LibTomCrypt Reference Manual

1.15

Generated by Doxygen 1.4.6

Tue Nov 21 12:28:35 2006

Contents

Chapter 1

LibTomCrypt Hierarchical Index

1.1 LibTomCrypt Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

LibTomCrypt Data Structure Index

2.1 LibTomCrypt Data Structures

Here are the data structures with brief descriptions:

Chapter 3

LibTomCrypt File Index

3.1 LibTomCrypt File List

Here is a list of all files with brief descriptions:

Chapter 4

LibTomCrypt Data Structure Documentation

4.1 edge Struct Reference

4.1.1 Detailed Description

Definition at line 20 of file der_encode_setof.c.

Data Fields

- unsigned char ∗ start
- unsigned long size

4.1.2 Field Documentation

4.1.2.1 unsigned long edge::size

Definition at line 22 of file der_encode_setof.c.

Referenced by dsa_decrypt_key(), ecc_ansi_x963_import(), ecc_decrypt_key(), ecc_make_key(), ecc_-sizes(), ecc_test(), pkcs_1_i2osp(), qsort_helper(), and rsa_verify_hash_ex().

4.1.2.2 unsigned char∗ edge::start

Definition at line 21 of file der_encode_setof.c.

Referenced by qsort_helper().

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

- pk/asn1/der/set/der_encode_setof.c

4.2 Hash_state Union Reference

`#include <tomcrypt_hash.h>`

4.2.1 Detailed Description

Definition at line 105 of file tomcrypt_hash.h.

Data Fields

- char dummy [1]
- void * data

4.2.2 Field Documentation

4.2.2.1 void* Hash_state::data

Definition at line 146 of file tomcrypt_hash.h.

4.2.2.2 char Hash_state::dummy[1]

Definition at line 106 of file tomcrypt_hash.h.

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

- headers/tomcrypt_hash.h

4.3 ltc_cipher_descriptor Struct Reference

`#include <tomcrypt_cipher.h>`

4.3.1 Detailed Description

cipher descriptor table, last entry has "name == NULL" to mark the end of table

Definition at line 318 of file tomcrypt_cipher.h.

Data Fields

- char ∗ name
  name of cipher
- unsigned char ID
  internal ID
- int min_key_length
  min keysize (octets)
- int max_key_length
  max keysize (octets)
- int block_length
  block size (octets)
- int default_rounds
  default number of rounds
- int(∗ setup)(const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)
  Setup the cipher.
- int(∗ ecb_encrypt)(const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)
  Encrypt a block.
- int(∗ ecb_decrypt)(const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)
  Decrypt a block.
- int(∗ test)(void)
  Test the block cipher.
- void(∗ done)(symmetric_key ∗skey)
  Terminate the context.
- int(∗ keysize)(int ∗keysize)
  Determine a key size.
- int(∗ accel_ecb_encrypt)(const unsigned char ∗pt, unsigned char ∗ct, unsigned long blocks, symmetric_key ∗skey)

Accelerated ECB encryption.

- int(∗ accel_ecb_decrypt)(const unsigned char ∗ct, unsigned char ∗pt, unsigned long blocks, symmetric_key ∗skey)

  Accelerated ECB decryption.

- int(∗ accel_cbc_encrypt)(const unsigned char ∗pt, unsigned char ∗ct, unsigned long blocks, unsigned char ∗IV, symmetric_key ∗skey)

  Accelerated CBC encryption.

- int(∗ accel_cbc_decrypt)(const unsigned char ∗ct, unsigned char ∗pt, unsigned long blocks, unsigned char ∗IV, symmetric_key ∗skey)

  Accelerated CBC decryption.

- int(∗ accel_ctr_encrypt)(const unsigned char ∗pt, unsigned char ∗ct, unsigned long blocks, unsigned char ∗IV, int mode, symmetric_key ∗skey)

  Accelerated CTR encryption.

- int(∗ accel_lrw_encrypt)(const unsigned char ∗pt, unsigned char ∗ct, unsigned long blocks, unsigned char ∗IV, const unsigned char ∗tweak, symmetric_key ∗skey)

  Accelerated LRW.

- int(∗ accel_lrw_decrypt)(const unsigned char ∗ct, unsigned char ∗pt, unsigned long blocks, unsigned char ∗IV, const unsigned char ∗tweak, symmetric_key ∗skey)

  Accelerated LRW.

- int(∗ accel_ccm_memory)(const unsigned char ∗key, unsigned long keylen, symmetric_key ∗uskey, const unsigned char ∗nonce, unsigned long noncelen, const unsigned char ∗header, unsigned long headerlen, unsigned char ∗pt, unsigned long ptlen, unsigned char ∗ct, unsigned char ∗tag, unsigned long ∗taglen, int direction)

  Accelerated CCM packet (one-shot).

- int(∗ accel_gcm_memory)(const unsigned char ∗key, unsigned long keylen, const unsigned char ∗IV, unsigned long IVlen, const unsigned char ∗adata, unsigned long adatalen, unsigned char ∗pt, unsigned long ptlen, unsigned char ∗ct, unsigned char ∗tag, unsigned long ∗taglen, int direction)

  Accelerated GCM packet (one shot).

- int(∗ omac_memory)(const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Accelerated one shot OMAC.

- int(∗ xcbc_memory)(const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Accelerated one shot XCBC.

- int(∗ f9_memory)(const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Accelerated one shot F9.

4.3.2 Field Documentation

4.3.2.1 int(∗ ltc_cipher_descriptor::accel_cbc_decrypt)(const unsigned char ∗ct, unsigned char ∗pt, unsigned long blocks, unsigned char ∗IV, symmetric_key ∗skey)

Accelerated CBC decryption.

Parameters:

pt Plaintext

ct Ciphertext

blocks The number of complete blocks to process

IV The initial value (input/output)

skey The scheduled key context

Returns:

CRYPT_OK if successful

Referenced by cbc_decrypt().

4.3.2.2 int(∗ ltc_cipher_descriptor::accel_cbc_encrypt)(const unsigned char ∗pt, unsigned char ∗ct, unsigned long blocks, unsigned char ∗IV, symmetric_key ∗skey)

Accelerated CBC encryption.

Parameters:

pt Plaintext

ct Ciphertext

blocks The number of complete blocks to process

IV The initial value (input/output)

skey The scheduled key context

Returns:

CRYPT_OK if successful

Referenced by cbc_encrypt().

4.3.2.3 int(∗ ltc_cipher_descriptor::accel_ccm_memory)(const unsigned char ∗key, unsigned long keylen, symmetric_key ∗uskey, const unsigned char ∗nonce, unsigned long noncelen, const unsigned char ∗header, unsigned long headerlen, unsigned char ∗pt, unsigned long ptlen, unsigned char ∗ct, unsigned char ∗tag, unsigned long ∗taglen, int direction)

Accelerated CCM packet (one-shot).

Parameters:

key The secret key to use

keylen The length of the secret key (octets)

uskey A previously scheduled key [optional can be NULL]

nonce The session nonce [use once]

noncelen The length of the nonce

header The header for the session

headerlen The length of the header (octets)

pt [out] The plaintext

ptlen The length of the plaintext (octets)

ct [out] The ciphertext

tag [out] The destination tag

taglen [in/out] The max size and resulting size of the authentication tag

direction Encrypt or Decrypt direction (0 or 1)

Returns:
CRYPT_OK if successful

Referenced by ccm_memory().

4.3.2.4 int(∗ ltc_cipher_descriptor::accel_ctr_encrypt)(const unsigned char ∗pt, unsigned char ∗ct, unsigned long blocks, unsigned char ∗IV, int mode, symmetric_key ∗skey)

Accelerated CTR encryption.

Parameters:

pt Plaintext

ct Ciphertext

blocks The number of complete blocks to process

IV The initial value (input/output)

mode little or big endian counter (mode=0 or mode=1)

skey The scheduled key context

Returns:
CRYPT_OK if successful

Referenced by ctr_encrypt().

4.3.2.5 int(∗ ltc_cipher_descriptor::accel_ecb_decrypt)(const unsigned char ∗ct, unsigned char ∗pt, unsigned long blocks, symmetric_key ∗skey)

Accelerated ECB decryption.

Parameters:

pt Plaintext

ct Ciphertext

blocks The number of complete blocks to process

skey The scheduled key context

Returns:
CRYPT_OK if successful

Referenced by ecb_decrypt().

4.3.2.6 int(* ltc_cipher_descriptor::accel_ecb_encrypt)(const unsigned char *pt, unsigned char *ct, unsigned long blocks, symmetric_key *skey)

Accelerated ECB encryption.

Parameters:

pt Plaintext

ct Ciphertext

blocks The number of complete blocks to process

skey The scheduled key context

Returns:

CRYPT_OK if successful

Referenced by ecb_encrypt().

4.3.2.7 int(* ltc_cipher_descriptor::accel_gcm_memory)(const unsigned char *key, unsigned long keylen, const unsigned char *IV, unsigned long IVlen, const unsigned char *adata, unsigned long adatalen, unsigned char *pt, unsigned long ptlen, unsigned char *ct, unsigned char *tag, unsigned long *taglen, int direction)

Accelerated GCM packet (one shot).

