Steganography on Vorbis' audio streams

Generated by Doxygen 1.6.1

Mon Aug 30 18:29:37 2010

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 cryptos_config_t Struct Reference

Defines the global configuration of the cryptographic protocol.

```
#include <include/cryptos_types.h>
```

Public Attributes

- int cipher_algo
- int md_algo
- int md_len
- int max_data
- int hmac
- gcry_cipher_hd_t chd
- gcry_md_hd_t mdhd
- cryptos_key_t ∗ key
- cryptos_key_t ∗ iv
- uint64_t emission
- uint64_t packet
- uint64_t default_data_size

3.1.1 Detailed Description

Defines the global configuration of the cryptographic protocol.

See also:

cryptos_key_t

3.1.2 Member Data Documentation

3.1.2.1 gcry_cipher_hd_t cryptos_config_t::chd

Handler for the ciphering algorithm

3.1.2.2 int cryptos_config_t::cipher_algo

Cipher algorithm to use

3.1.2.3 uint64_t cryptos_config_t::default_data_size

Default size of data, in bytes, to send in a given packet.

3.1.2.4 uint64_t cryptos_config_t::emission

Current emission ID

3.1.2.5 int cryptos_config_t::hmac

When active, indicate hmac variant of digest algorithm

3.1.2.6 cryptos_key_t∗ cryptos_config_t::iv

IV to use in the cipher algorithm

3.1.2.7 cryptos_key_t∗ cryptos_config_t::key

3.1.2.8 int cryptos_config_t::max_data

Max data to hide. Will depend on the cipher and digest algorithms

3.1.2.9 int cryptos_config_t::md_algo

Digest algorithm to use

3.1.2.10 int cryptos_config_t::md_len

Digest length

3.1.2.11 gcry_md_hd_t cryptos_config_t::mdhd

Handler for the message digest algorithm

3.1.2.12 uint64_t cryptos_config_t::packet

Current packet ID

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

- include/cryptos_types.h

3.2 cryptos_key_t Struct Reference

Defines the key structure to be used in the cryptographic layer.

```
#include <include/cryptos_types.h>
```

Public Attributes

- byte ∗ key
- int length

3.2.1 Detailed Description

Defines the key structure to be used in the cryptographic layer.

3.2.2 Member Data Documentation

3.2.2.1 byte∗ cryptos_key_t::key

Key value

3.2.2.2 int cryptos_key_t::length

Key length, in bytes.

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

- include/cryptos_types.h

3.3 cryptos_protocol_buffer Struct Reference

Buffer used within the cryptos protocol and which also serves as interface with the steganos protocol.

```
#include <cryptos_types.h>
```

3.3.1 Detailed Description

Buffer used within the cryptos protocol and which also serves as interface with the steganos protocol.

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

- include/cryptos_types.h

3.4 cryptos_protocol_buffer_t Struct Reference

```
#include <cryptos_types.h>
```

Public Attributes

- int fd
- size_t offset
- byte * buffer
- size_t buffer_size
- size_t buffer_used

3.4.1 Member Data Documentation

3.4.1.1 byte∗ cryptos_protocol_buffer_t::buffer

Buffer to store data not successfuly sent or received in previous packets.

3.4.1.2 size_t cryptos_protocol_buffer_t::buffer_size

Allocated size for buffer.

3.4.1.3 size_t cryptos_protocol_buffer_t::buffer_used

Current amount of bytes in buffer.

3.4.1.4 int cryptos_protocol_buffer_t::fd

The file from which we'll read the data to send at the emitter's side, or the file in which we'll write the data at the receiver's side.

3.4.1.5 size_t cryptos_protocol_buffer_t::offset

Offset inside fd, in bytes.

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

- include/cryptos_types.h

3.5 iss_cfg_t Struct Reference

```
#include <steganos_types.h>
```

Public Attributes

- float alpha
- float lambda
- float sigma
- float ∗ u
- float u_norm

3.5.1 Member Data Documentation

3.5.1.1 float iss_cfg_t::alpha

alpha in s = x + (alpha∗b - lambda∗x)∗u

3.5.1.2 float iss_cfg_t::lambda

lambda in s = x + (alpha∗b - lambda∗x)∗u

3.5.1.3 float iss_cfg_t::sigma

The deviation to use in the watermarking random sequence

3.5.1.4 float∗ iss_cfg_t::u

The watermark

3.5.1.5 float iss_cfg_t::u_norm

Norm of the watermark

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

- include/steganos_types.h

3.6 prng_t Struct Reference

Defines the structure to use as PRNG abstraction.

```
#include <include/numbers.h>
```

Public Attributes

- void ∗ seed
- long int iters

3.6.1 Detailed Description

Defines the structure to use as PRNG abstraction.

3.6.2 Member Data Documentation

3.6.2.1 long int prng_t::iters

Numbers of times used

3.6.2.2 void∗ prng_t::seed

Seed used in the PRNG

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

- include/numbers.h

3.7 steganos_key_t Struct Reference

Defines the key structure to be used in the steganographic layer.

```
#include <include/steganos_types.h>
```

Public Attributes

- byte * key
- int length

3.7.1 Detailed Description

Defines the key structure to be used in the steganographic layer.

3.7.2 Member Data Documentation

3.7.2.1 byte* steganos_key_t::key

Key value, of size $\lceil length/BITS\_PER\_BYTE \rceil$ bytes.

3.7.2.2 int steganos_key_t::length

Key length, in bits.

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

- include/steganos_types.h

3.8 steganos_state_t Struct Reference

Defines the global internal state of the steganosgraphic protocol.

```
#include <include/steganos_types.h>
```

Public Attributes

- vorbis_config_t ∗ vc
- steganos_key_t ∗ master_key
- steganos_key_t ∗ synchro_key
- steganos_key_t ∗ hiding_key
- synchro_method_et synchro_method
- sync_method synchronize
- desync_method desynchronize
- int desync
- hide_method_et hide_method
- hide_method hide
- unsigned int da
- float ra
- float variation_limit [VORBIS_MAX_BLOCK][2]
- float max_fc_capacity
- float min_fc_capacity
- int res_max_capacity [VORBIS_MAX_BLOCK]
- int res_min_capacity [VORBIS_MAX_BLOCK]
- int res_lineup [VORBIS_MAX_BLOCK]
- int res_occupied [VORBIS_MAX_BLOCK]
- unsigned long int sent
- unsigned long int read
- long int metadata_sent
- long int remaining
- int out [VIF_POSIT+2]
- int status
- int posts_mode
- int aligned
- long int total_sub_capacity
- int iters
- prng_t ∗ prng

3.8.1 Detailed Description

Defines the global internal state of the steganosgraphic protocol.

See also:

synchro_method_et
sync_method
desync_method
hide_method_et
hide_method
steganos_key_t

3.8.2 Member Data Documentation

3.8.2.1 int steganos_state_t::aligned

Boolean signaling that we've already calculated the residue lineup. Avoids recalculating it when considering several different stego-frame-channels.

3.8.2.2 unsigned int steganos_state_t::da

Desired aggressiveness of the protocol. Remains the same through the whole process. Ranges from 1 to 10, meaning we want to use the 10,20, ... 100% of the total subliminal capacity.

3.8.2.3 int steganos_state_t::desync

Boolean. Active when desynchronization must be done.

3.8.2.4 desync_method steganos_state_t::desynchronize

Desynchronization method (pointer to function)

3.8.2.5 hide_method steganos_state_t::hide

Hiding method (pointer to function) to use in the current frame and channel.

3.8.2.6 hide_method_et steganos_state_t::hide_method

Hiding method (enum type) to use in the current frame and channel residue vectors.

3.8.2.7 steganos_key_t∗ steganos_state_t::hiding_key

Key to use for hiding the data in the residues.

3.8.2.8 int steganos_state_t::iters

Usefull when debugging and printing statistics, stores how many iterations of the steganographic protocol has been run at a given instant.

3.8.2.9 steganos_key_t∗ steganos_state_t::master_key

Master key used to derive the subkeys.

3.8.2.10 float steganos_state_t::max_fc_capacity

Stores the maximum subliminal capacity for the current frame and channel residue, in bits.

3.8.2.11 long int steganos_state_t::metadata_sent

The same as sent, but including metadata

3.8.2.12 float steganos_state_t::min_fc_capacity

Stores the minimum subliminal capacity for the current frame and channel residue, in bits.

3.8.2.13 int steganos_state_t::out[VIF_POSIT+2]

Vector with the posts values to write in the ogg packet when using ISS.

3.8.2.14 int steganos_state_t::posts_mode

With ISS, this variable indicates the way we're trying to mark the floor posts. 1 means we're trying to mark it with a 1 bit, 0 means we're using the original posts, and -1 means we're trying to mark it with a 0 bit.

3.8.2.15 prng_t∗ steganos_state_t::prng

PRNG abstraction

3.8.2.16 float steganos_state_t::ra

The aggressiveness is used as a guide to the global share of the subliminal channel we desire to use, but due to the stochastic nature of the method, we shall use it as a guide rather than a strict quantity. The *real aggressiveness* will change from channel to channel and will be used as a correction indicator.

3.8.2.17 unsigned long int steganos_state_t::read

Total amount of _pure_ data (excluding metadata) already recoverred from the stego-audio stream.

3.8.2.18 long int steganos_state_t::remaining

Total amount of _pure_ data to send/receive.

3.8.2.19 int steganos_state_t::res_lineup[VORBIS_MAX_BLOCK]

Will store the ordering of the residual elements for hiding/unhiding purposes.

3.8.2.20 int steganos_state_t::res_max_capacity[VORBIS_MAX_BLOCK]

Will store the maximum number of subliminal bits a given residual element should shelter. This value should be seen also as a guide.

3.8.2.21 int steganos_state_t::res_min_capacity[VORBIS_MAX_BLOCK]

Will store the minimum number of subliminal bits a given residual element should shelter. This value should be seen also as a guide.

3.8.2.22 int steganos_state_t::res_occupied[VORBIS_MAX_BLOCK]

Will be used to mark the residual values which will be used to write in or read from whithin the same frame. This array will prevent from writing/ reading in/from the same residual value more than once.

