SEFile

Generated by Doxygen 1.8.5

Wed Nov 9 2016 19:23:54

Contents

1 Module Index **1**

1.1 Modules . . . 1

2 Data Structure Index **3**

2.1 Data Structures . . . 3

3 File Index **5**

3.1 File List . . . 5

4 Module Documentation **7**

4.1 SectorStruct . . . 7

4.1.1 Detailed Description . . . 7

4.2 EnvironmentalVars . . . 8

4.2.1 Detailed Description . . . 8

4.2.2 Variable Documentation . . . 8

4.2.2.1 EnvCrypto . . . 8

4.2.2.2 EnvKeyID . . . 8

4.2.2.3 EnvSession . . . 8

4.3 mode parameter for secure_open . . . 9

4.3.1 Detailed Description . . . 9

4.3.2 Macro Definition Documentation . . . 9

4.3.2.1 SEFILE_READ . . . 9

4.3.2.2 SEFILE_WRITE . . . 9

4.4 access parameter for secure_open . . . 10

4.4.1 Detailed Description . . . 10

4.4.2 Macro Definition Documentation . . . 10

4.4.2.1 SEFILE_NEWFILE . . . 10

4.4.2.2 SEFILE_OPEN . . . 10

4.5 whence parameter for secure_seek . . . 11

4.5.1 Detailed Description . . . 11

4.5.2 Macro Definition Documentation . . . 11

4.5.2.1 SEFILE_BEGIN . . . 11

4.5.2.2 SEFILE_CURRENT . . . 11
4.5.2.3 SEFILE_END . . . 11
4.6 error values . . . 12
4.6.1 Detailed Description . . . 12
4.7 Sector_Defines . . . 13
4.7.1 Detailed Description . . . 13
4.7.2 Macro Definition Documentation . . . 13
4.7.2.1 SEFILE_BLOCK_SIZE . . . 13
4.7.2.2 SEFILE_LOGIC_DATA . . . 13
4.7.2.3 SEFILE_SECTOR_DATA_SIZE . . . 13
4.7.2.4 SEFILE_SECTOR_OVERHEAD . . . 13
4.7.2.5 SEFILE_SECTOR_SIZE . . . 13

5 Data Structure Documentation **15**

5.1 SEFILE_HANDLE Struct Reference . . . 15
5.1.1 Detailed Description . . . 15
5.1.2 Field Documentation . . . 15
5.1.2.1 fd . . . 15
5.1.2.2 log_offset . . . 15
5.1.2.3 nonce_ctr . . . 15
5.1.2.4 nonce_pbkdf2 . . . 15
5.2 SEFILE_HEADER Struct Reference . . . 16
5.2.1 Detailed Description . . . 16
5.2.2 Field Documentation . . . 16
5.2.2.1 fname_len . . . 16
5.2.2.2 magic . . . 16
5.2.2.3 nonce_ctr . . . 16
5.2.2.4 nonce_pbkdf2 . . . 16
5.2.2.5 uid . . . 16
5.2.2.6 uid_cnt . . . 16
5.2.2.7 ver . . . 17
5.3 SEFILE_SECTOR Struct Reference . . . 17
5.3.1 Detailed Description . . . 17
5.3.2 Field Documentation . . . 17
5.3.2.1 data . . . 17
5.3.2.2 header . . . 17
5.3.2.3 len . . . 17
5.3.2.4 signature . . . 17

6 File Documentation **19**

6.1 /run/media/scaglia/5B106C33792E4440/polito/MAGISTRALE/current/tesi/SEfile_library/SEfile.c File Reference . . . 19
6.1.1 Detailed Description . . . 21
6.1.2 Macro Definition Documentation . . . 21
6.1.2.1 POS_TO_CIPHER_BLOCK . . . 21
6.1.3 Function Documentation . . . 21
6.1.3.1 check_env . . . 21
6.1.3.2 crypt_dirname . . . 21
6.1.3.3 crypt_header . . . 22
6.1.3.4 crypt_sectors . . . 22
6.1.3.5 crypto_filename . . . 22
6.1.3.6 decrypt_dirname . . . 23
6.1.3.7 decrypt_filehandle . . . 23
6.1.3.8 decrypt_filename . . . 23
6.1.3.9 decrypt_sectors . . . 23
6.1.3.10 get_filename . . . 24
6.1.3.11 get_filesize . . . 24
6.1.3.12 get_path . . . 24
6.1.3.13 secure_close . . . 24
6.1.3.14 secure_create . . . 25
6.1.3.15 secure_finit . . . 25
6.1.3.16 secure_getfilesize . . . 25
6.1.3.17 secure_init . . . 25
6.1.3.18 secure_ls . . . 26
6.1.3.19 secure_mkdir . . . 26
6.1.3.20 secure_open . . . 26
6.1.3.21 secure_read . . . 27
6.1.3.22 secure_seek . . . 27
6.1.3.23 secure_sync . . . 27
6.1.3.24 secure_truncate . . . 28
6.1.3.25 secure_update . . . 29
6.1.3.26 secure_write . . . 29
6.1.3.27 valid_name . . . 29
6.2 /run/media/scaglia/5B106C33792E4440/polito/MAGISTRALE/current/tesi/SEfile_library/SEfile.h File Reference . . . 30
6.2.1 Detailed Description . . . 32
6.2.2 Macro Definition Documentation . . . 32
6.2.2.1 MAX_PATHNAME . . . 32
6.2.3 Typedef Documentation . . . 32
6.2.3.1 SEFILE_FHANDLE . . . 32

6.2.4 Function Documentation 32
6.2.4.1 crypto_filename 32
6.2.4.2 secure_close 33
6.2.4.3 secure_create 33
6.2.4.4 secure_finit 33
6.2.4.5 secure_getfilesize 33
6.2.4.6 secure_init 34
6.2.4.7 secure_ls 34
6.2.4.8 secure_mkdir 34
6.2.4.9 secure_open 35
6.2.4.10 secure_read 36
6.2.4.11 secure_seek 36
6.2.4.12 secure_sync 36
6.2.4.13 secure_truncate 37
6.2.4.14 secure_update 37
6.2.4.15 secure_write 37

Index **38**

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

SectorStruct 7
EnvironmentalVars 8
mode parameter for secure_open 9
access parameter for secure_open 10
whence parameter for secure_seek 11
error values 12
Sector_Defines 13

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

SEFILE_HANDLE
: The SEFILE_HANDLE struct 15

SEFILE_HEADER
: The SEFILE_HEADER struct 16

SEFILE_SECTOR
: The SEFILE_SECTOR struct This data struct is the actual sector organization. The total size should ALWAYS be equal to SEFILE_SECTOR_SIZE. The first sector is used to hold ONLY the header. Thanks to the union data type, the developer can simply declare a sector and then choose if it is the header sector or not 17

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

/run/media/scaglia/5B106C33792E4440/polito/MAGISTRALE/current/tesi/SEfile_library/SEfile.c
In this file you will find the implementation of the public function already described in SEfile.h . 19
/run/media/scaglia/5B106C33792E4440/polito/MAGISTRALE/current/tesi/SEfile_library/SEfile.h
This file includes constants, return values and public functions used for implementing a secure file system 30

Chapter 4

Module Documentation

4.1 SectorStruct

Data Structures

- struct SEFILE_HEADER

  The SEFILE_HEADER struct.
- struct SEFILE_SECTOR

  The SEFILE_SECTOR struct This data struct is the actual sector organization. The total size should ALWAYS be equal to SEFILE_SECTOR_SIZE. The first sector is used to hold ONLY the header. Thanks to the union data type, the developer can simply declare a sector and then choose if it is the header sector or not.

