```

### OVERVIEW

hashtar: an encrypted archive utility designed for secure archiving to media vulnerable to corruption.

Recursively encrypt the files and directories passed as arguments. Rather than preserving the directory structure, or archiving to a single file as in tar, the files are encrypted to a single dir and named with the hash of their relative path. The file information (filename, permission mode, uid, gid) is encrypted and stored in the header of the file itself, and can be used to restore the original file with dir structure from the archive file. For example, the command

```
> hashtar.py -cvf tmp.htar finance/
```

prompts for a password and generates an encrypted recursive archive of the finance dir in the tmp.htar dir, with filenames mapped like

```
finance/irs/98/f1040.pdf -> tmp.htar/e5/e5ed546c0bc0191d80d791bc2f73c890
finance/sale_house/notes -> tmp.htar/58/580e89bad7563ae76c295f75aecea030
```

## Re: Using Python To Create An Encrypted Container

finance/online/accounts.gz.mcr -> tmp.htar/bb/bbf12f06dc3fcee04067d40b9781f4a8  
finance/phone/prepaid1242.doc -> tmp.htar/c1/c1fe52a9d8cbef55eff8840d379d972a

The encrypted files are placed in subdirs based on the first two characters in their hash name because if too many files are placed in one dir, it may not be possible to pass all of them as command line arguments to the restore command. The entire finance dir structure can later be restored with

```
> hashtar.py -xvf tmp.htar
```

The advantage of this method of encrypted archiving, as opposed to archiving to a single tar file and encrypting it, is that this method is not sensitive to single byte corruption, which becomes important especially on externally stored archives, such as on CDR, or DVDR. Any individual file contains all the information needed to restore itself, with directory structure, permission bits, etc. So only the specific files that are corrupted on the media will be lost.

The alternative strategy, encrypting all the files in place and then archiving to external media, doesn't suffer from single byte corruption but affords less privacy since the filenames, dir structure, and permission bits are available, and less security since a filename may indicate contents and thus expose the archive to a known plaintext attack.

A match string allows you to only extract files matching a given pattern. Eg, to only extract pdf and xls files, do

```
> hashtar.py -m pdf,xls -xvf tmp.htar
```

Because the filenames are stored in the header, only a small portion of the file needs to be decrypted to determine the match, so this is quite fast.

Data can be encrypted and decrypted across platforms (tested between linux and win32 and vice-versa) but of course some information may be lost, such as uid, gid for platforms that don't support it.

### USAGE:

```
> hashtar.py [OPTIONS] files
```

### OPTIONS

-h, --help Show help message and exit  
-fDIR, --arcdir=DIR Write hashed filenames to archive dir  
-pFILE, --passwdfile=FILE  
Get passwd from FILE, otherwise prompt  
-mPATTERN, --match=PATTERN

## Re: Using Python To Create An Encrypted Container

Only extract files that match PATTERN.

PATTERN is a comma separated list of strings,  
one of which must match the filename

- u, --unlink Delete files after archiving them
- c, --create Create archive dir
- t, --tell Report information about files
- x, --extract Extract files recursively from archive dir
- v, --verbose Verbose listing of filenames to stdout

### WARNING:

I think this software is suitable to protect your data from your  
sister, your boss, and even the nosy computer hacker next door, but  
not the NSA.

### REQUIREMENTS:

python2.3 – [python.org](http://python.org)  
yawPyCrypto and Flatten – <http://yawpycrypto.sourceforge.net/>  
pycrypto – <http://www.amk.ca/python/code/crypto.html>

The python dependencies are very easy to install; just do the usual  
> python setup.py install

### PLATFORMS:

Tested on linux and win32

### AUTHOR:

John D. Hunter <jdhunter@xxxxxxxxxxxxxxxxxxxxxxxx>

### LICENSE:

same as python2.3

### KNOWN BUGS:

Ignores symbolic links

### DEDICATION:

For Erik Curiel, who's life's work I lost when I volunteered to  
backup the only copy of his home dir on a CD containing a single  
encrypted gzipped tar file, which was subsequently corrupted.

.....