3.8.2.23 unsigned long int steganos_state_t::sent

Total amount of _pure_ data (excluding metadata) already hided in the audio stream.

3.8.2.24 int steganos_state_t::status

Status code. Used to control "inter fucntion" errors

3.8.2.25 steganos_key_t∗ steganos_state_t::synchro_key

Key to use for synchronization.

3.8.2.26 synchro_method_et steganos_state_t::synchro_method

Synchronization method (enum type). Can change from one frame and channel to another.

3.8.2.27 sync_method steganos_state_t::synchronize

Synchronization method (pointer to function) to use in the current frame and channel.

3.8.2.28 long int steganos_state_t::total_sub_capacity

The total subliminal capacity at a given instant. Useful for statistics and debugging. Only meaningful when encoding.

3.8.2.29 float steganos_state_t::variation_limit[VORBIS_MAX_BLOCK][2]

Variation limit per residual line. Stores the [-, +] maximum variations to introduce in each residual value. Changes from one frame and channel to another.

3.8.2.30 vorbis_config_t∗ steganos_state_t::vc

Vorbis configuration data needed during the steganographic protocol.

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

- include/steganos_types.h

3.9 vorbis_config_t Struct Reference

Stores the needed information about the Vorbis block and look configuration. Is used to avoid circular dependencies.

```
#include <global_types.h>
```

Public Attributes

- int rate
- int pcmend
- int mult
- int ∗ postlist
- int ∗ forward_index
- int posts_len

3.9.1 Detailed Description

Stores the needed information about the Vorbis block and look configuration. Is used to avoid circular dependencies.

3.9.2 Member Data Documentation

3.9.2.1 int∗ vorbis_config_t::forward_index

3.9.2.2 int vorbis_config_t::mult

3.9.2.3 int vorbis_config_t::pcmend

3.9.2.4 int∗ vorbis_config_t::postlist

3.9.2.5 int vorbis_config_t::posts_len

3.9.2.6 int vorbis_config_t::rate

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

- include/global_types.h

Chapter 4

File Documentation

4.1 include/cryptos_types.h File Reference

```
#include <gcrypt.h>
#include "global_types.h"
```

Classes

- struct cryptos_key_t

  Defines the key structure to be used in the cryptographic layer.
- struct cryptos_protocol_buffer_t
- struct cryptos_config_t

  Defines the global configuration of the cryptographic protocol.

Defines

- #define I_CRYPTOS_OK 0

  Error code for 'No error' to use in functions of integer return type.
- #define I_CRYPTOS_ERR 1

  Error code for 'Error occured' to use in functions of integer return type.
- #define I_CRYPTOS_CHECK_FAIL 2

  Error code for 'Integrity check failed'.

4.1.1 Define Documentation

4.1.1.1 #define I_CRYPTOS_CHECK_FAIL 2

Error code for 'Integrity check failed'.

4.1.1.2 #define I_CRYPTOS_ERR 1

Error code for 'Error occured' to use in functions of integer return type.

4.1.1.3 #define I_CRYPTOS_OK 0

Error code for 'No error' to use in functions of integer return type.

4.2 include/global_types.h File Reference

```
#include <stdint.h>
```

Classes

- struct vorbis_config_t

  Stores the needed information about the Vorbis block and look configuration. Is used to avoid circular dependencies.

Defines

- #define I_OK 0

  Error code for 'No error' to use in functions of integer return type.

- #define I_ERR 1

  Error code for 'Error occured' to use in functions of integer return type.

- #define BITS_PER_BYTE 8

  Modify for different byte sizes.

Typedefs

- typedef unsigned char byte

4.2.1 Define Documentation

4.2.1.1 #define BITS_PER_BYTE 8

Modify for different byte sizes.

4.2.1.2 #define I_ERR 1

Error code for 'Error occured' to use in functions of integer return type.

4.2.1.3 #define I_OK 0

Error code for 'No error' to use in functions of integer return type.

4.2.2 Typedef Documentation

4.2.2.1 typedef unsigned char byte

4.3 include/numbers.h File Reference

```
#include "global_types.h"
```

Classes

- struct prng_t

  Defines the structure to use as PRNG abstraction.

Functions

- int prng_init (prng_t ∗prng)

  Initializes the PRNG.

- int prng_free (prng_t ∗prng)

  Sets the seed to use to the seed, of size bits.

- int prng_reset_byte (prng_t ∗prng, byte ∗seed, int size, long int iters)

  Recalculate the state of the given PRNG seeded with seed and used prng->iters times.

- int prng_set_seed_byte (prng_t ∗prng, const byte ∗seed, const int size)

  Sets the seed to use to the seed, of size bits.

- int prng_set_seed_int (prng_t ∗prng, const int seed)

  Sets the seed to use to the seed.

- int prng_get_random_int (prng_t ∗prng, const int modulo, int ∗r)

  Feeds r with a pseudo random integer.

- int linear_interpolation_y (const float x1, const float y1, const float x2, const float y2, const float x, float ∗y)

  Stores the result of the lineal interpolation between the points [x1,y1] and [x2,y2] in [x,y].

- int seek (int ∗sequence, int seq_len, int number, int ∗exist)

  Looks for number in sequence, updating exist in consequence.

- int seek_interval (int ∗sequence, int seq_len, int lower, int higher, int ∗exist)

  Looks if there is any number in sequence belonging to the interval defined by [lower, higher], updating exist in consequence.

4.3.1 Function Documentation

4.3.1.1 int linear_interpolation_y (const float *x1*, const float *y1*, const float *x2*, const float *y2*, const float *x*, float ∗ *y*)

Stores the result of the lineal interpolation between the points [x1,y1] and [x2,y2] in [x,y]. Given [x1, y1], [x2, y2] and x, stores the result of the lineal interpolation between the points [x1,y1] and [x2,y2] in [x,y].

Parameters:

$\leftarrow$ ***x1*** The first point x-coordinate.

$\leftarrow$ ***y1*** The first point y-coordinate.

$\leftarrow$ ***x2*** The second point x-coordinate.

$\leftarrow$ ***y2*** The second point y-coordinate.

$\leftarrow$ ***x*** The x coordinate of the new point.

$\rightarrow$ ***y*** Will store the resulting new point y-coordinate.

Returns:

The corresponding error code for integer returning functions, i.e., I_OK if no error was present and I_ERR if an error occured with errno updated.

Return values:

I_OK with errno = 0 (No error).

I_ERR with errno = EINVAL (invalid argument).

4.3.1.2 int prng_free (prng_t ∗ *prng*)

Sets the seed to use to the *seed*, of *size* bits. Frees the memory allocated for the given PRNG

Parameters:

$\leftarrow$ ***prng*** PRNG to free.

Returns:

The corresponding error code for integer returning functions, i.e., I_OK if no error was present and I_ERR if an error occured with errno updated.

Return values:

I_OK with errno = 0 (No error).

4.3.1.3 int prng_get_random_int (prng_t ∗ *prng*, const int *modulo*, int ∗ *r*)

Feeds *r* with a pseudo random integer. Obtains a new pseudo random number and stores it in *r*.

Parameters:

$\leftarrow$ ***prng*** PRNG abstraction pointer.

$\leftarrow$ ***modulo*** The pseudo random number generated will take a value between 0 and *modulo-1*. If *modulo* equals 0, the function won't apply any modular arithmetic.

$\rightarrow$ ***r*** Will store the pseudo random number generated.

Returns:

The corresponding error code for integer returning functions, i.e., I_OK if no error was present and I_ERR if an error occured with errno updated.

Return values:

I_OK with errno = 0 (No error).

I_ERR with errno = EINVAL (invalid argument).

4.3.1.4 int prng_init (prng_t * *prng*)

Initializes the PRNG. Allocates for the given PRNG

Parameters:

$\leftarrow$ ***prng*** PRNG to allocate.

Returns:

The corresponding error code for integer returning functions, i.e., I_OK if no error was present and I_ERR if an error occured with errno updated.

Return values:

I_OK with errno = 0 (No error).

I_ERR with errno = EINVAL (invalid argument).

4.3.1.5 int prng_reset_byte (prng_t * *prng*, byte * *seed*, int *size*, long int *iters*)

Recalculate the state of the given PRNG seeded with *seed* and used *prng->iters* times. Recalculates the state of the PRNG *prng* after being seeded with *seed* and iterated *iters* times.

Parameters:

$\leftarrow$ ***prng*** The PRNG.

$\leftarrow$ ***seed*** The seed used to initialize prng.

$\leftarrow$ ***size*** The size of seed, in bits.

$\leftarrow$ ***iters*** The iterations to "discard".

Returns:

The corresponding error code for integer returning functions, i.e., I_OK if no error was present and I_ERR if an error occured with errno updated.

Return values:

I_OK with errno = 0 (No error).

I_ERR with errno = EINVAL (invalid argument).

4.3.1.6 int prng_set_seed_byte (prng_t * *prng*, const byte * *seed*, const int *size*)

Sets the seed to use to the *seed*, of *size* bits. Sets the seed to use for generating [CS]PRN sequences to *seed*, which will have *size* bits of length.

Parameters:

$\leftarrow$ ***prng*** PRNG abstraction pointer.

$\leftarrow$ ***seed*** The seed to use, in a byte representation

$\leftarrow$ ***size*** The size of the seed, in bits

Returns:

The corresponding error code for integer returning functions, i.e., I_OK if no error was present and I_ERR if an error occured with errno updated.

Return values:

I_OK with errno = 0 (No error).

I_ERR with errno = EINVAL (invalid argument).

4.3.1.7 int prng_set_seed_int (prng_t ∗ *prng*, const int *seed*)

Sets the seed to use to the *seed*. Sets the seed to use for generating [CS]PRN sequences to *seed*, which will have *size* bits of length.

Parameters:

← ***prng*** PRNG abstraction pointer.

← ***seed*** The seed to use, in an int representation

Returns:

The corresponding error code for integer returning functions, i.e., I_OK if no error was present and I_ERR if an error occured with errno updated.