4.1.1 Detailed Description

4.2 EnvironmentalVars

Environmental Variables

This static variables will store some data useful during the active session.

- static se3_session * EnvSession =NULL
- static int32_t * EnvKeyID =NULL
- static uint16_t * EnvCrypto =NULL

4.2.1 Detailed Description

4.2.2 Variable Documentation

4.2.2.1 uint16_t* EnvCrypto =NULL [static]

Which cipher algorithm and mode we want to use

4.2.2.2 int32_t* EnvKeyID =NULL [static]

Which KeyID we want to use

4.2.2.3 se3_session* EnvSession =NULL [static]

Which session we want to use

4.3 mode parameter for secure_open

Use this values as mode parameter for

secure_open().

- #define SEFILE_READ
- #define SEFILE_WRITE

4.3.1 Detailed Description

4.3.2 Macro Definition Documentation

4.3.2.1 #define SEFILE_READ

Open as Read only

4.3.2.2 #define SEFILE_WRITE

Open for Read/Write

4.4 access parameter for secure_open

Use this values as access parameter for

secure_open().

- #define SEFILE_NEWFILE
- #define SEFILE_OPEN

4.4.1 Detailed Description

4.4.2 Macro Definition Documentation

4.4.2.1 #define SEFILE_NEWFILE

Create new file and delete if existing

4.4.2.2 #define SEFILE_OPEN

Open an existing file, create it if not existing

4.5 whence parameter for secure_seek

Use this values as whence parameter for

secure_seek().

- #define SEFILE_BEGIN
- #define SEFILE_CURRENT
- #define SEFILE_END

4.5.1 Detailed Description

4.5.2 Macro Definition Documentation

4.5.2.1 #define SEFILE_BEGIN

Seek from file beginning

4.5.2.2 #define SEFILE_CURRENT

Seek from current position

4.5.2.3 #define SEFILE_END

Seek from file ending

4.6 error values

Returned error values

If something goes wrong, one of this values will be returned.

- #define **SEFILE_ENV_ALREADY_SET** 15
- #define **SEFILE_ENV_WRONG_PARAMETER** 16
- #define **SEFILE_ENV_MALLOC_ERROR** 17
- #define **SEFILE_ENV_NOT_SET** 18
- #define **SEFILE_SECTOR_MALLOC_ERR** 19
- #define **SEFILE_GET_FILEPOINTER_ERR** 20
- #define **SEFILE_HANDLE_MALLOC_ERR** 21
- #define **SEFILE_CLOSE_HANDLE_ERR** 22
- #define **SEFILE_CREATE_ERROR** 23
- #define **SEFILE_OPEN_ERROR** 24
- #define **SEFILE_WRITE_ERROR** 25
- #define **SEFILE_SEEK_ERROR** 26
- #define **SEFILE_READ_ERROR** 27
- #define **SEFILE_ILLEGAL_SEEK** 28
- #define **SEFILE_FILESIZE_ERROR** 29
- #define **SEFILE_BUFFER_MALLOC_ERR** 30
- #define **SEFILE_FILENAME_DEC_ERROR** 31
- #define **SEFILE_FILENAME_ENC_ERROR** 32
- #define **SEFILE_DIRNAME_ENC_ERROR** 33
- #define **SEFILE_DIRNAME_DEC_ERROR** 34
- #define **SEFILE_DIRNAME_TOO_LONG** 35
- #define **SEFILE_MKDIR_ERROR** 36
- #define **SEFILE_LS_ERROR** 37
- #define **SEFILE_USER_NOT_ALLOWED** 38
- #define **SEFILE_ENV_INIT_ERROR** 39
- #define **SEFILE_ENV_UPDATE_ERROR** 40
- #define **SEFILE_INTEGRITY_ERROR** 41
- #define **SEFILE_NAME_NOT_VALID** 42
- #define **SEFILE_TRUNCATE_ERROR** 43
- #define **SEFILE_DEVICE_SN_MISMATCH** 44
- #define **SEFILE_KEYID_NOT_PRESENT** 45
- #define **SEFILE_ALGID_NOT_PRESENT** 46
- #define **SEFILE_PATH_TOO_LONG** 47
- #define **SEFILE_SYNC_ERR** 48
- #define **SEFILE_SIGNATURE_MISMATCH** 49

4.6.1 Detailed Description

4.7 Sector_Defines

Constant used to define sector structure.

Do not change this unless you know what you are doing.

- #define SEFILE_SECTOR_SIZE 512
- #define SEFILE_SECTOR_DATA_SIZE (SEFILE_SECTOR_SIZE-B5_SHA256_DIGEST_SIZE)
- #define SEFILE_BLOCK_SIZE B5_AES_BLK_SIZE
- #define SEFILE_LOGIC_DATA (SEFILE_SECTOR_DATA_SIZE-2)
- #define SEFILE_SECTOR_OVERHEAD (SEFILE_SECTOR_SIZE-SEFILE_LOGIC_DATA)

4.7.1 Detailed Description

4.7.2 Macro Definition Documentation

4.7.2.1 #define SEFILE_BLOCK_SIZE B5_AES_BLK_SIZE

Cipher block algorithm requires to encrypt data whose size is a multiple of this block size

4.7.2.2 #define SEFILE_LOGIC_DATA (SEFILE_SECTOR_DATA_SIZE-2)

The largest multiple of SEFILE_BLOCK_SIZE that can fit in SEFILE_SECTOR_DATA_SIZE

4.7.2.3 #define SEFILE_SECTOR_DATA_SIZE (SEFILE_SECTOR_SIZE-B5_SHA256_DIGEST_SIZE)

The actual valid data may be as much as this, since the signature is coded on 32 bytes

4.7.2.4 #define SEFILE_SECTOR_OVERHEAD (SEFILE_SECTOR_SIZE-SEFILE_LOGIC_DATA)

The amount of Overhead created by SEFILE_SECTOR::len and SEFILE_SECTOR::signature

4.7.2.5 #define SEFILE_SECTOR_SIZE 512

Actual sector size. Use only power of 2

Chapter 5

Data Structure Documentation

5.1 SEFILE_HANDLE Struct Reference

The SEFILE_HANDLE struct.