```
import sys, os, random, struct, csv, time, glob
from md5 import md5
```

## Re: Using Python To Create An Encrypted Container

```
from optparse import OptionParser
from cStringIO import StringIO
from getpass import getpass
#def getpass(arg): pass
from yawPyCrypto.Cipher import DecryptCipher, EncryptCipher
from yawPyCrypto.Cipher import ZipDecryptCipher, ZipEncryptCipher
from yawPyCrypto.Constants import CIPHER_BLOWFISH, MODE_CBC

version = 0.3
pathsep = os.path.join('.', '')[1:] # is there a better way to get this?

def encrypt_str(passwd, s, enc=None):
    """
    Encrypt the string s using passwd and encryption cipher enc
    """
    if enc is None:
        enc = ZipEncryptCipher(passwd, CIPHER_BLOWFISH, MODE_CBC)
    enc.feed(s)
    enc.finish()
    return enc.data

def decrypt_str(passwd, s, dec=None):
    """
    Decrypt the string s using passwd and encryption cipher enc
    """
    if dec is None:
        dec = ZipDecryptCipher(passwd)
    dec.feed(s)
    dec.finish()
    return dec.data

def ends_with_pathsep(fname):
    """
    Return true if string fname ends in a path separator string
    """
    head, tail = os.path.split(fname)
    return tail == ""

junk = list('abcdefghijklmnopqrstuvwyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_ - /\\.\n'*20)
numJunk = 200

def info_to_str(info):
    """
    info is a fname, mode, uid, gid tuple

    Return a string for storage in the encrypted archive outfile
    """
    global junk
    # Here is the real info string
    sh = StringIO()
```

## Re: Using Python To Create An Encrypted Container

```
writer = csv.writer(sh)
writer.writerow(info)
s = sh.getvalue()

# because the info string will be fed thru zlib before encryption,
# I'm prepending numJunk garbage characters at the beginning of
# the key string to make it difficult for someone trying to do a
# known plaintext attack on the key file given that the info
# contains plain text that might be guessed and plain text that is
# common between archive file (such as base path names).
random.shuffle(junk)
junkStr = ".join(junk[:numJunk])

return junkStr + s

def str_to_info(s):
    """
    Takes string and returns (fname, mode, uid, gid)
    """
    sh = StringIO(s[numJunk:])
    reader = csv.reader(sh)
    for row in reader:
        fname,mode,uid,gid = row
    return fname, int(mode), int(uid), int(gid)

def encode(infile, arcdir, passwd, unlink=0, verbose=0, endian='little'):
    """
    Encrypt the file infile and hash it's filename to arcdir

    - infile: the input filename
    - arcdir is the dir that the files where the hashed filenames will
    be placed
    - passwd is the encryption passwd as string
    - unlink, if true, will try to remove the src file after
    archiving it

    """

    if not os.path.exists(infile):
        print >> sys.stderr, '%s does not exist: skipping' % infile
        return

    if os.path.isdir(infile):
        # mark dirs with pathsep at the end if they don't have one
        head, tail = os.path.split(infile)
        if not ends_with_pathsep(infile):
            infile = os.path.join(infile, ")

Re: Using Python To Create An Encrypted Container
```

## Re: Using Python To Create An Encrypted Container

```
m = md5(infile)
hd = m.hexdigest()
tup = os.stat(infile)

if pathsep=='/': storeName = infile
else: storeName = ''.join(infile.split(pathsep))
s = info_to_str( (storeName, tup.st_mode, tup.st_uid, tup.st_gid) )
s = encrypt_str(passwd, s)

outdir = os.path.join(arcdir, hd[:2])
if not os.path.isdir(outdir): os.mkdir(outdir, 0700)
outfile = os.path.join(outdir, hd)

oh = file(outfile, 'wb')

if endian=='little': fmt = '<'
else: fmt = '>'

oh.write(struct.pack(fmt+'fI', version, len(s)))
oh.write(s)

if verbose: print '%s -> %s' % (infile, outfile)
if os.path.isdir(infile): return 1 # nothing more to do for dirs

ih = file(infile, 'rb')
enc = EncryptCipher(passwd, CIPHER_BLOWFISH, MODE_CBC)
while 1:
    data = ih.read(1024)
    if len(data)==0: break
    enc.feed(data)
    oh.write(enc.data)
enc.finish()
oh.write(enc.data)
ih.close()
oh.close()

if unlink:
    try: os.remove(infile)
    except OSError, msg:
        print >> sys.stderr, 'Could not remove', fname
        print >> sys.stderr, msg

return 1

def decode(infile, passwd, unlink=0, verbose=0, match=None, endian='little'):
    """
    Restore the original file system from the archive dir