Parameters:

key The secret key

keylen The length of the secret key

IV The initial vector

IVlen The length of the initial vector

adata The additional authentication data (header)

adatalen The length of the adata

pt The plaintext

ptlen The length of the plaintext (ciphertext length is the same)

ct The ciphertext

tag [out] The MAC tag

taglen [in/out] The MAC tag length

direction Encrypt or Decrypt mode (GCM_ENCRYPT or GCM_DECRYPT)

Returns:

CRYPT_OK on success

Referenced by gcm_memory().

4.3.2.8 int(* ltc_cipher_descriptor::accel_lrw_decrypt)(const unsigned char *ct, unsigned char *pt, unsigned long blocks, unsigned char *IV, const unsigned char *tweak, symmetric_key *skey)

Accelerated LRW.

Parameters:

- ***ct*** Ciphertext
- ***pt*** Plaintext
- ***blocks*** The number of complete blocks to process
- ***IV*** The initial value (input/output)
- ***tweak*** The LRW tweak
- ***skey*** The scheduled key context

Returns:

CRYPT_OK if successful

Referenced by lrw_decrypt(), and lrw_setiv().

4.3.2.9 int(∗ ltc_cipher_descriptor::accel_lrw_encrypt)(const unsigned char ∗pt, unsigned char ∗ct, unsigned long blocks, unsigned char ∗IV, const unsigned char ∗tweak, symmetric_key ∗skey)

Accelerated LRW.

Parameters:

- ***pt*** Plaintext
- ***ct*** Ciphertext
- ***blocks*** The number of complete blocks to process
- ***IV*** The initial value (input/output)
- ***tweak*** The LRW tweak
- ***skey*** The scheduled key context

Returns:

CRYPT_OK if successful

Referenced by lrw_encrypt(), and lrw_setiv().

4.3.2.10 int ltc_cipher_descriptor::block_length

block size (octets)

Definition at line 324 of file tomcrypt_cipher.h.

Referenced by cbc_start(), ccm_memory(), cfb_start(), chc_register(), ctr_start(), eax_init(), ecb_decrypt(), ecb_encrypt(), ecb_start(), f8_start(), f9_done(), f9_process(), ocb_decrypt(), ocb_encrypt(), ocb_init(), ofb_start(), omac_init(), pmac_init(), s_ocb_done(), xcbc_done(), xcbc_init(), and xcbc_process().

4.3.2.11 int ltc_cipher_descriptor::default_rounds

default number of rounds

Definition at line 324 of file tomcrypt_cipher.h.

Referenced by rc5_setup().

4.3.2.12 void(* ltc_cipher_descriptor::done)(symmetric_key *skey)

Terminate the context.

Parameters:

skey The scheduled key

Referenced by cbc_done(), ccm_test(), cfb_done(), ctr_done(), ecb_done(), f8_done(), lrw_done(), ofb_done(), and omac_done().

4.3.2.13 int(* ltc_cipher_descriptor::ecb_decrypt)(const unsigned char *ct, unsigned char *pt, symmetric_key *skey)

Decrypt a block.

Parameters:

ct The ciphertext

pt [out] The plaintext

skey The scheduled key

Returns:

CRYPT_OK if successful

Referenced by cbc_decrypt(), ecb_decrypt(), and ocb_decrypt().

4.3.2.14 int(* ltc_cipher_descriptor::ecb_encrypt)(const unsigned char *pt, unsigned char *ct, symmetric_key *skey)

Encrypt a block.

Parameters:

pt The plaintext

ct [out] The ciphertext

skey The scheduled key

Returns:

CRYPT_OK if successful

Referenced by cbc_encrypt(), cfb_decrypt(), cfb_encrypt(), cfb_setiv(), chc_compress(), chc_init(), ctr_encrypt(), ctr_setiv(), ecb_encrypt(), f8_encrypt(), f8_setiv(), f9_done(), f9_process(), ofb_encrypt(), ofb_setiv(), omac_done(), omac_process(), pmac_process(), and xcbc_process().

4.3.2.15 int(* ltc_cipher_descriptor::f9_memory)(const unsigned char *key, unsigned long keylen, const unsigned char *in, unsigned long inlen, unsigned char *out, unsigned long *outlen)

Accelerated one shot F9.

Parameters:

key The secret key

keylen The key length (octets)

in The message

inlen Length of message (octets)

out [out] Destination for tag

outlen [in/out] Initial and final size of out

Returns:

CRYPT_OK on success

Remarks:

Requires manual padding

Referenced by f9_memory().

4.3.2.16 unsigned char ltc_cipher_descriptor::ID

internal ID

Definition at line 322 of file tomcrypt_cipher.h.

Referenced by register_cipher(), and unregister_cipher().

4.3.2.17 int(∗ ltc_cipher_descriptor::keysize)(int ∗keysize)

Determine a key size.

Parameters:

keysize [in/out] The size of the key desired and the suggested size

Returns:

CRYPT_OK if successful

4.3.2.18 int ltc_cipher_descriptor::max_key_length

max keysize (octets)

Definition at line 324 of file tomcrypt_cipher.h.

4.3.2.19 int ltc_cipher_descriptor::min_key_length

min keysize (octets)

Definition at line 324 of file tomcrypt_cipher.h.

4.3.2.20 char∗ ltc_cipher_descriptor::name

name of cipher

Definition at line 320 of file tomcrypt_cipher.h.

Referenced by cipher_is_valid(), find_cipher_id(), find_hash_id(), and find_hash_oid().

4.3.2.21 int(∗ ltc_cipher_descriptor::omac_memory)(const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

Accelerated one shot OMAC.

Parameters:

key The secret key

keylen The key length (octets)

in The message

inlen Length of message (octets)

out [out] Destination for tag

outlen [in/out] Initial and final size of out

Returns:

CRYPT_OK on success

Referenced by omac_memory().

4.3.2.22 int(∗ ltc_cipher_descriptor::setup)(const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

Setup the cipher.

Parameters:

key The input symmetric key

keylen The length of the input key (octets)

num_rounds The requested number of rounds (0==default)

skey [out] The destination of the scheduled key

Returns:

CRYPT_OK if successful

Referenced by ccm_memory(), and ecb_start().

4.3.2.23 int(∗ ltc_cipher_descriptor::test)(void)

Test the block cipher.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-testing has been disabled

4.3.2.24 int(∗ ltc_cipher_descriptor::xcbc_memory)(const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

Accelerated one shot XCBC.

Parameters:

key The secret key

keylen The key length (octets)

in The message

inlen Length of message (octets)

out [out] Destination for tag

outlen [in/out] Initial and final size of out

Returns:

CRYPT_OK on success

Referenced by xcbc_memory().

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

- headers/tomcrypt_cipher.h

4.4 ltc_hash_descriptor Struct Reference

```
#include <tomcrypt_hash.h>
```

4.4.1 Detailed Description

hash descriptor

Definition at line 150 of file tomcrypt_hash.h.

Data Fields

- char ∗ name

  name of hash

- unsigned char ID

  internal ID

- unsigned long hashsize

  Size of digest in octets.

- unsigned long blocksize

  Input block size in octets.

- unsigned long OID [16]

  ASN.1 OID.

- unsigned long OIDlen

  Length of DER encoding.

- int(∗ init)(hash_state ∗hash)

  Init a hash state.

- int(∗ process)(hash_state ∗hash, const unsigned char ∗in, unsigned long inlen)

  Process a block of data.

- int(∗ done)(hash_state ∗hash, unsigned char ∗out)

  Produce the digest and store it.

- int(∗ test)(void)

  Self-test.

- int(∗ hmac_block)(const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

4.4.2 Field Documentation

4.4.2.1 unsigned long ltc_hash_descriptor::blocksize

Input block size in octets.

Definition at line 158 of file tomcrypt_hash.h.

Referenced by chc_register().

4.4.2.2 int(∗ ltc_hash_descriptor::done)(hash_state ∗hash, unsigned char ∗out)

Produce the digest and store it.

Parameters:

hash The hash state

out [out] The destination of the digest

Returns:

CRYPT_OK if successful

Referenced by hash_filehandle(), and hash_memory().

4.4.2.3 unsigned long ltc_hash_descriptor::hashsize

Size of digest in octets.

Definition at line 156 of file tomcrypt_hash.h.

Referenced by chc_register(), hash_filehandle(), hash_memory(), hash_memory_multi(), hmac_done(), hmac_init(), pkcs_1_mgf1(), pkcs_1_oaep_decode(), pkcs_1_oaep_encode(), pkcs_1_pss_decode(), pkcs_1_pss_encode(), and pkcs_5_alg1().

4.4.2.4 int(∗ ltc_hash_descriptor::hmac_block)(const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

Referenced by hmac_memory().

4.4.2.5 unsigned char ltc_hash_descriptor::ID

internal ID

Definition at line 154 of file tomcrypt_hash.h.