Return values:

I_OK with errno = 0 (No error).

I_ERR with errno = EINVAL (invalid argument).

4.3.1.8 int seek (int ∗ *sequence*, int *seq_len*, int *number*, int ∗ *exist*)

Looks for *number* in *sequence*, updating *exist* in consequence. Looks for *number* in *sequence*. After a successful run, *exist* will equal to 1 if the number exists and 0 otherwise.

Parameters:

← ***sequence*** A list of numbers.

← ***seq_len*** The number of elements in sequence.

← ***number*** The number to look for.

→ ***exist*** Will be 1 if the number exist and 0 otherwise, in a successful run of the function.

Returns:

The corresponding error code for integer returning functions, i.e., I_OK if no error was present and I_ERR if an error occured with errno updated.

Return values:

I_OK with errno = 0 (No error).

I_ERR with errno = EINVAL (invalid argument).

4.3.1.9 int seek_interval (int $*$ *sequence*, int *seq_len*, int *lower*, int *higher*, int $*$ *exist*)

Looks if there is any number in *sequence* belonging to the interval defined by [*lower, higher*], updating *exist* in consequence. Looks if there is any number in *sequence* belgoning to the interval defined by [*lower, higher*]. After a successful run, *exist* will equal to 1 if the number exists and 0 otherwise.

Parameters:

$\leftarrow$ ***sequence*** A list of numbers.

$\leftarrow$ ***seq_len*** The number of elements in sequence.

$\leftarrow$ ***lower*** The lower end of the interval.

$\leftarrow$ ***higher*** The higher end of the interval.

$\rightarrow$ ***exist*** Will be 1 if the number exist and 0 otherwise, in a successful run of the function.

Returns:

The corresponding error code for integer returning functions, i.e., I_OK if no error was present and I_ERR if an error occured with errno updated.

Return values:

I_OK with errno = 0 (No error).

I_ERR with errno = EINVAL (invalid argument).

4.4 include/steganos_types.h File Reference

```
#include "global_types.h"
#include "numbers.h"
```

Classes

- struct steganos_key_t

  Defines the key structure to be used in the steganographic layer.

- struct iss_cfg_t
- struct steganos_state_t

  Defines the global internal state of the steganosgraphic protocol.

Defines

- #define I_STEGANOS_OK I_OK

  Error code for 'No error' to use in functions of integer return type.

- #define I_STEGANOS_ERR I_ERR

  Error code for 'Error occured' to use in functions of integer return type.

- #define I_STEGANOS_SYNC_FAIL 2

  Error code for 'Unable to synchronize', used when a synhcornization method is unable to synchronize due to is own nature, i.e., there is no computation error.

- #define I_STEGANOS_FRAME_SKIP 3

  Return ocde used to specify we're not using the current frame to hide data. This can be due, for example, to a short subliminal capacity of the frame or to "imperceptibility" reasons.

- #define INDETERMINATE_MARK -1

  Value used when there is not enough statistical evidence of the mark hided in the Vorbis's posts vector being 0 or 1. Used only with ISS synchronization method.

- #define VORBIS_MAX_BLOCK 8192

  Defines the Vorbis maximum number of samples per Vorbis block.

- #define VIF_POSIT 63

Typedefs

- typedef int(∗ hide_method)(byte ∗plain_mess, int p_m_len, int ∗floor, float ∗res, int res_len, steganos_key_t ∗key, byte ∗sub_mess, int ∗s_m_len, prng_t ∗prng)

  Type definition for hiding functions to use in the residual subliminal channel.

- typedef int(∗ sync_method)(vorbis_config_t ∗vc, steganos_key_t ∗key, int ∗posts, float ∗residue, void ∗cfg, int decoding, int ∗bit, prng_t ∗prng)

  Type definition for synchronization functions to use in the protocol.

- typedef int(∗ desync_method)(vorbis_config_t ∗vc, int ∗res_lineup, int ∗posts, int ∗floor, float ∗residue, void ∗cfg, steganos_key_t ∗hiding_key, hide_method hide, prng_t ∗prng)

  Type definition for desynchronization functions to use in the protocol.

Enumerations

- enum synchro_method_et { RES_HEADER = 0, ISS = 1, FORCED_RES_HEADER = 2 }

  Enumeration for the different synchronization modes available.

- enum hide_method_et { DIRECT_HIDING = 0, PARITY_BITS = 1 }

  Enumeration for the different methods of hiding data in the residues.

4.4.1 Define Documentation

4.4.1.1 #define I_STEGANOS_ERR I_ERR

Error code for 'Error occured' to use in functions of integer return type.

4.4.1.2 #define I_STEGANOS_FRAME_SKIP 3

Return ocde used to specify we're not using the current frame to hide data. This can be due, for example, to a short subliminal capacity of the frame or to "imperceptibility" reasons.

4.4.1.3 #define I_STEGANOS_OK I_OK

Error code for 'No error' to use in functions of integer return type.

4.4.1.4 #define I_STEGANOS_SYNC_FAIL 2

Error code for 'Unable to synchronize', used when a synhcornization method is unable to synchronize due to is own nature, i.e., there is no computation error.

4.4.1.5 #define INDETERMINATE_MARK -1

Value used when there is not enough statistical evidence of the mark hided in the Vorbis's posts vector being 0 or 1. Used only with ISS synchronization method.

4.4.1.6 #define VIF_POSIT 63

4.4.1.7 #define VORBIS_MAX_BLOCK 8192

Defines the Vorbis maximum number of samples per Vorbis block.

4.4.2 Typedef Documentation

4.4.2.1 int(∗ desync_method)(vorbis_config_t ∗vc, int ∗res_lineup, int ∗posts, int ∗floor, float ∗residue, void ∗cfg, steganos_key_t ∗hiding_key, hide_method hide, prng_t ∗prng)

Type definition for desynchronization functions to use in the protocol. Functions of this type shall solve the false positive synchronization problem that may happen depending on the synchro_method used. Depending on the specific method in use, the *floor* and *residue* vectors will or won't be used.

Parameters:

← ***vorbis_config_t*** Vorbis configuration data.

← ***res_lineup*** Residue ordering

↔ ***posts*** Current floor posts vector. May be subject to changes depending on the concrete method used. See synchronization methods.

↔ ***floor*** Current floor vector. May be subject to changes depending on the concrete method used. See synchronization methods.

↔ ***residue*** Current residue vector. May be subject to changes depending on the concrete method used. See synchronization methods.

← ***cfg*** This parameter will be used to pass any argument needed in a concrete synchronization method.

← ***hiding_key*** Key for hiding

← ***hide*** Hiding method (pointer to function) in use in the current frame and channel.

← ***prng*** PRNG in use

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

synchro_method_et
iss_synchronization

4.4.2.2 int(∗ hide_method)(byte ∗plain_mess, int p_m_len, int ∗floor, float ∗res, int res_len, steganos_key_t ∗key, byte ∗sub_mess, int ∗s_m_len, prng_t ∗prng)

Type definition for hiding functions to use in the residual subliminal channel. Functions of this type shall produce a stego message of the same length as the subliminal message, and may or may not use the cover object and/or the key during the process. In case the cover object and/or the key aren't necessary, the corresponding byte ∗ parameters representing them must be NULL and their corresponding int's be 0.

Parameters:

← ***plain_mess*** Plain message to hide.

← ***p_m_len*** Length of plain message in bits.

← ***floor*** Floor vector.

← ***floor_len*** The length of the floor vector (actually, the floor posts in Vorbis).

← ***res*** Residue vector.

← ***res_len*** Number of elements in *floor* and *res* vectors.

← ***key*** Key to use.

↔ ***sub_mess*** String in which the subliminal message will be stored. It has to be allocated previously to calling the function.

↔ ***s_m_len*** At the input, will store the allocated size (in bits) of *sub_mess*. At the output, the length of the resulting sub_message, in bits (it may be different than p_m_len as it may include de-synchronization bits).

← ***prng*** PRNG in use.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

4.4.2.3 int(∗ sync_method)(vorbis_config_t ∗vc, steganos_key_t ∗key, int ∗posts, float ∗residue, void ∗cfg, int decoding, int ∗bit, prng_t ∗prng)

Type definition for synchronization functions to use in the protocol. Functions of this type shall solve the synchronization problem between sender and receiver using the "both-known" data (stored in *ss* and the floor and residue vectors, both of length *res_len*. Depending on the specific method use, the *floor* and *residue* vectors will or won't be used.

Parameters:

← ***vc*** Vorbis block and look information.

← ***key*** Key to use

↔ ***posts*** Current floor posts vector. May be subject to changes depending on the concrete method used. See synchronization methods.

↔ ***residue*** Current residue vector. May be subject to changes depending on the concrete method used. See synchronization methods.

← ***cfg*** This parameter will be used to pass any argument needed in a concrete synchronization method.

← ***decoding*** Active if the function is called from the decoder's side,

↔ ***bit*** At the input, the desired bit to include if we are at the encoder's side; at the ouput, the received bit if we are at the decoder's side (i.e. 1 if we're telling the receiver there is hidden data and 0 if not).

← ***prng*** PRNG in use

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

I_STEGANOS_SYNC_FAIL with errno = 0 (Unable to synchronize).

See also:

synchro_method_et
iss_synchronization

4.4.3 Enumeration Type Documentation

4.4.3.1 enum hide_method_et

Enumeration for the different methods of hiding data in the residues.

Enumerator:

DIRECT_HIDING Just write the data in the given locations.

PARITY_BITS Applies the parity method by Anderson and Petitcolas.

4.4.3.2 enum synchro_method_et

Enumeration for the different synchronization modes available.

Enumerator:

RES_HEADER Classic mode: synchronization header in residue subliminal channel

ISS Improved Spread Spectrum algorithm within floor vector (uses synchro_key)

FORCED_RES_HEADER Used when ISS is the default option but for some reason we can't use it at a given moment, and also to prevent false negatives at the decoder's side.