Data Fields

- uint32_t log_offset
- int32_t fd
- uint8_t nonce_ctr [16]
- uint8_t nonce_pbkdf2 [SEFILE_NONCE_LEN]

5.1.1 Detailed Description

The SEFILE_HANDLE struct.

This abstract data type is used to hide from higher level of abstraction its implementation. The data stored in here are the current physical file pointer position and the file descriptor OS-dependent data type.

5.1.2 Field Documentation

5.1.2.1 int32_t SEFILE_HANDLE::fd

File descriptor in Unix environment

5.1.2.2 uint32_t SEFILE_HANDLE::log_offset

Actual pointer position in bytes

5.1.2.3 uint8_t SEFILE_HANDLE::nonce_ctr[16]

Nonce used for the CTR feedback

5.1.2.4 uint8_t SEFILE_HANDLE::nonce_pbkdf2[SEFILE_NONCE_LEN]

Nonce used for the PBKDF2

The documentation for this struct was generated from the following file:

- /run/media/scaglia/5B106C33792E4440/polito/MAGISTRALE/current/tesi/SEfile_library/SEfile.c

5.2 SEFILE_HEADER Struct Reference

The SEFILE_HEADER struct.

Data Fields

- uint8_t nonce_pbkdf2 [SEFILE_NONCE_LEN]
- uint8_t nonce_ctr [16]
- int32_t magic
- int16_t ver
- int32_t uid
- int32_t uid_cnt
- uint8_t fname_len

5.2.1 Detailed Description

The SEFILE_HEADER struct.

This data struct is used to define a 31 bytes field inside a sector while taking care of its inner composition.

5.2.2 Field Documentation

5.2.2.1 uint8_t SEFILE_HEADER::fname_len

1 byte to express how long is the filename.

5.2.2.2 int32_t SEFILE_HEADER::magic

4 bytes used to represent file type (not used yet)

5.2.2.3 uint8_t SEFILE_HEADER::nonce_ctr[16]

16 random bytes storing the IV for next sectors

5.2.2.4 uint8_t SEFILE_HEADER::nonce_pbkdf2[SEFILE_NONCE_LEN]

32 random bytes storing the IV for generating a different key

5.2.2.5 int32_t SEFILE_HEADER::uid

4 bytes not used yet

5.2.2.6 int32_t SEFILE_HEADER::uid_cnt

4 bytes not used yet

5.2.2.7 int16_t SEFILE_HEADER::ver

2 bytes used to represent current filesystem version (not used yet)

The documentation for this struct was generated from the following file:

- /run/media/scaglia/5B106C33792E4440/polito/MAGISTRALE/current/tesi/SEfile_library/SEfile.c

5.3 SEFILE_SECTOR Struct Reference

The SEFILE_SECTOR struct This data struct is the actual sector organization. The total size should ALWAYS be equal to SEFILE_SECTOR_SIZE. The first sector is used to hold ONLY the header. Thanks to the union data type, the developer can simply declare a sector and then choose if it is the header sector or not.

Data Fields

- union {
  SEFILE_HEADER header
  uint8_t data [SEFILE_LOGIC_DATA]
  };

- uint16_t len
- uint8_t signature [32]

5.3.1 Detailed Description

The SEFILE_SECTOR struct This data struct is the actual sector organization. The total size should ALWAYS be equal to SEFILE_SECTOR_SIZE. The first sector is used to hold ONLY the header. Thanks to the union data type, the developer can simply declare a sector and then choose if it is the header sector or not.

5.3.2 Field Documentation

5.3.2.1 uint8_t SEFILE_SECTOR::data[SEFILE_LOGIC_DATA]

```
In here it will be written the actual data.
```

Since it is inside a union data type, the filename will be written from 32nd byte.

5.3.2.2 SEFILE_HEADER SEFILE_SECTOR::header

See SEFILE_HEADER .

5.3.2.3 uint16_t SEFILE_SECTOR::len

How many bytes are actually stored in this sector.

5.3.2.4 uint8_t SEFILE_SECTOR::signature[32]

Authenticated digest generated by the device

The documentation for this struct was generated from the following file:

- /run/media/scaglia/5B106C33792E4440/polito/MAGISTRALE/current/tesi/SEfile_library/SEfile.c

Chapter 6

File Documentation

6.1 /run/media/scaglia/5B106C33792E4440/polito/MAGISTRALE/current/tesi/SEfile_-library/SEfile.c File Reference

In this file you will find the implementation of the public function already described in SEfile.h.

```
#include "SEfile.h"
```

Data Structures

- struct SEFILE_HANDLE

  The SEFILE_HANDLE struct.
- struct SEFILE_HEADER

  The SEFILE_HEADER struct.
- struct SEFILE_SECTOR

  The SEFILE_SECTOR struct This data struct is the actual sector organization. The total size should ALWAYS be equal to SEFILE_SECTOR_SIZE. The first sector is used to hold ONLY the header. Thanks to the union data type, the developer can simply declare a sector and then choose if it is the header sector or not.

Macros

- #define **_GNU_SOURCE**
- #define **SEFILE_NONCE_LEN** 32
- #define POS_TO_CIPHER_BLOCK(current_position) ((current_position / SEFILE_SECTOR_SIZE) - 1)∗(SEFILE_SECTOR_DATA_SIZE / SEFILE_BLOCK_SIZE)

Functions

- void get_filename (char ∗path, char ∗file_name)

  This function extract the filename pointed by path.
- void get_path (char ∗full_path, char ∗path)

  This function extract the path where the file is.
- uint16_t check_env ()

  This function check if the environmental variables are correctly initialized and set.
- uint16_t crypt_sectors (void ∗buff_decrypt, void ∗buff_crypt, size_t datain_len, size_t current_offset, uint8_t ∗nonce_ctr, uint8_t ∗nonce_pbkdf2)

  This function encrypts the buff_decrypt data by exploiting the functions provided by L1.h.