4.4.2.6 int(∗ ltc_hash_descriptor::init)(hash_state ∗hash)

Init a hash state.

Parameters:

hash The hash to initialize

Returns:

CRYPT_OK if successful

4.4.2.7 char* ltc_hash_descriptor::name

name of hash

Definition at line 152 of file tomcrypt_hash.h.

Referenced by hash_is_valid().

4.4.2.8 unsigned long ltc_hash_descriptor::OID[16]

ASN.1 OID.

Definition at line 160 of file tomcrypt_hash.h.

4.4.2.9 unsigned long ltc_hash_descriptor::OIDlen

Length of DER encoding.

Definition at line 162 of file tomcrypt_hash.h.

4.4.2.10 int(* ltc_hash_descriptor::process)(hash_state *hash, const unsigned char *in, unsigned long inlen)

Process a block of data.

Parameters:

hash The hash state

in The data to hash

inlen The length of the data (octets)

Returns:

CRYPT_OK if successful

Referenced by hmac_process().

4.4.2.11 int(* ltc_hash_descriptor::test)(void)

Self-test.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

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

- headers/tomcrypt_hash.h

4.5 ltc_math_descriptor Struct Reference

`#include <tomcrypt_math.h>`

4.5.1 Detailed Description

math descriptor

Definition at line 19 of file tomcrypt_math.h.

Data Fields

- char ∗ name

  Name of the math provider.

- int bits_per_digit

  Bits per digit, amount of bits must fit in an unsigned long.

- int(∗ init)(void ∗∗a)

  initialize a bignum

- int(∗ init_copy)(void ∗∗dst, void ∗src)

  init copy

- void(∗ deinit)(void ∗a)

  deinit

- int(∗ neg)(void ∗src, void ∗dst)

  negate

- int(∗ copy)(void ∗src, void ∗dst)

  copy

- int(∗ set_int)(void ∗a, unsigned long n)

  set small constant

- unsigned long(∗ get_int)(void ∗a)

  get small constant

- unsigned long(∗ get_digit)(void ∗a, int n)

  get digit n

- int(∗ get_digit_count)(void ∗a)

  Get the number of digits that represent the number.

- int(∗ compare)(void ∗a, void ∗b)

  compare two integers

- int(∗ compare_d)(void ∗a, unsigned long n) *compare against int*

- int(∗ count_bits)(void ∗a)

  Count the number of bits used to represent the integer.

- int(∗ count_lsb_bits)(void ∗a)

  Count the number of LSB bits which are zero.

- int(∗ twoexpt)(void ∗a, int n)

  Compute a power of two.

- int(∗ read_radix)(void ∗a, const char ∗str, int radix)

  read ascii string

- int(∗ write_radix)(void ∗a, char ∗str, int radix)

  write number to string

- unsigned long(∗ unsigned_size)(void ∗a)

  get size as unsigned char string

- int(∗ unsigned_write)(void ∗src, unsigned char ∗dst)

  store an integer as an array of octets

- int(∗ unsigned_read)(void ∗dst, unsigned char ∗src, unsigned long len)

  read an array of octets and store as integer

- int(∗ add)(void ∗a, void ∗b, void ∗c)

  add two integers

- int(∗ addi)(void ∗a, unsigned long b, void ∗c)

  add two integers

- int(∗ sub)(void ∗a, void ∗b, void ∗c)

  subtract two integers

- int(∗ subi)(void ∗a, unsigned long b, void ∗c)

  subtract two integers

- int(∗ mul)(void ∗a, void ∗b, void ∗c)

  multiply two integers

- int(∗ muli)(void ∗a, unsigned long b, void ∗c)

  multiply two integers

- int(∗ sqr)(void ∗a, void ∗b)

  Square an integer.

- int(∗ mpdiv)(void ∗a, void ∗b, void ∗c, void ∗d)

  Divide an integer.

- int(∗ div_2)(void ∗a, void ∗b)

  divide by two

- int(∗ modi)(void ∗a, unsigned long b, unsigned long ∗c)

  Get remainder (small value).

- int(∗ gcd)(void ∗a, void ∗b, void ∗c)

  gcd

- int(∗ lcm)(void ∗a, void ∗b, void ∗c)

  lcm

- int(∗ mulmod)(void ∗a, void ∗b, void ∗c, void ∗d)

  Modular multiplication.

- int(∗ sqrmod)(void ∗a, void ∗b, void ∗c)

  Modular squaring.

- int(∗ invmod)(void ∗, void ∗, void ∗)

  Modular inversion.

- int(∗ montgomery_setup)(void ∗a, void ∗∗b)

  setup montgomery

- int(∗ montgomery_normalization)(void ∗a, void ∗b)

  get normalization value

- int(∗ montgomery_reduce)(void ∗a, void ∗b, void ∗c)

  reduce a number

- void(∗ montgomery_deinit)(void ∗a)

  clean up (frees memory)

- int(∗ exptmod)(void ∗a, void ∗b, void ∗c, void ∗d)

  Modular exponentiation.

- int(∗ isprime)(void ∗a, int ∗b)

  Primality testing.

- int(∗ ecc_ptmul)(void ∗k, ecc_point ∗G, ecc_point ∗R, void ∗modulus, int map)

  ECC GF(p) point multiplication (from the NIST curves).

- int(∗ ecc_ptadd)(ecc_point ∗P, ecc_point ∗Q, ecc_point ∗R, void ∗modulus, void ∗mp)

  ECC GF(p) point addition.

- int(∗ ecc_ptdbl)(ecc_point ∗P, ecc_point ∗R, void ∗modulus, void ∗mp)

  ECC GF(p) point double.

- int(∗ ecc_map)(ecc_point ∗P, void ∗modulus, void ∗mp)

  ECC mapping from projective to affine, currently uses (x,y,z) => (x/z^2, y/z^3, 1).

- int(∗ rsa_keygen)(prng_state ∗prng, int wprng, int size, long e, rsa_key ∗key)

    RSA Key Generation.

- int(∗ rsa_me)(const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, int which, rsa_key ∗key)

    RSA exponentiation.

4.5.2 Field Documentation

4.5.2.1 int(∗ ltc_math_descriptor::add)(void ∗a, void ∗b, void ∗c)

add two integers

Parameters:

a The first source integer

b The second source integer

c The destination of "a + b"

Returns:

CRYPT_OK on success

4.5.2.2 int(∗ ltc_math_descriptor::addi)(void ∗a, unsigned long b, void ∗c)

add two integers

Parameters:

a The first source integer

b The second source integer (single digit of upto bits_per_digit in length)

c The destination of "a + b"

Returns:

CRYPT_OK on success

4.5.2.3 int ltc_math_descriptor::bits_per_digit

Bits per digit, amount of bits must fit in an unsigned long.

Definition at line 24 of file tomcrypt_math.h.

4.5.2.4 int(∗ ltc_math_descriptor::compare)(void ∗a, void ∗b)

compare two integers

Parameters:

a The left side integer

b The right side integer

Returns:

LTC_MP_LT if a < b, LTC_MP_GT if a > b and LTC_MP_EQ otherwise. (signed comparison)

4.5.2.5 int(* ltc_math_descriptor::compare_d)(void *a, unsigned long n)

compare against int

Parameters:

a The left side integer

b The right side integer (upto bits_per_digit)

Returns:

LTC_MP_LT if a < b, LTC_MP_GT if a > b and LTC_MP_EQ otherwise. (signed comparison)

4.5.2.6 int(* ltc_math_descriptor::copy)(void *src, void *dst)

copy

Parameters:

src The number to copy from

dst The number to write to

Returns:

CRYPT_OK on success

4.5.2.7 int(* ltc_math_descriptor::count_bits)(void *a)

Count the number of bits used to represent the integer.

Parameters:

a The integer to count

Returns:

The number of bits required to represent the integer

4.5.2.8 int(* ltc_math_descriptor::count_lsb_bits)(void *a)

Count the number of LSB bits which are zero.

Parameters:

a The integer to count

Returns:

The number of contiguous zero LSB bits

4.5.2.9 void(* ltc_math_descriptor::deinit)(void *a)

deinit

Parameters:

a The number to free

Returns:

CRYPT_OK on success

4.5.2.10 int(* ltc_math_descriptor::div_2)(void *a, void *b)

divide by two

Parameters:

a The integer to divide (shift right)

b The destination

Returns:

CRYPT_OK on success

4.5.2.11 int(* ltc_math_descriptor::ecc_map)(ecc_point *P, void *modulus, void *mp)

ECC mapping from projective to affine, currently uses (x,y,z) => (x/z^2, y/z^3, 1).

Parameters:

P The point to map

modulus The modulus

mp The "b" value from montgomery_setup()

Returns:

CRYPT_OK on success

Remarks:

The mapping can be different but keep in mind a ecc_point only has three integers (x,y,z) so if you use a different mapping you have to make it fit.

4.5.2.12 int(* ltc_math_descriptor::ecc_ptadd)(ecc_point *P, ecc_point *Q, ecc_point *R, void *modulus, void *mp)

ECC GF(p) point addition.

