LibGolle

0.0.0

Generated by Doxygen 1.8.1.2

Fri May 30 2014 00:32:46

Contents

1 Golle 1

2 ChangeLog 3

3 Authors 5

4 News 7

5 COPYING 9

6 Module Index 11

6.1 Modules . . . 11

7 Data Structure Index 13

7.1 Data Structures . . . 13

8 File Index 15

8.1 File List . . . 15

9 Module Documentation 17

9.1 Binary buffers . . . 17

9.1.1 Detailed Description . . . 17

9.1.2 Function Documentation . . . 18

9.1.2.1 golle_bin_copy . . . 18

9.1.2.2 golle_bin_delete . . . 18

9.1.2.3 golle_bin_init . . . 18

9.1.2.4 golle_bin_new . . . 18

9.1.2.5 golle_bin_release . . . 19

9.1.2.6 golle_bin_resize . . . 19

9.2 Bit Commitment . . . 20

9.2.1 Detailed Description . . . 20

9.2.2 Function Documentation . . . 20

9.2.2.1 golle_commit_clear . . . 20

9.2.2.2 golle_commit_copy . . . 20

9.2.2.3 golle_commit_delete . . . 21
9.2.2.4 golle_commit_new . . . 21
9.2.2.5 golle_commit_verify . . . 21
9.3 Disjunctive Schnorr Identification . . . 22
9.3.1 Detailed Description . . . 22
9.3.2 Function Documentation . . . 22
9.3.2.1 golle_disj_clear . . . 22
9.3.2.2 golle_disj_commit . . . 22
9.3.2.3 golle_disj_prove . . . 23
9.3.2.4 golle_disj_verify . . . 23
9.4 Disjunctive Plaintext Equivalence Proof . . . 24
9.4.1 Detailed Description . . . 24
9.4.2 Function Documentation . . . 24
9.4.2.1 golle_dispep_setup . . . 24
9.5 Key Generation and Distribution . . . 25
9.5.1 Detailed Description . . . 25
9.5.2 Function Documentation . . . 25
9.5.2.1 golle_key_accum_h . . . 25
9.5.2.2 golle_key_cleanup . . . 26
9.5.2.3 golle_key_gen_private . . . 26
9.5.2.4 golle_key_gen_public . . . 26
9.5.2.5 golle_key_set_public . . . 27
9.6 ElGamal . . . 28
9.6.1 Detailed Description . . . 28
9.6.2 Function Documentation . . . 28
9.6.2.1 golle_eg_clear . . . 28
9.6.2.2 golle_eg_decrypt . . . 29
9.6.2.3 golle_eg_encrypt . . . 29
9.6.2.4 golle_eg_reencrypt . . . 29
9.7 The Golle protocol interface . . . 31
9.7.1 Detailed Description . . . 32
9.7.2 Macro Definition Documentation . . . 32
9.7.2.1 GOLLE_FACE_UP . . . 32
9.7.3 Function Documentation . . . 32
9.7.3.1 golle_check_selection . . . 32
9.7.3.2 golle_clear . . . 32
9.7.3.3 golle_generate . . . 33
9.7.3.4 golle_initialise . . . 33
9.7.3.5 golle_reduce_selection . . . 33
9.7.3.6 golle_reveal_selection . . . 34

9.8 Singly-linked lists . . . 35
9.8.1 Detailed Description . . . 35
9.8.2 Function Documentation . . . 36
9.8.2.1 golle_list_delete . . . 36
9.8.2.2 golle_list_erase_at . . . 36
9.8.2.3 golle_list_insert_at . . . 36
9.8.2.4 golle_list_iterator . . . 36
9.8.2.5 golle_list_iterator_free . . . 37
9.8.2.6 golle_list_iterator_next . . . 37
9.8.2.7 golle_list_iterator_reset . . . 37
9.8.2.8 golle_list_new . . . 37
9.8.2.9 golle_list_pop . . . 38
9.8.2.10 golle_list_pop_all . . . 38
9.8.2.11 golle_list_pop_many . . . 38
9.8.2.12 golle_list_push . . . 38
9.8.2.13 golle_list_push_many . . . 39
9.8.2.14 golle_list_size . . . 39
9.8.2.15 golle_list_top . . . 39
9.9 Large Numbers . . . 40
9.9.1 Detailed Description . . . 41
9.9.2 Function Documentation . . . 41
9.9.2.1 golle_bin_to_num . . . 41
9.9.2.2 golle_find_generator . . . 41
9.9.2.3 golle_generate_prime . . . 41
9.9.2.4 golle_num_cmp . . . 42
9.9.2.5 golle_num_cpy . . . 42
9.9.2.6 golle_num_dup . . . 42
9.9.2.7 golle_num_generate_rand . . . 43
9.9.2.8 golle_num_mod_exp . . . 43
9.9.2.9 golle_num_new . . . 43
9.9.2.10 golle_num_new_int . . . 43
9.9.2.11 golle_num_print . . . 43
9.9.2.12 golle_num_rand . . . 44
9.9.2.13 golle_num_rand_bits . . . 44
9.9.2.14 golle_num_to_bin . . . 44
9.9.2.15 golle_num_xor . . . 44
9.9.2.16 golle_test_prime . . . 45
9.10 Plaintext Equivalence Proof . . . 46
9.10.1 Detailed Description . . . 46
9.10.2 Function Documentation . . . 46

9.10.2.1 golle_pep_prover . . . 46
9.10.2.2 golle_pep_verifier . . . 46
9.11 Random Data . . . 48
9.11.1 Detailed Description . . . 48
9.11.2 Function Documentation . . . 48
9.11.2.1 golle_random_clear . . . 48
9.11.2.2 golle_random_generate . . . 48
9.11.2.3 golle_random_seed . . . 48
9.12 Schnorr Identification Algorithm . . . 49
9.12.1 Detailed Description . . . 49
9.12.2 Function Documentation . . . 49
9.12.2.1 golle_schnorr_clear . . . 49
9.12.2.2 golle_schnorr_commit . . . 49
9.12.2.3 golle_schnorr_prove . . . 50
9.12.2.4 golle_schnorr_verify . . . 50

10 Data Structure Documentation **51**

10.1 golle_bin_t Struct Reference . . . 51
10.1.1 Detailed Description . . . 51
10.1.2 Field Documentation . . . 51
10.1.2.1 bin . . . 51
10.1.2.2 size . . . 51
10.2 golle_commit_t Struct Reference . . . 51
10.2.1 Detailed Description . . . 52
10.2.2 Field Documentation . . . 52
10.2.2.1 hash . . . 52
10.2.2.2 rkeep . . . 52
10.2.2.3 rsend . . . 52
10.2.2.4 secret . . . 52
10.3 golle_disj_t Struct Reference . . . 52
10.3.1 Detailed Description . . . 52
10.3.2 Field Documentation . . . 53
10.3.2.1 c1 . . . 53
10.3.2.2 c2 . . . 53
10.3.2.3 r1 . . . 53
10.3.2.4 s1 . . . 53
10.3.2.5 s2 . . . 53
10.3.2.6 t1 . . . 53
10.3.2.7 t2 . . . 53
10.4 golle_eg_t Struct Reference . . . 53

10.4.1 Detailed Description . . . 53
10.4.2 Field Documentation . . . 54
10.4.2.1 a . . . 54
10.4.2.2 b . . . 54
10.5 golle_key_t Struct Reference . . . 54
10.5.1 Detailed Description . . . 54
10.5.2 Field Documentation . . . 54
10.5.2.1 g . . . 54
10.5.2.2 h . . . 54
10.5.2.3 h_product . . . 54
10.5.2.4 p . . . 54
10.5.2.5 q . . . 55
10.5.2.6 x . . . 55
10.6 golle_list_iterator_t Struct Reference . . . 55
10.6.1 Detailed Description . . . 55
10.7 golle_list_t Struct Reference . . . 55
10.7.1 Detailed Description . . . 55
10.8 golle_schnorr_t Struct Reference . . . 55
10.8.1 Detailed Description . . . 56
10.8.2 Field Documentation . . . 56
10.8.2.1 G . . . 56
10.8.2.2 p . . . 56
10.8.2.3 q . . . 56
10.8.2.4 x . . . 56
10.8.2.5 Y . . . 56
10.9 golle_t Struct Reference . . . 56
10.9.1 Detailed Description . . . 57
10.9.2 Field Documentation . . . 57
10.9.2.1 accept_commit . . . 57
10.9.2.2 accept_crypt . . . 57
10.9.2.3 accept_eg . . . 57
10.9.2.4 accept_rand . . . 57
10.9.2.5 bcast_commit . . . 57
10.9.2.6 bcast_crypt . . . 58
10.9.2.7 bcast_secret . . . 58
10.9.2.8 key . . . 58
10.9.2.9 num_items . . . 58
10.9.2.10 num_peers . . . 58
10.9.2.11 reserved . . . 58
10.9.2.12 reveal_rand . . . 58

11 File Documentation **59**

11.1 include/golle/bin.h File Reference . . . 59
11.1.1 Detailed Description . . . 60
11.2 include/golle/commit.h File Reference . . . 60
11.2.1 Detailed Description . . . 60
11.3 include/golle/disj.h File Reference . . . 61
11.3.1 Detailed Description . . . 61
11.4 include/golle/dispep.h File Reference . . . 62
11.4.1 Detailed Description . . . 62
11.5 include/golle/distribute.h File Reference . . . 62
11.5.1 Detailed Description . . . 63
11.6 include/golle/elgamal.h File Reference . . . 63
11.6.1 Detailed Description . . . 64
11.7 include/golle/errors.h File Reference . . . 64
11.7.1 Detailed Description . . . 65
11.7.2 Macro Definition Documentation . . . 65
11.7.2.1 GOLLE_ASSERT . . . 65
11.7.2.2 GOLLE_UNUSED . . . 65
11.7.3 Enumeration Type Documentation . . . 65
11.7.3.1 golle_error . . . 65
11.8 include/golle/golle.h File Reference . . . 65
11.8.1 Detailed Description . . . 67
11.9 include/golle/list.h File Reference . . . 67
11.9.1 Detailed Description . . . 68
11.10 include/golle/numbers.h File Reference . . . 68
11.10.1 Detailed Description . . . 69
11.11 include/golle/pep.h File Reference . . . 70
11.11.1 Detailed Description . . . 70
11.12 include/golle/platform.h File Reference . . . 70
11.12.1 Detailed Description . . . 70
11.12.2 Macro Definition Documentation . . . 71
11.12.2.1 GOLLE_BEGIN_C . . . 71
11.12.2.2 GOLLE_END_C . . . 71
11.12.2.3 GOLLE_EXTERN . . . 71
11.12.2.4 GOLLE_INLINE . . . 71
11.13 include/golle/random.h File Reference . . . 71
11.13.1 Detailed Description . . . 72
11.14 include/golle/schnorr.h File Reference . . . 72
11.14.1 Detailed Description . . . 72

Chapter 1

Golle

libgolle is a library that allows individual nodes on a network to deal cards in a fully distributed way.

The algorithm was described by Phillipe Golle in his paper Dealing Cards in Poker Games. This library provides an implementation of the algorithm in order to develop distributed applications where nodes a randomly "dealt" distinct elements from a set.

A note about copying

This project uses OpenSSL for large numbers and some cryptography algorithms. As such:

> This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)

And when compiling for Windows:

> This product includes software written by Tim Hudson (tjs@cryptsoft.com)

The Autoconf macros under the `aclocal` directory are part of the Autoconf Archive and are subject to the licensing and copyright thereof.

Chapter 2

ChangeLog

Chapter 3

Authors

- Anthony Arnold (anthony.arnold@uqconnect.edu.au)

Chapter 4

News

- 30 Mar 2014, libgolle 0.0.0
  - General
    - Development begins.

Chapter 5

COPYING

Chapter 6

Module Index

6.1 Modules

Here is a list of all modules:

Binary buffers . . . 17
Bit Commitment . . . 20
Disjunctive Schnorr Identification . . . 22
Disjunctive Plaintext Equivalence Proof . . . 24
Key Generation and Distribution . . . 25
ElGamal . . . 28
The Golle protocol interface . . . 31
Singly-linked lists . . . 35
Large Numbers . . . 40
Plaintext Equivalence Proof . . . 46
Random Data . . . 48
Schnorr Identification Algorithm . . . 49

Chapter 7

Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

golle_bin_t
: Represents a binary buffer . . . 51

golle_commit_t
: Holds the values for a bit commitment . . . 51

golle_disj_t
: A structure to store all of the revelant values required by the Disjunctive Schnorr Identification protocol . . . 52

golle_eg_t
: ElGamal ciphertex . . . 53

golle_key_t
: A peer's key. Contains the peer's portion of the private key and the public key elements . . . 54

golle_list_iterator_t
: A type used for iterating through all the items in a list . . . 55

golle_list_t
: An opaque pointer to a singly-linked list . . . 55

golle_schnorr_t
: A key used for the Schnorr Identification Algorithm . . . 55

golle_t
: The main Golle structure . . . 56

Chapter 8

File Index

8.1 File List

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

include/**golle.h** . . . . . **??**
include/golle/bin.h
Defines a structure for a binary buffer 59
include/golle/commit.h
Defines functions for a commitment scheme 60
include/golle/**config.h** . . . . . **??**
include/golle/disj.h
Disjunctive Schnorr Identification 61
include/golle/dispep.h
DISPEP protocol 62
include/golle/distribute.h
Defines the protocol for generating a distributed public/private key pair 62
include/golle/elgamal.h
Describes functions for performing distributed ElGamal cryptography 63
include/golle/errors.h
Error constants 64
include/golle/golle.h
Golle interface 65
include/golle/list.h
Describes the structures and operations for working with singly-linked lists 67
include/golle/numbers.h
Describes various available number functions including primality functions, generator finding, and arithmetic of large numbers 68
include/golle/pep.h
PEP protocol 70
include/golle/platform.h
Macros for platform-dependant behaviour 70
include/golle/random.h
Wrapper functions for collecting random data 71
include/golle/schnorr.h
Schnorr Identification 72
include/golle/**types.h** . . . . . **??**

Chapter 9

Module Documentation

9.1 Binary buffers

Data Structures

- struct golle_bin_t

  Represents a binary buffer.