- uint16_t crypt_header (void *buff1, void *buff2, size_t datain_len, uint16_t direction)

  This function encrypts a header buffer by exploiting the functions provided by L1.h.
- uint16_t decrypt_sectors (void *buff_crypt, void *buff_decrypt, size_t datain_len, size_t current_offset, uint8_t *nonce_ctr, uint8_t *nonce_pbkdf2)

  This function decrypts the buff_crypt data by exploiting the functions provided by L1.h.
- uint16_t get_filesize (SEFILE_FHANDLE *hFile, uint32_t *length)

  This function is used to compute the total logic size of an open file handle.
- uint16_t decrypt_filename (char *path, char *filename)

  This function is used to compute the plaintext of a encrypted filename stored in path.
- uint16_t decrypt_filehandle (SEFILE_FHANDLE *hFile, char *filename)

  This function is used to compute the plaintext of a encrypted filename stored in an already open hFile header.
- uint16_t crypt_dirname (char *dirpath, char *encDirname)

  This function is used to compute the ciphertext of a directory name stored in dirname.
- uint16_t decrypt_dirname (char *dirpath, char *decDirname)

  This function is used to compute the plaintext of a encrypted directory name stored in dirname.
- uint16_t valid_name (char *name)

  This function checks if the given name can be a valid encrypted filename/directory name.
- uint16_t secure_init (se3_session *s, uint32_t keyID, uint16_t crypto)

  This function creates a new secure environment, by allocating statically the parameters needed by the following functions.
- uint16_t secure_update (se3_session *s, int32_t keyID, uint16_t crypto)

  This function can be called only after the secure_init() function and give to the user the possibility to overwrite the Environment variables with new ones.
- uint16_t secure_finit ()

  This function deallocate the structures defined by the secure_init(). Should be called at the end of a session. No parameters are needed;.
- uint16_t secure_open (char *path, SEFILE_FHANDLE *hFile, int32_t mode, int32_t creation)

  This function opens a secure file and create a SEFILE_FHANDLE that can be used in future.
- uint16_t secure_create (char *path, SEFILE_FHANDLE *hFile, int mode)

  This function creates a new secure file and creates a SEFILE_FHANDLE that can be used in future. If the file already exists, it is overwritten with an empty one, all previous data are lost.
- uint16_t secure_write (SEFILE_FHANDLE *hFile, uint8_t *dataIn, uint32_t dataIn_len)

  This function writes the characters given by dataIn to the encrypted file hFile. Before writing them, dataIn is encrypted according to the environmental parameters.
- uint16_t secure_read (SEFILE_FHANDLE *hFile, uint8_t *dataOut, uint32_t dataOut_len, uint32_t *bytesRead)

  This function reads from hFile bytesRead characters out of dataOut_len correctly decrypted ones and stores them in dataOut string.
- uint16_t secure_seek (SEFILE_FHANDLE *hFile, int32_t offset, int32_t *position, uint8_t whence)

  This function is used to move correctly the file pointer.
- uint16_t secure_truncate (SEFILE_FHANDLE *hFile, uint32_t size)

  This function resizes the file pointed by hFile to size. If size is bigger than its current size the gap is filled with 0s.
- uint16_t secure_close (SEFILE_FHANDLE *hFile)

  This function releases resources related to hFile.
- uint16_t secure_ls (char *path, char *list, uint32_t *list_length)

  This function identifies which encrypted files and encrypted directories are present in the directory pointed by path and writes them in list. It only recognizes the ones encrypted with the current environmental parameters.
- uint16_t secure_getfilesize (char *path, uint32_t *position)

  This function is used to get the total logic size of an encrypted file pointed by path. Logic size will always be smaller than physical size.
- uint16_t secure_mkdir (char *path)

  This function creates a directory with an encrypted name.

- uint16_t crypto_filename (char *path, char *enc_name, uint16_t *encoded_length)

  This function computes the encrypted name of the file specified at position path and its length.
- void **compute_blk_offset** (size_t current_offset, uint8_t *nonce)
- uint16_t **encrypt_name** (void *buff1, void *buff2, size_t size, uint16_t direction)
- uint16_t secure_sync (SEFILE_FHANDLE *hFile)

  This function is used in case we want to be sure that the physical file is synced with the OS buffers.

Variables

Environmental Variables

This static variables will store some data useful during the active session.

- static se3_session * EnvSession =NULL
- static int32_t * EnvKeyID =NULL
- static uint16_t * EnvCrypto =NULL

6.1.1 Detailed Description

In this file you will find the implementation of the public function already described in SEfile.h.

Authors

Francesco Giavatto, Nicolò Maunero, Giulio Scalia

Date

17/09/2016

6.1.2 Macro Definition Documentation

6.1.2.1 #define POS_TO_CIPHER_BLOCK(*current_position*) ((current_position / SEFILE_SECTOR_SIZE) - 1)*(SEFILE_SECTOR_DATA_SIZE / SEFILE_BLOCK_SIZE)

Macro used to convert the actual pointer position to the cipher blocks amount

6.1.3 Function Documentation

6.1.3.1 uint16_t check_env ( )

This function check if the environmental variables are correctly initialized and set.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.2 uint16_t crypt_dirname (char * *dirpath*, char * *encDirname*)

This function is used to compute the ciphertext of a directory name stored in dirname.

Parameters

in	*dirpath*	Path to the directory whose name has to be encrypted. No encrypted directory are allowed inside the path.
out	*encDirname*	A preallocated string where to store the encrypted directory name

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.3 uint16_t crypt_header (void * *buff1*, void * *buff2*, size_t *datain_len*, uint16_t *direction*)

This function encrypts a header buffer by exploiting the functions provided by L1.h.

Parameters

in	*buff1*	Pointer to the header we want to encrypt/decrypt.
out	*buff2*	Pointer to an allocated header where to store the result.
in	*datain_len*	How big is the amount of data.
in	*direction*	See SE3_DIR.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.4 uint16_t crypt_sectors (void * *buff_decrypt*, void * *buff_crypt*, size_t *datain_len*, size_t *current_offset*, uint8_t * *nonce_ctr*, uint8_t * *nonce_pbkdf2*)

This function encrypts the buff_decrypt data by exploiting the functions provided by L1.h.

Parameters

in	*buff_decrypt*	The plaintext data to be encrypted
out	*buff_crypt*	The preallocated buffer where to store the encrypted data.
in	*datain_len*	Specify how many data we want to encrypt.
in	*current_offset*	Current position inside the file expressed as number of cipher blocks
in	*nonce_ctr*	Initialization vector, see SEFILE_HEADER
in	*nonce_pbkdf2*	Initialization vector, see SEFILE_HEADER

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.5 uint16_t crypto_filename (char * *path*, char * *enc_name*, uint16_t * *encoded_length*)

This function computes the encrypted name of the file specified at position path and its length.

Parameters

in	*path*	It can be absolute or relative but it can not be a directory. No encrypted directory are allowed inside the path.

out	*enc_name*	Already allocate string where the encrypted filename should be stored.
out	*encoded_length*	Pointer to an allocated uint16_t where the length of the encrypted filename is stored.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.6 uint16_t decrypt_dirname (char * *dirpath*, char * *decDirname*)

This function is used to compute the plaintext of a encrypted directory name stored in dirname.

Parameters

in	*dirpath*	Path to the directory whose name has to be decrypted. No encrypted directory are allowed inside the path.
out	*decDirname*	A preallocated string where to store the decrypted directory name

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.7 uint16_t decrypt_filehandle (SEFILE_FHANDLE * *hFile*, char * *filename*)

This function is used to compute the plaintext of a encrypted filename stored in an already open hFile header.