Parameters:

P The first point

Q The second point

R The destination of P + Q

modulus The modulus

mp The "b" value from montgomery_setup()

Returns:

CRYPT_OK on success

4.5.2.13 int(∗ ltc_math_descriptor::ecc_ptdbl)(ecc_point ∗P, ecc_point ∗R, void ∗modulus, void ∗mp)

ECC GF(p) point double.

Parameters:

P The first point

R The destination of 2P

modulus The modulus

mp The "b" value from montgomery_setup()

Returns:

CRYPT_OK on success

4.5.2.14 int(∗ ltc_math_descriptor::ecc_ptmul)(void ∗k, ecc_point ∗G, ecc_point ∗R, void ∗modulus, int map)

ECC GF(p) point multiplication (from the NIST curves).

Parameters:

k The integer to multiply the point by

G The point to multiply

R The destination for kG

modulus The modulus for the field

map Boolean indicated whether to map back to affine or not (can be ignored if you work in affine only)

Returns:

CRYPT_OK on success

Referenced by ecc_shared_secret(), and ecc_verify_hash().

4.5.2.15 int(∗ ltc_math_descriptor::exptmod)(void ∗a, void ∗b, void ∗c, void ∗d)

Modular exponentiation.

Parameters:

a The base integer

b The power (can be negative) integer

c The modulus integer

d The destination

Returns:

CRYPT_OK on success

4.5.2.16 int(* ltc_math_descriptor::gcd)(void *a, void *b, void *c)

gcd

Parameters:

a The first integer

b The second integer

c The destination for (a, b)

Returns:

CRYPT_OK on success

4.5.2.17 unsigned long(* ltc_math_descriptor::get_digit)(void *a, int n)

get digit n

Parameters:

a The number to read from

n The number of the digit to fetch

Returns:

The bits_per_digit sized n'th digit of a

4.5.2.18 int(* ltc_math_descriptor::get_digit_count)(void *a)

Get the number of digits that represent the number.

Parameters:

a The number to count

Returns:

The number of digits used to represent the number

4.5.2.19 unsigned long(* ltc_math_descriptor::get_int)(void *a)

get small constant

Parameters:

a Number to read, only fetches upto bits_per_digit from the number

Returns:

The lower bits_per_digit of the integer (unsigned)

4.5.2.20 int(* ltc_math_descriptor::init)(void **a)

initialize a bignum

Parameters:

a The number to initialize

Returns:

CRYPT_OK on success

4.5.2.21 int(* ltc_math_descriptor::init_copy)(void **dst, void *src)

init copy

Parameters:

dst The number to initialize and write to

src The number to copy from

Returns:

CRYPT_OK on success

4.5.2.22 int(* ltc_math_descriptor::invmod)(void *, void *, void *)

Modular inversion.

Parameters:

a The value to invert

b The modulus

c The destination (1/a mod b)

Returns:

CRYPT_OK on success

4.5.2.23 int(* ltc_math_descriptor::isprime)(void *a, int *b)

Primality testing.

Parameters:

a The integer to test

b The destination of the result (FP_YES if prime)

Returns:

CRYPT_OK on success

4.5.2.24 int(* ltc_math_descriptor::lcm)(void *a, void *b, void *c)

lcm

Parameters:

a The first integer

b The second integer

c The destination for [a, b]

Returns:

CRYPT_OK on success

4.5.2.25 int(* ltc_math_descriptor::modi)(void *a, unsigned long b, unsigned long *c)

Get remainder (small value).

Parameters:

a The integer to reduce

b The modulus (upto bits_per_digit in length)

c The destination for the residue

Returns:

CRYPT_OK on success

4.5.2.26 void(* ltc_math_descriptor::montgomery_deinit)(void *a)

clean up (frees memory)

Parameters:

a The value "b" from montgomery_setup()

Returns:

CRYPT_OK on success

4.5.2.27 int(* ltc_math_descriptor::montgomery_normalization)(void *a, void *b)

get normalization value

Parameters:

a The destination for the normalization value

b The modulus

Returns:

CRYPT_OK on success

4.5.2.28 int($*$ ltc_math_descriptor::montgomery_reduce)(void $*$a, void $*$b, void $*$c)

reduce a number

Parameters:

- ***a*** The number [and dest] to reduce
- ***b*** The modulus
- ***c*** The value "b" from montgomery_setup()

Returns:

CRYPT_OK on success

4.5.2.29 int($*$ ltc_math_descriptor::montgomery_setup)(void $*$a, void $**$b)

setup montgomery

Parameters:

- ***a*** The modulus
- ***b*** The destination for the reduction digit

Returns:

CRYPT_OK on success

4.5.2.30 int($*$ ltc_math_descriptor::mpdiv)(void $*$a, void $*$b, void $*$c, void $*$d)

Divide an integer.

Parameters:

- ***a*** The dividend
- ***b*** The divisor
- ***c*** The quotient (can be NULL to signify don't care)
- ***d*** The remainder (can be NULL to signify don't care)

Returns:

CRYPT_OK on success

4.5.2.31 int($*$ ltc_math_descriptor::mul)(void $*$a, void $*$b, void $*$c)

multiply two integers

Parameters:

- ***a*** The first source integer
- ***b*** The second source integer (single digit of upto bits_per_digit in length)
- ***c*** The destination of "a $*$ b"

Returns:

CRYPT_OK on success

4.5.2.32 int(* ltc_math_descriptor::muli)(void *a, unsigned long b, void *c)

multiply two integers

Parameters:

a The first source integer

b The second source integer (single digit of upto bits_per_digit in length)

c The destination of "a * b"

Returns:

CRYPT_OK on success

4.5.2.33 int(* ltc_math_descriptor::mulmod)(void *a, void *b, void *c, void *d)

Modular multiplication.

Parameters:

a The first source

b The second source

c The modulus

d The destination (a*b mod c)

Returns:

CRYPT_OK on success

4.5.2.34 char* ltc_math_descriptor::name

Name of the math provider.

Definition at line 21 of file tomcrypt_math.h.

Referenced by dsa_import(), dsa_make_key(), ecc_import(), ecc_make_key(), rsa_import(), and rsa_-make_key().

4.5.2.35 int(* ltc_math_descriptor::neg)(void *src, void *dst)

negate

Parameters:

src The number to negate

dst The destination

Returns:

CRYPT_OK on success

4.5.2.36 int(* ltc_math_descriptor::read_radix)(void *a, const char *str, int radix)

read ascii string

Parameters:

a The integer to store into

str The string to read

radix The radix the integer has been represented in (2-64)

Returns:

CRYPT_OK on success

4.5.2.37 int(* ltc_math_descriptor::rsa_keygen)(prng_state *prng, int wprng, int size, long e, rsa_key *key)

RSA Key Generation.

Parameters:

prng An active PRNG state

wprng The index of the PRNG desired

size The size of the modulus (key size) desired (octets)

e The "e" value (public key). e==65537 is a good choice

key [out] Destination of a newly created private key pair

Returns:

CRYPT_OK if successful, upon error all allocated ram is freed

4.5.2.38 int(* ltc_math_descriptor::rsa_me)(const unsigned char *in, unsigned long inlen, unsigned char *out, unsigned long *outlen, int which, rsa_key *key)

RSA exponentiation.

Parameters:

in The octet array representing the base

inlen The length of the input

out The destination (to be stored in an octet array format)

outlen The length of the output buffer and the resulting size (zero padded to the size of the modulus)

which PK_PUBLIC for public RSA and PK_PRIVATE for private RSA

key The RSA key to use

Returns:

CRYPT_OK on success

Referenced by rsa_decrypt_key_ex(), rsa_encrypt_key_ex(), rsa_sign_hash_ex(), and rsa_verify_hash_-ex().

4.5.2.39 int(* ltc_math_descriptor::set_int)(void *a, unsigned long n)

set small constant

Parameters:

a Number to write to

n Source upto bits_per_digit (actually meant for very small constants)

Returns:

CRYPT_OK on succcess

4.5.2.40 int(* ltc_math_descriptor::sqr)(void *a, void *b)

Square an integer.

Parameters:

a The integer to square

b The destination

Returns:

CRYPT_OK on success

4.5.2.41 int(* ltc_math_descriptor::sqrmod)(void *a, void *b, void *c)

Modular squaring.

Parameters:

a The first source

b The modulus

c The destination (a*a mod b)

Returns:

CRYPT_OK on success

4.5.2.42 int(* ltc_math_descriptor::sub)(void *a, void *b, void *c)

subtract two integers

Parameters:

a The first source integer

b The second source integer

c The destination of "a - b"

Returns:

CRYPT_OK on success

4.5.2.43 int(* ltc_math_descriptor::subi)(void *a, unsigned long b, void *c)

subtract two integers

Parameters:

a The first source integer

b The second source integer (single digit of upto bits_per_digit in length)

c The destination of "a - b"

Returns:

CRYPT_OK on success

4.5.2.44 int(* ltc_math_descriptor::twoexpt)(void *a, int n)

Compute a power of two.