4.5 lib/cryptos_channel.c File Reference

4.6 lib/cryptos_channel.h File Reference

```
#include "cryptos_types.h"
```

Defines

- #define CRYPTOS_SYNC_HEADER_LEN 3

  Defines the number of bytes of the SYNC header field.

- #define CRYPTOS_LENGTH_HEADER_LEN 4

  Defines the number of bytes of the "length" header field.

- #define CRYPTOS_IV_HEADER_LEN 16

  Defines the number of bytes of the IV header field.

- #define CRYPTOS_EMISSION_HEADER_LEN sizeof(uint64_t)

  Defines the number of bytes of the EMISSION header field.

- #define CRYPTOS_PACKET_HEADER_LEN sizeof(uint64_t)

  Defines the number of bytes of the PACKET header field.

- #define CRYPTOS_MAX_DATA 0xFFFFFFFF

  Defines the maximum number of bytes of the data field, which will depend on the CRYPTOS_LENGTH_-HEADER_LEN bytes.

- #define CRYPTOS_DEFAULT_DATA_SIZE 512

  Defines the default size of the data field in crypto packets.

- #define CRYPTOS_MIN_DIGEST_LEN 3

  Defines the size of the message digest algorithm which produces the minimum length digest, in bytes. Currently, CRC24.

- #define CRYPTOS_MAX_DIGEST_LEN 64

  Defines the size of the message digest algorithm which produces the maximum length digest, in bytes. Currently, SHA512.

- #define RATIO_DD 16

  Defines the number of bytes of data per byte of message digest.

- #define CRYPTOS_BUFFER_MAX_SIZE CRYPTOS_PACKET_MAX_LEN∗2

  The max size, in bytes, of the cryptos buffer. It is set to twice the size of a cryptos packet because it'll never store two whole packets.

Functions

- int cryptos_buffer_init (cryptos_protocol_buffer_t ∗cb, int fd, size_t size)

  Initializes cryptographic layer buffer.

- int cryptos_buffer_free (cryptos_protocol_buffer_t ∗cb)

  Frees the resources allocated for the cryptographic layer buffer.

- int cryptos_config_init (cryptos_config_t ∗cc, int cipher_algo, byte ∗key, int keylen, int md_algo, int hmac, byte ∗iv, int ivlen, uint64_t emission, uint64_t packet, uint64_t default_data_size)

  Initializes the internal variables of the cryptos_config_t structure needed during the protocol.

- int cryptos_config_free (cryptos_config_t ∗cc)

  Frees any internal variable of the structure.

- int cryptos_config_set_key (cryptos_config_t ∗cc, cryptos_key_t ∗key)

  Sets the key attribute of cryptos_config_t to the key specified, freeing the previous iv when necessary.

- int cryptos_config_set_iv (cryptos_config_t ∗cc, cryptos_key_t ∗iv)

  Sets the key attribute of cryptos_config_t to the key specified, freeing the previous iv when necessary.

- int produce_packet (cryptos_config_t ∗cc, byte ∗data, uint64_t d_len, byte ∗packet, uint64_t p_len, uint64_t ∗w_data)

  Produces a new packet ready to be sent.

- int parse_packet (cryptos_config_t ∗cc, byte ∗packet, uint64_t p_len, byte ∗data, uint64_t ∗d_len, uint64_t ∗r_data)

  Parses the received data, returning the data in it.

- int cryptos_key_init (byte ∗byte_key, const int size, cryptos_key_t ∗key)

  Allocates memory for the key fields and sets them to the received values.

- int cryptos_key_free (cryptos_key_t ∗key)

  Frees the memory allocated for the internal variables of the given cryptos_key.

- int cryptos_cipher_algo_code (const char ∗cipher_algo_name, int ∗code)

  Returns the GCRY code for the algorithm with cipher_algo_name name. If the specified name is NULL, returns the code for ARCFOUR, which is the default cipher algorithm in this system.

- int cryptos_md_algo_code (const char ∗md_algo_name, int ∗code)

  Returns the GCRY code for the algorithm with md_algo_name name. If the specified name is NULL, returns the code for SHA1, which is the default message digest algorithm in this system.

4.6.1 Define Documentation

4.6.1.1 #define CRYPTOS_BUFFER_MAX_SIZE CRYPTOS_PACKET_MAX_LEN∗2

The max size, in bytes, of the cryptos buffer. It is set to twice the size of a cryptos packet because it'll never store two whole packets.

4.6.1.2 #define CRYPTOS_DEFAULT_DATA_SIZE 512

Defines the default size of the data field in crypto packets.

4.6.1.3 #define CRYPTOS_EMISSION_HEADER_LEN sizeof(uint64_t)

Defines the number of bytes of the EMISSION header field.

4.6.1.4 #define CRYPTOS_IV_HEADER_LEN 16

Defines the number of bytes of the IV header field.

4.6.1.5 #define CRYPTOS_LENGTH_HEADER_LEN 4

Defines the number of bytes of the "length" header field.

4.6.1.6 #define CRYPTOS_MAX_DATA 0xFFFFFFFF

Defines the maximum number of bytes of the data field, which will depend on the CRYPTOS_LENGTH_-HEADER_LEN bytes.

4.6.1.7 #define CRYPTOS_MAX_DIGEST_LEN 64

Defines the size of the message digest algorithm which produces the maximum length digest, in bytes. Currently, SHA512.

4.6.1.8 #define CRYPTOS_MIN_DIGEST_LEN 3

Defines the size of the message digest algorithm which produces the minimum length digest, in bytes. Currently, CRC24.

4.6.1.9 #define CRYPTOS_PACKET_HEADER_LEN sizeof(uint64_t)

Defines the number of bytes of the PACKET header field.

4.6.1.10 #define CRYPTOS_SYNC_HEADER_LEN 3

Defines the number of bytes of the SYNC header field.

4.6.1.11 #define RATIO_DD 16

Defines the number of bytes of data per byte of message digest.

4.6.2 Function Documentation

4.6.2.1 int cryptos_buffer_free (cryptos_protocol_buffer_t ∗ *cb*)

Frees the resources allocated for the cryptographic layer buffer.

Parameters:

← ***cb*** The cryptographic layer buffer structure to free

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

I_CRYPTOS_ERR with errno = EINVAL (invalid argument).

I_CRYPTOS_ERR with errno != 0 && errno != EINVAL (see corresponding error code)

4.6.2.2 int cryptos_buffer_init (cryptos_protocol_buffer_t ∗ *cb*, int *fd*, size_t *size*)

Initializes cryptographic layer buffer. Initializes the internal variables of the cryptos_config_t structure needed during the protocol.

Parameters:

← ***cb*** The cryptographic layer buffer structure to initialize

← ***fd*** The file descriptor from which the cryptographic layer will retrieve the data to produce crypto packets

← ***size*** The desired size of the buffer, in bytes.

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

I_CRYPTOS_ERR with errno = EINVAL (invalid argument).

I_CRYPTOS_ERR with errno != 0 && errno != EINVAL (see corresponding error code)

4.6.2.3 int cryptos_cipher_algo_code (const char ∗ *cipher_algo_name*, int ∗ *code*)

Returns the GCRY code for the algorithm with *cipher_algo_name* name. If the specified name is NULL, returns the code for ARCFOUR, which is the default cipher algorithm in this system.

Parameters:

← ***cipher_algo_name*** The name of the ciphering algorithm, in char∗ representation.

→ ***code*** Variable to store the corresponding code.

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

4.6.2.4 int cryptos_config_free (cryptos_config_t ∗ *cc*)

Frees any internal variable of the structure.

Parameters:

cc The crytpos config structure to free.

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

I_CRYPTOS_ERR with errno = EINVAL (invalid argument).

4.6.2.5 int cryptos_config_init (cryptos_config_t ∗ *cc*, int *cipher_algo*, byte ∗ *key*, int *keylen*, int *md_algo*, int *hmac*, byte ∗ *iv*, int *ivlen*, uint64_t *emission*, uint64_t *packet*, uint64_t *default_data_size*)

Initializes the internal variables of the cryptos_config_t structure needed during the protocol. Initializes the internal variables of the cryptos_config_t structure needed during the protocol.

Parameters:

← ***cc*** The cryptos_config_t structure to initialize

← ***cipher_algo*** The cipher algorithm to use

← ***key*** The key to use, must have at least 16 bytes

← ***keylen*** Length of key, in bytes

← ***md_algo*** The message digest algorithm

← ***hmac*** Boolean. When set, indicates the hmac variant of the md_algo wants to be used (if possible)

← ***iv*** Initialization vector. It is not a key, but uses the same structure. Must have a length of 16 bytes (128 bits). Will be NULL at the receiver's side.

← ***ivlen*** Length of iv, in bytes. Will be 0 at the receiver's side.

← ***emission*** Emission id

← ***packet*** Packet id initial value. Will be 1 at the receiver's side.

← ***default_data_size*** Default data size, in bytes, to send in each packet.

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

I_CRYPTOS_ERR with errno = EINVAL (invalid argument).

I_CRYPTOS_ERR with errno != 0 && errno != EINVAL (see corresponding error code)

See also:

cryptos_config_t

4.6.2.6 int cryptos_config_set_iv (cryptos_config_t ∗ *cc*, cryptos_key_t ∗ *iv*)

Sets the key attribute of cryptos_config_t to the key specified, freeing the previous iv when necessary.

Parameters:

cc The crytpos config structure

iv The IV

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

I_CRYPTOS_ERR with errno = EINVAL (invalid argument).

4.6.2.7 int cryptos_config_set_key (cryptos_config_t ∗ *cc*, cryptos_key_t ∗ *key*)

Sets the key attribute of cryptos_config_t to the key specified, freeing the previous iv when necessary.

Parameters:

cc The crytpos config structure

key The key

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

I_CRYPTOS_ERR with errno = EINVAL (invalid argument).

4.6.2.8 int cryptos_key_free (cryptos_key_t ∗ *key*)

Frees the memory allocated for the internal variables of the given cryptos_key.

Parameters:

← *key* A pointer to the key to free.

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

4.6.2.9 int cryptos_key_init (byte ∗ *byte_key*, const int *size*, cryptos_key_t ∗ *key*)

Allocates memory for the key fields and sets them to the received values. Allocates memory for the key fields and sets them to the received values.