Parameters

in	*hFile*	Already opened file handle to be read in order to obtain its plaintext filename.
out	*filename*	A preallocated string where to store the plaintext filename.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.8 uint16_t decrypt_filename (char * *path*, char * *filename*)

This function is used to compute the plaintext of a encrypted filename stored in path.

Parameters

in	*path*	Where the encrypted file is stored, it can be an absolute or relative path. No encrypted directory are allowed inside the path.
out	*filename*	A preallocated string where to store the plaintext filename

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.9 uint16_t decrypt_sectors (void * *buff_crypt*, void * *buff_decrypt*, size_t *datain_len*, size_t *current_offset*, uint8_t * *nonce_ctr*, uint8_t * *nonce_pbkdf2*)

This function decrypts the buff_crypt data by exploiting the functions provided by L1.h.

Parameters

in	*buff_crypt*	The cipher text data to be decrypted
out	*buff_decrypt*	The preallocated buffer where to store the decrypted data.
in	*datain_len*	Specify how many data we want to decrypt.
in	*current_offset*	Current position inside the file expressed as number of cipher blocks
in	*nonce_ctr*	Initialization vector, see SEFILE_HEADER
in	*nonce_pbkdf2*	Initialization vector, see SEFILE_HEADER

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.10 void get_filename (char * *path*, char * *file_name*)

This function extract the filename pointed by path.

Parameters

in	*path*	It can be both an absolute or relative path. No encrypted directory are allowed inside the path.
out	*file_name*	A preallocated string where to store the filename.

6.1.3.11 uint16_t get_filesize (SEFILE_FHANDLE * *hFile*, uint32_t * *length*)

This function is used to compute the total logic size of an open file handle.

Parameters

in	*hFile*	Open file handle whose size shall be computed.
out	*length*	Pointer to a preallocated variable where to store the logic size.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.12 void get_path (char * *full_path*, char * *path*)

This function extract the path where the file is.

Parameters

in	*full_path*	It can be both an absolute or relative path. No encrypted directory are allowed inside the path.
out	*path*	A preallocated string where to store the path

6.1.3.13 uint16_t secure_close (SEFILE_FHANDLE * *hFile*)

This function releases resources related to hFile.

Parameters

in	*hFile*	The handle to the file we do not want to manipulate no more.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.14 uint16_t secure_create (char * *path*, SEFILE_FHANDLE * *hFile*, int *mode*)

This function creates a new secure file and creates a SEFILE_FHANDLE that can be used in future. If the file already exists, it is overwritten with an empty one, all previous data are lost.

Parameters

in	*path*	Specify the absolute/relative path where to create the file. No encrypted directory are allowed inside the path.
out	*hFile*	The pointer in which the file handle to the new opened file is placed after a success, NULL in case of failure.
in	*mode*	The mode in which the file should be created. See mode parameter for secure_open.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.15 uint16_t secure_finit ( )

This function deallocate the structures defined by the secure_init(). Should be called at the end of a session. No parameters are needed;.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.16 uint16_t secure_getfilesize (char * *path*, uint32_t * *position*)

This function is used to get the total logic size of an encrypted file pointed by path. Logic size will always be smaller than physical size.

Parameters

in	*path*	Absolute or relative path the file. No encrypted directory are allowed inside the path.
out	*position*	Pointer to an allocated uint32_t variable where will be stored the file size.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.17 uint16_t secure_init (se3_session * *s*, uint32_t *keyID*, uint16_t *crypto*)

This function creates a new secure environment, by allocating statically the parameters needed by the following functions.

Parameters

in	*s*	Contains the pointer to the se3_session structure that must be used during the session.
in	*keyID*	Contains the ID number of the key that must be used during the session.
in	*crypto*	Contains the id to specify which algorithm to use. See AlgorithmAvail, it can be SE3_ALGO_MAX + 1 if you don't know which algorithm to choose. See error values for error list.

All the data passed to this function must be allocated and filled with valid data. Once secure_init succeed it is possible to destroy these data, since a copy has been made. N.B. Remember to call the secure_finit function to deallocate these data once you have finished.

6.1.3.18 uint16_t secure_ls (char * *path*, char * *list*, uint32_t * *list_length*)

This function identifies which encrypted files and encrypted directories are present in the directory pointed by path and writes them in list. It only recognizes the ones encrypted with the current environmental parameters.

Parameters

in	*path*	Absolute or relative path to the directory to browse. No encrypted directory are allowed inside the path.
out	*list*	Already allocated array where to store filenames and directory names. Each entry is separated by '\0'.
out	*list_length*	Pointer to a uint32_t used to stored total number of characters written in list.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.19 uint16_t secure_mkdir (char * *path*)

This function creates a directory with an encrypted name.

Parameters

in	*path*	Absolute or relative path of the new directory. No encrypted directory are allowed inside the path.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.20 uint16_t secure_open (char * *path*, SEFILE_FHANDLE * *hFile*, int32_t *mode*, int32_t *access*)

This function opens a secure file and create a SEFILE_FHANDLE that can be used in future.

Parameters

in	*path*	Specify the absolute/relative path where to retrieve the file to open. No encrypted directory are allowed inside the path.
out	*hFile*	The pointer in which the file handle to the opened file is placed after a success, NULL in case of failure.

in	*mode*	The mode in which the file should be opened. See mode parameter for secure_open.
in	*access*	Define if the file should be created or it should already exist. See access parameter for secure_open.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.21 uint16_t secure_read (SEFILE_FHANDLE * *hFile*, uint8_t * *dataOut*, uint32_t *dataOut_len*, uint32_t * *bytesRead*)

This function reads from hFile bytesRead characters out of dataOut_len correctly decrypted ones and stores them in dataOut string.

Parameters

in	*hFile*	The handle to an already opened file to be read.
out	*dataOut*	An already allocated array of characters where to store data read.
in	*dataOut_len*	Number of characters we want to read.
out	*bytesRead*	Number of effective characters read, MUST NOT BE NULL.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.22 uint16_t secure_seek (SEFILE_FHANDLE * *hFile*, int32_t *offset*, int32_t * *position*, uint8_t *whence*)

This function is used to move correctly the file pointer.

Parameters

in	*hFile*	The handle to the file to manipulate.
in	*offset*	Amount of character we want to move.
out	*position*	Pointer to a int32_t variable where the final position is stored, MUST NOT BE NULL.
in	*whence*	According to this parameter we can choose if we want to move from the file beginning, file ending or current file pointer position. See whence parameter for secure_seek.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.23 uint16_t secure_sync (SEFILE_FHANDLE * *hFile*)

This function is used in case we want to be sure that the physical file is synced with the OS buffers.

Parameters

hFile	Handle to the secure file to be synced.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.24 uint16_t secure_truncate (SEFILE_FHANDLE * *hFile,* uint32_t *size*)

This function resizes the file pointed by hFile to size. If size is bigger than its current size the gap is filled with 0s.