Parameters:

a The integer to store the power in

n The power of two you want to store (a = 2^n)

Returns:

CRYPT_OK on success

4.5.2.45 int(* ltc_math_descriptor::unsigned_read)(void *dst, unsigned char *src, unsigned long len)

read an array of octets and store as integer

Parameters:

dst The integer to load

src The array of octets

len The number of octets

Returns:

CRYPT_OK on success

4.5.2.46 unsigned long(* ltc_math_descriptor::unsigned_size)(void *a)

get size as unsigned char string

Parameters:

a The integer to get the size (when stored in array of octets)

Returns:

The length of the integer

4.5.2.47 int(* ltc_math_descriptor::unsigned_write)(void *src, unsigned char *dst)

store an integer as an array of octets

Parameters:

src The integer to store

dst The buffer to store the integer in

Returns:

CRYPT_OK on success

4.5.2.48 int(* ltc_math_descriptor::write_radix)(void *a, char *str, int radix)

write number to string

Parameters:

a The integer to store

str The destination for the string

radix The radix the integer is to be represented in (2-64)

Returns:

CRYPT_OK on success

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

- headers/tomcrypt_math.h

4.6 ltc_prng_descriptor Struct Reference

```
#include <tomcrypt_prng.h>
```

4.6.1 Detailed Description

PRNG descriptor.

Definition at line 67 of file tomcrypt_prng.h.

Data Fields

- char ∗ name

  Name of the PRNG.

- int export_size

  size in bytes of exported state

- int(∗ start)(prng_state ∗prng)

  Start a PRNG state.

- int(∗ add_entropy)(const unsigned char ∗in, unsigned long inlen, prng_state ∗prng)

  Add entropy to the PRNG.

- int(∗ ready)(prng_state ∗prng)

  Ready a PRNG state to read from.

- unsigned long(∗ read)(unsigned char ∗out, unsigned long outlen, prng_state ∗prng)

  Read from the PRNG.

- int(∗ done)(prng_state ∗prng)

  Terminate a PRNG state.

- int(∗ pexport)(unsigned char ∗out, unsigned long ∗outlen, prng_state ∗prng)

  Export a PRNG state.

- int(∗ pimport)(const unsigned char ∗in, unsigned long inlen, prng_state ∗prng)

  Import a PRNG state.

- int(∗ test)(void)

  Self-test the PRNG.

4.6.2 Field Documentation

4.6.2.1 int(∗ ltc_prng_descriptor::add_entropy)(const unsigned char ∗in, unsigned long inlen, prng_state ∗prng)

Add entropy to the PRNG.

Parameters:

in The entropy

inlen Length of the entropy (octets)\

prng The PRNG state

Returns:

CRYPT_OK if successful

4.6.2.2 int(∗ ltc_prng_descriptor::done)(prng_state ∗prng)

Terminate a PRNG state.

Parameters:

prng The PRNG state to terminate

Returns:

CRYPT_OK if successful

Referenced by f9_init(), f9_memory(), and pkcs_5_alg1().

4.6.2.3 int ltc_prng_descriptor::export_size

size in bytes of exported state

Definition at line 71 of file tomcrypt_prng.h.

4.6.2.4 char∗ ltc_prng_descriptor::name

Name of the PRNG.

Definition at line 69 of file tomcrypt_prng.h.

Referenced by prng_is_valid(), register_cipher(), unregister_cipher(), unregister_hash(), and unregister_-prng().

4.6.2.5 int(∗ ltc_prng_descriptor::pexport)(unsigned char ∗out, unsigned long ∗outlen, prng_state ∗prng)

Export a PRNG state.

Parameters:

out [out] The destination for the state

outlen [in/out] The max size and resulting size of the PRNG state

prng The PRNG to export

Returns:

CRYPT_OK if successful

4.6.2.6 int(* ltc_prng_descriptor::pimport)(const unsigned char *in, unsigned long inlen, prng_state *prng)

Import a PRNG state.

Parameters:

in The data to import

inlen The length of the data to import (octets)

prng The PRNG to initialize/import

Returns:

CRYPT_OK if successful

4.6.2.7 unsigned long(* ltc_prng_descriptor::read)(unsigned char *out, unsigned long outlen, prng_state *prng)

Read from the PRNG.

Parameters:

out [out] Where to store the data

outlen Length of data desired (octets)

prng The PRNG state to read from

Returns:

Number of octets read

4.6.2.8 int(* ltc_prng_descriptor::ready)(prng_state *prng)

Ready a PRNG state to read from.

Parameters:

prng The PRNG state to ready

Returns:

CRYPT_OK if successful

4.6.2.9 int(* ltc_prng_descriptor::start)(prng_state *prng)

Start a PRNG state.

Parameters:

prng [out] The state to initialize

Returns:

CRYPT_OK if successful

4.6.2.10 int(∗ ltc_prng_descriptor::test)(void)

Self-test the PRNG.

Returns:
CRYPT_OK if successful, CRYPT_NOP if self-testing has been disabled

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

- headers/tomcrypt_prng.h

4.7 Prng_state Union Reference

```
#include <tomcrypt_prng.h>
```

4.7.1 Detailed Description

Definition at line 50 of file tomcrypt_prng.h.

Data Fields

- char dummy [1]

4.7.2 Field Documentation

4.7.2.1 char Prng_state::dummy[1]

Definition at line 51 of file tomcrypt_prng.h.

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

- headers/tomcrypt_prng.h

4.8 Symmetric_key Union Reference

```
#include <tomcrypt_cipher.h>
```

4.8.1 Detailed Description

Definition at line 134 of file tomcrypt_cipher.h.

Data Fields

- void * data

4.8.2 Field Documentation

4.8.2.1 void* Symmetric_key::data

Definition at line 187 of file tomcrypt_cipher.h.

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

- headers/tomcrypt_cipher.h

Chapter 5

LibTomCrypt File Documentation

5.1 ciphers/aes/aes.c File Reference

5.1.1 Detailed Description

Implementation of AES.

Definition in file aes.c.

```
#include "tomcrypt.h"
```

```
#include "aes_tab.c"
```

Include dependency graph for aes.c:

Defines

- #define SETUP rijndael_setup
- #define ECB_ENC rijndael_ecb_encrypt
- #define ECB_DEC rijndael_ecb_decrypt
- #define ECB_DONE rijndael_done
- #define ECB_TEST rijndael_test
- #define ECB_KS rijndael_keysize

Functions

- static ulong32 setup_mix (ulong32 temp)
- int SETUP (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the AES (Rijndael) block cipher.

- int ECB_ENC (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with AES.

- int ECB_DEC (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with AES.

- int ECB_TEST (void)

Performs a self-test of the AES block cipher.

- void ECB_DONE (symmetric_key *skey)

  Terminate the context.

- int ECB_KS (int *keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor rijndael_desc
- const struct ltc_cipher_descriptor aes_desc

5.1.2 Define Documentation

5.1.2.1 #define ECB_DEC rijndael_ecb_decrypt

Definition at line 41 of file aes.c.

5.1.2.2 #define ECB_DONE rijndael_done

Definition at line 42 of file aes.c.

5.1.2.3 #define ECB_ENC rijndael_ecb_encrypt

Definition at line 40 of file aes.c.

5.1.2.4 #define ECB_KS rijndael_keysize

Definition at line 44 of file aes.c.

5.1.2.5 #define ECB_TEST rijndael_test

Definition at line 43 of file aes.c.

5.1.2.6 #define SETUP rijndael_setup

Definition at line 39 of file aes.c.

5.1.3 Function Documentation

5.1.3.1 int ECB_DEC (const unsigned char * *ct*, unsigned char * *pt*, symmetric_key * *skey*)

Decrypts a block of text with AES.

Parameters:

ct The input ciphertext (16 bytes)

pt The output plaintext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 468 of file aes.c.