    - keys is a list of file information tuples; see str_to_keys.
```

## Re: Using Python To Create An Encrypted Container

- arcdir is the dir that the files with hashed filenames live
- passwd is the decryption passwd as string
- unlink, if true, will try to remove the archive file after restoring it

```
"""
```

```
ih = file(infile, 'rb')
if endian=='little': fmt = '<'
else: fmt = '>'

thisVersion, lenHeader = struct.unpack(fmt+'fI', ih.read(8))
if lenHeader>2000:
print '%s header size %d too large; aborting (try flipping endian)'%(infile, lenHeader)
sys.exit()

try: header = decrypt_str(passwd, ih.read(lenHeader))
except MemoryError, msg:
print >>sys.stderr, 'Could not decode %s; skipping' % infile
return 0
except ValueError, msg:
print >>sys.stderr, 'Could not decode %s; bad passwd or file?' % infile
print >>sys.stderr, '\t', msg
return 0
fname, mode, uid, gid = str_to_info(header)

if match is not None:
for pattern in match.split(','):
if fname.find(pattern)!=-1: break
else: return 0

if verbose: print '%s -> %s' % (infile,fname)
if ends_with_pathsep(fname): # it's a dir
if not os.path.isdir(fname):
os.makedirs(fname)
os.chmod(fname, mode)
try: os.chown(fname, uid, gid)
except AttributeError, msg: pass
except OSError, msg: pass # if coming from win32, uid,gid=0

return 1 # nothing more to do

thedir, thename = os.path.split(fname)
if not os.path.isdir(thedir) and len(thedir): os.makedirs(thedir)

dec = DecryptCipher(passwd)
oh = file(fname, 'wb')

while 1:
```

## Re: Using Python To Create An Encrypted Container

```
data = ih.read(1024)
if len(data)==0: break
dec.feed(data)
oh.write(dec.data)
dec.finish()
oh.write(dec.data)
ih.close()
oh.close()

os.chmod(fname, mode)
try: os.chown(fname, uid, gid)
except AttributeError: pass
except OSError, msg: pass # if coming from win32, uid,gid=0

if unlink:
try: os.remove(infile)
except OSError, msg:
print >> sys.stderr, 'Could not remove', infile

def tell(infile, passwd, verbose=0, endian='little'):
"""
Report the information about infile
"""

if endian=='little': fmt = '<'
else: fmt = '>'

ih = file(infile, 'rb')
thisVersion, lenHeader = struct.unpack(fmt+'fI', ih.read(8))
if lenHeader>2000:
print '%s header size %d too large; aborting (try flipping endian)'%(infile, lenHeader)
sys.exit()
try: header = decrypt_str(passwd, ih.read(lenHeader))
except MemoryError, msg:
print >>sys.stderr, 'Could not decode %s; skipping' % infile
return 0
except ValueError, msg:
print >>sys.stderr, 'Could not decode %s; bad passwd or file?' % infile
print >>sys.stderr, '\t', msg
return 0
size = os.path.getsize(infile)-lenHeader
fname, mode, uid, gid = str_to_info(header)

print '%d\t%s'%(size, fname)

def getpass2():
"""
Prompt for a passwd twice, returning the string only when they
match
```

## Re: Using Python To Create An Encrypted Container

```
"""
p1 = getpass('Password: ')
p2 = getpass('Confirm: ')
if p1!=p2:
print >> sys.stderr, '\nPasswords do not match. Try again.\n'
return getpass2()
else:
return p1

def listFiles(root, patterns='*', recurse=1, return_folders=0):
# from Parmar and Martelli in the Python Cookbook
import os.path, fnmatch
# Expand patterns from semicolon-separated string to list
pattern_list = patterns.split(';')
# Collect input and output arguments into one bunch
class Bunch:
def __init__(self, **kwds): self.__dict__.update(kwds)
arg = Bunch(recurse=recurse, pattern_list=pattern_list,
return_folders=return_folders, results=[])