Macros

- #define golle_bin_clear(b) golle_bin_release(b)

  An alias for golle_bin_release.

Functions

- GOLLE_EXTERN golle_error golle_bin_init (golle_bin_t *buff, size_t size)

  Initialise the inner buffer of a non-dynamic buffer. A typical use case is declaring on the stack or as a non-pointer member of another structure.
- GOLLE_EXTERN void golle_bin_release (golle_bin_t *buff)

  Releases the inner buffer without releasing the golle_bin_t structure itself. Useful for releasing resources allocated with golle_bin_init().
- GOLLE_EXTERN golle_bin_t * golle_bin_new (size_t size)

  Create a new binary buffer of a given size. The data block is zeroed out before returning.
- GOLLE_EXTERN void golle_bin_delete (golle_bin_t *buff)

  Deallocates resources held by a golle_bin_t structure.
- GOLLE_EXTERN golle_bin_t * golle_bin_copy (const golle_bin_t *buff)

  Makes a copy of `buff` via golle_bin_new.
- GOLLE_EXTERN golle_error golle_bin_resize (golle_bin_t *buff, size_t size)

  Resize the buffer.

9.1.1 Detailed Description

Many parts of this library, and client applications, need to work with a block of data of arbitrary size. It is important to always know how large the block is and to always work with it in the context of its size. Here we present the very simple golle_bin_t, which marries the size of a data block to the block itself. By using the golle_bin_new() and golle_bin_delete() functions, it's not so hard to screw up; as long as you always check the size of the buffer before using it.

9.1.2 Function Documentation

9.1.2.1 GOLLE_EXTERN golle_bin_t* golle_bin_copy (const golle_bin_t * *buff*)

Makes a copy of `buff` via golle_bin_new.

Parameters

buff	The buffer to copy.

Returns

A new buffer, or `NULL` if either buff was `NULL` or if golle_bin_new returned `NULL`.

Note

You should delete this copy with golle_bin_delete.

9.1.2.2 GOLLE_EXTERN void golle_bin_delete (golle_bin_t * *buff*)

Deallocates resources held by a golle_bin_t structure.

Parameters

buff	The structure to free.

Note

This function will check the address of the `bin` member of `buff`. If it points to the address just passed the object, it will free the object. If it points elsewhere, it will *also* free the `bin` member separately.

9.1.2.3 GOLLE_EXTERN golle_error golle_bin_init (golle_bin_t * *buff*, size_t *size*)

Initialise the inner buffer of a non-dynamic buffer. A typical use case is declaring on the stack or as a non-pointer member of another structure.

golle_bin_t buffer; golle_bin_init (&buffer, size);

Parameters

buff	The buffer to initialise. It is assumed that the `bin` member of `buff` is invalid (i.e. this function won't free it).
size	The requested size to initialise the buffer to.

Returns

GOLLE_OK if everything succeeded. GOLLE_EMEM if memory allocation failed. GOLLE_ERROR if `buffer` is NULL or `size` is 0.

9.1.2.4 GOLLE_EXTERN golle_bin_t* golle_bin_new (size_t *size*)

Create a new binary buffer of a given size. The data block is zeroed out before returning.

Parameters

size	The size of the buffer to allocate.

Returns

The allocated buffer, or NULL if allocation failed.

Note

This function only performs one `malloc`. It allocates enough space for the structure *and* the `bin` data itself. The returned object's `bin` member will point to the address just after the object.

Warning

Do no independantly `free` the `bin` member of a golle_bin_t structure allocated with this function. Call golle_bin_delete instead.
It is not usually a good idea to set the members of a structure returned by this function.

9.1.2.5 GOLLE_EXTERN void golle_bin_release (golle_bin_t * *buff*)

Releases the inner buffer without releasing the golle_bin_t structure itself. Useful for releasing resources allocated with golle_bin_init().

Parameters

buff	The buffer to release.

9.1.2.6 GOLLE_EXTERN golle_error golle_bin_resize (golle_bin_t * *buff*, size_t *size*)

Resize the buffer.

Parameters

buff	The buffer to resize.
size	The new size of the buffer.

Returns

GOLLE_OK if successful. GOLLE_ERROR if `buff` is `NULL` or size is 0. GOLLE_EMEM if memory reallocation failed.

9.2 Bit Commitment

Data Structures

- struct golle_commit_t

  Holds the values for a bit commitment.

Functions

- GOLLE_EXTERN golle_commit_t * golle_commit_new (const golle_bin_t *secret)

  Generate a new bit commitment to a value.
- GOLLE_EXTERN void golle_commit_delete (golle_commit_t *commitment)

  Free resources allocated by a call to golle_commit_new.
- GOLLE_EXTERN golle_error golle_commit_verify (const golle_commit_t *commitment)

  Verify a commmitment.
- GOLLE_INLINE void golle_commit_clear (golle_commit_t *commit)

  Release the buffers associated with a commit without freeing the commit structure itself.
- GOLLE_EXTERN golle_error golle_commit_copy (golle_commit_t *dest, const golle_commit_t *src)

  Copy a commit structure, bin for bin.

9.2.1 Detailed Description

The protocol for bit commitment scheme is as such:

- The user creates a `secret` golle_bin_t
- The user creates a golle_commit_t using the `secret`, which generates two random sequences, `r1` and `r2`, and the hash `h` of (`r1`, `r2`, `secret`).
- The user sends `r1` and `h` to one or more parties. At this point, the commitment is non-malleable.
- The external parties cannot determine the contents of the `secret`, even if the `secret` is one bit.
- Later, the user may reveal the `secret` by sending `r1`, `r2`, and the `secret` to the external parties.
- The external parties can verify that the `secret` has not changed by checking the hash (see golle_commit_verify).

9.2.2 Function Documentation

9.2.2.1 GOLLE_INLINE void golle_commit_clear (golle_commit_t * *commit*)

Release the buffers associated with a commit without freeing the commit structure itself.

Parameters

commit	The commit structure whose buffers should be freed.

9.2.2.2 GOLLE_EXTERN golle_error golle_commit_copy (golle_commit_t * *dest*, const golle_commit_t * *src*)

Copy a commit structure, bin for bin.

Parameters

out	*dest*	Will have each member set to copied buffer.
	src	Contains the buffers to copy from.

Returns

GOLLE_OK if OK, :GOLLE_ERROR if either param is NULL, GOLLE_EMEM if memory failed.

9.2.2.3 GOLLE_EXTERN void golle_commit_delete (golle_commit_t * *commitment*)

Free resources allocated by a call to golle_commit_new.

Parameters

commitment	A pointer returned by golle_commit_new().

9.2.2.4 GOLLE_EXTERN golle_commit_t* golle_commit_new (const golle_bin_t * *secret*)

Generate a new bit commitment to a value.

Parameters

secret	The secret that is to be commited to.

Returns

A new golle_commit_t, or NULL if allocation failed or secret is NULL or secret is of size zero. The other three members of the structure will be set.

Note

The new returned structure will have a *copy* of the secret passed in to the function.

9.2.2.5 GOLLE_EXTERN golle_error golle_commit_verify (const golle_commit_t * *commitment*)

Verify a commmitment.

Parameters

commitment	The commitment to verify.

Returns

GOLLE_COMMIT_PASSED if the commitment was verified. GOLLE_COMMIT_FAILED if the commitment verification did not pass. GOLLE_ERROR if any commitment, or any member of commitment is NULL. GOLLE_ECRYPTO if hash checking failed.

Note

The caller should have first received rsend and hash, then independantly received secret and rkeep. A full golle_commit_t is required to verify that the secret value is correct.

9.3 Disjunctive Schnorr Identification

Data Structures

- struct golle_disj_t

  A structure to store all of the revelant values required by the Disjunctive Schnorr Identification protocol.

Functions

- GOLLE_INLINE void golle_disj_clear (golle_disj_t *d)

  Clear all numbers out of a disjuntive schnorr structure.
- GOLLE_EXTERN golle_error golle_disj_commit (const golle_schnorr_t *unknown, const golle_schnorr_t *known, golle_disj_t *d)

  Generate the commitments `t1` and `s2` to be sent to the verifier.
- GOLLE_EXTERN golle_error golle_disj_prove (const golle_schnorr_t *unknown, const golle_schnorr_t *known, const golle_num_t c, golle_disj_t *d)

  Output the proof that `x` is known. `s1`, `s2`, `c1`, and `c2` are sent to the verifier.
- GOLLE_EXTERN golle_error golle_disj_verify (const golle_schnorr_t *k1, const golle_schnorr_t *k2, const golle_disj_t *d)

  Verify a proof sent by a prover.

9.3.1 Detailed Description

The disjunctive Schnorr Identification protocol is similar to the Schnorr Identification Algorithm. However, it allows the prover to use one of two different keys without the verifier knowing which key was used.

9.3.2 Function Documentation

9.3.2.1 GOLLE_INLINE void golle_disj_clear (golle_disj_t * *d*)

Clear all numbers out of a disjuntive schnorr structure.

Parameters

d	The structure to clear.

9.3.2.2 GOLLE_EXTERN golle_error golle_disj_commit (const golle_schnorr_t * *unknown*, const golle_schnorr_t * *known*, golle_disj_t * *d*)

Generate the commitments `t1` and `s2` to be sent to the verifier.

Parameters

unknown	The Schnorr key, containing the `G` and `Y` values, that the secret key is *not* associated with.
known	The schnorr key, containing the `G` and `Y` values,
d	The disjunct structure that will receive the `t1`, `r1`, `t2`, `c2`, and `s2` values.

Returns

GOLLE_ERROR, GOLLE_OK, or GOLLE_EMEM.

9.3.2.3 GOLLE_EXTERN golle_error golle_disj_prove (const golle_schnorr_t * *unknown*, const golle_schnorr_t * *known*, const golle_num_t *c*, golle_disj_t * *d*)

Output the proof that `x` is known. `s1`, `s2`, `c1`, and `c2` are sent to the verifier.

Parameters

unknown	The schnorr key, containing the `G`, and `Y` values, that the secret key is *not* associated with; the same key used with golle_disj_commit().
known	The schnorr key, containing the `G` and `Y` values, and the secrete key value `x` that is associated with them.
c	The random c value sent by the verifier.
d	The disjunct structure that will receive values `c1` and `s1`. ‘

Returns

GOLLE_ERROR, GOLLE_OK, or GOLLE_EMEM.

9.3.2.4 GOLLE_EXTERN golle_error golle_disj_verify (const golle_schnorr_t * *k1*, const golle_schnorr_t * *k2*, const golle_disj_t * *d*)

Verify a proof sent by a prover.

Parameters

k1	The first Schnorr public key.
k2	The second Schnorr public key.
d	The collection of values received from the prover.

Returns

GOLLE_OK, GOLLE_ERROR for `NULL`, or GOLLE_EMEM. If the verification fails, returns GOLLE_ECRYPTO.

9.4 Disjunctive Plaintext Equivalence Proof

Functions

- GOLLE_EXTERN golle_error golle_dispep_setup (const golle_eg_t *r, const golle_eg_t *e1, const golle_eg_t *e2, golle_schnorr_t *k1, golle_schnorr_t *k2, const golle_key_t *key)

  Prepare the disjunctive schnorr key for use by a prover and verifier.

9.4.1 Detailed Description

The DISPEP protocol described in Jakobsson M. and Juels A., Millimix: Mixing in Small Batches, DIMACS Technical Report 99-33, June 1999.

The DISPEP protocol leverages the security of Disjunctive Schnorr Identification in order to prove that an El Gamal ciphertext (α, β) is a re-encryption of one of two difference ciphertexts, without revealing which one.

To use DISPEP, use golle_dispep_setup() to set up the two golle_schnorr_t structures, and then use the results to perform Disjuntive Schnorr proof.

9.4.2 Function Documentation

9.4.2.1 GOLLE_EXTERN golle_error golle_dispep_setup (const golle_eg_t * *r*, const golle_eg_t * *e1*, const golle_eg_t * *e2*, golle_schnorr_t * *k1*, golle_schnorr_t * *k2*, const golle_key_t * *key*)

Prepare the disjunctive schnorr key for use by a prover and verifier.

Parameters

	r	The El Gamal ciphertext that is a re-encryption of either `e1` or `e2`.
	e1	The first potential base ciphertext.
	e2	The second potential base ciphertext.
`out`	*k1*	The first Schnorr key.
`out`	*k2*	The second Schnorr key.
	key	The El Gamal key associated with the re-encryption. Must contain `p`.

Returns

GOLLE_OK for success. GOLLE_ERROR for unexpected `NULL`. GOLLE_EMEM for memory errors. GOLLE_ECRYPTO for encryption errors.

9.5 Key Generation and Distribution

Data Structures

- struct golle_key_t

  A peer's key. Contains the peer's portion of the private key and the public key elements.

Macros

- #define golle_key_clear(k) golle_key_cleanup(k)

  An alias for golle_key_cleanup()

Functions

- GOLLE_INLINE void golle_key_cleanup (golle_key_t *k)

  Frees each member of the golle_key_t k.
- GOLLE_EXTERN golle_error golle_key_gen_public (golle_key_t *key, int bits, int n)

  Generate a full public key description. This should usually be done once, and be distributed amongst each peer for verification.
- GOLLE_EXTERN golle_error golle_key_set_public (golle_key_t *key, const golle_num_t p, const golle_num_t g)

  Set the public key description.
- GOLLE_EXTERN golle_error golle_key_gen_private (golle_key_t *key)

  Generate a private key $x \in \mathbb{Z}_q$ and calculate $h = g^x$. The `h` and `h_product` members will be set.
- GOLLE_EXTERN golle_error golle_key_accum_h (golle_key_t *key, const golle_num_t h)

  *Calculate the product of all h_i in order to get $h = \prod_i h_i$. Call this function successively for each h_i, **not** including the `h` member of the local key.*

9.5.1 Detailed Description

The key distribution protocol is an implementation of Torben Pedersen's Threshold Cryptosystem without a Trusted Party. D.W. Davies (Ed.): Advances in Cryptology - EUROCRYPT'91. LNCS 547, pp. 522-526, 1991.