Parameters:

← ***byte_key*** The key value.

← ***size*** The size of the *byte_key* argument, in bytes.

↔ ***key*** A pointer to a previously allocated cryptos_key_t variable in which we'll store the key's members.

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

I_CRYPTOS_ERR with errno = EINVAL (invalid argument).

I_CRYPTOS_ERR with errno = ERANGE (invalid key size).

4.6.2.10 int cryptos_md_algo_code (const char ∗ *md_algo_name*, int ∗ *code*)

Returns the GCRY code for the algorithm with *md_algo_name* name. If the specified name is NULL, returns the code for SHA1, which is the default message digest algorithm in this system.

Parameters:

← ***md_algo_name*** The name of the digest algorithm, in char∗ representation.

→ ***code*** Variable to store the corresponding code.

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

4.6.2.11 int parse_packet (cryptos_config_t ∗ *cc*, byte ∗ *packet*, uint64_t *p_len*, byte ∗ *data*, uint64_t ∗ *d_len*, uint64_t ∗ *r_data*)

Parses the received data, returning the data in it. Parses the packet *packet*, of *p_len* bytes, using the cryptos context defined by *cc*, verifying it's integrity and that matches with the expected emission and packet IDs. If everything goes OK, the recoverd data will be stored in *data*, of *d_len* allocated bytes, which will be reallocated (and *d_len* updated consequently) if more space is needed. The total amount of pure data recoverd, in bytes, will be stored in *r_data*.

A packet will have the structure depicted below:

-- | SYNC | DATA_LENGTH | IV | EMISSION_ID | PACKET_ID |.. DATA .. | DIGEST | ---

- The field SYNC will be equal to CRYPTOS_SYNC_HEADER and will be of CRYPTOS_SYNC_-HEADER_LEN bytes long.
- The field DATA_LENGTH will mark the length of the field DATA and will be of CRYPTOS_-LENGTH_HEADER_LEN bytes long.
- The field IV will be of CRYPTOS_IV_LEN bytes
- The field EMISSION_ID will be CRYPTOS_EMISSION_LEN bytes long
- The field PACKET_ID will be CRYPTOS_PACKET_LEN bytes long
- The field DATA will have the size marked in the DATA_LENGTH field
- The DIGEST field will have a variable size, depending on the digest algorithm in use, and will vary from 24 bits (CRC24) to 64 bytes (SHA512) although high sizes are not likely to happen.

Parameters:

cc Cryptos context.

packet Packet to parse

p_len Allocated size, in bytes, for *packet*

data Array to store the recovered data in

d_len Allocated size, in bytes, for *data*

r_data Final successfully recovered amount of data, in bytes.

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

I_CRYPTOS_ERR with errno = EINVAL (invalid argument).

I_CRYPTOS_ERR with errno != 0 && errno != EINVAL (see corresponding error code)

4.6.2.12 int produce_packet (cryptos_config_t ∗ *cc*, byte ∗ *data*, uint64_t *d_len*, byte ∗ *packet*, uint64_t *p_len*, uint64_t ∗ *w_data*)

Produces a new packet ready to be sent. Prepares a new packet ready to be sent. The data field will be ciphered and inserted in the packet. The header fields and integrity checks will be generated accordingly to the cryptos configuration received, and, upon successful execution, the packet id will be increased for the next packet.

A packet will have the structure depicted below:

--- | SYNC | DATA_LENGTH | IV | EMISSION_ID | PACKET_ID |.. DATA .. | DIGEST | ---

- The field SYNC will be equal to CRYPTOS_SYNC_HEADER and will be of CRYPTOS_SYNC_-HEADER_LEN bytes long.

- The field DATA_LENGTH will mark the length of the field DATA and will be of CRYPTOS_-LENGTH_HEADER_LEN bytes long.
- The field IV will be of CRYPTOS_IV_LEN bytes
- The field EMISSION_ID will be CRYPTOS_EMISSION_LEN bytes long
- The field PACKET_ID will be CRYPTOS_PACKET_LEN bytes long
- The field DATA will have the size marked in the DATA_LENGTH field and will be the only field ciphered.
- The DIGEST field will have a variable size, depending on the digest algorithm in use, and will vary from 24 bits (CRC24) to 64 bytes (SHA512) although high sizes are not likely to happen. The digest includes the whole packet except the SYNC field.

Parameters:

cc Cryptos config structure

data The (plain) data to write

d_len The length of *data*, in bytes

packet The byte array in which store the result

p_len The memory allocated for *packet*, in bytes.

w_data The number of data bytes successfuly included in the current packet. Note that the amount of received plain data can exceed the amount of data that can be included in the current packet.

Returns:

The corresponding error code for integer returning functions, i.e., I_CRYPTOS_OK if no error was present and I_CRYPTOS_ERR if an error occured with errno updated.

Return values:

I_CRYPTOS_OK with errno = 0 (No error).

I_CRYPTOS_ERR with errno = EINVAL (invalid argument).

I_CRYPTOS_ERR with errno != 0 && errno != EINVAL (see corresponding error code)

4.7 lib/miscellaneous.c File Reference

4.8 lib/miscellaneous.h File Reference

```
#include "steganos_types.h"
#include "global_types.h"
```

Defines

- #define I_MISC_OK 0

  Error code for 'No error' to use in functions of integer return type.

- #define I_MISC_ERR 1

  Error code for 'Error occured' to use in functions of integer return type.

- #define I_MISC_INVALID_XML -1

  Error code for 'Invalid xml file' while validating an xml file.

- #define XML_KEY "key"

  String representing the field 'key' in xml config. files.

- #define XML_KEY_LENGTH "length"

  String representing the field 'length' within key elements in xml config. files.

- #define XML_KEY_BITSTREAM "bitstream"

  String representing the field 'bitstream' within key elements in xml config. files.

- #define XML_SYNCHROMODE "synchromode"

  String representing the field 'synchromode' in xml config. files.

- #define XML_HIDEMETHOD "hidemethod"

  String representing the field 'hidemethod' in xml config. files.

- #define XML_AGGRESS "aggressiveness"

  String representing the field 'agressiveness' in xml config. files.

- #define XML_LEAFS 5

  Number of "leaf nodes" (i.e. elements of type other than child) in the steganos DTD.

Functions

- int to_byte (void ∗stream, size_t nmemb, size_t size, byte ∗byte_stream)

  Converts the given pointer of generic type to a byte pointer.

- int print_bytestream (int fd, byte ∗stream, int length)

  Prints a bytestream in fd.

- int bitstream_rol (byte ∗bitstream, int length, int rot)

  Rotates bitstream rot bits to the left.

- int bitstream_ror (byte ∗bitstream, int length, int rot)

  Rotates bitstream rot bits to the right.

- int message_log (const char ∗caller, const char ∗message)

  Writes the specified message in the log file log_<pid>.

- int zlib_def (FILE ∗source, FILE ∗dest, int level)
- int zlib_inf (FILE ∗source, FILE ∗dest)
- void zlib_zerr (int ret)

4.8.1 Define Documentation

4.8.1.1 #define I_MISC_ERR 1

Error code for 'Error occured' to use in functions of integer return type.

4.8.1.2 #define I_MISC_INVALID_XML -1

Error code for 'Invalid xml file' while validating an xml file.

4.8.1.3 #define I_MISC_OK 0

Error code for 'No error' to use in functions of integer return type.

4.8.1.4 #define XML_AGGRESS "aggressiveness"

String representing the field 'agressiveness' in xml config. files.

4.8.1.5 #define XML_HIDEMETHOD "hidemethod"

String representing the field 'hidemethod' in xml config. files.

4.8.1.6 #define XML_KEY "key"

String representing the field 'key' in xml config. files.

4.8.1.7 #define XML_KEY_BITSTREAM "bitstream"

String representing the field 'bitstream' within key elements in xml config. files.

4.8.1.8 #define XML_KEY_LENGTH "length"

String representing the field 'length' within key elements in xml config. files.

4.8.1.9 #define XML_LEAFS 5

Number of "leaf nodes" (i.e. elements of type other than child) in the steganos DTD.

4.8.1.10 #define XML_SYNCHROMODE "synchromode"

String representing the field 'synchromode' in xml config. files.

4.8.2 Function Documentation

4.8.2.1 int bitstream_rol (byte ∗ *bitstream*, int *length*, int *rot*)

Rotates bitstream rot bits to the left.

Parameters:

↔ ***bitstream*** The bit stream to rotate at the input, the rotated resulting bitstream at the output.

← ***length*** The length of bitstream, in bits.

← ***rot*** The number of bits to rotate to the left.

Returns:

The corresponding error code for integer returning functions, i.e., I_MISC_OK if no error was present and I_MISC_ERR if an error occured with errno updated.

Return values:

I_MISC_OK with errno = 0 (No error).

I_MISC_ERR if unable to open log file, with errno updated to consequently (see fopen man page).

4.8.2.2 int bitstream_ror (byte ∗ *bitstream*, int *length*, int *rot*)

Rotates bitstream rot bits to the right.

Parameters:

↔ ***bitstream*** The bit stream to rotate at the input, the rotated resulting bitstream at the output.

← ***length*** The length of bitstream, in bits.

← ***rot*** The number of bits to rotate to the right.

Returns:

The corresponding error code for integer returning functions, i.e., I_MISC_OK if no error was present and I_MISC_ERR if an error occured with errno updated.

Return values:

I_MISC_OK with errno = 0 (No error).

I_MISC_ERR if unable to open log file, with errno updated to consequently (see fopen man page).

4.8.2.3 int message_log (const char ∗ *caller*, const char ∗ *message*)

Writes the specified message in the log file log_<pid>.

Parameters:

← ***caller*** Name of the calling function.

← ***message*** The message to write in the log.

Returns:

The corresponding error code for integer returning functions, i.e., I_MISC_OK if no error was present and I_MISC_ERR if an error occured with errno updated.

Return values:

I_MISC_OK with errno = 0 (No error).