Parameters

in	*hFile*	The handle to the file to manipulate.
in	*size*	New size of the file.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.25 uint16_t secure_update (se3_session * *s*, int32_t *keyID*, uint16_t *crypto*)

This function can be called only after the secure_init() function and give to the user the possibility to overwrite the Environment variables with new ones.

Parameters

in	*s*	Contains the pointer to the se3_session structure that must be used during the session. Can be NULL.
in	*keyID*	keyID Contains the ID number of the key that must be used during the session. Can be -1.
in	*crypto*	Contains the id to specify which algorithm to use.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

Parameters from 1 to 3 can be a NULL pointer or '-1' value, if all parameters are NULL (or '-1' in case of keyID) the function is a No-Operation one.

6.1.3.26 uint16_t secure_write (SEFILE_FHANDLE * *hFile*, uint8_t * *dataIn*, uint32_t *dataIn_len*)

This function writes the characters given by dataIn to the encrypted file hFile. Before writing them, dataIn is encrypted according to the environmental parameters.

Parameters

in	*hFile*	The handle to an already opened file to be written.
in	*dataIn*	The string of characters that have to be written.
in	*dataIn_len*	The length, in bytes, of the data that have to be written.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.1.3.27 uint16_t valid_name (char * *name*)

This function checks if the given name can be a valid encrypted filename/directory name.

Parameters

in	*Name*	of the file/directory.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2 /run/media/scaglia/5B106C33792E4440/polito/MAGISTRALE/current/tesi/SEfile_-library/SEfile.h File Reference

This file includes constants, return values and public functions used for implementing a secure file system.

```
#include "se3/L0.h"
#include "se3/L1.h"
#include "se3/se3c1def.h"
#include <string.h>
#include <ctype.h>
```

Macros

- #define MAX_PATHNAME 256

Use this values as mode parameter for

secure_open().

- #define SEFILE_READ
- #define SEFILE_WRITE

Use this values as access parameter for

secure_open().

- #define SEFILE_NEWFILE
- #define SEFILE_OPEN

Use this values as whence parameter for

secure_seek().

- #define SEFILE_BEGIN
- #define SEFILE_CURRENT
- #define SEFILE_END

Returned error values

If something goes wrong, one of this values will be returned.

- #define **SEFILE_ENV_ALREADY_SET** 15
- #define **SEFILE_ENV_WRONG_PARAMETER** 16
- #define **SEFILE_ENV_MALLOC_ERROR** 17
- #define **SEFILE_ENV_NOT_SET** 18
- #define **SEFILE_SECTOR_MALLOC_ERR** 19
- #define **SEFILE_GET_FILEPOINTER_ERR** 20
- #define **SEFILE_HANDLE_MALLOC_ERR** 21
- #define **SEFILE_CLOSE_HANDLE_ERR** 22
- #define **SEFILE_CREATE_ERROR** 23
- #define **SEFILE_OPEN_ERROR** 24
- #define **SEFILE_WRITE_ERROR** 25
- #define **SEFILE_SEEK_ERROR** 26
- #define **SEFILE_READ_ERROR** 27
- #define **SEFILE_ILLEGAL_SEEK** 28
- #define **SEFILE_FILESIZE_ERROR** 29
- #define **SEFILE_BUFFER_MALLOC_ERR** 30
- #define **SEFILE_FILENAME_DEC_ERROR** 31
- #define **SEFILE_FILENAME_ENC_ERROR** 32
- #define **SEFILE_DIRNAME_ENC_ERROR** 33
- #define **SEFILE_DIRNAME_DEC_ERROR** 34
- #define **SEFILE_DIRNAME_TOO_LONG** 35

- #define **SEFILE_MKDIR_ERROR** 36
- #define **SEFILE_LS_ERROR** 37
- #define **SEFILE_USER_NOT_ALLOWED** 38
- #define **SEFILE_ENV_INIT_ERROR** 39
- #define **SEFILE_ENV_UPDATE_ERROR** 40
- #define **SEFILE_INTEGRITY_ERROR** 41
- #define **SEFILE_NAME_NOT_VALID** 42
- #define **SEFILE_TRUNCATE_ERROR** 43
- #define **SEFILE_DEVICE_SN_MISMATCH** 44
- #define **SEFILE_KEYID_NOT_PRESENT** 45
- #define **SEFILE_ALGID_NOT_PRESENT** 46
- #define **SEFILE_PATH_TOO_LONG** 47
- #define **SEFILE_SYNC_ERR** 48
- #define **SEFILE_SIGNATURE_MISMATCH** 49

Constant used to define sector structure.

Do not change this unless you know what you are doing.

- #define SEFILE_SECTOR_SIZE 512
- #define SEFILE_SECTOR_DATA_SIZE (SEFILE_SECTOR_SIZE-B5_SHA256_DIGEST_SIZE)
- #define SEFILE_BLOCK_SIZE B5_AES_BLK_SIZE
- #define SEFILE_LOGIC_DATA (SEFILE_SECTOR_DATA_SIZE-2)
- #define SEFILE_SECTOR_OVERHEAD (SEFILE_SECTOR_SIZE-SEFILE_LOGIC_DATA)

Typedefs

- typedef struct SEFILE_HANDLE * SEFILE_FHANDLE

Functions

- uint16_t secure_init (se3_session *s, uint32_t keyID, uint16_t crypto)

  This function creates a new secure environment, by allocating statically the parameters needed by the following functions.
- uint16_t secure_update (se3_session *s, int32_t keyID, uint16_t crypto)

  This function can be called only after the secure_init() function and give to the user the possibility to overwrite the Environment variables with new ones.
- uint16_t secure_finit ()

  This function deallocate the structures defined by the secure_init(). Should be called at the end of a session. No parameters are needed;.
- uint16_t crypto_filename (char *path, char *enc_name, uint16_t *encoded_length)

  This function computes the encrypted name of the file specified at position path and its length.
- uint16_t secure_open (char *path, SEFILE_FHANDLE *hFile, int32_t mode, int32_t access)

  This function opens a secure file and create a SEFILE_FHANDLE that can be used in future.
- uint16_t secure_create (char *path, SEFILE_FHANDLE *hFile, int mode)

  This function creates a new secure file and creates a SEFILE_FHANDLE that can be used in future. If the file already exists, it is overwritten with an empty one, all previous data are lost.
- uint16_t secure_write (SEFILE_FHANDLE *hFile, uint8_t *dataIn, uint32_t dataIn_len)

  This function writes the characters given by dataIn to the encrypted file hFile. Before writing them, dataIn is encrypted according to the environmental parameters.
- uint16_t secure_read (SEFILE_FHANDLE *hFile, uint8_t *dataOut, uint32_t dataOut_len, uint32_t *bytesRead)

  This function reads from hFile bytesRead characters out of dataOut_len correctly decrypted ones and stores them in dataOut string.
- uint16_t secure_seek (SEFILE_FHANDLE *hFile, int32_t offset, int32_t *position, uint8_t whence)

  This function is used to move correctly the file pointer.
- uint16_t secure_truncate (SEFILE_FHANDLE *hFile, uint32_t size)

This function resizes the file pointed by hFile to size. If size is bigger than its current size the gap is filled with 0s.