First, all peers must agree on primes p and q, and a generator g of G_q.

A peer P_i selects a random private key $x_i \in \mathbb{Z}_q$ and calculates $h_i = g^{x_i}$. P_i then publishes a non-malleable commitment to h_i (see Bit Commitment).

Once each other peer has received the commitment, P_i then reveals h_i and the commitment is verified.

For each peer P_i, the public key $h = \prod_i h_i$.

9.5.2 Function Documentation

9.5.2.1 GOLLE_EXTERN golle_error golle_key_accum_h (golle_key_t * *key*, const golle_num_t *h*)

Calculate the product of all h_i in order to get $h = \prod_i h_i$. Call this function successively for *each* h_i, **not** including the `h` member of the local key.

Parameters

key	The key to multiple the `h_product` for.
h	The h value to multiply by the `h_product` of the `key`.

Returns

GOLLE_OK upon success. GOLLE_ERROR if any value is NULL. GOLLE_EMEM if memory couldn't be allocated.

9.5.2.2 GOLLE_INLINE void golle_key_cleanup (golle_key_t * *k*)

Frees each member of the golle_key_t k.

Parameters

k	The key to free.

9.5.2.3 GOLLE_EXTERN golle_error golle_key_gen_private (golle_key_t * *key*)

Generate a private key $x \in \mathbb{Z}_q$ and calculate $h = g^x$. The h and h_product members will be set.

Parameters

key	The key to generate an x for.

Returns

GOLLE_OK if successful. GOLLE_EMEM if a value couldn't be allocated. GOLLE_ERROR if key, or any public key member is NULL.

Warning

This function will overwrite any existing private key value.
This function assumes that the public key values are valid. Always check the return values of golle_key_gen_public and golle_key_set_public.

Note

The h_product member is set to the value of h, but it is not ready to be used until the h of each other peer is received and included in the product using golle_key_accum_h.

9.5.2.4 GOLLE_EXTERN golle_error golle_key_gen_public (golle_key_t * *key*, int *bits*, int *n*)

Generate a full public key description. This should usually be done once, and be distributed amongst each peer for verification.

Parameters

key	The key to generate public values for.
bits	The number of bits in the key. If <= 0, defaults to 1024.
n	The number of attempts to try to find a generator before failing.

Returns

GOLLE_OK if successful, GOLLE_EMEM if any memory failed to be allocated. GOLLE_ERROR if key is NULL. GOLLE_ECRYPTO if something went wrong in the cryptography library. GOLLE_ENOTFOUND if a generator couldn't be found within n attempts.

Warning

This function contains an implicit call to golle_key_cleanup.
Finding a large safe prime `and` a generator can be slow.

9.5.2.5 GOLLE_EXTERN golle_error golle_key_set_public (golle_key_t * *key*, const golle_num_t *p*, const golle_num_t *g*)

Set the public key description.

Parameters

key	The key to set public values for.
p	The value for p
g	The value for g

Returns

GOLLE_OK if all values are valid. GOLLE_ERROR if any parameter is `NULL`. GOLLE_EMEM if a value couldn't be allocated. GOLLE_ENOTPRIME if either p or q fail the test for primality. GOLLE_ECRYPTO if an error occurred during cryptography. Cryptography failures include $q \nmid (p-1)$, and g is not a generator of $\mathbb{G}_q$.

Warning

This function contains an implicit call to golle_key_cleanup.

9.6 ElGamal

Data Structures

- struct golle_eg_t

  ElGamal ciphertex.

Macros

- #define GOLLE_EG_FULL(C) ((C) && (C)->a && (C)->b)

  Check whether the structure is full (both parameters are present).

Functions

- GOLLE_INLINE void golle_eg_clear (golle_eg_t *cipher)

  Clear memory allocated for the ciphertext.
- GOLLE_EXTERN golle_error golle_eg_encrypt (const golle_key_t *key, const golle_num_t m, golle_eg_t *cipher, golle_num_t *rand)

  Encrypt a number $m \in \mathbb{G}_q$.
- GOLLE_EXTERN golle_error golle_eg_reencrypt (const golle_key_t *key, const golle_eg_t *e1, golle_eg_t *e2, golle_num_t *rand)

  Re-encrypt a message (as in the Plaintext Equivalence Proof). The resulting ciphertext, for a random $r \in \mathbb{Z}_q$ will be (ag^t, bh^r).
- GOLLE_EXTERN golle_error golle_eg_decrypt (const golle_key_t *key, const golle_num_t *xi, size_t len, const golle_eg_t *cipher, golle_num_t m)

  Decrypt a message.

9.6.1 Detailed Description

Given a golle_key_t structure properly generated so that the user has a the full public key in `h_product`, this module allows the user to encrypt a message that the group can then work together to decrypt.

Given a golle_key_t struct properly generated so that the user has part of the private key in `x`, this module allows the user to partially decrypt a message that was encrypted by another group member, as above.

To encrypt a message $m \in \mathbb{G}$ using ElGamal, we select $r \xleftarrow{R} \{\mathbb{Z}_q^*\}$, then calculate the ciphertext $c = (g^r, mh^r)$.

To decrypt a ciphertext (a,b), we calculate b/a^x, where $a^x = \prod_{i=1}^{k} a^{x_i}$ for each of the k members of the group.

Decryption is not needed by the Golle protocol, although we include it here for verification and completeness.

9.6.2 Function Documentation

9.6.2.1 GOLLE_INLINE void golle_eg_clear (golle_eg_t * *cipher*)

Clear memory allocated for the ciphertext.

Parameters

cipher	The ciphertext to clear.

9.6.2.2 GOLLE_EXTERN golle_error golle_eg_decrypt (const golle_key_t * *key*, const golle_num_t * *xi*, size_t *len*, const golle_eg_t * *cipher*, golle_num_t *m*)

Decrypt a message.

Parameters

key	The key containing the primes used for modulus operations.
xi	An array of private key values, for each member of the group.
len	The number of keys in `xi`.
cipher	A non-`NULL` ciphertext value from golle_eg_encrypt().
m	The decrypted number.

Returns

GOLLE_ERROR if any parameter is `NULL` or if `len` is `0`. GOLLE_ECRYPTO if an error occurs during cryptography. GOLLE_EMEM if memory allocation fails. GOLLE_OK if successful.

Warning

This function is never actually called during the Golle protocol. it is provided here for completeness of the ElGamal cryptosystem and for the purposes of testing. You may, however, use it as a general purpose cryptosystem if encryption is asymmetric, with one encryptor and one decryptor.

9.6.2.3 GOLLE_EXTERN golle_error golle_eg_encrypt (const golle_key_t * *key*, const golle_num_t *m*, golle_eg_t * *cipher*, golle_num_t * *rand*)

Encrypt a number $m \in \mathbb{G}_q$.

Parameters

	key	The ElGamal public key to use during encryption.
	m	The number to encrypt. $msg \in \mathbb{G}_q$
`out`	*cipher*	A non-`NULL` golle_eg_t structure.
	rand	If the value pointed to is not `NULL`, it will be used as the random value $r \in \mathbb{Z}_q^*$. Otherwise, a random value will be collected and returned as new number, via golle_num_new(). If the argument itself is `NULL`, then a random value will be generated but not returned.

Returns

GOLLE_ERROR if any parameter is `NULL`. GOLLE_EOUTOFRANGE if $m >= q$. GOLLE_ECRYPTO if an error happens during cryptography. GOLLE_EMEM if memory allocation fails. GOLLE_OK if successful.

Note

It is assumed that $m \in \mathbb{G}_q$ by computing $m = g^n \mod q$ prior to encrypting.

9.6.2.4 GOLLE_EXTERN golle_error golle_eg_reencrypt (const golle_key_t * *key*, const golle_eg_t * *e1*, golle_eg_t * *e2*, golle_num_t * *rand*)

Re-encrypt a message (as in the Plaintext Equivalence Proof). The resulting ciphertext, for a random $r \in \mathbb{Z}_q$ will be (ag^t, bh^r).

Parameters

	key	The ElGamal public key used to encrypt the first ciphertext.
	e1	The first ciphertext, already encrypted.
out	*e2*	A non-NULL golle_eg_t structure.
	rand	If the value pointed to is not NULL, it will be used as the random value $r \in \mathbb{Z}_q^*$. Otherwise, a random value will be collected and returned as new number, via golle_num_new(). If the argument itself is NULL, then a random value will be generated but not returned.

Returns

GOLLE_ERROR if any parameter is NULL. GOLLE_ECRYPTO if an error happens during cryptography. GOLLE_EMEM if memory allocation fails. GOLLE_OK if successful.

9.7 The Golle protocol interface

Data Structures

- struct golle_t

  The main Golle structure.

Macros

- #define GOLLE_FACE_UP SIZE_MAX

Typedefs

- typedef golle_error(∗ golle_bcast_commit_t)(golle_t ∗, golle_bin_t ∗, golle_bin_t ∗)

  A callback used to broadcast a commitment.
- typedef golle_error(∗ golle_bcast_secret_t)(golle_t ∗, golle_eg_t ∗, golle_bin_t ∗)

  A callback for broadcasting the secret parts of a commitment.
- typedef golle_error(∗ golle_accept_commit_t)(golle_t ∗, size_t, golle_bin_t ∗, golle_bin_t ∗)

  A callback for accepting the commitment of a peer.
- typedef golle_error(∗ golle_accept_eg_t)(golle_t ∗, size_t, golle_eg_t ∗, golle_bin_t ∗)

  A callback for accepting ciphertext from a peer.
- typedef golle_error(∗ golle_reveal_rand_t)(golle_t ∗, size_t, size_t, golle_num_t)

  A callback for revealing a random number and the randomness used to encrypt it in a previous operation.
- typedef golle_error(∗ golle_accept_rand_t)(golle_t ∗, size_t, size_t ∗, golle_num_t)

  A callback for accepting a random number and the randomness used to encrypt it in a previous operation.
- typedef golle_error(∗ golle_accept_crypt_t)(golle_t ∗, golle_eg_t ∗, size_t)

  A callback for accepting a encrypted selection from a peer.
- typedef golle_error(∗ golle_bcast_crypt_t)(golle_t ∗, const golle_eg_t ∗)

  A callback for broadcasting an encrypted selection.

Functions

- GOLLE_EXTERN golle_error golle_initialise (golle_t ∗golle)

  Establish the group structure. This is the first step to perform before dealing any rounds.
- GOLLE_EXTERN void golle_clear (golle_t ∗golle)

  Releases any memory used by the Golle interface stored in the `reserved` *member.*
- GOLLE_EXTERN golle_error golle_generate (golle_t ∗golle, size_t round, size_t peer)

  Participate in selecting a random element from the set. The behaviour of the implementation will depend on the round number.
- GOLLE_EXTERN golle_error golle_reveal_selection (golle_t ∗golle, size_t ∗selection)

  Call this function after golle_generate() if the local peer is to reveal received random selections as an actual item in the set.
- GOLLE_EXTERN golle_error golle_reduce_selection (golle_t ∗golle, size_t c, size_t ∗collision)

  Call this function after golle_reveal_selection() if the local peer is the only peer receiving a random selection. The peer must reduce the selection and output a proof that it has been done correctly.
- GOLLE_EXTERN golle_error golle_check_selection (golle_t ∗golle, size_t peer, size_t ∗collision)

  Call this function in the golle_reveal_rand_t callback when some other peer is receiving the reduced item. The function will accept proof from the other peer that the item was reduced correctly and will check for collisions.

9.7.1 Detailed Description

The Golle interface is a strong wrapper around most of the subprotocols that make up the Golle protocol. It allows the client code to set up a series of callbacks and have the library perform most of the work. The callbacks are raised when input is required from the client or when data is required to be sent.

Because this interface is basically the implementation of the protocol, if all the client wants to do is "play Mental Poker", then this, and the key distribution module, are the only interfaces needed. All of the other interfaces are provided as a handy reference implementation and as a description of how the inner machinery of the Golle protocol works. However if the client wishes more fine-grained control over the protocol then the headers for each subprotocol are available; this interface can be used as a reference implementation for the protocol as a whole.

Note

There are aspects of this protocol currently missing. As a result, some functionality is **not** available. The shortcomings of the current implementation can be summarised as follows:

- No proof of subset membership or proof of correct decryption is performed.
- Millimix is not implemented, so multiple rounds are not allowed.

The building blocks for these features are implemented in Disjunctive Plaintext Equivalence Proof.

9.7.2 Macro Definition Documentation

9.7.2.1 #define GOLLE_FACE_UP SIZE_MAX

Used to indicate that a selected item is for all peers.

9.7.3 Function Documentation

9.7.3.1 GOLLE_EXTERN golle_error golle_check_selection (golle_t * *golle*, size_t *peer*, size_t * *collision*)

Call this function in the golle_reveal_rand_t callback when some other peer is receiving the reduced item. The function will accept proof from the other peer that the item was reduced correctly and will check for collisions.

Parameters

golle	The golle structure.
peer	The peer from which to accept proof.
collision	If a collision occurs, will be populated with the index of the found collision.

Returns

GOLLE_EMEM for memory errors. GOLLE_ERROR for `NULL` errors. GOLLE_ECRYPTO for cryptography errors. GOLLE_ECOLLISION if the reduced item has already been 'dealt'. GOLLE_OK for success.

Note

If a collision occurs, the selection inditicated by `collision` will be discarded and must be done again. The selection id will not be reused.

9.7.3.2 GOLLE_EXTERN void golle_clear (golle_t * *golle*)

Releases any memory used by the Golle interface stored in the `reserved` member.

Parameters

golle	The structure to release.

Note

This function does not release the key, and does not free the golle structure.

Warning

The Golle structure must be reinitialised with golle_initialise() if it is to be used again.

9.7.3.3 GOLLE_EXTERN golle_error golle_generate (golle_t * *golle*, size_t *round*, size_t *peer*)

Participate in selecting a random element from the set. The behaviour of the implementation will depend on the round number.

Parameters

golle	The golle structure.
round	The round number, zero-based.
peer	The peer who is to receive the selected item. Set to SIZE_MAX if the item is meant to be broadcast.