I_MISC_ERR if unable to open log file, with errno updated to consequently (see fopen man page).

4.8.2.4 int print_bytestream (int *fd*, byte ∗ *stream*, int *length*)

Prints a bytestream in fd.

Parameters:

← ***fd*** The file descriptor to print in. -1 to write in the system log

← ***stream*** The stream to print.

← ***length*** The length of *stream* in bytes.

Returns:

The corresponding error code for integer returning functions, i.e., I_MISC_OK if no error was present and I_MISC_ERR if an error occured with errno updated.

Return values:

I_MISC_OK with errno = 0 (No error).

I_MISC_ERR with errno = EINVAL (Invalid argument).

4.8.2.5 int to_byte (void ∗ *stream*, size_t *nmemb*, size_t *size*, byte ∗ *byte_stream*)

Converts the given pointer of generic type to a byte pointer. Converts the pointer *stream*, composed by *nmemb* elements of size *size* bytes each into a byte pointer, stored in *byte_stream*. Uses big endian ordering.

Parameters:

← ***stream*** The void pointer to convert.

← ***nmemb*** The number of elements in *stream*.

← ***size*** The size, in bytes, of each *stream*'s element.

↔ ***byte_stream*** The pointer which will store the byte conversion of *stream*. Enough memory must have been allocated previously to calling the function, which must be, at least nmemb∗size bytes.

Returns:

The corresponding error code for integer returning functions, i.e., I_MISC_OK if no error was present and I_MISC_ERR if an error occured with errno updated.

Return values:

I_MISC_OK with errno = 0 (No error).

I_MISC_ERR with errno = EINVAL (invalid argument).

4.8.2.6 int zlib_def (FILE ∗ *source*, FILE ∗ *dest*, int *level*)

4.8.2.7 int zlib_inf (FILE ∗ *source*, FILE ∗ *dest*)

4.8.2.8 void zlib_zerr (int *ret*)

4.9 lib/numbers.c File Reference

4.10 lib/protocols.c File Reference

4.11 lib/protocols.h File Reference

```
#include "vorbis/codec.h"
#include "steganos_types.h"
#include "cryptos_types.h"
#include "global_types.h"
```

Functions

- int cryptos_forward (cryptos_config_t ∗cc, cryptos_protocol_buffer_t ∗cb, uint64_t data_size)

  Produces a new cryptographic packet when necessary.

- int cryptos_inverse (cryptos_config_t ∗cc, cryptos_protocol_buffer_t ∗clb)

  Tries to recover a new crypto packet from the cryptographic layer buffer.

- int steganos_vorbis_config_init (vorbis_config_t ∗vc, int rate, int pcmend, int mult, int ∗postlist, int ∗forward_index, int posts)

  Initializes the structure containing basic vorbis codec info needed in the steganographic layer.

- int steganos_vorbis_config_free (vorbis_config_t ∗vc)

  Frees the given vorbis_config_t structure.

- int steganos_forward (steganos_state_t ∗ss, vorbis_config_t ∗vc, int ∗floor, int ∗posts, float ∗residue, byte ∗buffer, size_t buffer_len, int ∗hided)

  Interface to the sender's steganographic functionality.

- int steganos_inverse (steganos_state_t ∗ss, vorbis_config_t ∗vc, int ∗posts, int ∗floor, float ∗residue, float sigma, byte ∗buffer, int buffer_sz, int ∗sd_read)

  Interface to the receiver's steganographic functionality.

4.11.1 Function Documentation

4.11.1.1 int cryptos_forward (cryptos_config_t ∗ *cc*, cryptos_protocol_buffer_t ∗ *cb*, uint64_t *data_size*)

Produces a new cryptographic packet when necessary. When this function is called, and the cryptographic layer buffer is at risk of underflow (i.e., if it has less than MAX_SUBLIMINAL_SIZE bits stores), produces a new crypto packet, storing it in the cryptographic layer buffer. The data to include is read from the file descriptor stored in the cryptographic layer buffer.

Parameters:

← ***cc*** Cryptographic layer configuration structure.

← ***cb*** Cryptographic layer buffer.

← ***data_size*** The number of bits of data to include in the current crypto-packet. If a value of 0 is passed, or it exceeds the maximum data size per packet specified in *cc*, the default data size will be used instead.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

4.11.1.2 int cryptos_inverse (cryptos_config_t ∗ *cc*, cryptos_protocol_buffer_t ∗ *cb*)

Tries to recover a new crypto packet from the cryptographic layer buffer. When called, this function explores the cryptographic layer buffer, using the context defined by the cryptographic configuration structure *cc*, and if a complete new packet is successfully parsed, writes the recovered data in the file descriptor stored in the cryptographic layer buffer.

Parameters:

← ***cc*** Cryptographic layer configuration structure.

← ***cb*** Cryptographic layer buffer.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

4.11.1.3 int steganos_forward (steganos_state_t ∗ *ss*, vorbis_config_t ∗ *vc*, int ∗ *floor*, int ∗ *posts*, float ∗ *residue*, byte ∗ *buffer*, size_t *buffer_len*, int ∗ *hided*)

Interface to the sender's steganographic functionality. Through this interface, a sender program will be able to hide in the floor and residue received vectors, from a Vorbis audio stream, the bitstream stored in the field indicated by the *input* field within the 'steganos_config.xml' file. The steganographic layer will try to hide as much data as possible, and will update *sent* with the number of bits successfully hided.

Parameters:

← ***ss*** The steganographic protocol internal structure

← ***vc*** Vorbis block and look info.

← ***floor*** The Vorbis audio floor vector.

↔ ***posts*** The Vorbis posts vector.

↔ ***residue*** The Vorbis audio residue vector.

← ***buffer*** Buffer containing the data to send.

← ***buffer_len*** Length of buffer, in bytes, containing initialized data.

↔ ***hided*** Pure data successfully hided, in bits.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

4.11.1.4 int steganos_inverse (steganos_state_t * *ss*, vorbis_config_t * *vc*, int * *posts*, int * *floor*, float * *residue*, float *sigma*, byte * *buffer*, int *buffer_sz*, int * *sd_read*)

Interface to the receiver's steganographic functionality. Through this interface, a receiver program can unhide as much subliminal data as may had been hidden in the Vorbis audio *floor* and/or *residue* vectors, if any.

Parameters:

$\leftarrow$ ***ss*** The steganographic protocol internal structure

$\leftarrow$ ***vc*** Vorbis block and look info.

$\leftarrow$ ***posts*** The Vorbis audio floor posts vector.

$\leftarrow$ ***floor*** The Vorbis audio floor vector.

$\leftarrow$ ***residue*** The Vorbis audio residue vector.

$\leftarrow$ ***sigma*** Sigma parameter if ISS.

$\leftrightarrow$ ***buffer*** Buffer to store the subliminal data.

$\leftarrow$ ***buffer_sz*** Size of *buffer*, in bytes.

$\rightarrow$ ***sd_read*** Number of bytes recovered

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

steganos_state_t

4.11.1.5 int steganos_vorbis_config_free (vorbis_config_t * *vc*)

Frees the given vorbis_config_t structure.

Parameters:

$\leftarrow$ ***vc*** Structure to free

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

4.11.1.6 int steganos_vorbis_config_init (vorbis_config_t ∗ *vc*, int *rate*, int *pcmend*, int *mult*, int ∗ *postlist*, int ∗ *forward_index*, int *posts*)

Initializes the structure containing basic vorbis codec info needed in the steganographic layer. Initializes the structure containing basic vorbis codec info needed in the steganographic layer. It is very important to note that, unless the other ∗_init functions, this one DOES NOT allocate memory for the internal arrays, as they are just make point the Vorbis ones. Therefore, there exists no respective ∗_free function, as the inner variables are supposed to be freed by Vorbis code.

Parameters:

↔ ***vc*** Structure that whil group all the essential Vorbis variables

← ***rate*** Sampling rate

← ***pcmend*** Length of the floor and residue vectors

← ***mult*** Multiplier used in the posts calculation

← ***postlist*** Array of posts

← ***forward_index*** Array of posts X axis ordering

← ***posts*** Number of elements in the postlist array

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

4.12 lib/steganos_channel.c File Reference

4.13 lib/steganos_channel.h File Reference

```
#include "global_types.h"
#include "steganos_types.h"
#include "numbers.h"
```

Defines

- #define DISTRESS_SYNC_BYTES 3

  Size, in bytes, of the "distress" header, used to resynchronize when using streaming mode.

- #define DISTRESS_SYNC_BITS DISTRESS_SYNC_BYTES∗BITS_PER_BYTE

  Size, in bits, of the "distress" header, used to resynchronize when using streaming mode.

- #define FILE_SIZE_BYTES 4

  Size, in bytes, of the field that will mark the amount of subliminal data that we will try to hide at a given audio file.

- #define FILE_SIZE_BITS FILE_SIZE_BYTES∗BITS_PER_BYTE

  Size, in bits, of the field that will mark the amount of subliminal data that we will try to hide at a given audio file.

- #define SYNCHRO_HEADER_BYTES_RES 1

  Length of the synchronization header field when using RES_HEADER synchronization mode. Doesn't include the size field bytes.

- #define SIZE_FIELD_BITS 8

  Length of the synchronization field addressed to indicate the number of subliminal bits hided in a given frame and channel.

- #define RES_HEADER_BITS SYNCHRO_HEADER_BYTES_RES∗BITS_PER_BYTE+SIZE_-FIELD_BITS

  The size in bits of THE WHOLE header when using RES_HEADER synchronization method.

- #define MAX_SUBLIMINAL_SIZE ((1 << SIZE_FIELD_BITS) - 1)

  Maximum number of subliminal bits that could be hidden in a residue.

- #define BITS_PARITY 2

  Number of bits to use to calculate the parity in the parity bit method.

- #define ITU_R_BS_468_SIZE 20

  Size of the global static array ITU_R_BS_468.

Functions

- int steganos_state_init (steganos_state_t ∗ss, int da, int hide_method, int sync_method, char ∗key, int keylen)

  Initializes the members of the received stegano_state_t structure.

