CodeMagus

secfile: Code Magus Secure File

CML00124-01

CodeMagus

January 6, 2025

Contents

1 Introduction

secfile: Code Magus Secure File is the Code Magus utility used for securely sharing files. The data is triple-DES CBC encrypted, with varying key lengths options (that is, number of keys). The data is compressed prior to encryption in a loss-less manner and decompressed post decryption before restoring the contents in the clear to a file. Integrity is ensured by computing signatures using an MD5 hash[1] on both the clear data and the compressed and encrypted file data. The tool is available for sharing data across platforms including Windows, AIX, Linux, and z/OS.

There are various modes available for key management. Known symmetric DES keys can be supplied using the Code Magus Key Management library (a text format containing possibly multiple keys identified each identified by a unique key-set-id). Additionally the DES keys can be generated by `secfile` using a cryptographically secure random number generator (see `RAND_bytes` in OpenSSL[2]). In this modes, an RSA asymmetric cipher pubic key is expected which is used to encrypt the generated one-time keys and places the encrypted keys in a separate file for sharing with the party that holds the corresponding private key.

[1] RSA Data Security, Inc. MD5 Message-Digest Algorithm.

[2] OpenSSL DES, RSA functions from the OpenSSL project (www.openssl.org)

2 Running `secfile`

The program runs from the command line, either under a UNIX/Linux shell, at the Windows command prompt or bat-file, or under OMVS or BPXBATCH when running under z/OS.

```
secfile --function=encrypt \
   --clear-text-file-name=/tmp/testfile.txt \
   --clear-text-file-mode=r \
   --cipher-text-file-name=/tmp/testfile.enc \
   --key-set-file=/tmp/keyset.keys \
   --key-set-name=mykeyset
```

Where `/tmp/testfile.txt` is the name of the clear text file to be encrypted and `/tmp/testfile.enc` is the name of the resultant compressed and encrypted file. The file `/tmp/keyset.keys` is the file containing the symmetric cipher triple-DES keys to use to encrypt the data, and where `mykeyset` is the name of the specific key set to use for encryption. The following is an example of a key management key set file:

```
set(mykeyset)
   -- Keys file name: /tmp/keyset.keys
   mode(live)
   keys(X"2F6D9289E0085BC7",X"0D01A8EA2943E545",X"B6B03D29F8FEC7FE")
   ;
```

For this example, the clear-text file can be recovered as `/tmp/testfile_clear.txt` using the following command:

```
secfile --function=decrypt \
   --clear-text-file-name=/tmp/testfile_clear.txt \
   --clear-text-file-mode=w \
   --cipher-text-file-name=/tmp/testfile.enc \
   --key-set-file=/tmp/keyset.keys \
   --key-set-name=mykeyset
```

The full set of `secfile` command line options can be shown by running the command with the `--help` command line option:

```
[stephen@developer secfile]$ secfile --help
Code Magus Limited secfile V2.0: build 2024-12-10-11.32.48
[secfile Dec 10 2024 11:32:48]
Copyright (c) 2002--2024 by Code Magus Limited. All rights reserved.
[Assistance stephen@codemagus.com].
$Id: secfile_CML0012401.tex,v 1.4 2024/12/16 22:38:14 stephen Exp $
[OpenSSL DES, RSA functions from the OpenSSL project (www.openssl.org).        ]
[RSA Data Security, Inc. MD5 Message-Digest Algorithm.                         ]
[zlib general purpose compression library. (www.cdrom.com/pub/infozip/zlib/).  ]
[Code Magus Limited RDWIO I/O library. (www.codemagus.com).                    ]
Usage: secfile [OPTION...]
  -f, --function={encrypt|decrypt|deskeys}                Function to be performed
      --clear-text-file-name=<file-name>                  Name of the clear text file
  -m, --clear-text-file-mode=[{<in-mode>|r|rb,type=record}]     Input file open mode string
      --clear-text-file-desc=<file-desc>                  File descriptor of the clear text
                                                          file
      --cipher-text-file-name=<file-name>                 Name of the cipher text file
  -K, --key-set-file=<key-set-file>                       File containing named DES key sets
  -k, --key-set-name=<key-set-name>                            DES key set name to used from key
                                                               set file
      --private-key=<private-key-file>                         Private key of public/private key
                                                               pair file
      --public-key=<public-key-file>                           Public key of public/private key
                                                               pair file
      --encrypted-key=<encryption-key-file>                    Public key encrypted encryption
                                                               keys file
  -s, --clear-text-record-length=<clear-reclen>                Logical record length of clear text
                                                               file
  -r, --clear-text-uses-rdwio                                  Use the RDW I/O library for clear
                                                               text file
  -x, --clear-text-translate={ascii|ebcdic}                    Clear text character set. Cypher
                                                               text opposite,
  -a, --clear-text-line-feed                                   Clear text file records terminated
                                                               as text (LF/CRLF)
  -l, --record-limit=<record count>                            Limit number of records processed
      --help-split-encrypted-output                            Help on using the log splitting
                                                               options
  -T, --enc-out-name-strftime                                  Use strftime on pattern in output
                                                               encrypted files
  -B, --enc-out-segment-bytes=<byte-count>                     Switch output encrypted files after
                                                               byte count reached
  -S, --enc-out-segment-seconds=<seconds>                      Switch output encrypted files after
                                                               time in seconds expires
  -R, --enc-out-segment-records=<records>                      Switch output encrypted files after
                                                               record count reached

Help options:
  -?, --help                                                   Show this help message
      --usage                                                  Display brief usage message
```