Returns

GOLLE_ERROR for any NULLs or if `peer` is too large, or if `round > 0` and the first round hasn't been finished yet. GOLLE_ECRYPTO for internal cryptographic errors. GOLLE_ENOCOMMIT if a commitment from a peer is invalid. GOLLE_OK for success.

Note

Selections are indexed internally, starting at zero and incrementing. If a collision occurs, the collision will be discarded but the index will not be reused.

9.7.3.4 GOLLE_EXTERN golle_error golle_initialise (golle_t * *golle*)

Establish the group structure. This is the first step to perform before dealing any rounds.

Parameters

golle	The Golle Structure. Must have a valid key, and `num_peers` and `num_items` must be > 0.

Returns

GOLLE_ERROR if `golle` is `NULL`, or a member is invalid. GOLLE_EMEM if memory allocation fails. GOLLE_ECRYPTO if any internal crypto operation fails (indicates a bad key). Upon success, returns GOLLE_OK.

9.7.3.5 GOLLE_EXTERN golle_error golle_reduce_selection (golle_t * *golle*, size_t *c*, size_t * *collision*)

Call this function after golle_reveal_selection() if the local peer is the only peer receiving a random selection. The peer must reduce the selection and output a proof that it has been done correctly.

Parameters

golle	The golle structure.
c	The reduced item, returned from golle_reveal_selection().
collision	If a collision occurs, will be populated with the index of the found collision.

Returns

GOLLE_EMEM for memory errors. GOLLE_ERROR for `NULL` errors. GOLLE_ECRYPTO for cryptography errors. GOLLE_ECOLLISION if the reduced item has already been 'dealt'. GOLLE_OK for success.

Note

If a collision occurs, the selection inditicated by `collision` will be discarded and must be done again. The selection id will not be reused.

9.7.3.6 GOLLE_EXTERN golle_error golle_reveal_selection (golle_t * *golle*, size_t * *selection*)

Call this function after golle_generate() if the local peer is to reveal received random selections as an actual item in the set.

Parameters

	golle	The golle structure.
`out`	*selection*	Receives the revealed selection.

Returns

GOLLE_EMEM for memory errors. GOLLE_ERROR for `NULL` errors. GOLLE_ECRYPTO for cryptography errors. GOLLE_OK for success.

9.8 Singly-linked lists

Data Structures

- struct golle_list_t

  An opaque pointer to a singly-linked list.
- struct golle_list_iterator_t

  A type used for iterating through all the items in a list.

Functions

- GOLLE_EXTERN golle_error golle_list_new (golle_list_t **list)

  Allocate a new list.
- GOLLE_EXTERN void golle_list_delete (golle_list_t *list)

  Deallocate a list.
- GOLLE_EXTERN size_t golle_list_size (const golle_list_t *list)

  Get the number of items in a list.
- GOLLE_EXTERN golle_error golle_list_top (const golle_list_t *list, void **item)

  Get the element at the head of the list.
- GOLLE_EXTERN golle_error golle_list_push (golle_list_t *list, const void *item, size_t size)

  Append an item to the list. The item will be copied.
- GOLLE_EXTERN golle_error golle_list_push_many (golle_list_t *list, const void *item, size_t size, size_t count)

  Append many identical items to the list. The item will be copied multiple times.
- GOLLE_EXTERN golle_error golle_list_pop (golle_list_t *list)

  Remove the first item in the list.
- GOLLE_EXTERN golle_error golle_list_pop_many (golle_list_t *list, size_t count)

  Remove the first count items from the front of the list.
- GOLLE_EXTERN golle_error golle_list_pop_all (golle_list_t *list)

  Remove all items from a list. The equivalent of.
- GOLLE_EXTERN golle_error golle_list_iterator (golle_list_t *list, golle_list_iterator_t **iter)

  Create an iterator to iterate over the given list. The iterator begins by pointing to before the first element in the list.
- GOLLE_EXTERN void golle_list_iterator_free (golle_list_iterator_t *iter)

  Free any resources associated with an iterator.
- GOLLE_EXTERN golle_error golle_list_iterator_next (golle_list_iterator_t *iter, void **item)

  Get the next value of the iterator.
- GOLLE_EXTERN golle_error golle_list_iterator_reset (golle_list_iterator_t *iter)

  Set the iterator back to its initial state.
- GOLLE_EXTERN golle_error golle_list_insert_at (golle_list_iterator_t *iter, const void *item, size_t size)

  Insert an item into the list at the given location. If the operation is successful, a call to golle_list_iterator_next with the given iter parameter will return the inserted item.
- GOLLE_EXTERN golle_error golle_list_erase_at (golle_list_iterator_t *iter)

  Erase the item at the given position. If the operation is successful, a call to golle_list_iterator_next with the same iter parameter will return the item that previously came after the removed item (i.e. the iterator is set to the item that preceeds the removed item.)

9.8.1 Detailed Description

Contains structures and functions for maintaining a singly-linked list (or a LIFO queue).

9.8.2 Function Documentation

9.8.2.1 GOLLE_EXTERN void golle_list_delete (golle_list_t * *list*)

Deallocate a list.

Parameters

list	The list to be destroyed.

9.8.2.2 GOLLE_EXTERN golle_error golle_list_erase_at (golle_list_iterator_t * *iter*)

Erase the item at the given position. If the operation is successful, a call to golle_list_iterator_next with the same iter parameter will return the item that previously came after the removed item (i.e. the iterator is set to the item that preceeds the removed item.)

Parameters

iter	The location to remove an item from.

Returns

GOLLE_OK if the operation was successful. GOLLE_ENOTFOUND if the iterator is not pointing to an element (it is at the very start or very end of the list). GOLLE_ERROR if `iter` is NULL.

9.8.2.3 GOLLE_EXTERN golle_error golle_list_insert_at (golle_list_iterator_t * *iter,* const void * *item,* size_t *size*)

Insert an item into the list at the given location. If the operation is successful, a call to golle_list_iterator_next with the given iter parameter will return the inserted item.

Parameters

iter	The location to insert the item at. The item will be inserted just after the node that iter currently points to. If iter is in its initial state, the item will be prepended. If iter is at the end of the list, the item will be appended.
item	The item to insert into the list.
size	The size of item.

Returns

GOLLE_OK if the operation was successful. GOLLE_EMEM if the new element couldn't be allocated. GOLLE_ERROR if `iter` is NULL.

9.8.2.4 GOLLE_EXTERN golle_error golle_list_iterator (golle_list_t * *list,* golle_list_iterator_t ** *iter*)

Create an iterator to iterate over the given list. The iterator begins by pointing to before the first element in the list.

Parameters

	list	The list to iterate over.
`out`	*iter*	Receives the address of the new iterator.

Returns

GOLLE_OK if the iterator was created. GOLLE_EMEM if the iterator could't be allocated. GOLLE_ERROR if `list` or `iter` is NULL.

9.8.2.5 GOLLE_EXTERN void golle_list_iterator_free (golle_list_iterator_t * *iter*)

Free any resources associated with an iterator.

Parameters

	iter	The iterator to free.

9.8.2.6 GOLLE_EXTERN golle_error golle_list_iterator_next (golle_list_iterator_t * *iter,* void ** *item*)

Get the next value of the iterator.

Parameters

	iter	The iterator.
`out`	*item*	Is populated with the next item pointed to by the iterator.

Returns

GOLLE_OK if the operation was successful. GOLLE_ERROR if `iter` or `value` is NULL. GOLLE_END if the iterator is at the end of the list.

Warning

The returned pointer is the address of the item in the list. Be wary.

9.8.2.7 GOLLE_EXTERN golle_error golle_list_iterator_reset (golle_list_iterator_t * *iter*)

Set the iterator back to its initial state.

Parameters

	iter	The iterator to rest.

Returns

GOLLE_OK if the operation was successful. GOLLE_ERROR if `iter` is NULL.

9.8.2.8 GOLLE_EXTERN golle_error golle_list_new (golle_list_t ** *list*)

Allocate a new list.

Parameters

`out`	*list*	Pointer which will hold the address of the list.

Returns

GOLLE_OK if successful. GOLLE_EMEM if memory couldn't be allocated. GOLLE_ERROR if `list` is NULL.

9.8.2.9 GOLLE_EXTERN golle_error golle_list_pop (golle_list_t * *list*)

Remove the first item in the list.

Parameters

list	The list to remove from.

Returns

GOLLE_OK if an item was removed. GOLLE_EEMPTY if the `list` is empty. GOLLE_ERROR if `list` is NULL.

9.8.2.10 GOLLE_EXTERN golle_error golle_list_pop_all (golle_list_t * *list*)

Remove all items from a list. The equivalent of.

golle_list_pop_many(list, golle_list_size(list));

Parameters

list	The list to clear.

Returns

GOLLE_OK if the `list` was cleared. GOLLE_ERROR if `list` was NULL.

9.8.2.11 GOLLE_EXTERN golle_error golle_list_pop_many (golle_list_t * *list*, size_t *count*)

Remove the first count items from the front of the list.

Parameters

list	The list to remove from.
count	The number of items to remove.

Returns

GOLLE_OK if an item was removed. GOLLE_EEMPTY if there are less than `count` items in the `list` (note, they will not be removed). GOLLE_ERROR if `list` is NULL.

9.8.2.12 GOLLE_EXTERN golle_error golle_list_push (golle_list_t * *list*, const void * *item*, size_t *size*)

Append an item to the list. The item will be copied.

Parameters

list	The list to append to.
item	The element to append into the list.
size	The size of the element (used in memcpy).

Returns

GOLLE_OK if the item was appended. GOLLE_EMEM if memory couldn't be allocated, or GOLLE_ERROR if `list` if NULL.

9.8.2.13 GOLLE_EXTERN golle_error golle_list_push_many (golle_list_t * *list*, const void * *item*, size_t *size*, size_t *count*)

Append many identical items to the list. The item will be copied multiple times.

Parameters

list	The list to append to.
item	The element to append into the list. If `item` is NULL, `size` is ignored and the item in the list is set to NULL.
size	The size of the element (used in `memcpy`). If `size` is 0, the item will be set to NULL in the `list`.
count	The number of new items to append.

Returns

GOLLE_OK if the item was appended. GOLLE_EMEM if memory couldn't be allocated, or GOLLE_ERROR if `list` if NULL.

9.8.2.14 GOLLE_EXTERN size_t golle_list_size (const golle_list_t * *list*)

Get the number of items in a list.

Parameters

list	The list to test.

Returns

The number of items in the list. If `list` is NULL, returns 0.

9.8.2.15 GOLLE_EXTERN golle_error golle_list_top (const golle_list_t * *list*, void ** *item*)

Get the element at the head of the list.

Parameters

	list	The list to query.
`out`	*item*	Recieves the item value.

Returns

GOLLE_OK if the operation was successful. GOLLE_ERROR if any parameter is NULL. GOLLE_EEMPTY if the `list` is empty.

9.9 Large Numbers

Typedefs

- typedef void * golle_num_t

  Wraps `BIGNUM` in an opaque (OK, maybe translucent) way.

Functions

- GOLLE_EXTERN golle_num_t golle_num_new (void)

  Create a new number.
- GOLLE_EXTERN void golle_num_delete (golle_num_t n)

  Free a number.
- GOLLE_EXTERN golle_num_t golle_num_dup (const golle_num_t i)

  Make a copy of the input.
- GOLLE_EXTERN golle_error golle_num_cpy (golle_num_t dst, const golle_num_t src)

  Make a copy of the input.
- GOLLE_EXTERN golle_num_t golle_num_new_int (size_t i)

  Create a new number from a given native integer.
- GOLLE_EXTERN golle_error golle_num_generate_rand (golle_num_t r, const golle_num_t n)

  Generate a new random number in the range [0, n).
- GOLLE_EXTERN golle_num_t golle_num_rand (const golle_num_t n)

  Generate a new random number in the range [0, n).
- GOLLE_EXTERN golle_error golle_num_rand_bits (golle_num_t r, int bits)

  Generate a new random number of the given size.
- GOLLE_EXTERN int golle_num_cmp (const golle_num_t n1, const golle_num_t n2)

  Compare two numbers.
- GOLLE_EXTERN golle_num_t golle_generate_prime (int bits, int safe, golle_num_t div)

  Generate a pseudo-random `size`-bit prime number.
- GOLLE_EXTERN golle_error golle_test_prime (const golle_num_t p)

  Test a number for proabable primality.
- GOLLE_EXTERN golle_error golle_find_generator (golle_num_t g, const golle_num_t p, const golle_num_t q, int n)

  Find a generator for the multiplicative subgroup of $\mathbb{Z}_p^*$ of order q ($\mathbb{G}_q$).
- GOLLE_EXTERN golle_error golle_num_to_bin (const golle_num_t n, golle_bin_t *bin)

  Write the big-endian binary representation of a number into the given binary buffer. The buffer will be resized to the number of bytes required.
- GOLLE_EXTERN golle_error golle_bin_to_num (const golle_bin_t *bin, golle_num_t n)

  Convert a big-endian binary buffer into a number.
- GOLLE_EXTERN golle_error golle_num_mod_exp (golle_num_t out, const golle_num_t base, const golle_num_t exp, const golle_num_t mod)

  Calculate $m = g^n \mod q$.
- GOLLE_EXTERN golle_error golle_num_print (FILE *file, const golle_num_t num)

  Print a number, in big-endian hexadecimal, to the given file pointer.
- GOLLE_EXTERN golle_error golle_num_xor (golle_num_t out, const golle_num_t x1, const golle_num_t x2)

  XOR two numbers. Increases the size of a number if required.

9.9.1 Detailed Description

This module wraps OpenSSL's `BIGNUM` type. However, it leaves the type opaque, so that these headers do not rely on the OpenSSL headers. If access is required, golle_num_t will cast to a `BIGNUM*`.

Many functions here are simply wrappers around their OpenSSL analogues. This is done for the same reason that we hide the `BIGNUM` type.

9.9.2 Function Documentation

9.9.2.1 GOLLE_EXTERN golle_error golle_bin_to_num (const golle_bin_t * *bin*, golle_num_t *n*)

Convert a big-endian binary buffer into a number.