- uint16_t secure_close (SEFILE_FHANDLE ∗hFile)

  This function releases resources related to hFile.

- uint16_t secure_ls (char ∗path, char ∗list, uint32_t ∗list_length)

  This function identifies which encrypted files and encrypted directories are present in the directory pointed by path and writes them in list. It only recognizes the ones encrypted with the current environmental parameters.

- uint16_t secure_getfilesize (char ∗path, uint32_t ∗position)

  This function is used to get the total logic size of an encrypted file pointed by path. Logic size will always be smaller than physical size.

- uint16_t secure_mkdir (char ∗path)

  This function creates a directory with an encrypted name.

- uint16_t secure_sync (SEFILE_FHANDLE ∗hFile)

  This function is used in case we want to be sure that the physical file is synced with the OS buffers.

6.2.1 Detailed Description

This file includes constants, return values and public functions used for implementing a secure file system.

Authors

Francesco Giavatto, Nicolò Maunero, Giulio Scalia

Date

17/09/2016

In this library you will find wrappers to common OS system calls, in order to manage encrypted files using the SECube Board. In this files are also reported the constant used as parameter and all the possible return values.

6.2.2 Macro Definition Documentation

6.2.2.1 #define MAX_PATHNAME 256

Maximum length for pathname string

6.2.3 Typedef Documentation

6.2.3.1 typedef struct SEFILE_HANDLE∗ SEFILE_FHANDLE

Data struct used to access encrypted files

6.2.4 Function Documentation

6.2.4.1 uint16_t crypto_filename (char ∗ *path,* char ∗ *enc_name,* uint16_t ∗ *encoded_length*)

This function computes the encrypted name of the file specified at position path and its length.

Parameters

in	*path*	It can be absolute or relative but it can not be a directory. No encrypted directory are allowed inside the path.
out	*enc_name*	Already allocate string where the encrypted filename should be stored.
out	*encoded_length*	Pointer to an allocated uint16_t where the length of the encrypted filename is stored.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.2 uint16_t secure_close (SEFILE_FHANDLE * *hFile*)

This function releases resources related to hFile.

Parameters

in	*hFile*	The handle to the file we do not want to manipulate no more.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.3 uint16_t secure_create (char * *path*, SEFILE_FHANDLE * *hFile*, int *mode*)

This function creates a new secure file and creates a SEFILE_FHANDLE that can be used in future. If the file already exists, it is overwritten with an empty one, all previous data are lost.

Parameters

in	*path*	Specify the absolute/relative path where to create the file. No encrypted directory are allowed inside the path.
out	*hFile*	The pointer in which the file handle to the new opened file is placed after a success, NULL in case of failure.
in	*mode*	The mode in which the file should be created. See mode parameter for secure_open.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.4 uint16_t secure_finit ( )

This function deallocate the structures defined by the secure_init(). Should be called at the end of a session. No parameters are needed;.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.5 uint16_t secure_getfilesize (char * *path*, uint32_t * *position*)

This function is used to get the total logic size of an encrypted file pointed by path. Logic size will always be smaller than physical size.

Parameters

in	*path*	Absolute or relative path the file. No encrypted directory are allowed inside the path.
out	*position*	Pointer to an allocated uint32_t variable where will be stored the file size.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.6 uint16_t secure_init (se3_session * *s*, uint32_t *keyID*, uint16_t *crypto*)

This function creates a new secure environment, by allocating statically the parameters needed by the following functions.

Parameters

in	*s*	Contains the pointer to the se3_session structure that must be used during the session.
in	*keyID*	Contains the ID number of the key that must be used during the session.
in	*crypto*	Contains the id to specify which algorithm to use. See AlgorithmAvail, it can be SE3_ALGO_MAX + 1 if you don't know which algorithm to choose. See error values for error list.

All the data passed to this function must be allocated and filled with valid data. Once secure_init succeed it is possible to destroy these data, since a copy has been made. N.B. Remember to call the secure_finit function to deallocate these data once you have finished.

6.2.4.7 uint16_t secure_ls (char * *path*, char * *list*, uint32_t * *list_length*)

This function identifies which encrypted files and encrypted directories are present in the directory pointed by path and writes them in list. It only recognizes the ones encrypted with the current environmental parameters.

Parameters

in	*path*	Absolute or relative path to the directory to browse. No encrypted directory are allowed inside the path.
out	*list*	Already allocated array where to store filenames and directory names. Each entry is separated by '\0'.
out	*list_length*	Pointer to a uint32_t used to stored total number of characters written in list.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.8 uint16_t secure_mkdir (char * *path*)

This function creates a directory with an encrypted name.

Parameters

in	*path*	Absolute or relative path of the new directory. No encrypted directory are allowed inside the path.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.9 uint16_t secure_open (char * *path,* SEFILE_FHANDLE * *hFile,* int32_t *mode,* int32_t *access*)

This function opens a secure file and create a SEFILE_FHANDLE that can be used in future.

Parameters

in	*path*	Specify the absolute/relative path where to retrieve the file to open. No encrypted directory are allowed inside the path.
out	*hFile*	The pointer in which the file handle to the opened file is placed after a success, NULL in case of failure.
in	*mode*	The mode in which the file should be opened. See mode parameter for secure_open.
in	*access*	Define if the file should be created or it should already exist. See access parameter for secure_open.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.10 uint16_t secure_read (SEFILE_FHANDLE * *hFile*, uint8_t * *dataOut*, uint32_t *dataOut_len*, uint32_t * *bytesRead*)

This function reads from hFile bytesRead characters out of dataOut_len correctly decrypted ones and stores them in dataOut string.

Parameters

in	*hFile*	The handle to an already opened file to be read.
out	*dataOut*	An already allocated array of characters where to store data read.
in	*dataOut_len*	Number of characters we want to read.
out	*bytesRead*	Number of effective characters read, MUST NOT BE NULL.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.11 uint16_t secure_seek (SEFILE_FHANDLE * *hFile*, int32_t *offset*, int32_t * *position*, uint8_t *whence*)

This function is used to move correctly the file pointer.

Parameters

in	*hFile*	The handle to the file to manipulate.
in	*offset*	Amount of character we want to move.
out	*position*	Pointer to a int32_t variable where the final position is stored, MUST NOT BE NULL.
in	*whence*	According to this parameter we can choose if we want to move from the file beginning, file ending or current file pointer position. See whence parameter for secure_seek.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.12 uint16_t secure_sync (SEFILE_FHANDLE * *hFile*)

This function is used in case we want to be sure that the physical file is synced with the OS buffers.