3 Usecase scenarios

Depending on the particular setting and security policies and procedures in place, there are a number of modes of operation which may satisfy the requirements of a particular instance. The following sections detail these additional scenarios in addition to the basic scenario illustrated in Section 1.

In these additional scenarios, the presence of the following files is assumed:

File name	Description
`/tmp/testfile.txt`	The name of the original clear text file, *the data*.
`/tmp/private.pem`	A private key of a key-pair generated by and held securely by the securing officer responsible for the first party[a] [a]Here, the *first party* is the party that originates the clear text data. The *second party* is the party that consumes or processes the data.
`/tmp/public.pem`	A public key of the same key pair as the `/tmp/private.pem` made available by the security officer responsible for the first party.
`/tmp/return_private.pem`	A private key of a key-pair generated by and held securely by the securing officer responsible for the second party. For some usecases, this may be the second party required to consume or process the data.
`/tmp/return_public.pem`	A public key of the same key pair as the `/tmp/return_private.pem` made available to the security officer responsible for the first party.

As used here, a sample cleat text file can be generated from a local random source and base-64 encoded as follows:

```
base64 /dev/urandom | head -c 100000000 > /tmp/testfile.txt
```

For the examples used in here, the key-pairs are generated using the respective OpenSSL commands:

```
openssl genrsa -out /tmp/private.pem 8192
openssl rsa -pubout -in /tmp/private.pem \
      -out /tmp/public.pem
openssl genrsa -out /tmp/return_private.pem 1024
openssl rsa -pubout -in /tmp/return_private.pem \
      -out /tmp/return_public.pem
```

There are a number of modes that the compression-encryption and the decryption-uncompress can operate. Each independently. The compression-encryption function can read both a file or a file-descriptor. Being able to read a file descriptor allows the compression-encryption process to read a pipe in which case clear-text data does not have to hardened at any point during the data production/gathering process.

In the mode that one-time-keys are generated for the compression and encryption function, then the generated keys are encrypted with a supplied public key of an RSA key pair (a public key generated by the security officer responsible for the first party). If this mode is used by the first party whilst readying a pipe, then not only is the clear-text data not hardened on the system in which the data is generated/gathered, but without access to the corresponding public key, it is not possible for the generating or gathering system or the first party, to decrypt the data produced.

In the mode where one-time keys are generated for the data encryption, it is possible for a party with access to the corresponding private key (that is, the security officer responsible for the first party) to use the `secfile` program to generate a key-set file containing the tripe-DES keys in the clear. This clear key-set file can be shared with the second party with the compressed-encrypted file in hand so that the original clear text file can be recovered.

A further option would be to have the first party security officer with access to the private key decrypt the encrypted key-set file with that private key, and then re-encrypt the file with a public key shared by the security officer of the second party, that is from another key-pair. This would allow only the intended second party to have access to the clear text data — provided the second party can be trusted to keep the corresponding private key secure.

When compression and encrypting file, the output file name can be specified as a template, and in which case the compressed-encrypted data is written out into multiple files. When this output mode is selected with the one-time key set option, each individual file is encrypted with a unique set of triple-DES keys, and hence multiple encrypted one-time key set files are produced, one corresponding to each of the compressed-encrypted files.

The required commands to illustrate these scenarios are shown in the following sections.