Parameters

bin	The binary buffer.
n	The number to populate.

Returns

GOLLE_OK on success. GOLLE_ERROR if any parameter is `NULL`. GOLLE_EMEM if memory for the number couldn't be allocated.

9.9.2.2 GOLLE_EXTERN golle_error golle_find_generator (golle_num_t *g*, const golle_num_t *p*, const golle_num_t *q*, int *n*)

Find a generator for the multiplicative subgroup of $\mathbb{Z}_p^*$ of order q ($\mathbb{G}_q$).

Parameters

g	If not `NULL`, will be populated with the found generator.
p	A large prime.
q	Another prime with divides `p`.
n	The number of attempts before failing with GOLLE_ENOTFOUND.

Returns

GOLLE_OK if a generator was found. GOLLE_ERROR if `p` or `g` is `NULL`. GOLLE_EMEM if memory failed to allocate. GOLLE_ECRYPTO if the crypto library fails. GOLLE_ENOTFOUND if a generator could not be found in n attempts.

Warning

This function assumes that `p` and `q` are valid primes, and that `q` divides `p`.

Note

A generator is calculated by taking a random number $h \in \mathbb{Z}_p^*$ and computing $g = h^{(p-1)/q} \mod q$. If $g \neq 1$ then g is a generator. This technique is described in H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications*, 2007, pp. 303-304.

9.9.2.3 GOLLE_EXTERN golle_num_t golle_generate_prime (int *bits*, int *safe*, golle_num_t *div*)

Generate a pseudo-random `size`-bit prime number.

Parameters

bits	The number of bits required.
safe	If non-zero, the algorithm is required to select a safe prime.
div	If not `NULL`, `div` must divide the prime - 1 (i.e. $prime \mod div = 1$).

Returns

A prime golle_num_t, or `NULL` if generation failed.

9.9.2.4 GOLLE_EXTERN int golle_num_cmp (const golle_num_t *n1*, const golle_num_t *n2*)

Compare two numbers.

Parameters

n1	The first number (left-hand side).
n2	The second number (right-hand side).

Returns

`-1` if `n1 < n2`. `1` if `n1 > n2`. `0` if `n1 == n2`.

Warning

No parameter checking is done. If `n1` or `n2` is `NULL`, behaviour is undefined.

9.9.2.5 GOLLE_EXTERN golle_error golle_num_cpy (golle_num_t *dst*, const golle_num_t *src*)

Make a copy of the input.

Parameters

src	The input number to copy.
dst	The output number to copy to.

Returns

GOLLE_OK if successful. GOLLE_ERROR if `src` or `dst` are `NULL`. GOLLE_EMEM if memory allocation failed.

9.9.2.6 GOLLE_EXTERN golle_num_t golle_num_dup (const golle_num_t *i*)

Make a copy of the input.

Parameters

i	The input number to copy.

Returns

A copy of `i`, or `NULL` if allocation fails or `i` is `NULL`.

9.9.2.7 GOLLE_EXTERN golle_error golle_num_generate_rand (golle_num_t *r,* const golle_num_t *n*)

Generate a new random number in the range [0, n).

Parameters

r	The number to store the random value.
n	The upper range of random values.

Returns

GOLLE_ERROR, GOLLE_EMEM, or GOLLE_OK.

9.9.2.8 GOLLE_EXTERN golle_error golle_num_mod_exp (golle_num_t *out,* const golle_num_t *base,* const golle_num_t *exp,* const golle_num_t *mod*)

Calculate $m = g^n \mod q$.

Parameters

out	m
base	g
exp	n
mod	q

Returns

GOLLE_ERROR if any argument is `NULL`. GOLLE_ECRYPTO if the operation fails. GOLLE_EMEM if resources run out. GOLLE_OK if successful.

9.9.2.9 GOLLE_EXTERN golle_num_t golle_num_new (void)

Create a new number.

Returns

A newly-allocated number, or `NULL` if failed.

9.9.2.10 GOLLE_EXTERN golle_num_t golle_num_new_int (size_t *i*)

Create a new number from a given native integer.

Parameters

i	The value to set the newly allocated number.

Returns

A newly-allocated number, or `NULL` if failed.

9.9.2.11 GOLLE_EXTERN golle_error golle_num_print (FILE * *file,* const golle_num_t *num*)

Print a number, in big-endian hexadecimal, to the given file pointer.

Parameters

file	The file pointer to print to.
num	The number to print.

Returns

GOLLE_ERROR if either argument is NULL. GOLLE_EMEM if the buffer allocation failed. GOLLE_OK otherwise.

9.9.2.12 GOLLE_EXTERN golle_num_t golle_num_rand (const golle_num_t *n*)

Generate a new random number in the range [0, n).

Parameters

n	The upper range of random values.

Returns

A newly-allocated random number, or NULL if failed.

9.9.2.13 GOLLE_EXTERN golle_error golle_num_rand_bits (golle_num_t *r,* int *bits*)

Generate a new random number of the given size.

Parameters

r	The number to store the bits in.
bits	The number of bits of randomness to generate.

Returns

GOLLE_ERROR, GOLLE_ECRYPTO, or GOLLE_OK.

9.9.2.14 GOLLE_EXTERN golle_error golle_num_to_bin (const golle_num_t *n,* golle_bin_t * *bin*)

Write the big-endian binary representation of a number into the given binary buffer. The buffer will be resized to the number of bytes required.

Parameters

n	The number to write out.
bin	The buffer that will be filled with the number.

Returns

GOLLE_OK on success. GOLLE_ERROR if any parameter is NULL. GOLLE_EMEM if memory for the buffer couldn't be allocated.

9.9.2.15 GOLLE_EXTERN golle_error golle_num_xor (golle_num_t *out,* const golle_num_t *x1,* const golle_num_t *x2*)

XOR two numbers. Increases the size of a number if required.

Parameters

out	*out*	The result of x1 ^ x2
	x1	The first number.
	x2	The second number.

Returns

GOLLE_OK, or GOLLE_EMEM, or GOLLE_ERROR.

9.9.2.16 GOLLE_EXTERN golle_error golle_test_prime (const golle_num_t *p*)

Test a number for proabable primality.

Parameters

p	A possible prime.

Returns

GOLLE_ERROR if the number is NULL. GOLLE_EMEM if memory allocation fails. GOLLE_NOT_PRIME if the number is definitely composite, or GOLLE_PROBABLY_PRIME if the number passes the primality test.

9.10 Plaintext Equivalence Proof

Functions

- GOLLE_EXTERN golle_error golle_pep_prover (const golle_key_t *egKey, const golle_num_t k, const golle_num_t z, golle_schnorr_t *key)

  Make a Schnorr public key, (G,Y) out of the ElGamal public key and store the private key.
- GOLLE_EXTERN golle_error golle_pep_verifier (const golle_key_t *egKey, const golle_num_t z, const golle_eg_t *e1, const golle_eg_t *e2, golle_schnorr_t *key)

  Make a Schnorr public key, (G,Y) from two ciphertexts. The private key (i.e. the reencryption factor) is not known. This function is used by the verifier to check if the two ciphertexts are the same.

9.10.1 Detailed Description

The PEP protocol described in Jakobsson M. and Juels A., Millimix: Mixing in Small Batches, DIMACS Technical Report 99-33, June 1999.

Consider the El Gamal encryptions (a,b) and (c,d) for some plaintext m. PEP allows the prover to prove that both ciphertexts are encryptions of the same plaintext, without revealing what the plaintext is.

First, it's easy to see due to the homomorphic property of El Gamal that if (a,b) and (c,d) are encryptions of the same plaintext, then $(a/c,b/d)$ is an encryption of 1 and forms a Schnorr Identification Algorithm public key. The Schnorr Identification Algorithm is then used to complete the PEP protocol.

9.10.2 Function Documentation

9.10.2.1 GOLLE_EXTERN golle_error golle_pep_prover (const golle_key_t * *egKey*, const golle_num_t *k*, const golle_num_t *z*, golle_schnorr_t * *key*)

Make a Schnorr public key, (G,Y) out of the ElGamal public key and store the private key.

Parameters

	egKey	The key used in the encryption and reencryption.
	k	The random number used in the reencryption. Becomes the secret key x.
	z	A random number in $\mathbb{Z}_q$ chosen by the verifier.
out	*key*	The key to construct.

Returns

GOLLE_OK, GOLLE_EMEM, or GOLLE_ERROR.

9.10.2.2 GOLLE_EXTERN golle_error golle_pep_verifier (const golle_key_t * *egKey*, const golle_num_t *z*, const golle_eg_t * *e1*, const golle_eg_t * *e2*, golle_schnorr_t * *key*)

Make a Schnorr public key, (G,Y) from two ciphertexts. The private key (i.e. the reencryption factor) is not known. This function is used by the verifier to check if the two ciphertexts are the same.

Parameters

	egKey	The ElGamal key used in the encryption.
	z	A random number in $\mathbb{Z}_q$ chosen by the verifier.
	e1	The first ciphertext.
	e2	The second ciphertext.
out	*key*	The key to construct.

Returns

GOLLE_OK, GOLLE_EMEM, or GOLLE_ERROR.

9.11 Random Data

Functions

- GOLLE_EXTERN golle_error golle_random_seed (void)

  Seed the system's random generator.
- GOLLE_EXTERN golle_error golle_random_generate (golle_bin_t *buffer)

  Fill the buffer with random data.
- GOLLE_EXTERN golle_error golle_random_clear (void)

  Safely destroy the random state. This function should be called before application exit.

9.11.1 Detailed Description

Functions provide wrappers around OpenSSL's random data generation functions. The golle_random_seed() function will attempt to set up a hardware random number generator if one is available.

A well-behaved application will call golle_random_clear() before exiting.

9.11.2 Function Documentation

9.11.2.1 GOLLE_EXTERN golle_error golle_random_clear (void)

Safely destroy the random state. This function should be called before application exit.

Returns

Always returns GOLLE_OK.

9.11.2.2 GOLLE_EXTERN golle_error golle_random_generate (golle_bin_t * *buffer*)

Fill the buffer with random data.

Parameters

buffer	A buffer to be filled. The `bin` part will be filled with `size` bytes.

Returns

GOLLE_ERROR or GOLLE_OK.

9.11.2.3 GOLLE_EXTERN golle_error golle_random_seed (void)

Seed the system's random generator.

Returns

GOLLE_OK or GOLLE_ERROR.

9.12 Schnorr Identification Algorithm

Data Structures

- struct golle_schnorr_t

  A key used for the Schnorr Identification Algorithm.

Functions

- GOLLE_INLINE void golle_schnorr_clear (golle_schnorr_t *key)

  Clear all number values in a Schnorr key.
- GOLLE_EXTERN golle_error golle_schnorr_commit (const golle_schnorr_t *key, golle_num_t r, golle_num_t t)

  For an ElGamal public key, generate a random r and calculate t.
- GOLLE_EXTERN golle_error golle_schnorr_prove (const golle_schnorr_t *key, golle_num_t s, const golle_num_t r, const golle_num_t c)

  Calculate $s = r + cx$.
- GOLLE_EXTERN golle_error golle_schnorr_verify (const golle_schnorr_t *key, const golle_num_t s, const golle_num_t t, const golle_num_t c)

  Verify $g^s = ty^c$.

9.12.1 Detailed Description

Given a cyclic group $\mathbb{G}_q$ of order q with generator g (e.g. from an El Gamal public key), and plaintext $x = log_g y$, the prover picks $t = g^r$ for some random $r \in \mathbb{Z}_q$ and send t as a commitment. The verifier sends a challenge $c \in \mathbb{Z}_q$, and the prover then responds with $s = cx + r$. The verifier must then verify that $g^s = ty^c$.

9.12.2 Function Documentation

9.12.2.1 GOLLE_INLINE void golle_schnorr_clear (golle_schnorr_t * *key*)

Clear all number values in a Schnorr key.

Parameters

key	The key to clear.

9.12.2.2 GOLLE_EXTERN golle_error golle_schnorr_commit (const golle_schnorr_t * *key*, golle_num_t *r*, golle_num_t *t*)

For an ElGamal public key, generate a random r and calculate t.

Parameters

	key	A key containing g and q.
out	*r*	r
out	*t*	t

Returns

GOLLE_OK, GOLLE_ERROR for NULL, or GOLLE_EMEM.

9.12.2.3 GOLLE_EXTERN golle_error golle_schnorr_prove (const golle_schnorr_t * *key*, golle_num_t *s*, const golle_num_t *r*, const golle_num_t *c*)

Calculate $s = r + cx$.

Parameters

	key	A key containing q, and private key x.
out	*s*	s
	r	r
	c	c

Returns

GOLLE_OK, GOLLE_ERROR for NULL, or GOLLE_EMEM.

9.12.2.4 GOLLE_EXTERN golle_error golle_schnorr_verify (const golle_schnorr_t * *key*, const golle_num_t *s*, const golle_num_t *t*, const golle_num_t *c*)

Verify $g^s = ty^c$.

Parameters

key	A key containing g, h, and q.
s	s
t	t
c	c

Returns

GOLLE_OK, GOLLE_ERROR for NULL, or GOLLE_EMEM. If the verification fails, returns GOLLE_ECRYPTO.

Chapter 10

Data Structure Documentation

10.1 golle_bin_t Struct Reference

Represents a binary buffer.

`#include <golle/bin.h>`

Data Fields

- size_t size
- void * bin

10.1.1 Detailed Description

Represents a binary buffer.

Warning

You can fill in the values of this structure if necessary, but use the golle_bin_new function whenever possible to avoid disparity between the `size` member and the allocated size of `bin`.

10.1.2 Field Documentation

10.1.2.1 void* golle_bin_t::bin

Binary bytes.

10.1.2.2 size_t golle_bin_t::size

Size, in bytes, of bin.

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

- include/golle/bin.h

10.2 golle_commit_t Struct Reference

Holds the values for a bit commitment.

`#include <golle/commit.h>`

Data Fields

- golle_bin_t * secret
- golle_bin_t * rsend
- golle_bin_t * rkeep
- golle_bin_t * hash

10.2.1 Detailed Description

Holds the values for a bit commitment.