References byte, LTC_ARGCHK, Td0, Td1, Td2, and Td3.

```
{
    ulong32 s0, s1, s2, s3, t0, t1, t2, t3, *rk;
    int Nr, r;

    LTC_ARGCHK(pt != NULL);
    LTC_ARGCHK(ct != NULL);
    LTC_ARGCHK(skey != NULL);

    Nr = skey->rijndael.Nr;
    rk = skey->rijndael.dK;

    /*
     * map byte array block to cipher state
     * and add initial round key:
     */
    LOAD32H(s0, ct      ); s0 ^= rk[0];
    LOAD32H(s1, ct  +  4); s1 ^= rk[1];
    LOAD32H(s2, ct  +  8); s2 ^= rk[2];
    LOAD32H(s3, ct  + 12); s3 ^= rk[3];

#ifdef LTC_SMALL_CODE
    for (r = 0; ; r++) {
        rk += 4;
        t0 =
            Td0(byte(s0, 3)) ^
            Td1(byte(s3, 2)) ^
            Td2(byte(s2, 1)) ^
            Td3(byte(s1, 0)) ^
            rk[0];
        t1 =
            Td0(byte(s1, 3)) ^
            Td1(byte(s0, 2)) ^
            Td2(byte(s3, 1)) ^
            Td3(byte(s2, 0)) ^
            rk[1];
        t2 =
            Td0(byte(s2, 3)) ^
            Td1(byte(s1, 2)) ^
            Td2(byte(s0, 1)) ^
            Td3(byte(s3, 0)) ^
            rk[2];
        t3 =
            Td0(byte(s3, 3)) ^
            Td1(byte(s2, 2)) ^
            Td2(byte(s1, 1)) ^
            Td3(byte(s0, 0)) ^
            rk[3];
        if (r == Nr-2) {
           break;
        }
        s0 = t0; s1 = t1; s2 = t2; s3 = t3;
    }
```

```
    rk += 4;

#else

    /*
     * Nr - 1 full rounds:
     */
    r = Nr >> 1;
    for (;;) {

        t0 =
            Td0(byte(s0, 3)) ^
            Td1(byte(s3, 2)) ^
            Td2(byte(s2, 1)) ^
            Td3(byte(s1, 0)) ^
            rk[4];
        t1 =
            Td0(byte(s1, 3)) ^
            Td1(byte(s0, 2)) ^
            Td2(byte(s3, 1)) ^
            Td3(byte(s2, 0)) ^
            rk[5];
        t2 =
            Td0(byte(s2, 3)) ^
            Td1(byte(s1, 2)) ^
            Td2(byte(s0, 1)) ^
            Td3(byte(s3, 0)) ^
            rk[6];
        t3 =
            Td0(byte(s3, 3)) ^
            Td1(byte(s2, 2)) ^
            Td2(byte(s1, 1)) ^
            Td3(byte(s0, 0)) ^
            rk[7];

        rk += 8;
        if (--r == 0) {
            break;
        }


        s0 =
            Td0(byte(t0, 3)) ^
            Td1(byte(t3, 2)) ^
            Td2(byte(t2, 1)) ^
            Td3(byte(t1, 0)) ^
            rk[0];
        s1 =
            Td0(byte(t1, 3)) ^
            Td1(byte(t0, 2)) ^
            Td2(byte(t3, 1)) ^
            Td3(byte(t2, 0)) ^
            rk[1];
        s2 =
            Td0(byte(t2, 3)) ^
            Td1(byte(t1, 2)) ^
            Td2(byte(t0, 1)) ^
            Td3(byte(t3, 0)) ^
            rk[2];
        s3 =
            Td0(byte(t3, 3)) ^
            Td1(byte(t2, 2)) ^
            Td2(byte(t1, 1)) ^
            Td3(byte(t0, 0)) ^
            rk[3];
    }
#endif
```

```

    /*
     * apply last round and
     * map cipher state to byte array block:
     */
    s0 =
        (Td4[byte(t0, 3)] & 0xff000000) ^
        (Td4[byte(t3, 2)] & 0x00ff0000) ^
        (Td4[byte(t2, 1)] & 0x0000ff00) ^
        (Td4[byte(t1, 0)] & 0x000000ff) ^
        rk[0];
    STORE32H(s0, pt);
    s1 =
        (Td4[byte(t1, 3)] & 0xff000000) ^
        (Td4[byte(t0, 2)] & 0x00ff0000) ^
        (Td4[byte(t3, 1)] & 0x0000ff00) ^
        (Td4[byte(t2, 0)] & 0x000000ff) ^
        rk[1];
    STORE32H(s1, pt+4);
    s2 =
        (Td4[byte(t2, 3)] & 0xff000000) ^
        (Td4[byte(t1, 2)] & 0x00ff0000) ^
        (Td4[byte(t0, 1)] & 0x0000ff00) ^
        (Td4[byte(t3, 0)] & 0x000000ff) ^
        rk[2];
    STORE32H(s2, pt+8);
    s3 =
        (Td4[byte(t3, 3)] & 0xff000000) ^
        (Td4[byte(t2, 2)] & 0x00ff0000) ^
        (Td4[byte(t1, 1)] & 0x0000ff00) ^
        (Td4[byte(t0, 0)] & 0x000000ff) ^
        rk[3];
    STORE32H(s3, pt+12);

    return CRYPT_OK;
}
```

5.1.3.2 void ECB_DONE (symmetric_key ∗ *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 727 of file aes.c.

```
728 {
729 }
```

5.1.3.3 int ECB_ENC (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with AES.

Parameters:

pt The input plaintext (16 bytes)

ct The output ciphertext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 289 of file aes.c.

References byte, LTC_ARGCHK, t1, t2, t3, Te0, Te1, Te2, and Te3.

```
{
    ulong32 s0, s1, s2, s3, t0, t1, t2, t3, *rk;
    int Nr, r;

    LTC_ARGCHK(pt != NULL);
    LTC_ARGCHK(ct != NULL);
    LTC_ARGCHK(skey != NULL);

    Nr = skey->rijndael.Nr;
    rk = skey->rijndael.eK;

    /*
     * map byte array block to cipher state
     * and add initial round key:
     */
    LOAD32H(s0, pt      ); s0 ^= rk[0];
    LOAD32H(s1, pt  +  4); s1 ^= rk[1];
    LOAD32H(s2, pt  +  8); s2 ^= rk[2];
    LOAD32H(s3, pt  + 12); s3 ^= rk[3];

#ifdef LTC_SMALL_CODE

    for (r = 0; ; r++) {
        rk += 4;
        t0 =
            Te0(byte(s0, 3)) ^
            Te1(byte(s1, 2)) ^
            Te2(byte(s2, 1)) ^
            Te3(byte(s3, 0)) ^
            rk[0];
        t1 =
            Te0(byte(s1, 3)) ^
            Te1(byte(s2, 2)) ^
            Te2(byte(s3, 1)) ^
            Te3(byte(s0, 0)) ^
            rk[1];
        t2 =
            Te0(byte(s2, 3)) ^
            Te1(byte(s3, 2)) ^
            Te2(byte(s0, 1)) ^
            Te3(byte(s1, 0)) ^
            rk[2];
        t3 =
            Te0(byte(s3, 3)) ^
            Te1(byte(s0, 2)) ^
            Te2(byte(s1, 1)) ^
            Te3(byte(s2, 0)) ^
            rk[3];
        if (r == Nr-2) {
           break;
        }
        s0 = t0; s1 = t1; s2 = t2; s3 = t3;
    }
    rk += 4;

#else

    /*
     * Nr - 1 full rounds:
     */
```

```
    r = Nr >> 1;
    for (;;) {
        t0 =
            Te0(byte(s0, 3)) ^
            Te1(byte(s1, 2)) ^
            Te2(byte(s2, 1)) ^
            Te3(byte(s3, 0)) ^
            rk[4];
        t1 =
            Te0(byte(s1, 3)) ^
            Te1(byte(s2, 2)) ^
            Te2(byte(s3, 1)) ^
            Te3(byte(s0, 0)) ^
            rk[5];
        t2 =
            Te0(byte(s2, 3)) ^
            Te1(byte(s3, 2)) ^
            Te2(byte(s0, 1)) ^
            Te3(byte(s1, 0)) ^
            rk[6];
        t3 =
            Te0(byte(s3, 3)) ^
            Te1(byte(s0, 2)) ^
            Te2(byte(s1, 1)) ^
            Te3(byte(s2, 0)) ^
            rk[7];

        rk += 8;
        if (--r == 0) {
            break;
        }

        s0 =
            Te0(byte(t0, 3)) ^
            Te1(byte(t1, 2)) ^
            Te2(byte(t2, 1)) ^
            Te3(byte(t3, 0)) ^
            rk[0];
        s1 =
            Te0(byte(t1, 3)) ^
            Te1(byte(t2, 2)) ^
            Te2(byte(t3, 1)) ^
            Te3(byte(t0, 0)) ^
            rk[1];
        s2 =
            Te0(byte(t2, 3)) ^
            Te1(byte(t3, 2)) ^
            Te2(byte(t0, 1)) ^
            Te3(byte(t1, 0)) ^
            rk[2];
        s3 =
            Te0(byte(t3, 3)) ^
            Te1(byte(t0, 2)) ^
            Te2(byte(t1, 1)) ^
            Te3(byte(t2, 0)) ^
            rk[3];
    }

#endif

    /*
     * apply last round and
     * map cipher state to byte array block:
     */
    s0 =
        (Te4_3[byte(t0, 3)]) ^
        (Te4_2[byte(t1, 2)]) ^
```

```
        (Te4_1[byte(t2, 1)]) ^
        (Te4_0[byte(t3, 0)]) ^
        rk[0];
    STORE32H(s0, ct);
    s1 =
        (Te4_3[byte(t1, 3)]) ^
        (Te4_2[byte(t2, 2)]) ^
        (Te4_1[byte(t3, 1)]) ^
        (Te4_0[byte(t0, 0)]) ^
        rk[1];
    STORE32H(s1, ct+4);
    s2 =
        (Te4_3[byte(t2, 3)]) ^
        (Te4_2[byte(t3, 2)]) ^
        (Te4_1[byte(t0, 1)]) ^
        (Te4_0[byte(t1, 0)]) ^
        rk[2];
    STORE32H(s2, ct+8);
    s3 =
        (Te4_3[byte(t3, 3)]) ^
        (Te4_2[byte(t0, 2)]) ^
        (Te4_1[byte(t1, 1)]) ^
        (Te4_0[byte(t2, 0)]) ^
        rk[3];
    STORE32H(s3, ct+12);

    return CRYPT_OK;
}
```