3.1 Exchange of data using only symmetric keys

This is the scenario introduced in the Section 2. Following an agreed key exchange policy, the security officer of the either party exchanges keys with the security officer of the other party. Since the symmetric key set is required for the encryption process and the decryption process, the party or process producing or preparing the data for sharing and the party of process for consuming or preparing the data for consumption both require copies of the keys. The security of this scenario requires a secure exchange of the symmetric cipher keys (in this case DES or triple-DES encryption keys) as well as being able to maintain the security of the keys once installed on the respective processing platforms.

Assuming, as before, the following keys had been exchanged and saved in `/tmp/keyset.keys` (which in a real situation would not be stored in a file in `/tmp`). In this case, a triple length set of DES keys are being used:

```
set(mykeyset)
   -- Keys file name: /tmp/keyset.keys
   mode(live)
   keys(X"2F6D9289E0085BC7",X"0D01A8EA2943E545",X"B6B03D29F8FEC7FE")
   ;
```

The following command would be used by the first party in order to compress and encrypt the file. The MD5 signature values should be captured and shared with the second party in a manner that can be verified. The command reads in clear text file `/tmp/testfile.txt` and produces the output file `/tmp/testfile.enc` of the same contents by compressing and encrypting using triple-DES using the supplied symmetric keys.

```
secfile --function=encrypt \
   --clear-text-file-name=/tmp/testfile.txt \
   --clear-text-file-mode=r \
   --cipher-text-file-name=/tmp/testfile.enc \
   --key-set-file=/tmp/keyset.keys \
   --key-set-name=mykeyset
```

```
Code Magus Limited secfile V2.0: build 2024-12-10-11.32.48
[secfile Dec 10 2024 11:32:48]
Copyright (c) 2002--2024 by Code Magus Limited. All rights reserved.
[Assistance stephen@codemagus.com].
$Id: secfile_CML0012401.tex,v 1.4 2024/12/16 22:38:14 stephen Exp $
[OpenSSL DES, RSA functions from the OpenSSL project (www.openssl.org).
[RSA Data Security, Inc. MD5 Message-Digest Algorithm.
[zlib general purpose compression library. (www.cdrom.com/pub/infozip/zlib/).
[Code Magus Limited RDWIO I/O library. (www.codemagus.com).
DES_set_key_checked rc = 0
DES_set_key_checked rc = 0
DES_set_key_checked rc = 0
Number of records processed = 3053
Cypher text total = 76219616 bytes, clear text total = 100000000 bytes

Encryption computed file details:
   Start time   = Mon Dec 16 17:22:18 2024
   End time     = Mon Dec 16 17:22:26 2024
   Record count = 3053
   Computed clear text file MD5  = 0F045F10F02769E176F8D46B06BBE58E
   Computed cipher text file MD5 = C7F3EF2959D0420415ABEFF0AD142E36
```

The following command would be used by the second party to decrypt and uncompress the file. The shared MD5 values can be used to verify that the file remains intact, and has not been changed by comparing the values to the independently shared values generated when the file was compressed and encrypted. This ensures that the has not been tampered with and is indeed the file originally put forward for compression and encryption. To ensure that it is generated from the same set of keys, and also not tampered with the decryption and decompressing process re-computes the MD5 values and compares them to the values stored in the file and generated during the corresponding compression and encryption process. The uncompressed and decrypted file `/tmp/testfile_clear.txt` should exactly match the original text file `/tmp/testfile.txt`.

```
secfile --function=decrypt \
   --clear-text-file-name=/tmp/testfile_clear.txt \
   --clear-text-file-mode=w \
   --cipher-text-file-name=/tmp/testfile.enc \
   --key-set-file=/tmp/keyset.keys \
   --key-set-name=mykeyset
```

```
Code Magus Limited secfile V2.0: build 2024-12-10-11.32.48
[secfile Dec 10 2024 11:32:48]
Copyright (c) 2002--2024 by Code Magus Limited. All rights reserved.
[Assistance stephen@codemagus.com].
$Id: secfile_CML0012401.tex,v 1.4 2024/12/16 22:38:14 stephen Exp $
[OpenSSL DES, RSA functions from the OpenSSL project (www.openssl.org).
[RSA Data Security, Inc. MD5 Message-Digest Algorithm.
[zlib general purpose compression library. (www.cdrom.com/pub/infozip/zlib/).
[Code Magus Limited RDWIO I/O library. (www.codemagus.com).
DES_set_key_checked rc = 0
DES_set_key_checked rc = 0
DES_set_key_checked rc = 0
Number of records processed = 3053
Cypher text total = 76219616 bytes, clear text total = 100000000 bytes

Encryption computed file details:
   Start time   = Mon Dec 16 17:22:18 2024
   End time     = Mon Dec 16 17:22:26 2024
   Record count = 3053
   Computed clear text file MD5  = 0F045F10F02769E176F8D46B06BBE58E
   Computed cipher text file MD5 = C7F3EF2959D0420415ABEFF0AD142E36

Decryption computed file details:
   Start time   = Mon Dec 16 17:22:26 2024
   End time     = Mon Dec 16 17:22:31 2024
   Record count = 3053
   Computed clear text file MD5  = 0F045F10F02769E176F8D46B06BBE58E
   Computed cipher text file MD5 = C7F3EF2959D0420415ABEFF0AD142E36
```