10.2.2 Field Documentation

10.2.2.1 golle_bin_t* golle_commit_t::hash

The hash of the other members.

10.2.2.2 golle_bin_t* golle_commit_t::rkeep

The second random value. Kept secret.

10.2.2.3 golle_bin_t* golle_commit_t::rsend

The first random value. Sent along with `hash`.

10.2.2.4 golle_bin_t* golle_commit_t::secret

Holds the secret of the originating user.

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

- include/golle/commit.h

10.3 golle_disj_t Struct Reference

A structure to store all of the revelant values required by the Disjunctive Schnorr Identification protocol.

`#include <golle/disj.h>`

Data Fields

- golle_num_t r1
- golle_num_t c1
- golle_num_t c2
- golle_num_t t1
- golle_num_t t2
- golle_num_t s1
- golle_num_t s2

10.3.1 Detailed Description

A structure to store all of the revelant values required by the Disjunctive Schnorr Identification protocol.

10.3.2 Field Documentation

10.3.2.1 golle_num_t golle_disj_t::c1

The first generated challenge value.

10.3.2.2 golle_num_t golle_disj_t::c2

The second generated challenge value.

10.3.2.3 golle_num_t golle_disj_t::r1

The first generated random value.

10.3.2.4 golle_num_t golle_disj_t::s1

The first calculated s value.

10.3.2.5 golle_num_t golle_disj_t::s2

The second calculated s value.

10.3.2.6 golle_num_t golle_disj_t::t1

The first calculated t value.

10.3.2.7 golle_num_t golle_disj_t::t2

The second calculated t value.

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

- include/golle/disj.h

10.4 golle_eg_t Struct Reference

ElGamal ciphertex.

`#include <golle/elgamal.h>`

Data Fields

- golle_num_t a
- golle_num_t b

10.4.1 Detailed Description

ElGamal ciphertex.

10.4.2 Field Documentation

10.4.2.1 golle_num_t golle_eg_t::a

The first ciphertext element.

10.4.2.2 golle_num_t golle_eg_t::b

The second ciphertext element.

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

- include/golle/elgamal.h

10.5 golle_key_t Struct Reference

A peer's key. Contains the peer's portion of the private key and the public key elements.

`#include <golle/distribute.h>`

Data Fields

- golle_num_t p
- golle_num_t q
- golle_num_t g
- golle_num_t x
- golle_num_t h
- golle_num_t h_product

10.5.1 Detailed Description

A peer's key. Contains the peer's portion of the private key and the public key elements.

10.5.2 Field Documentation

10.5.2.1 golle_num_t golle_key_t::g

A generator for $\mathbb{G}_q$

10.5.2.2 golle_num_t golle_key_t::h

The value g^x. Computed when x is generated.

10.5.2.3 golle_num_t golle_key_t::h_product

The computed $\prod_i h_i$ from successive calls to golle_key_accum_h.

10.5.2.4 golle_num_t golle_key_t::p

A 1024-bit prime st. $\alpha q + 1 = p$.

10.5.2.5 golle_num_t golle_key_t::q

The value $q = (p-1)/2$.

10.5.2.6 golle_num_t golle_key_t::x

A value $x \in \mathbb{Z}_q$.

Warning

This is the private key.

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

- include/golle/distribute.h

10.6 golle_list_iterator_t Struct Reference

A type used for iterating through all the items in a list.

`#include <golle/list.h>`

10.6.1 Detailed Description

A type used for iterating through all the items in a list.

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

- include/golle/list.h

10.7 golle_list_t Struct Reference

An opaque pointer to a singly-linked list.

`#include <golle/list.h>`

10.7.1 Detailed Description

An opaque pointer to a singly-linked list.

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

- include/golle/list.h

10.8 golle_schnorr_t Struct Reference

A key used for the Schnorr Identification Algorithm.

`#include <golle/schnorr.h>`

Data Fields

- golle_num_t G
- golle_num_t Y
- golle_num_t x
- golle_num_t p
- golle_num_t q

10.8.1 Detailed Description

A key used for the Schnorr Identification Algorithm.

10.8.2 Field Documentation

10.8.2.1 golle_num_t golle_schnorr_t::G

The value G, in the algorithm.

10.8.2.2 golle_num_t golle_schnorr_t::p

The p value, a large prime.

10.8.2.3 golle_num_t golle_schnorr_t::q

The q value, the group order.

10.8.2.4 golle_num_t golle_schnorr_t::x

The private key.

10.8.2.5 golle_num_t golle_schnorr_t::Y

The value Y, in the algorithm.

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

- include/golle/schnorr.h

10.9 golle_t Struct Reference

The main Golle structure.

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

Data Fields

- size_t num_peers
- size_t num_items
- golle_key_t * key
- golle_bcast_commit_t bcast_commit
- golle_bcast_secret_t bcast_secret

- golle_accept_commit_t accept_commit
- golle_accept_eg_t accept_eg
- golle_reveal_rand_t reveal_rand
- golle_accept_rand_t accept_rand
- golle_accept_crypt_t accept_crypt
- golle_bcast_crypt_t bcast_crypt
- void * reserved

10.9.1 Detailed Description

The main Golle structure.

Note

All of the callbacks must be filled out in order the the protocol to work. If the protocol comes across a `NULL` callback at any point, it will fail and the fail will propagate all the way to the callsite. It can be difficult for the client to figure out at what point the protocol failed, so always check that the callbacks are set appropriately. Also note that the number of peers, including the local client, must be invariant. If at any point the number of peers changes, the current "game" will be invalid and must be started from the first round again.

10.9.2 Field Documentation

10.9.2.1 golle_accept_commit_t golle_t::accept_commit

The callback which will be invoked when the protocol requires a commitment from a peer. The client should receive the commitment from the designated peer in the first parameter and return it by populating the final two buffers which correspond to `rsend` and `hash` respectively.

10.9.2.2 golle_accept_crypt_t golle_t::accept_crypt

The callback which will be invoked when the protocol needs to receive an encrypted selection from another peer. This occurs when a selection has been revealed to one peer secretly (e.g. revealing a card face-down to one player). The callback should receive the ciphertext from the peer in paramter 2 and populate the argument in parameter 1.

10.9.2.3 golle_accept_eg_t golle_t::accept_eg

The callback which will be invoked when the protocol requires a ciphertext from a peer. The client should receive the ciphertext buffer from the peer indicated in the first parameter and return it in the second parameter. The ciphertext corresponds to the `secret` member of a commitment. The third parameter corresponds to the `rkeep` buffer of a commitment. Thus the protocol will receive the full commitment for verification.

10.9.2.4 golle_accept_rand_t golle_t::accept_rand

The callback which will be invoked when the protocol needs to receive a random number and the randomness that was used to encrypt it in a previous operation. The first parameter will indicate the peer to receive it from.

10.9.2.5 golle_bcast_commit_t golle_t::bcast_commit

The callback which will be invoked when a commitment should be send to all peers. The parameters are `rsend` and `hash`.

10.9.2.6 golle_bcast_crypt_t golle_t::bcast_crypt

The callback which will be invoked when the protocol needs to send an encrypted selection to every other peer. This occurs when a selection has been revealed to one peer secretly (e.g. revealing a card face-down to one player). The callback should broadcast the ciphertext argument to all other peers.

10.9.2.7 golle_bcast_secret_t golle_t::bcast_secret

The callback which will be invoked when a commitment's secret values should be revealed to all peers.

10.9.2.8 golle_key_t* golle_t::key

The ElGamal key, which must be set up via the Key Generation and Distribution module prior to using the Golle interface.

10.9.2.9 size_t golle_t::num_items

The number of distinct items in the set (e.g. number of cards in a deck).

10.9.2.10 size_t golle_t::num_peers

The number of peers connected to.

10.9.2.11 void* golle_t::reserved

Reserved for private data used by the implementation. Do not set. Do not clear. Just leave it alone.

10.9.2.12 golle_reveal_rand_t golle_t::reveal_rand

The callback which will be invoked when the protocol needs to reveal a random number and the randomness that was used to encrypt it in a previous operation. The first parameter will indicate the peer to send it to (or GOLLE_FACE_UP if it is to be broadcast). After being sent, the local client should do one of two things:

1. If the peers that reveal the value do not include the local client, then the local client should call golle_check_selection().
2. If the local client must receive the selection, then the callback should call golle_reveal_selection(). If the local client is the *only* peer to reveal the selection, it must follow up with a call to golle_reduce_selection().

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

- include/golle/golle.h

Chapter 11

File Documentation

11.1 include/golle/bin.h File Reference

Defines a structure for a binary buffer.

```
#include "errors.h"
#include "types.h"
#include "platform.h"
```

Data Structures

- struct golle_bin_t

  Represents a binary buffer.

Macros

- #define golle_bin_clear(b) golle_bin_release(b)

  An alias for golle_bin_release.

Functions

- GOLLE_EXTERN golle_error golle_bin_init (golle_bin_t *buff, size_t size)

  Initialise the inner buffer of a non-dynamic buffer. A typical use case is declaring on the stack or as a non-pointer member of another structure.
- GOLLE_EXTERN void golle_bin_release (golle_bin_t *buff)

  Releases the inner buffer without releasing the golle_bin_t structure itself. Useful for releasing resources allocated with golle_bin_init().
- GOLLE_EXTERN golle_bin_t * golle_bin_new (size_t size)

  Create a new binary buffer of a given size. The data block is zeroed out before returning.
- GOLLE_EXTERN void golle_bin_delete (golle_bin_t *buff)

  Deallocates resources held by a golle_bin_t structure.
- GOLLE_EXTERN golle_bin_t * golle_bin_copy (const golle_bin_t *buff)

  Makes a copy of `buff` via golle_bin_new.
- GOLLE_EXTERN golle_error golle_bin_resize (golle_bin_t *buff, size_t size)

  Resize the buffer.

11.1.1 Detailed Description

Defines a structure for a binary buffer.

Author

Anthony Arnold

Copyright

MIT License

Date

2014 It is best to use the golle_bin_t struct in two distinct ways. Either allocate your own buffer and then assign the `bin` and `size` members of the struct yourself (ensuring that `size` is correct) and then free the buffer yourself. Or, use the golle_bin_new and golle_bin_delete functions to handle the allocation and deallocation for you. Mixing these two techniques *could* lead to problems, even though the golle_bin_delete function is defined to try to detect such problems.

11.2 include/golle/commit.h File Reference

Defines functions for a commitment scheme.

```
#include "bin.h"
#include "platform.h"
#include "errors.h"
```

Data Structures

- struct golle_commit_t

  Holds the values for a bit commitment.

Functions

- GOLLE_EXTERN golle_commit_t * golle_commit_new (const golle_bin_t *secret)

  Generate a new bit commitment to a value.
- GOLLE_EXTERN void golle_commit_delete (golle_commit_t *commitment)

  Free resources allocated by a call to golle_commit_new.
- GOLLE_EXTERN golle_error golle_commit_verify (const golle_commit_t *commitment)

  Verify a commmitment.
- GOLLE_INLINE void golle_commit_clear (golle_commit_t *commit)

  Release the buffers associated with a commit without freeing the commit structure itself.
- GOLLE_EXTERN golle_error golle_commit_copy (golle_commit_t *dest, const golle_commit_t *src)

  Copy a commit structure, bin for bin.

11.2.1 Detailed Description

Defines functions for a commitment scheme.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.3 include/golle/disj.h File Reference

Disjunctive Schnorr Identification.

```
#include "platform.h"
#include "schnorr.h"
#include "numbers.h"
#include "types.h"
```

Data Structures

- struct golle_disj_t

  A structure to store all of the revelant values required by the Disjunctive Schnorr Identification protocol.

Functions

- GOLLE_INLINE void golle_disj_clear (golle_disj_t *d)

  Clear all numbers out of a disjuntive schnorr structure.
- GOLLE_EXTERN golle_error golle_disj_commit (const golle_schnorr_t *unknown, const golle_schnorr_t *known, golle_disj_t *d)

  Generate the commitments `t1` and `s2` to be sent to the verifier.
- GOLLE_EXTERN golle_error golle_disj_prove (const golle_schnorr_t *unknown, const golle_schnorr_t *known, const golle_num_t c, golle_disj_t *d)

  Output the proof that `x` is known. `s1`, `s2`, `c1`, and `c2` are sent to the verifier.
- GOLLE_EXTERN golle_error golle_disj_verify (const golle_schnorr_t *k1, const golle_schnorr_t *k2, const golle_disj_t *d)

  Verify a proof sent by a prover.

11.3.1 Detailed Description

Disjunctive Schnorr Identification.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.4 include/golle/dispep.h File Reference

DISPEP protocol.

```
#include "platform.h"
#include "elgamal.h"
#include "schnorr.h"
#include "errors.h"
```

Functions

- GOLLE_EXTERN golle_error golle_dispep_setup (const golle_eg_t *r, const golle_eg_t *e1, const golle_eg_t *e2, golle_schnorr_t *k1, golle_schnorr_t *k2, const golle_key_t *key)

  Prepare the disjunctive schnorr key for use by a prover and verifier.

11.4.1 Detailed Description

DISPEP protocol.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.5 include/golle/distribute.h File Reference

Defines the protocol for generating a distributed public/private key pair.

```
#include "platform.h"
#include "errors.h"
#include "numbers.h"
```

Data Structures

- struct golle_key_t

  A peer's key. Contains the peer's portion of the private key and the public key elements.

Macros

- #define golle_key_clear(k) golle_key_cleanup(k)

  An alias for golle_key_cleanup()

Functions

- GOLLE_INLINE void golle_key_cleanup (golle_key_t *k)

  Frees each member of the golle_key_t k.
- GOLLE_EXTERN golle_error golle_key_gen_public (golle_key_t *key, int bits, int n)

  Generate a full public key description. This should usually be done once, and be distributed amongst each peer for verification.
- GOLLE_EXTERN golle_error golle_key_set_public (golle_key_t *key, const golle_num_t p, const golle_num_t g)

  Set the public key description.
- GOLLE_EXTERN golle_error golle_key_gen_private (golle_key_t *key)

  Generate a private key $x \in \mathbb{Z}_q$ and calculate $h = g^x$. The `h` and `h_product` members will be set.
- GOLLE_EXTERN golle_error golle_key_accum_h (golle_key_t *key, const golle_num_t h)

  *Calculate the product of all h_i in order to get $h = \prod_i h_i$. Call this function successively for each h_i, **not** including the `h` member of the local key.*

11.5.1 Detailed Description

Defines the protocol for generating a distributed public/private key pair.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.6 include/golle/elgamal.h File Reference

Describes functions for performing distributed ElGamal cryptography.

```
#include "platform.h"
#include "distribute.h"
#include "numbers.h"
#include "errors.h"
```

Data Structures

- struct golle_eg_t

  ElGamal ciphertex.