5.1.3.4 int ECB_KS (int ∗ *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 737 of file aes.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);

   if (*keysize < 16)
      return CRYPT_INVALID_KEYSIZE;
   if (*keysize < 24) {
      *keysize = 16;
      return CRYPT_OK;
   } else if (*keysize < 32) {
      *keysize = 24;
      return CRYPT_OK;
   } else {
      *keysize = 32;
      return CRYPT_OK;
   }
}
```

5.1.3.5 int ECB_TEST (void)

Performs a self-test of the AES block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 640 of file aes.c.

References CRYPT_NOP, CRYPT_OK, and zeromem().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
 int err;
 static const struct {
     int keylen;
     unsigned char key[32], pt[16], ct[16];
 } tests[] = {
    { 16,
      { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
      { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff },
      { 0x69, 0xc4, 0xe0, 0xd8, 0x6a, 0x7b, 0x04, 0x30,
        0xd8, 0xcd, 0xb7, 0x80, 0x70, 0xb4, 0xc5, 0x5a }
    }, {
      24,
      { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
        0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17 },
      { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff },
      { 0xdd, 0xa9, 0x7c, 0xa4, 0x86, 0x4c, 0xdf, 0xe0,
        0x6e, 0xaf, 0x70, 0xa0, 0xec, 0x0d, 0x71, 0x91 }
    }, {
      32,
      { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
        0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
        0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f },
      { 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff },
      { 0x8e, 0xa2, 0xb7, 0xca, 0x51, 0x67, 0x45, 0xbf,
        0xea, 0xfc, 0x49, 0x90, 0x4b, 0x49, 0x60, 0x89 }
    }
 };

 symmetric_key key;
 unsigned char tmp[2][16];
 int i, y;

 for (i = 0; i < (int)(sizeof(tests)/sizeof(tests[0])); i++) {
    zeromem(&key, sizeof(key));
    if ((err = rijndael_setup(tests[i].key, tests[i].keylen, 0, &key)) != CRYPT_OK) {
       return err;
    }

    rijndael_ecb_encrypt(tests[i].pt, tmp[0], &key);
    rijndael_ecb_decrypt(tmp[0], tmp[1], &key);
    if (XMEMCMP(tmp[0], tests[i].ct, 16) || XMEMCMP(tmp[1], tests[i].pt, 16)) {
#if 0
       printf("\n\nTest %d failed\n", i);
       if (XMEMCMP(tmp[0], tests[i].ct, 16)) {
```

```
          printf("CT: ");
          for (i = 0; i < 16; i++) {
             printf("%02x ", tmp[0][i]);
          }
          printf("\n");
       } else {
          printf("PT: ");
          for (i = 0; i < 16; i++) {
             printf("%02x ", tmp[1][i]);
          }
          printf("\n");
       }
#endif
        return CRYPT_FAIL_TESTVECTOR;
    }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 16; y++) tmp[0][y] = 0;
      for (y = 0; y < 1000; y++) rijndael_ecb_encrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 1000; y++) rijndael_ecb_decrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 16; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
 }
 return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.1.3.6 int SETUP (const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_key ∗ *skey*)

Initialize the AES (Rijndael) block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 121 of file aes.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, LTC_ARGCHK, rcon, and setup_mix().

```
{
    int i, j;
    ulong32 temp, *rk;
#ifndef ENCRYPT_ONLY
    ulong32 *rrk;
#endif
    LTC_ARGCHK(key  != NULL);
    LTC_ARGCHK(skey != NULL);

    if (keylen != 16 && keylen != 24 && keylen != 32) {
       return CRYPT_INVALID_KEYSIZE;
    }

```

```
    if (num_rounds != 0 && num_rounds != (10 + ((keylen/8)-2)*2)) {
       return CRYPT_INVALID_ROUNDS;
    }

    skey->rijndael.Nr = 10 + ((keylen/8)-2)*2;

    /* setup the forward key */
    i                 = 0;
    rk                = skey->rijndael.eK;
    LOAD32H(rk[0], key     );
    LOAD32H(rk[1], key +  4);
    LOAD32H(rk[2], key +  8);
    LOAD32H(rk[3], key + 12);
    if (keylen == 16) {
        j = 44;
        for (;;) {
            temp  = rk[3];
            rk[4] = rk[0] ^ setup_mix(temp) ^ rcon[i];
            rk[5] = rk[1] ^ rk[4];
            rk[6] = rk[2] ^ rk[5];
            rk[7] = rk[3] ^ rk[6];
            if (++i == 10) {
               break;
            }
            rk += 4;
        }
    } else if (keylen == 24) {
        j = 52;
        LOAD32H(rk[4], key + 16);
        LOAD32H(rk[5], key + 20);
        for (;;) {
        #ifdef _MSC_VER
            temp = skey->rijndael.eK[rk - skey->rijndael.eK + 5];
        #else
            temp = rk[5];
        #endif
            rk[ 6] = rk[ 0] ^ setup_mix(temp) ^ rcon[i];
            rk[ 7] = rk[ 1] ^ rk[ 6];
            rk[ 8] = rk[ 2] ^ rk[ 7];
            rk[ 9] = rk[ 3] ^ rk[ 8];
            if (++i == 8) {
                break;
            }
            rk[10] = rk[ 4] ^ rk[ 9];
            rk[11] = rk[ 5] ^ rk[10];
            rk += 6;
        }
    } else if (keylen == 32) {
        j = 60;
        LOAD32H(rk[4], key + 16);
        LOAD32H(rk[5], key + 20);
        LOAD32H(rk[6], key + 24);
        LOAD32H(rk[7], key + 28);
        for (;;) {
        #ifdef _MSC_VER
            temp = skey->rijndael.eK[rk - skey->rijndael.eK + 7];
        #else
            temp = rk[7];
        #endif
            rk[ 8] = rk[ 0] ^ setup_mix(temp) ^ rcon[i];
            rk[ 9] = rk[ 1] ^ rk[ 8];
            rk[10] = rk[ 2] ^ rk[ 9];
            rk[11] = rk[ 3] ^ rk[10];
            if (++i == 7) {
                break;
            }
            temp = rk[11];
```

```
            rk[12] = rk[ 4] ^ setup_mix(RORc(temp, 8));
            rk[13] = rk[ 5] ^ rk[12];
            rk[14] = rk[ 6] ^ rk[13];
            rk[15] = rk[ 7] ^ rk[14];
            rk += 8;
        }
    } else {
       /* this can't happen */
       return CRYPT_ERROR;
    }

#ifndef ENCRYPT_ONLY
    /* setup the inverse key now */
    rk   = skey->rijndael.dK;
    rrk  = skey->rijndael.eK + j - 4;

    /* apply the inverse MixColumn transform to all round keys but the first and the last: */
    /* copy first */
    *rk++ = *rrk++;
    *rk++ = *rrk++;
    *rk++ = *rrk++;
    *rk   = *rrk;
    rk -= 3; rrk -= 3;

    for (i = 1; i < skey->rijndael.Nr; i++) {
        rrk -= 4;
        rk  += 4;
    #ifdef LTC_SMALL_CODE
        temp = rrk[0];
        rk[0] = setup_mix2(temp);
        temp = rrk[1];
        rk[1] = setup_mix2(temp);
        temp = rrk[2];
        rk[2] = setup_mix2(temp);
        temp = rrk[3];
        rk[3] = setup_mix2(temp);
     #else
        temp = rrk[0];
        rk[0] =
            Tks0[byte(temp, 3)] ^
            Tks1[byte(temp, 2)] ^
            Tks2[byte(temp, 1)] ^
            Tks3[byte(temp, 0)];
        temp = rrk[1];
        rk[1] =
            Tks0[byte(temp, 3)] ^
            Tks1[byte(temp, 2)] ^
            Tks2[byte(temp, 1)] ^
            Tks3[byte(temp, 0)];
        temp = rrk[2];
        rk[2] =
            Tks0[byte(temp, 3)] ^
            Tks1[byte(temp, 2)] ^
            Tks2[byte(temp, 1)] ^
            Tks3[byte(temp, 0)];
        temp = rrk[3];
        rk[3] =
            Tks0[byte(temp, 3)] ^
            Tks1[byte(temp, 2)] ^
            Tks2[byte(temp, 1)] ^
            Tks3[byte(temp, 0)];
      #endif

    }

    /* copy last */
    rrk -= 4;
```

```
    rk  += 4;
    *rk++ = *rrk++;
    *rk++ = *rrk++;
    *rk++ = *rrk++;
    *rk   = *rrk;
#endif /* ENCRYPT_ONLY */

    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.1.3.7 static ulong32 setup_mix (ulong32 *temp*) [static]

Definition at line 93 of file aes.c.