Parameters

hFile	Handle to the secure file to be synced.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.13 uint16_t secure_truncate (SEFILE_FHANDLE * *hFile*, uint32_t *size*)

This function resizes the file pointed by hFile to size. If size is bigger than its current size the gap is filled with 0s.

Parameters

in	*hFile*	The handle to the file to manipulate.
in	*size*	New size of the file.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

6.2.4.14 uint16_t secure_update (se3_session * *s*, int32_t *keyID*, uint16_t *crypto*)

This function can be called only after the secure_init() function and give to the user the possibility to overwrite the Environment variables with new ones.

Parameters

in	*s*	Contains the pointer to the se3_session structure that must be used during the session. Can be NULL.
in	*keyID*	keyID Contains the ID number of the key that must be used during the session. Can be -1.
in	*crypto*	Contains the id to specify which algorithm to use.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

Parameters from 1 to 3 can be a NULL pointer or '-1' value, if all parameters are NULL (or '-1' in case of keyID) the function is a No-Operation one.

6.2.4.15 uint16_t secure_write (SEFILE_FHANDLE * *hFile*, uint8_t * *dataIn*, uint32_t *dataIn_len*)

This function writes the characters given by dataIn to the encrypted file hFile. Before writing them, dataIn is encrypted according to the environmental parameters.

Parameters

in	*hFile*	The handle to an already opened file to be written.
in	*dataIn*	The string of characters that have to be written.
in	*dataIn_len*	The length, in bytes, of the data that have to be written.

Returns

The function returns a (uint16_t) '0' in case of success. See error values for error list.

Index

access parameter for secure_open, 10
    SEFILE_NEWFILE, 10
    SEFILE_OPEN, 10

check_env
    SEfile.c, 21
crypt_dirname
    SEfile.c, 21
crypt_header
    SEfile.c, 22
crypt_sectors
    SEfile.c, 22
crypto_filename
    SEfile.c, 22
    SEfile.h, 32

data
    SEFILE_SECTOR, 17
decrypt_dirname
    SEfile.c, 23
decrypt_filehandle
    SEfile.c, 23
decrypt_filename
    SEfile.c, 23
decrypt_sectors
    SEfile.c, 23

EnvCrypto
    EnvironmentalVars, 8
EnvKeyID
    EnvironmentalVars, 8
EnvSession
    EnvironmentalVars, 8
EnvironmentalVars, 8
    EnvCrypto, 8
    EnvKeyID, 8
    EnvSession, 8
error values, 12

fd
    SEFILE_HANDLE, 15
fname_len
    SEFILE_HEADER, 16

get_filename
    SEfile.c, 24
get_filesize
    SEfile.c, 24
get_path
    SEfile.c, 24

header
    SEFILE_SECTOR, 17

len
    SEFILE_SECTOR, 17
log_offset
    SEFILE_HANDLE, 15

MAX_PATHNAME
    SEfile.h, 32
magic
    SEFILE_HEADER, 16
mode parameter for secure_open, 9
    SEFILE_READ, 9
    SEFILE_WRITE, 9

nonce_ctr
    SEFILE_HANDLE, 15
    SEFILE_HEADER, 16
nonce_pbkdf2
    SEFILE_HANDLE, 15
    SEFILE_HEADER, 16

SEFILE_BEGIN
    whence parameter for secure_seek, 11
SEFILE_BLOCK_SIZE
    Sector_Defines, 13
SEFILE_CURRENT
    whence parameter for secure_seek, 11
SEFILE_END
    whence parameter for secure_seek, 11
SEFILE_FHANDLE
    SEfile.h, 32
SEFILE_HANDLE, 15
    fd, 15
    log_offset, 15
    nonce_ctr, 15
    nonce_pbkdf2, 15
SEFILE_HEADER, 16
    fname_len, 16
    magic, 16
    nonce_ctr, 16
    nonce_pbkdf2, 16
    uid, 16
    uid_cnt, 16
    ver, 16
SEFILE_LOGIC_DATA
    Sector_Defines, 13
SEFILE_NEWFILE
    access parameter for secure_open, 10

SEFILE_OPEN
access parameter for secure_open, 10
SEFILE_READ
mode parameter for secure_open, 9
SEFILE_SECTOR, 17
data, 17
header, 17
len, 17
signature, 17
SEFILE_SECTOR_SIZE
Sector_Defines, 13
SEFILE_WRITE
mode parameter for secure_open, 9
SEfile.c
check_env, 21
crypt_dirname, 21
crypt_header, 22
crypt_sectors, 22
crypto_filename, 22
decrypt_dirname, 23
decrypt_filehandle, 23
decrypt_filename, 23
decrypt_sectors, 23
get_filename, 24
get_filesize, 24
get_path, 24
secure_close, 24
secure_create, 25
secure_finit, 25
secure_getfilesize, 25
secure_init, 25
secure_ls, 26
secure_mkdir, 26
secure_open, 26
secure_read, 27
secure_seek, 27
secure_sync, 27
secure_truncate, 27
secure_update, 29
secure_write, 29
valid_name, 29
SEfile.h
crypto_filename, 32
MAX_PATHNAME, 32
SEFILE_FHANDLE, 32
secure_close, 33
secure_create, 33
secure_finit, 33
secure_getfilesize, 33
secure_init, 34
secure_ls, 34
secure_mkdir, 34
secure_open, 34
secure_read, 36
secure_seek, 36
secure_sync, 36
secure_truncate, 37
secure_update, 37
secure_write, 37
Sector_Defines, 13
SEFILE_BLOCK_SIZE, 13
SEFILE_LOGIC_DATA, 13
SectorStruct, 7
secure_close
SEfile.c, 24
SEfile.h, 33
secure_create
SEfile.c, 25
SEfile.h, 33
secure_finit
SEfile.c, 25
SEfile.h, 33
secure_getfilesize
SEfile.c, 25
SEfile.h, 33
secure_init
SEfile.c, 25
SEfile.h, 34
secure_ls
SEfile.c, 26
SEfile.h, 34
secure_mkdir
SEfile.c, 26
SEfile.h, 34
secure_open
SEfile.c, 26
SEfile.h, 34
secure_read
SEfile.c, 27
SEfile.h, 36
secure_seek
SEfile.c, 27
SEfile.h, 36
secure_sync
SEfile.c, 27
SEfile.h, 36
secure_truncate
SEfile.c, 27
SEfile.h, 37
secure_update
SEfile.c, 29
SEfile.h, 37
secure_write
SEfile.c, 29
SEfile.h, 37
signature
SEFILE_SECTOR, 17

uid
SEFILE_HEADER, 16
uid_cnt
SEFILE_HEADER, 16

valid_name
SEfile.c, 29
ver
SEFILE_HEADER, 16

whence parameter for secure_seek, 11
    SEFILE_BEGIN, 11
    SEFILE_CURRENT, 11
    SEFILE_END, 11