The respective MD5 values should match the values produced when the file was compressed and encrypted, and the two sets of MD5 values shown on the decrypt and uncompress should match each other.

3.2 One-time symmetric keys and asymmetric key encryption

It is not necessary to exchange symmetric keys up-front. If the first party security officer shares the public key of a asymmetric key-pair (here, RSA key-pairs are used), then a mode of operation of `secfile` can be used to generate a set of triple-DES keys (using an OpenSSL method for generating cryptographically secure random bytes) to be used as a one-time set of keys to encrypt the data. The generated set of keys are then encrypted with the supplied public key and saved in a file. This scenario renders it impossible for the first party producer to recover the data from the compressed and encrypted file without the contents of the private key (in this case `/tmp/private.pem`) held by the security officer responsible for the first party.

The following command illustrates this mode of operation:

```
secfile --function=encrypt \
   --clear-text-file-name=/tmp/testfile.txt \
   --clear-text-file-mode=r \
   --cipher-text-file-name=/tmp/testfile.pub_enc \
   --public-key=/tmp/public.pem \
   --encrypted-key=/tmp/one_time_keys.pub_enc
```

```
Code Magus Limited secfile V2.0: build 2024-12-10-11.32.48
[secfile Dec 10 2024 11:32:48]
Copyright (c) 2002--2024 by Code Magus Limited. All rights reserved.
[Assistance stephen@codemagus.com].
$Id: secfile_CML0012401.tex,v 1.4 2024/12/16 22:38:14 stephen Exp $
[OpenSSL DES, RSA functions from the OpenSSL project (www.openssl.org).
[RSA Data Security, Inc. MD5 Message-Digest Algorithm.
[zlib general purpose compression library. (www.cdrom.com/pub/infozip/zlib/).
[Code Magus Limited RDWIO I/O library. (www.codemagus.com).
DES_set_key_checked rc = 0
DES_set_key_checked rc = 0
DES_set_key_checked rc = 0
Number of records processed = 3334
Cypher text total = 76207304 bytes, clear text total = 100000000 bytes

Encryption computed file details:
   Start time   = Mon Dec 16 17:22:10 2024
   End time     = Mon Dec 16 17:22:18 2024
   Record count = 3334
   Computed clear text file MD5  = 0F045F10F02769E176F8D46B06BBE58E
   Computed cipher text file MD5 = 28D447622AADABD4FB44EF9EF1A2AC9A
```

This command compresses and encrypts the supplied clear text file `/tmp/testfile.txt`, this time producing both a compressed encrypted file and a file containing the one-time key-set file `/tmp/one_time_keys.pub_enc` which is encrypted under the supplied public key. In all other respects, processing is the same as that used in the scenario explained in Section 3.1.

3.3 Decryption by sharing clear text one-time keys

In this scenario, the security officer of the first party, or any party with access to the public key generated by the security officer of the first party, can have the encrypted file sent to then and a corresponding key-set file can be extracted from the encrypted key set file. This clear text triple-DES key set can be sent to the third party that is required to decrypt the corresponding compressed and encrypted file for processing.