def visit(arg, dirname, files):
# Append to arg.results all relevant files (and perhaps folders)
for name in files:
fullname = os.path.normpath(os.path.join(dirname, name))
if arg.return_folders or os.path.isfile(fullname):
for pattern in arg.pattern_list:
if fnmatch.fnmatch(name, pattern):
arg.results.append(fullname)
break
# Block recursion if recursion was disallowed
if not arg.recurse: files[:]=[]

os.path.walk(root, visit, arg)

return arg.results

def get_recursive_filelist(args):
"""
Recurs all the files and dirs in args ignoring symbolic links and
return the files as a list of strings
"""
files = []

for arg in args:
if os.path.isfile(arg):
files.append(arg)
continue
if os.path.isdir(arg):
newfiles = listFiles(arg, recurse=1, return_folders=1)
files.extend(newfiles)
```

## Re: Using Python To Create An Encrypted Container

```
return [f for f in files if not os.path.islink(f)]

if __name__ == '__main__':
    parser = OptionParser()

    parser.add_option("-f", "--arcdir", dest="arcdir",
                    help="Write hashed filenames to archive dir",
                    metavar="DIR",
                    default=os.getcwd())

    parser.add_option("-e", "--endian", dest="endian",
                    help="big|little",
                    default="little")

    parser.add_option("-p", "--passwdfile", dest="passwdfile",
                    help="Get passwd from FILE, otherwise prompt",
                    metavar="FILE", default=None)

    parser.add_option("-m", "--match", dest="match",
                    help="Only extract files that match PATTERN. PATTERN is a comma separated list of strings, one of which
                    must match the filename",
                    metavar="PATTERN", default=None)

    parser.add_option("-u", "--unlink",
                    action="store_true", dest="unlink", default=False,
                    help="Delete files after archiving them")

    parser.add_option("-c", "--create",
                    action="store_true", dest="create", default=False,
                    help="Create archive dir")

    parser.add_option("-x", "--extract",
                    action="store_true", dest="extract", default=False,
                    help="Extract files recursively from archive dir")

    parser.add_option("-t", "--tell",
                    action="store_true", dest="tell", default=False,
                    help="Report information about file but do not extract")

    parser.add_option("-v", "--verbose",
                    action="store_true", dest="verbose", default=False,
                    help="Verbose listing of filenames to stdout")

    (options, args) = parser.parse_args()

    if options.create and options.extract:
        print >>sys.stderr, 'Cannot create and extract archive simultaneously!'
        sys.exit()
    if not( options.create or options.extract or options.tell):
```

## Re: Using Python To Create An Encrypted Container

```
print >>sys.stderr, 'You must specify either -c or -x!'
sys.exit()

if os.path.exists(options.arcdir) and not os.path.isdir(options.arcdir):
print '%s exists and is not a dir' % options.arcdir

if not os.path.exists(options.arcdir):
os.makedirs(options.arcdir)

if options.passwdfile is not None:
passwd = file(options.passwdfile, 'rb').read()
else:
if options.create: passwd = getpass2()
else: passwd = getpass()

if options.extract:
if len(args)==0:
args = [options.arcdir]
files = get_recursive_filelist(args)

for thisFile in files:
head, tail = os.path.split(thisFile)
if not os.path.isfile(thisFile): continue
if not len(tail)==32:
print >>sys.stderr, '%s does not look like a hashname; skipping' % thisFile
continue
decode(thisFile, passwd,
unlink=options.unlink,
verbose=options.verbose,
match=options.match,
endian=options.endian
)
elif options.tell:
if len(args)==0:
args = [options.arcdir]
files = get_recursive_filelist(args)

for thisFile in files:
head, tail = os.path.split(thisFile)
if not os.path.isfile(thisFile): continue
if not len(tail)==32:
print >>sys.stderr, '%s does not look like a hashname; skipping' % thisFile
continue
tell(thisFile, passwd, verbose=options.verbose, endian=options.endian)

else:

if sys.platform=='win32':
# do glob expansion manually
expand = []
for arg in args:
```

## Re: Using Python To Create An Encrypted Container

```
expand.extend(glob.glob(arg))
args = expand
files = get_recursive_filelist(args)
for thisFile in files:
    encode(thisFile, options.arcdir, passwd,
    unlink=options.unlink,
    verbose=options.verbose,
    endian=options.endian)
```