- int steganos_state_reset_iter (steganos_state_t ∗ss)

  Resets the members of the steganos state structure that change from one iteration to the next.

- int steganos_state_free (steganos_state_t ∗ss)

  Frees the members of the received stegano_state_t structure. The structure itself must be freed separately.

- int steganos_prepare_packet_keys (vorbis_config_t ∗vc, steganos_state_t ∗ss)

  Manipulates the keys to be ready to be used in a new steganos packet.

- int steganos_free_packet_keys (steganos_state_t ∗ss)

  Frees the keys used in the current steganos packet.

- int set_subliminal_capacity_limit (steganos_state_t ∗ss, float ∗residue, const long rate, const int res_-len)

  Determines and sets the subliminal capacity limit for the current frame and channel.

- int hide_data (steganos_state_t ∗ss, byte ∗data, const int d_len, int ∗floor, float ∗residue, const int res_len, int ∗size)

  Determines the final usage of the residue subliminal channel for the current frame and channel by getting the desired size of subliminal data to hide in it.

- int unhide_data (steganos_state_t ∗ss, int ∗floor, float ∗residue, const int res_len, byte ∗∗data, int ∗read)

  Reads subliminal data from a subliminal residue.

- int parity_bits_method (byte ∗plain_mess, int p_m_len, int ∗floor, float ∗res, int res_len, steganos_-key_t ∗key, byte ∗sub_mess, int ∗s_m_len, prng_t ∗prng)

  Implementation of Anderson's and Petitcolas's parity bit method.

- int void_method (byte ∗plain_mess, int p_m_len, int ∗floor, float ∗res, int res_len, steganos_key_t ∗key, byte ∗sub_mess, int ∗s_m_len, prng_t ∗prng)

  Applies no hiding method to the subliminal data.

- int synchro_iss (vorbis_config_t ∗vc, steganos_key_t ∗key, int ∗posts, float ∗residue, void ∗cfg, int decoding, int ∗bit, prng_t ∗prng)

  Implementation of ISS watermarking for synchronization method.

- int iss_cfg_init (steganos_state_t ∗ss, int ∗posts, int posts_len, float sigma, iss_cfg_t ∗iss_cfg)

  Initializes the sigma, lambda and alpha values needed for ISS accordingly to the formulas from Malvar and Florencio's.

- int iss_cfg_free (iss_cfg_t ∗iss_cfg)

  Frees the memory allocated for a iss_cfg_t structure.

- int iss_simulate_floor (vorbis_config_t ∗vc, int ∗posts, int ∗floor)

  Simulates the floor that will be calculated by te receiver may the post vector be like posts.

- int desynchro_res_header (vorbis_config_t ∗vc, int ∗res_lineup, int ∗posts, int ∗floor, float ∗residue, void ∗cfg, steganos_key_t ∗hiding_key, hide_method hide, prng_t ∗prng)

  Type definition for desynchronization functions to use in the protocol.

- int desynchro_iss (vorbis_config_t ∗vc, int ∗res_lineup, int ∗posts, int ∗floor, float ∗residue, void ∗cfg, steganos_key_t ∗hiding_key, hide_method hide, prng_t ∗prng)

  Desynchronization function when using ISS synchronization method.

- int steganos_key_init (byte ∗byte_key, const int key_len, steganos_key_t ∗key)

  Allocates memory for the key fields and sets them to the received values.

- int calculate_residue_lineup (steganos_state_t ∗ss, const int res_len)

  Calculates the residue lineup using the prng in ss.

4.13.1 Define Documentation

4.13.1.1 #define BITS_PARITY 2

Number of bits to use to calculate the parity in the parity bit method.

4.13.1.2 #define DISTRESS_SYNC_BITS DISTRESS_SYNC_BYTES∗BITS_PER_BYTE

Size, in bits, of the "distress" header, used to resynchronize when using streaming mode.

4.13.1.3 #define DISTRESS_SYNC_BYTES 3

Size, in bytes, of the "distress" header, used to resynchronize when using streaming mode.

4.13.1.4 #define FILE_SIZE_BITS FILE_SIZE_BYTES∗BITS_PER_BYTE

Size, in bits, of the field that will mark the amount of subliminal data that we will try to hide at a given audio file.

4.13.1.5 #define FILE_SIZE_BYTES 4

Size, in bytes, of the field that will mark the amount of subliminal data that we will try to hide at a given audio file.

4.13.1.6 #define ITU_R_BS_468_SIZE 20

Size of the global static array ITU_R_BS_468.

4.13.1.7 #define MAX_SUBLIMINAL_SIZE ((1 << SIZE_FIELD_BITS) - 1)

Maximum number of subliminal bits that could be hidden in a residue.

4.13.1.8 #define RES_HEADER_BITS SYNCHRO_HEADER_BYTES_RES∗BITS_PER_-BYTE+SIZE_FIELD_BITS

The size in bits of THE WHOLE header when using RES_HEADER synchronization method.

4.13.1.9 #define SIZE_FIELD_BITS 8

Length of the synchronization field addressed to indicate the number of subliminal bits hided in a given frame and channel.

4.13.1.10 #define SYNCHRO_HEADER_BYTES_RES 1

Length of the synchronization header field when using RES_HEADER synchronization mode. Doesn't include the size field bytes.

4.13.2 Function Documentation

4.13.2.1 calculate_residue_lineup (steganos_state_t ∗ *ss*, const int *res_len*)

Calculates the residue lineup using the prng in ss.

Parameters:

← ***ss*** Steganos state structure.

← ***res_len*** Length of the current residue vector.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK

I_STEGANOS_ERR

4.13.2.2 int desynchro_iss (vorbis_config_t ∗ *vc*, int ∗ *res_lineup*, int ∗ *posts*, int ∗ *floor*, float ∗ *residue*, void ∗ *cfg*, steganos_key_t ∗ *hiding_key*, hide_method *hide*, prng_t ∗ *prng*)

Desynchronization function when using ISS synchronization method. This function should be called when it is not possible to hide information in a given frame-channel and the synchronization method in use is ISS. This function will set the first SIZE_FIELD_BITS of the given residue to 0, ensuring this way that, despite the receiver will receive a 1-marked floor, he'll read 0 bytes of data.

Parameters:

← ***vc*** Vorbis block and look info.

← ***res_lineup*** Residue ordering

↔ ***posts*** Current floor posts vector.

← ***floor*** Floor vector

↔ ***residue*** Current residue vector. Will be changed if it comes "naturally" synchronized.

← ***cfg*** ISS configuration options (sigma, alpha, etc.)

← ***hiding_key*** Key for hiding

← ***hide*** Hiding method (pointer to function) in use in the current frame and channel.

← ***prng*** PRNG in use

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

synchro_method_et
iss_synchronization

4.13.2.3 int desynchro_res_header (vorbis_config_t * *vc*, int * *res_lineup*, int * *posts*, int * *floor*, float * *residue*, void * *cfg*, steganos_key_t * *hiding_key*, hide_method *hide*, prng_t * *prng*)

Type definition for desynchronization functions to use in the protocol. Functions of this type shall solve the false positive synchronization problem that may happen depending on the synchro_method used. Depending on the specific method in use, the *floor* and *residue* vectors will or won't be used.

Parameters:

← ***vc*** Vorbis block and look info.

← ***res_lineup*** Residue ordering

↔ ***posts*** Current floor posts vector.

← ***floor*** Floor vector

↔ ***residue*** Current residue vector. Will be changed if it comes "naturally" synchronized.

← ***cfg*** This parameter will be used to pass any argument needed in a concrete synchronization method.

← ***hiding_key*** Key for hiding

← ***hide*** Hiding method (pointer to function) in use in the current frame and channel.

← ***prng*** PRNG in use

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

synchro_method_et
iss_synchronization

4.13.2.4 int hide_data (steganos_state_t ∗ *ss*, byte ∗ *data*, const int *d_len*, int ∗ *floor*, float ∗ *residue*, const int *res_len*, int ∗ *size*)

Determines the final usage of the residue subliminal channel for the current frame and channel by getting the desired size of subliminal data to hide in it. This function first retrieves the next 'ss->aggressiveness∗ss->max_fc_capacity' bits of subliminal data to hide in the current frame and channel residue, using the chosen hiding function. This let us make a more precise estimation of the final effects of the changes we will introduce in the original data. Once this is done, the nearest integer size of bits is hided in the residue vector, giving the final subliminal-residue vector. Note that this function shouldn't be called if ss->aggressiveness∗ss->max_fc_capacity is not big enough to store the meta-information required (plus some subliminal info).

Parameters:

↔ ***ss*** Internal state structure. The function updates the ss->protocol_send_state field, which must be SYNCHRONIZED at the input and will be CONTROLLED after a successful execution.

← ***data*** The remaining bitstream to hide. Note that can have greater, equal or less size than the subliminal channel usage.

← ***d_len*** The length of the 'data' stream, in bits.

← ***floor*** The current frame and channel floor vecotr. Depending on the hiding method selected, it may or may not be used.

↔ ***residue*** The current frame and channel residue vector. After a successful execution of the function, it will store the subliminal residue.

← ***res_len*** The length of the residue (and floor) vector.

→ ***size*** The final number of bits of pure data (excluding headers) hided in the current frame and channel residue.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated

Return values:

I_STEGANOS_OK with errno = 0 (No error)

I_STEGANOS_ERR with errno = EINVAL (invalid argument)

See also:

steganos_state_t

4.13.2.5 int iss_cfg_free (iss_cfg_t ∗ *iss_cfg*)

Frees the memory allocated for a iss_cfg_t structure.

Parameters:

↔ ***iss_cfg*** ISS configuration structure to free.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

See also:

iss_synchronization

4.13.2.6 int iss_cfg_init (steganos_state_t ∗ *ss*, int ∗ *posts*, int *posts_len*, float *sigma*, iss_cfg_t ∗ *iss_cfg*)

Initializes the sigma, lambda and alpha values needed for ISS accordingly to the formulas from Malvar and Florencio's.

Parameters:

← ***ss*** Internal state structure.

← ***posts*** Post vector.