In order to accomplish this, the `deskeys` function of `secfile` is used with a supplied private key:

```
secfile --function=deskeys \
   --key-set-file=/tmp/keyset.keys \
   --key-set-name=mykeyset \
   --private-key=/tmp/private.pem \
   --encrypted-key=/tmp/one_time_keys.pub_enc
```

```
Code Magus Limited secfile V2.0: build 2024-12-10-11.32.48
[secfile Dec 10 2024 11:32:48]
Copyright (c) 2002--2024 by Code Magus Limited. All rights reserved.
[Assistance stephen@codemagus.com].
$Id: secfile_CML0012401.tex,v 1.4 2024/12/16 22:38:14 stephen Exp $
[OpenSSL DES, RSA functions from the OpenSSL project (www.openssl.org).
[RSA Data Security, Inc. MD5 Message-Digest Algorithm.
[zlib general purpose compression library. (www.cdrom.com/pub/infozip/zlib/).
[Code Magus Limited RDWIO I/O library. (www.codemagus.com).
```

Here, the file `/tmp/one_time_keys.pub_enc` contains the one-time triple-DES keys encrypted under the public key provided by the security officer responsible for the first party. This command creates the file `/tmp/keyset.keys` into which the clear-text triple-DES keys are placed as a key-set with key-set id of `mykeyset`.

```
[stephen@developer secfile]$ cat /tmp/keyset.keys
set(mykeyset)
   -- Keys file name: /tmp/keyset.keys
   mode(live)
   keys(X"3838C21F2616640B",X"BF64BA316B46BADA",X"384F5DCBCD2F2319")
   ;
```

This method requires the party holding the private key to have the `secfile` program installed. The following method does not require this if OpenSSL is available on the platform that contains the private key. If secfile is available on the platform that contains the private key generated by the security officer responsible for the first party, then if the security officer responsible for the second party has shared a public key with the security officer responsible for the first party, then the first party security officer can return the one-time key file encrypted under this public key. This would ensure that no other party of privy to the data contents (provided the public keys remain secret).

3.4 Decryption by sharing encrypted one-time keys

There is another method of sending the one-time keys to the second party which is applicable in the case where the `secfile` program is not available on the platform where the first party private key is available, but where OpenSSL is available. Similar to the method described at the end of Section 3.3. In this method OpenSSL and the first party private key is used to decrypt the one-time keys file, and then the one-time keys file is immediately re-encrypted with the public key shared by the security officer responsible for the second party. The data is then decrypted and uncompressed using the private key of the second party:

```
openssl rsautl -decrypt -in /tmp/one_time_keys.pub_enc \
           -out /tmp/checkfile_otp  \
           -inkey /tmp/private.pem
openssl rsautl -encrypt -in /tmp/checkfile_otp \
           -out /tmp/one_time_keys.pub_re_enc \
           -pubin -inkey /tmp/return_public.pem
```

Once the second party receives the re-encrypted key file `/tmp/one_time_keys.pub_re_enc` and has the compressed and encrypted data file `/tmp/testfile.pub_enc` in hand, the clear text can be decrypted and uncompressed into a clear text file `/tmp/testfile.pub3_txt`:

```
secfile --function=decrypt \
   --clear-text-file-name=/tmp/testfile.pub3_txt \
   --clear-text-file-mode=w \
   --cipher-text-file-name=/tmp/testfile.pub_enc \
   --encrypted-key=/tmp/one_time_keys.pub_re_enc \
   --private-key=/tmp/return_private.pem
```

```
Code Magus Limited secfile V2.0: build 2024-12-10-11.32.48
[secfile Dec 10 2024 11:32:48]
Copyright (c) 2002--2024 by Code Magus Limited. All rights reserved.
[Assistance stephen@codemagus.com].
$Id: secfile_CML0012401.tex,v 1.4 2024/12/16 22:38:14 stephen Exp $
[OpenSSL DES, RSA functions from the OpenSSL project (www.openssl.org).
[RSA Data Security, Inc. MD5 Message-Digest Algorithm.
[zlib general purpose compression library. (www.cdrom.com/pub/infozip/zlib/).
[Code Magus Limited RDWIO I/O library. (www.codemagus.com).
DES_set_key_checked rc = 0
DES_set_key_checked rc = 0
DES_set_key_checked rc = 0
```

```
Number of records processed = 3334
Cypher text total = 76207304 bytes, clear text total = 100000000 bytes

Encryption computed file details:
   Start time   = Mon Dec 16 17:22:10 2024
   End time     = Mon Dec 16 17:22:18 2024
   Record count = 3334
   Computed clear text file MD5  = 0F045F10F02769E176F8D46B06BBE58E
   Computed cipher text file MD5 = 28D447622AADABD4FB44EF9EF1A2AC9A

Decryption computed file details:
   Start time   = Mon Dec 16 17:22:57 2024
   End time     = Mon Dec 16 17:23:02 2024
   Record count = 3334
   Computed clear text file MD5  = 0F045F10F02769E176F8D46B06BBE58E
   Computed cipher text file MD5 = 28D447622AADABD4FB44EF9EF1A2AC9A
```