Macros

- #define GOLLE_EG_FULL(C) ((C) && (C)->a && (C)->b)

  Check whether the structure is full (both parameters are present).

Functions

- GOLLE_INLINE void golle_eg_clear (golle_eg_t *cipher)

  Clear memory allocated for the ciphertext.
- GOLLE_EXTERN golle_error golle_eg_encrypt (const golle_key_t *key, const golle_num_t m, golle_eg_t *cipher, golle_num_t *rand)

  Encrypt a number $m \in \mathbb{G}_q$.
- GOLLE_EXTERN golle_error golle_eg_reencrypt (const golle_key_t *key, const golle_eg_t *e1, golle_eg_t *e2, golle_num_t *rand)

  Re-encrypt a message (as in the Plaintext Equivalence Proof). The resulting ciphertext, for a random $r \in \mathbb{Z}_q$ will be (ag^t, bh^r).
- GOLLE_EXTERN golle_error golle_eg_decrypt (const golle_key_t *key, const golle_num_t *xi, size_t len, const golle_eg_t *cipher, golle_num_t m)

  Decrypt a message.

11.6.1 Detailed Description

Describes functions for performing distributed ElGamal cryptography.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.7 include/golle/errors.h File Reference

Error constants.

```
#include "platform.h"
```

Macros

- #define GOLLE_ASSERT(x, r) do { if ((x) == 0) { return (r); } } while(0)
- #define GOLLE_UNUSED(x) (void)(x)

Enumerations

- enum golle_error {
  GOLLE_OK = 0, GOLLE_ERROR = -1, GOLLE_EMEM = -2, GOLLE_EEXISTS = -3,
  GOLLE_ENOTFOUND = -4, GOLLE_EEMPTY = -5, GOLLE_EOUTOFRANGE = -6, GOLLE_ETOOFEW = -7,
  GOLLE_EINVALID = -8, GOLLE_ENOTPRIME = -9, GOLLE_ENOCOMMIT = -10, GOLLE_ECRYPTO = -11,
  GOLLE_EABORT = -12, GOLLE_ECOLLISION = -13, GOLLE_END = 1, GOLLE_COMMIT_PASSED = 1,
  GOLLE_COMMIT_FAILED = 0, GOLLE_PROBABLY_PRIME = 1, GOLLE_NOT_PRIME = 0 }

11.7.1 Detailed Description

Error constants.

11.7.2 Macro Definition Documentation

11.7.2.1 #define GOLLE_ASSERT(*x, r*) do { if ((x) == 0) { return (r); } } while(0)

A shorthand way of stating assertions. Only use inside a function which returns non-void.

11.7.2.2 #define GOLLE_UNUSED(*x*) (void)(x)

Avoid compile-time warnings about unused parameters.

11.7.3 Enumeration Type Documentation

11.7.3.1 enum golle_error

Error codes and return values.

Enumerator:

GOLLE_OK Success

GOLLE_ERROR General error code.

GOLLE_EMEM Out of memory or resources.

GOLLE_EEXISTS The specified element already exists.

GOLLE_ENOTFOUND The specified element does not exist.

GOLLE_EEMPTY The container has no elements.

GOLLE_EOUTOFRANGE The given size or index is invalid.

GOLLE_ETOOFEW There are not enough elements available.

GOLLE_EINVALID Requested an invalid operation.

GOLLE_ENOTPRIME The given number failed the test for primality.

GOLLE_ENOCOMMIT The given commitment failed.

GOLLE_ECRYPTO An error occurred during cryptography.

GOLLE_EABORT The operation should abort.

GOLLE_ECOLLISION A collision between two selections occured.

GOLLE_END An iterator has reached the end of a sequence.

GOLLE_COMMIT_PASSED Bit commitment verification passed.

GOLLE_COMMIT_FAILED Bit commitment verification failed.

GOLLE_PROBABLY_PRIME The number has passed the primality test.

GOLLE_NOT_PRIME The number is definitely not prime.

11.8 include/golle/golle.h File Reference

Golle interface.

```
#include "platform.h"
#include "types.h"
#include "distribute.h"
#include "errors.h"
#include "commit.h"
#include "elgamal.h"
```

Data Structures

- struct golle_t

  The main Golle structure.

Macros

- #define GOLLE_FACE_UP SIZE_MAX

Typedefs

- typedef golle_error(∗ golle_bcast_commit_t)(golle_t ∗, golle_bin_t ∗, golle_bin_t ∗)

  A callback used to broadcast a commitment.
- typedef golle_error(∗ golle_bcast_secret_t)(golle_t ∗, golle_eg_t ∗, golle_bin_t ∗)

  A callback for broadcasting the secret parts of a commitment.
- typedef golle_error(∗ golle_accept_commit_t)(golle_t ∗, size_t, golle_bin_t ∗, golle_bin_t ∗)

  A callback for accepting the commitment of a peer.
- typedef golle_error(∗ golle_accept_eg_t)(golle_t ∗, size_t, golle_eg_t ∗, golle_bin_t ∗)

  A callback for accepting ciphertext from a peer.
- typedef golle_error(∗ golle_reveal_rand_t)(golle_t ∗, size_t, size_t, golle_num_t)

  A callback for revealing a random number and the randomness used to encrypt it in a previous operation.
- typedef golle_error(∗ golle_accept_rand_t)(golle_t ∗, size_t, size_t ∗, golle_num_t)

  A callback for accepting a random number and the randomness used to encrypt it in a previous operation.
- typedef golle_error(∗ golle_accept_crypt_t)(golle_t ∗, golle_eg_t ∗, size_t)

  A callback for accepting a encrypted selection from a peer.
- typedef golle_error(∗ golle_bcast_crypt_t)(golle_t ∗, const golle_eg_t ∗)

  A callback for broadcasting an encrypted selection.

Functions

- GOLLE_EXTERN golle_error golle_initialise (golle_t ∗golle)

  Establish the group structure. This is the first step to perform before dealing any rounds.
- GOLLE_EXTERN void golle_clear (golle_t ∗golle)

  Releases any memory used by the Golle interface stored in the `reserved` member.
- GOLLE_EXTERN golle_error golle_generate (golle_t ∗golle, size_t round, size_t peer)

  Participate in selecting a random element from the set. The behaviour of the implementation will depend on the round number.
- GOLLE_EXTERN golle_error golle_reveal_selection (golle_t ∗golle, size_t ∗selection)

  Call this function after golle_generate() if the local peer is to reveal received random selections as an actual item in the set.
- GOLLE_EXTERN golle_error golle_reduce_selection (golle_t ∗golle, size_t c, size_t ∗collision)

  Call this function after golle_reveal_selection() if the local peer is the only peer receiving a random selection. The peer must reduce the selection and output a proof that it has been done correctly.

- GOLLE_EXTERN golle_error golle_check_selection (golle_t *golle, size_t peer, size_t *collision)

  Call this function in the golle_reveal_rand_t callback when some other peer is receiving the reduced item. The function will accept proof from the other peer that the item was reduced correctly and will check for collisions.

11.8.1 Detailed Description

Golle interface.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.9 include/golle/list.h File Reference

Describes the structures and operations for working with singly-linked lists.

```
#include "platform.h"
#include "errors.h"
#include "types.h"
```

Functions

- GOLLE_EXTERN golle_error golle_list_new (golle_list_t **list)

  Allocate a new list.
- GOLLE_EXTERN void golle_list_delete (golle_list_t *list)

  Deallocate a list.
- GOLLE_EXTERN size_t golle_list_size (const golle_list_t *list)

  Get the number of items in a list.
- GOLLE_EXTERN golle_error golle_list_top (const golle_list_t *list, void **item)

  Get the element at the head of the list.
- GOLLE_EXTERN golle_error golle_list_push (golle_list_t *list, const void *item, size_t size)

  Append an item to the list. The item will be copied.
- GOLLE_EXTERN golle_error golle_list_push_many (golle_list_t *list, const void *item, size_t size, size_t count)

  Append many identical items to the list. The item will be copied multiple times.
- GOLLE_EXTERN golle_error golle_list_pop (golle_list_t *list)

  Remove the first item in the list.
- GOLLE_EXTERN golle_error golle_list_pop_many (golle_list_t *list, size_t count)

  Remove the first count items from the front of the list.
- GOLLE_EXTERN golle_error golle_list_pop_all (golle_list_t *list)

  Remove all items from a list. The equivalent of.
- GOLLE_EXTERN golle_error golle_list_iterator (golle_list_t *list, golle_list_iterator_t **iter)

  Create an iterator to iterate over the given list. The iterator begins by pointing to before the first element in the list.

- GOLLE_EXTERN void golle_list_iterator_free (golle_list_iterator_t *iter)

  Free any resources associated with an iterator.
- GOLLE_EXTERN golle_error golle_list_iterator_next (golle_list_iterator_t *iter, void **item)

  Get the next value of the iterator.
- GOLLE_EXTERN golle_error golle_list_iterator_reset (golle_list_iterator_t *iter)

  Set the iterator back to its initial state.
- GOLLE_EXTERN golle_error golle_list_insert_at (golle_list_iterator_t *iter, const void *item, size_t size)

  Insert an item into the list at the given location. If the operation is successful, a call to golle_list_iterator_next with the given iter parameter will return the inserted item.
- GOLLE_EXTERN golle_error golle_list_erase_at (golle_list_iterator_t *iter)

  Erase the item at the given position. If the operation is successful, a call to golle_list_iterator_next with the same iter parameter will return the item that previously came after the removed item (i.e. the iterator is set to the item that preceeds the removed item.)

11.9.1 Detailed Description

Describes the structures and operations for working with singly-linked lists.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.10 include/golle/numbers.h File Reference

Describes various available number functions including primality functions, generator finding, and arithmetic of large numbers.

```
#include "platform.h"
#include "errors.h"
#include "bin.h"
#include <stdio.h>
```

Typedefs

- typedef void * golle_num_t

  Wraps `BIGNUM` in an opaque (OK, maybe translucent) way.

Functions

- GOLLE_EXTERN golle_num_t golle_num_new (void)

  Create a new number.
- GOLLE_EXTERN void golle_num_delete (golle_num_t n)

  Free a number.

- GOLLE_EXTERN golle_num_t golle_num_dup (const golle_num_t i)

  Make a copy of the input.
- GOLLE_EXTERN golle_error golle_num_cpy (golle_num_t dst, const golle_num_t src)

  Make a copy of the input.
- GOLLE_EXTERN golle_num_t golle_num_new_int (size_t i)

  Create a new number from a given native integer.
- GOLLE_EXTERN golle_error golle_num_generate_rand (golle_num_t r, const golle_num_t n)

  Generate a new random number in the range [0, n).
- GOLLE_EXTERN golle_num_t golle_num_rand (const golle_num_t n)

  Generate a new random number in the range [0, n).
- GOLLE_EXTERN golle_error golle_num_rand_bits (golle_num_t r, int bits)

  Generate a new random number of the given size.
- GOLLE_EXTERN int golle_num_cmp (const golle_num_t n1, const golle_num_t n2)

  Compare two numbers.
- GOLLE_EXTERN golle_num_t golle_generate_prime (int bits, int safe, golle_num_t div)

  Generate a pseudo-random `size`-bit prime number.
- GOLLE_EXTERN golle_error golle_test_prime (const golle_num_t p)

  Test a number for proabable primality.
- GOLLE_EXTERN golle_error golle_find_generator (golle_num_t g, const golle_num_t p, const golle_num_t q, int n)

  Find a generator for the multiplicative subgroup of $\mathbb{Z}_p^*$ of order q ($\mathbb{G}_q$).
- GOLLE_EXTERN golle_error golle_num_to_bin (const golle_num_t n, golle_bin_t *bin)

  Write the big-endian binary representation of a number into the given binary buffer. The buffer will be resized to the number of bytes required.
- GOLLE_EXTERN golle_error golle_bin_to_num (const golle_bin_t *bin, golle_num_t n)

  Convert a big-endian binary buffer into a number.
- GOLLE_EXTERN golle_error golle_num_mod_exp (golle_num_t out, const golle_num_t base, const golle_num_t exp, const golle_num_t mod)

  Calculate $m = g^n \mod q$.
- GOLLE_EXTERN golle_error golle_num_print (FILE *file, const golle_num_t num)

  Print a number, in big-endian hexadecimal, to the given file pointer.
- GOLLE_EXTERN golle_error golle_num_xor (golle_num_t out, const golle_num_t x1, const golle_num_t x2)

  XOR two numbers. Increases the size of a number if required.

11.10.1 Detailed Description

Describes various available number functions including primality functions, generator finding, and arithmetic of large numbers.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.11 include/golle/pep.h File Reference

PEP protocol.

```
#include "platform.h"
#include "distribute.h"
#include "schnorr.h"
#include "elgamal.h"
```

Functions

- GOLLE_EXTERN golle_error golle_pep_prover (const golle_key_t *egKey, const golle_num_t k, const golle_num_t z, golle_schnorr_t *key)

  Make a Schnorr public key, (G,Y) out of the ElGamal public key and store the private key.
- GOLLE_EXTERN golle_error golle_pep_verifier (const golle_key_t *egKey, const golle_num_t z, const golle_eg_t *e1, const golle_eg_t *e2, golle_schnorr_t *key)

  Make a Schnorr public key, (G,Y) from two ciphertexts. The private key (i.e. the reencryption factor) is not known. This function is used by the verifier to check if the two ciphertexts are the same.