← ***posts_len*** Number of elements in the posts vector.

← ***sigma*** Sigma parameter (watermark's strength)

↔ ***iss_cfg*** ISS configuration structure.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

iss_synchronization

4.13.2.7 int iss_simulate_floor (vorbis_config_t ∗ *vc*, int ∗ *posts*, int ∗ *floor*)

Simulates the floor that will be calculated by te receiver may the post vector be like *posts*.

Parameters:

← ***vc*** Voribs block and look configuration data.

↔ ***floor*** The floor vector, as will be calculated by the receiver. The memory must be allocated previously to calling the function.

← ***posts*** The posts vector.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

4.13.2.8 int parity_bits_method (byte ∗ *plain_mess*, int *p_m_len*, int ∗ *floor*, float ∗ *res*, int *res_len*, steganos_key_t ∗ *key*, byte ∗ *sub_mess*, int ∗ *s_m_len*, prng_t ∗ *prng*)

Implementation of Anderson's and Petitcolas's parity bit method. Implementation of Anderson's and Petitcolas's parity bit method. See "On The Limits of Steganosgraphy" by Anderson and Petitcolas. The function keeps the (∗hide_method) typedef format.

Note:

This function uses the PRNG function *get_random_int*, therefore, the PRNG must have been seeded previously with the same key used for hiding in order to get the correct data from the residue/floor vectors.

Parameters:

← ***plain_mess*** Plain message to hide.

← ***p_m_len*** Length of plain message in bits.

← ***floor*** Floor vector.

← ***res*** Residue vector.

← ***res_len*** Number of elements in *floor* and *res* vectors.

← ***key*** Key to use.

↔ ***sub_mess*** String in which the subliminal message will be stored. It has to be allocated previously to calling the function.

↔ ***s_m_len*** At the input, will store the allocated size (in bits) of *sub_mess*. At the output, the length of the resulting sub_message, in bits (it may be different than p_m_len as it may include desynchronization bits), or discard them.

← ***prng*** PRNG in use.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

hide_method

4.13.2.9 int set_subliminal_capacity_limit (steganos_state_t ∗ *ss*, float ∗ *residue*, const long *rate*, const int *res_len*)

Determines and sets the subliminal capacity limit for the current frame and channel. This function determines the subliminal capacity limit for the current frame and channel residue, updating the steganos_state_t internal state structure with the values obtained. This limits will be stored in the (float ∗)[2] field as [-, +] maximum variations to introduce in the corresponding residual value of the frame and channel's residue vector. Also updates the maximum subliminal capacity for the whole frame and channel current residue. The protocol state must be IDLE at the input and MEASURED after a successful execution. Also initializes the ss->res_mask variable, which can be seen as some kind of "steganosgraphic mask", calculated initially from the maximum residual variations here obtained. The protocol state structure variables must have been allocated and reset previously.

Parameters:

↔ ***ss*** Internal state structure. It's field capacity_limit will be updated in a successful execution. The protocol state must be IDLE at the input and MEASURED after a successful execution.

← ***residue*** The current frame and channel residue vector.

← ***rate*** The sampling rate used (in Hz), to obtain the frequency corresponding to each residue.

← ***res_len*** Length of the residue and frequencies vectors.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

steganos_state_t

4.13.2.10 static int steganos_free_packet_keys (steganos_state_t ∗ *ss*)

Frees the keys used in the current steganos packet.

Parameters:

← ***ss*** Steganos state structure

Returns:

I_STEGANOS_OK

4.13.2.11 int steganos_key_init (byte ∗ *byte_key*, const int *key_len*, steganos_key_t ∗ *key*)

Allocates memory for the key fields and sets them to the received values. Allocates memory for the key fields and sets them to the received values.

Parameters:

← ***byte_key*** The key value.

← ***key_len*** The length of the *byte_key* argument, in bits.

↔ ***key*** A pointer to a previously allocated steganos_key_t variable in which we'll store the key's members.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

I_STEGANOS_ERR with errno = ERANGE (invalid key size).

4.13.2.12 int steganos_prepare_packet_keys (vorbis_config_t ∗ *vc*, steganos_state_t ∗ *ss*)

Manipulates the keys to be ready to be used in a new steganos packet. The key used to hide/sync is obtained following this flow: 1) Cipher md5(vc->forward_index) with ARCFOUR, using the master key 2) Use the resulting ciphertext as frame specific keys

Parameters:

← ***vc*** Vorbis config structure

↔ ***ss*** Steganos state structure. The internal variables ss->hiding_key and ss->synchro_key will be updated upon successful execution.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

4.13.2.13 int steganos_state_free (steganos_state_t ∗ *ss*)

Frees the members of the received stegano_state_t structure. The structure itself must be freed separately.

Parameters:

← ***ss*** Pointer to the steganographic protocol state variable.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

steganos_state_t

4.13.2.14 int steganos_state_init (steganos_state_t ∗ *ss*, int *da*, int *hide_method*, int *sync_method*, char ∗ *key*, int *keylen*)

Initializes the members of the received stegano_state_t structure. This function prepares the steganographic protocol for a new whole step, allocating and initializing the steganos state structure to the default values or those specified as arguments.

Parameters:

← ***ss*** Pointer to the steganographic protocol state variable.

← ***da*** Desired aggressiveness, i.e., the share of subliminal capacity to use, in 1/10 (e.g. 2 means use 20%)

← ***hide_method*** Hide method to use, must be one of the defined in hide_method_et

← ***sync_method*** Sync method to use, must be one of the defined in sync_method_et

← ***key*** Master key to use for deriving subkeys

← ***keylen*** Length of *key* in bits. Must be at least 128.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

steganos_state_t

4.13.2.15 int steganos_state_reset_iter (steganos_state_t ∗ *ss*)

Resets the members of the steganos state structure that change from one iteration to the next.

Parameters:

← ***ss*** Pointer to the steganographic protocol state variable.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

steganos_state_t

4.13.2.16 int synchro_iss (vorbis_config_t ∗ *vc*, steganos_key_t ∗ *key*, int ∗ *posts*, float ∗ *residue*, void ∗ *cfg*, int *decoding*, int ∗ *bit*, prng_t ∗ *prng*)

Implementation of ISS watermarking for synchronization method. Implements the Improved Spread Spectrum watermarking technique introduced by Malvar and Florencio (in "Improved Spread Spectrum: a new modulation technique for robust watermarking") to synchronize sender and receiver by marking the floor vector.

Parameters:

$\leftarrow$ ***vc*** Vorbis block and look information.

$\leftarrow$ ***key*** Key to use

$\leftrightarrow$ ***posts*** Current floor posts vector. After a successful execution, it will be modified as needed.

$\leftrightarrow$ ***residue*** Current residue vector. Not used here.

$\leftarrow$ ***cfg*** Pointer to a structure of iss_cfg_t with the alpha, lambda and sigma parameters needed for ISS.

$\leftarrow$ ***decoding*** Active if the function is called from the decoder's side,

$\leftrightarrow$ ***bit*** At the input, the desired bit to include if we are at the encoder's side; at the ouput, the received bit if we are at the decoder's side (i.e. 1 if we're telling the receiver there is hidden data and 0 if not).

$\leftarrow$ ***prng*** PRNG in use.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

I_STEGANOS_SYNC_FAIL with errno = 0 (Unable to synchronize).

See also:

synchro_method_et
iss_synchronization

4.13.2.17 int unhide_data (steganos_state_t ∗ *ss*, int ∗ *floor*, float ∗ *residue*, const int *res_len*, byte ∗∗ *data*, int ∗ *read*)

Reads subliminal data from a subliminal residue. This function recovers the subliminal data hided in the current residue. If the ISS synchronization mode is in use, and this function is called, means the residue surely shelters subliminal data. If the RES_HEADER synchro mode is in use, we have to look for the SYNCHRO_HEADER first, which will tell us if we have to recover any data. At the end of a successful execution, *data* will point to the recovered data and *read* store the number of bits read. If no data has been recovered, *read* will be 0.

Note:

This function uses the PRNG function *get_random_int*, therefore, the PRNG must have been seeded previously with the same key used for hiding in order to get the correct data from the residue/floor vectors.

Parameters:

$\leftrightarrow$ ***ss*** Internal state structure. The protocol state must be ANALYZED at the input and READ after a successful execution.

$\leftarrow$ ***floor*** The floor vector of the current frame and channel.

$\leftarrow$ ***residue*** The residue vector of the current frame and channel, containing the subliminal data, if any.

← ***res_len*** The length of the residue and floor vectors.

↔ ***data*** Pointer to the bitstream which will store the subliminal data read after a successful execution.

↔ ***read*** Will store the final amount of data that has been possible to read, in bits.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

steganos_state_t

4.13.2.18 int void_method (byte ∗ *plain_mess*, int *p_m_len*, int ∗ *floor*, float ∗ *res*, int *res_len*, steganos_key_t ∗ *key*, byte ∗ *sub_mess*, int ∗ *s_m_len*, prng_t ∗ *prng*)

Applies no hiding method to the subliminal data. This hiding method simply returns the first p_m_len bits of plain_mess.

Parameters:

← ***plain_mess*** Plain message to hide.

← ***p_m_len*** Length of plain message in bits.

← ***floor*** Floor vector.

← ***res*** Residue vector.

← ***res_len*** Number of elements in *floor* and *res* vectors.

← ***key*** Key to use.

↔ ***sub_mess*** String in which the subliminal message will be stored. It has to be allocated previously to calling the function.

↔ ***s_m_len*** At the input, will store the allocated size (in bits) of *sub_mess*. At the output, the length of the resulting sub_message, in bits (it may be different than p_m_len as it may include desynchronization bits).

← ***prng*** PRNG in use. Here, NULL.

Returns:

The corresponding error code for integer returning functions, i.e., I_STEGANOS_OK if no error was present and I_STEGANOS_ERR if an error occured with errno updated.

Return values:

I_STEGANOS_OK with errno = 0 (No error).

I_STEGANOS_ERR with errno = EINVAL (invalid argument).

See also:

hide_method

Index