3.5 Reading clear text from a file descriptor

When compression and encrypting data, `secfile` allows the clear test data to be supplied using a file descriptor. This is useful where the clear text data is not to be hardened on the generating/gathering system and can only be hardened once suitably encrypted. This is useful when recording data from trace programs which may contain sensitive data. For example, when gathering network packets using `tcpdump`. It is also possible to supply the data to be decrypted using a file descriptor rather than a disk file. The following example shows the compression and encryption function using generated one-time keys while reading the cleat text data from a file descriptor:

```
cat /tmp/testfile.txt | secfile --function=encrypt \
   --clear-text-file-desc=0 \
   --clear-text-file-mode=r \
   --cipher-text-file-name=/tmp/testfile_fd.pub_enc \
   --public-key=/tmp/public.pem \
   --encrypted-key=/tmp/one_time_keys_fd.pub_enc
```

3.6 Writing multiple encrypted files with a template

By specifying the output file as a template, `secfile` is able to create many compressed and encrypted output files from a single input file or file descriptor. This is useful for long or permanently running data collectors. Each output file is given a unique file name which includes a sequence number and a time-stamp (if required). If a public key is provided and one-time keys are being used to encrypt the data, then every compressed and encrypted output file is generated with a separate set of triple-DES keys, and hence, for every encrypted file containing a portion of the input, there is an encrypted one-time keys file. The following command illustrates this scenario:

```
./secfile \
   --function=encrypt \
   --clear-text-file-name=/tmp/testfile.pub_txt \
   --cipher-text-file-name=/tmp/testfile_ENC_file \
   --encrypted-key=/tmp/testfile_ENC_key \
   --public-key=/tmp/public.pem \
   --enc-out-segment-records=10
```

Here, the input is split into chunks of 10 lines each, and creates the following files as output. The files with the suffix `CMLz` are the compressed and encrypted files, and the files with suffix `CMLotk` are the corresponding one-time key files encrypted under the public key `/tmp/public.pem`.

```
[stephen@developer secfile]$ ls -lat /tmp/ | head
-rw-rw-r--  1 stephen  stephen     259280 Dec 16 22:14 testfile_ENC_file_00018674_00000001.CMLz
-rw-rw-r--  1 stephen  stephen       1024 Dec 16 22:14 testfile_ENC_key_00018674_00000001.CMLotk
-rw-rw-r--  1 stephen  stephen     259280 Dec 16 22:14 testfile_ENC_file_00018674_00000002.CMLz
-rw-rw-r--  1 stephen  stephen       1024 Dec 16 22:14 testfile_ENC_key_00018674_00000002.CMLotk
-rw-rw-r--  1 stephen  stephen     259280 Dec 16 22:14 testfile_ENC_file_00018674_00000003.CMLz
-rw-rw-r--  1 stephen  stephen       1024 Dec 16 22:14 testfile_ENC_key_00018674_00000003.CMLotk
-rw-rw-r--  1 stephen  stephen     259360 Dec 16 22:14 testfile_ENC_file_00018674_00000004.CMLz
-rw-rw-r--  1 stephen  stephen       1024 Dec 16 22:14 testfile_ENC_key_00018674_00000004.CMLotk
-rw-rw-r--  1 stephen  stephen     259280 Dec 16 22:14 testfile_ENC_file_00018674_00000005.CMLz
-rw-rw-r--  1 stephen  stephen       1024 Dec 16 22:14 testfile_ENC_key_00018674_00000005.CMLotk
-rw-rw-r--  1 stephen  stephen     259360 Dec 16 22:14 testfile_ENC_file_00018674_00000006.CMLz
-rw-rw-r--  1 stephen  stephen       1024 Dec 16 22:14 testfile_ENC_key_00018674_00000006.CMLotk
-rw-rw-r--  1 stephen  stephen     259280 Dec 16 22:14 testfile_ENC_file_00018674_00000007.CMLz
-rw-rw-r--  1 stephen  stephen       1024 Dec 16 22:14 testfile_ENC_key_00018674_00000007.CMLotk
...
```