11.11.1 Detailed Description

PEP protocol.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.12 include/golle/platform.h File Reference

Macros for platform-dependant behaviour.

Macros

- #define GOLLE_EXTERN extern
- #define GOLLE_INLINE static inline
- #define GOLLE_BEGIN_C
- #define GOLLE_END_C

11.12.1 Detailed Description

Macros for platform-dependant behaviour.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.12.2 Macro Definition Documentation

11.12.2.1 #define GOLLE_BEGIN_C

For C++ compilers, begins an `extern "C"` block.

11.12.2.2 #define GOLLE_END_C

For C++ compilers, ends and `extern "C"` block.

11.12.2.3 #define GOLLE_EXTERN extern

Platform-specific definition for shared library exports.

11.12.2.4 #define GOLLE_INLINE static inline

Platform-specific definition proper inlining.

11.13 include/golle/random.h File Reference

Wrapper functions for collecting random data.

```
#include "bin.h"
#include "errors.h"
#include "platform.h"
```

Functions

- GOLLE_EXTERN golle_error golle_random_seed (void)

  Seed the system's random generator.
- GOLLE_EXTERN golle_error golle_random_generate (golle_bin_t *buffer)

  Fill the buffer with random data.
- GOLLE_EXTERN golle_error golle_random_clear (void)

  Safely destroy the random state. This function should be called before application exit.

11.13.1 Detailed Description

Wrapper functions for collecting random data.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

11.14 include/golle/schnorr.h File Reference

Schnorr Identification.

```
#include "platform.h"
#include "numbers.h"
#include "errors.h"
#include "distribute.h"
```

Data Structures

- struct golle_schnorr_t

  A key used for the Schnorr Identification Algorithm.

Functions

- GOLLE_INLINE void golle_schnorr_clear (golle_schnorr_t *key)

  Clear all number values in a Schnorr key.
- GOLLE_EXTERN golle_error golle_schnorr_commit (const golle_schnorr_t *key, golle_num_t r, golle_num_t t)

  For an ElGamal public key, generate a random r and calculate t.
- GOLLE_EXTERN golle_error golle_schnorr_prove (const golle_schnorr_t *key, golle_num_t s, const golle_num_t r, const golle_num_t c)

  Calculate $s = r + cx$.
- GOLLE_EXTERN golle_error golle_schnorr_verify (const golle_schnorr_t *key, const golle_num_t s, const golle_num_t t, const golle_num_t c)

  Verify $g^s = ty^c$.

11.14.1 Detailed Description

Schnorr Identification.

Author

Anthony Arnold

Copyright

MIT License

Date

2014

Index

a
    golle_eg_t, 54
accept_commit
    golle_t, 57
accept_crypt
    golle_t, 57
accept_eg
    golle_t, 57
accept_rand
    golle_t, 57

b
    golle_eg_t, 54
bcast_commit
    golle_t, 57
bcast_crypt
    golle_t, 57
bcast_secret
    golle_t, 58
bin
    golle_bin_t, 51
Binary buffers, 17
    golle_bin_copy, 18
    golle_bin_delete, 18
    golle_bin_init, 18
    golle_bin_new, 18
    golle_bin_release, 19
    golle_bin_resize, 19
Bit Commitment, 20
    golle_commit_clear, 20
    golle_commit_copy, 20
    golle_commit_delete, 21
    golle_commit_new, 21
    golle_commit_verify, 21

c1
    golle_disj_t, 53
c2
    golle_disj_t, 53

Disjunctive Plaintext Equivalence Proof, 24
    golle_dispep_setup, 24
Disjunctive Schnorr Identification, 22
    golle_disj_clear, 22
    golle_disj_commit, 22
    golle_disj_prove, 22
    golle_disj_verify, 23

ElGamal, 28
    golle_eg_clear, 28
    golle_eg_decrypt, 28
    golle_eg_encrypt, 29
    golle_eg_reencrypt, 29
errors.h
    GOLLE_COMMIT_FAILED, 65
    GOLLE_COMMIT_PASSED, 65
    GOLLE_EABORT, 65
    GOLLE_ECOLLISION, 65
    GOLLE_ECRYPTO, 65
    GOLLE_EEMPTY, 65
    GOLLE_EEXISTS, 65
    GOLLE_EINVALID, 65
    GOLLE_EMEM, 65
    GOLLE_END, 65
    GOLLE_ENOCOMMIT, 65
    GOLLE_ENOTFOUND, 65
    GOLLE_ENOTPRIME, 65
    GOLLE_EOUTOFRANGE, 65
    GOLLE_ERROR, 65
    GOLLE_ETOOFEW, 65
    GOLLE_NOT_PRIME, 65
    GOLLE_OK, 65
    GOLLE_PROBABLY_PRIME, 65
errors.h
    GOLLE_ASSERT, 65
    GOLLE_UNUSED, 65
    golle_error, 65

G
    golle_schnorr_t, 56
g
    golle_key_t, 54
GOLLE_COMMIT_FAILED
    errors.h, 65
GOLLE_COMMIT_PASSED
    errors.h, 65
GOLLE_EABORT
    errors.h, 65
GOLLE_ECOLLISION
    errors.h, 65
GOLLE_ECRYPTO
    errors.h, 65
GOLLE_EEMPTY
    errors.h, 65
GOLLE_EEXISTS
    errors.h, 65
GOLLE_EINVALID
    errors.h, 65
GOLLE_EMEM
    errors.h, 65

GOLLE_END
errors.h, 65
GOLLE_ENOCOMMIT
errors.h, 65
GOLLE_ENOTFOUND
errors.h, 65
GOLLE_ENOTPRIME
errors.h, 65
GOLLE_EOUTOFRANGE
errors.h, 65
GOLLE_ERROR
errors.h, 65
GOLLE_ETOOFEW
errors.h, 65
GOLLE_NOT_PRIME
errors.h, 65
GOLLE_OK
errors.h, 65
GOLLE_PROBABLY_PRIME
errors.h, 65
GOLLE_ASSERT
errors.h, 65
GOLLE_BEGIN_C
platform.h, 71
GOLLE_END_C
platform.h, 71
GOLLE_EXTERN
platform.h, 71
GOLLE_FACE_UP
The Golle protocol interface, 32
GOLLE_INLINE
platform.h, 71
GOLLE_UNUSED
errors.h, 65
golle_bin_copy
Binary buffers, 18
golle_bin_delete
Binary buffers, 18
golle_bin_init
Binary buffers, 18
golle_bin_new
Binary buffers, 18
golle_bin_release
Binary buffers, 19
golle_bin_resize
Binary buffers, 19
golle_bin_t, 51
bin, 51
size, 51
golle_bin_to_num
Large Numbers, 41
golle_check_selection
The Golle protocol interface, 32
golle_clear
The Golle protocol interface, 32
golle_commit_clear
Bit Commitment, 20
golle_commit_copy
Bit Commitment, 20
golle_commit_delete
Bit Commitment, 21
golle_commit_new
Bit Commitment, 21
golle_commit_t, 51
hash, 52
rkeep, 52
rsend, 52
secret, 52
golle_commit_verify
Bit Commitment, 21
golle_disj_clear
Disjunctive Schnorr Identification, 22
golle_disj_commit
Disjunctive Schnorr Identification, 22
golle_disj_prove
Disjunctive Schnorr Identification, 22
golle_disj_t, 52
c1, 53
c2, 53
r1, 53
s1, 53
s2, 53
t1, 53
t2, 53
golle_disj_verify
Disjunctive Schnorr Identification, 23
golle_dispep_setup
Disjunctive Plaintext Equivalence Proof, 24
golle_eg_clear
ElGamal, 28
golle_eg_decrypt
ElGamal, 28
golle_eg_encrypt
ElGamal, 29
golle_eg_reencrypt
ElGamal, 29
golle_eg_t, 53
a, 54
b, 54
golle_error
errors.h, 65
golle_find_generator
Large Numbers, 41
golle_generate
The Golle protocol interface, 33
golle_generate_prime
Large Numbers, 41
golle_initialise
The Golle protocol interface, 33
golle_key_accum_h
Key Generation and Distribution, 25
golle_key_cleanup
Key Generation and Distribution, 26
golle_key_gen_private
Key Generation and Distribution, 26
golle_key_gen_public

Key Generation and Distribution, 26
golle_key_set_public
Key Generation and Distribution, 27
golle_key_t, 54
g, 54
h, 54
h_product, 54
p, 54
q, 54
x, 55
golle_list_delete
Singly-linked lists, 36
golle_list_erase_at
Singly-linked lists, 36
golle_list_insert_at
Singly-linked lists, 36
golle_list_iterator
Singly-linked lists, 36
golle_list_iterator_free
Singly-linked lists, 37
golle_list_iterator_next
Singly-linked lists, 37
golle_list_iterator_reset
Singly-linked lists, 37
golle_list_iterator_t, 55
golle_list_new
Singly-linked lists, 37
golle_list_pop
Singly-linked lists, 38
golle_list_pop_all
Singly-linked lists, 38
golle_list_pop_many
Singly-linked lists, 38
golle_list_push
Singly-linked lists, 38
golle_list_push_many
Singly-linked lists, 39
golle_list_size
Singly-linked lists, 39
golle_list_t, 55
golle_list_top
Singly-linked lists, 39
golle_num_cmp
Large Numbers, 42
golle_num_cpy
Large Numbers, 42
golle_num_dup
Large Numbers, 42
golle_num_generate_rand
Large Numbers, 42
golle_num_mod_exp
Large Numbers, 43
golle_num_new
Large Numbers, 43
golle_num_new_int
Large Numbers, 43
golle_num_print
Large Numbers, 43
golle_num_rand
Large Numbers, 44
golle_num_rand_bits
Large Numbers, 44
golle_num_to_bin
Large Numbers, 44
golle_num_xor
Large Numbers, 44
golle_pep_prover
Plaintext Equivalence Proof, 46
golle_pep_verifier
Plaintext Equivalence Proof, 46
golle_random_clear
Random Data, 48
golle_random_generate
Random Data, 48
golle_random_seed
Random Data, 48
golle_reduce_selection
The Golle protocol interface, 33
golle_reveal_selection
The Golle protocol interface, 34
golle_schnorr_clear
Schnorr Identification Algorithm, 49
golle_schnorr_commit
Schnorr Identification Algorithm, 49
golle_schnorr_prove
Schnorr Identification Algorithm, 49
golle_schnorr_t, 55
G, 56
p, 56
q, 56
x, 56
Y, 56
golle_schnorr_verify
Schnorr Identification Algorithm, 50
golle_t, 56
accept_commit, 57
accept_crypt, 57
accept_eg, 57
accept_rand, 57
bcast_commit, 57
bcast_crypt, 57
bcast_secret, 58
key, 58
num_items, 58
num_peers, 58
reserved, 58
reveal_rand, 58
golle_test_prime
Large Numbers, 45

h
golle_key_t, 54
h_product
golle_key_t, 54
hash
golle_commit_t, 52

include/golle/bin.h, 59
include/golle/commit.h, 60
include/golle/disj.h, 61
include/golle/dispep.h, 62
include/golle/distribute.h, 62
include/golle/elgamal.h, 63
include/golle/errors.h, 64
include/golle/golle.h, 65
include/golle/list.h, 67
include/golle/numbers.h, 68
include/golle/pep.h, 70
include/golle/platform.h, 70
include/golle/random.h, 71
include/golle/schnorr.h, 72

key
    golle_t, 58
Key Generation and Distribution, 25
    golle_key_accum_h, 25
    golle_key_cleanup, 26
    golle_key_gen_private, 26
    golle_key_gen_public, 26
    golle_key_set_public, 27

Large Numbers, 40
    golle_bin_to_num, 41
    golle_find_generator, 41
    golle_generate_prime, 41
    golle_num_cmp, 42
    golle_num_cpy, 42
    golle_num_dup, 42
    golle_num_generate_rand, 42
    golle_num_mod_exp, 43
    golle_num_new, 43
    golle_num_new_int, 43
    golle_num_print, 43
    golle_num_rand, 44
    golle_num_rand_bits, 44
    golle_num_to_bin, 44
    golle_num_xor, 44
    golle_test_prime, 45

num_items
    golle_t, 58
num_peers
    golle_t, 58

p
    golle_key_t, 54
    golle_schnorr_t, 56
Plaintext Equivalence Proof, 46
    golle_pep_prover, 46
    golle_pep_verifier, 46
platform.h
    GOLLE_BEGIN_C, 71
    GOLLE_END_C, 71
    GOLLE_EXTERN, 71
    GOLLE_INLINE, 71

q
    golle_key_t, 54
    golle_schnorr_t, 56

r1
    golle_disj_t, 53
Random Data, 48
    golle_random_clear, 48
    golle_random_generate, 48
    golle_random_seed, 48
reserved
    golle_t, 58
reveal_rand
    golle_t, 58
rkeep
    golle_commit_t, 52
rsend
    golle_commit_t, 52

s1
    golle_disj_t, 53
s2
    golle_disj_t, 53
Schnorr Identification Algorithm, 49
    golle_schnorr_clear, 49
    golle_schnorr_commit, 49
    golle_schnorr_prove, 49
    golle_schnorr_verify, 50
secret
    golle_commit_t, 52
Singly-linked lists, 35
    golle_list_delete, 36
    golle_list_erase_at, 36
    golle_list_insert_at, 36
    golle_list_iterator, 36
    golle_list_iterator_free, 37
    golle_list_iterator_next, 37
    golle_list_iterator_reset, 37
    golle_list_new, 37
    golle_list_pop, 38
    golle_list_pop_all, 38
    golle_list_pop_many, 38
    golle_list_push, 38
    golle_list_push_many, 39
    golle_list_size, 39
    golle_list_top, 39
size
    golle_bin_t, 51

t1
    golle_disj_t, 53
t2
    golle_disj_t, 53
The Golle protocol interface, 31
    GOLLE_FACE_UP, 32
    golle_check_selection, 32
    golle_clear, 32
    golle_generate, 33
    golle_initialise, 33
    golle_reduce_selection, 33

golle_reveal_selection, 34

x
golle_key_t, 55
golle_schnorr_t, 56

Y
golle_schnorr_t, 56