References byte, Te4_0, Te4_1, Te4_2, and Te4_3.

Referenced by SETUP().

```
{
   return (Te4_3[byte(temp, 2)]) ^
          (Te4_2[byte(temp, 1)]) ^
          (Te4_1[byte(temp, 0)]) ^
          (Te4_0[byte(temp, 3)]);
}
```

5.1.4 Variable Documentation

5.1.4.1 const struct ltc_cipher_descriptor aes_desc

Initial value:

```
{
    "aes",
    6,
    16, 32, 16, 10,
    SETUP, ECB_ENC, ECB_DEC, ECB_TEST, ECB_DONE, ECB_KS,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 55 of file aes.c.

Referenced by yarrow_start().

5.1.4.2 const struct ltc_cipher_descriptor rijndael_desc

Initial value:

```
{
    "rijndael",
    6,
    16, 32, 16, 10,
    SETUP, ECB_ENC, ECB_DEC, ECB_TEST, ECB_DONE, ECB_KS,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 46 of file aes.c.

Referenced by yarrow_start().

5.2 ciphers/aes/aes_tab.c File Reference

5.2.1 Detailed Description

AES tables.

Definition in file aes_tab.c.

This graph shows which files directly or indirectly include this file:

Defines

- #define Te0(x) TE0[x]
- #define Te1(x) TE1[x]
- #define Te2(x) TE2[x]
- #define Te3(x) TE3[x]
- #define Td0(x) TD0[x]
- #define Td1(x) TD1[x]
- #define Td2(x) TD2[x]
- #define Td3(x) TD3[x]

Variables

- static const ulong32 TE0 [256]
- static const ulong32 Te4 [256]
- static const ulong32 TD0 [256]
- static const ulong32 Td4 [256]
- static const ulong32 TE1 [256]
- static const ulong32 TE2 [256]
- static const ulong32 TE3 [256]
- static const ulong32 Te4_0 [ ]
- static const ulong32 Te4_1 [ ]
- static const ulong32 Te4_2 [ ]
- static const ulong32 Te4_3 [ ]
- static const ulong32 TD1 [256]
- static const ulong32 TD2 [256]
- static const ulong32 TD3 [256]
- static const ulong32 Tks0 [ ]
- static const ulong32 Tks1 [ ]
- static const ulong32 Tks2 [ ]
- static const ulong32 Tks3 [ ]
- static const ulong32 rcon [ ]

5.2.2 Define Documentation

5.2.2.1 #define Td0(x) TD0[x]

Definition at line 328 of file aes_tab.c.

Referenced by ECB_DEC().

5.2.2.2 #define Td1(x) TD1[x]

Definition at line 329 of file aes_tab.c.

Referenced by ECB_DEC().

5.2.2.3 #define Td2(x) TD2[x]

Definition at line 330 of file aes_tab.c.

Referenced by ECB_DEC().

5.2.2.4 #define Td3(x) TD3[x]

Definition at line 331 of file aes_tab.c.

Referenced by ECB_DEC().

5.2.2.5 #define Te0(x) TE0[x]

Definition at line 323 of file aes_tab.c.

Referenced by ECB_ENC(), and four_rounds().

5.2.2.6 #define Te1(x) TE1[x]

Definition at line 324 of file aes_tab.c.

Referenced by ECB_ENC(), and four_rounds().

5.2.2.7 #define Te2(x) TE2[x]

Definition at line 325 of file aes_tab.c.

Referenced by ECB_ENC(), and four_rounds().

5.2.2.8 #define Te3(x) TE3[x]

Definition at line 326 of file aes_tab.c.

Referenced by ECB_ENC(), and four_rounds().

5.2.3 Variable Documentation

5.2.3.1 const ulong32 rcon[ ] [static]

Initial value:

```
 {
    0x01000000UL, 0x02000000UL, 0x04000000UL, 0x08000000UL,
    0x10000000UL, 0x20000000UL, 0x40000000UL, 0x80000000UL,
    0x1B000000UL, 0x36000000UL,
}
```

Definition at line 1020 of file aes_tab.c.

Referenced by SETUP().

5.2.3.2 const ulong32 TD0[256] [static]

Definition at line 168 of file aes_tab.c.

5.2.3.3 const ulong32 TD1[256] [static]

Definition at line 677 of file aes_tab.c.

5.2.3.4 const ulong32 TD2[256] [static]

Definition at line 743 of file aes_tab.c.

5.2.3.5 const ulong32 TD3[256] [static]

Definition at line 809 of file aes_tab.c.

5.2.3.6 const ulong32 Td4[256] [static]

Definition at line 235 of file aes_tab.c.

5.2.3.7 const ulong32 TE0[256] [static]

Definition at line 30 of file aes_tab.c.

5.2.3.8 const ulong32 TE1[256] [static]

Definition at line 333 of file aes_tab.c.

5.2.3.9 const ulong32 TE2[256] [static]

Definition at line 399 of file aes_tab.c.

5.2.3.10 const ulong32 TE3[256] [static]

Definition at line 465 of file aes_tab.c.

5.2.3.11 const ulong32 Te4[256] [static]

Definition at line 98 of file aes_tab.c.

5.2.3.12 const ulong32 Te4_0[ ] [static]

Definition at line 534 of file aes_tab.c.

Referenced by setup_mix().

5.2.3.13 const ulong32 Te4_1[ ] [static]

Definition at line 569 of file aes_tab.c.

Referenced by setup_mix().

5.2.3.14 const ulong32 Te4_2[ ] [static]

Definition at line 604 of file aes_tab.c.

Referenced by setup_mix().

5.2.3.15 const ulong32 Te4_3[ ] [static]

Definition at line 639 of file aes_tab.c.

Referenced by setup_mix().

5.2.3.16 const ulong32 Tks0[ ] [static]

Definition at line 876 of file aes_tab.c.

5.2.3.17 const ulong32 Tks1[ ] [static]

Definition at line 911 of file aes_tab.c.

5.2.3.18 const ulong32 Tks2[ ] [static]

Definition at line 946 of file aes_tab.c.

5.2.3.19 const ulong32 Tks3[ ] [static]

Definition at line 981 of file aes_tab.c.

5.3 ciphers/anubis.c File Reference

5.3.1 Detailed Description

Anubis implementation derived from public domain source Authors: Paulo S.L.M.

Barreto and Vincent Rijmen.

Definition in file anubis.c.

`#include "tomcrypt.h"`

Include dependency graph for anubis.c:

Defines

- #define MIN_N 4
- #define MAX_N 10
- #define MIN_ROUNDS (8 + MIN_N)
- #define MAX_ROUNDS (8 + MAX_N)
- #define MIN_KEYSIZEB (4∗MIN_N)
- #define MAX_KEYSIZEB (4∗MAX_N)
- #define BLOCKSIZE 128
- #define BLOCKSIZEB (BLOCKSIZE/8)

Functions

- int anubis_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the Anubis block cipher.

- static void anubis_crypt (const unsigned char ∗plaintext, unsigned char ∗ciphertext, ulong32 roundKey[18+1][4], int R)
- int anubis_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with Anubis.

- int anubis_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with Anubis.

- int anubis_test (void)

  Performs a self-test of the Anubis block cipher.

- void anubis_done (symmetric_key ∗skey)

  Terminate the context.

- int anubis_keysize (int ∗keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor anubis_desc
- static const ulong32 T0 [256]
- static const ulong32 T1 [256]
- static const ulong32 T2 [256]
- static const ulong32 T3 [256]
- static const ulong32 T4 [256]
- static const ulong32 T5 [256]
- static const ulong32 rc [ ]

  The round constants.

5.3.2 Define Documentation

5.3.2.1 #define BLOCKSIZE 128

Definition at line 41 of file anubis.c.

5.3.2.2 #define BLOCKSIZEB (BLOCKSIZE/8)

Definition at line 42 of file anubis.c.

5.3.2.3 #define MAX_KEYSIZEB (4*MAX_N)

Definition at line 40 of file anubis.c.

5.3.2.4 #define MAX_N 10

Definition at line 36 of file anubis.c.

Referenced by anubis_setup().

5.3.2.5 #define MAX_ROUNDS (8 + MAX_N)

Definition at line 38 of file anubis.c.

5.3.2.6 #define MIN_KEYSIZEB (4*MIN_N)

Definition at line 39 of file anubis.c.

5.3.2.7 #define MIN_N 4

Definition at line 35 of file anubis.c.

5.3.2.8 #define MIN_ROUNDS (8 + MIN_N)

Definition at line 37 of file anubis.c.

5.3.3 Function Documentation

5.3.3.1 static void anubis_crypt (const unsigned char * *plaintext*, unsigned char * *ciphertext*, ulong32 *roundKey*[18+1][4], int *R*) [static]

Definition at line 1039 of file anubis.c.

Referenced by anubis_ecb_decrypt(), and anubis_ecb_encrypt().

```
{
    int i, pos, r;
    ulong32 state[4];
    ulong32 inter[4];

     /*
     * map plaintext block to cipher state (mu)
     * and add initial round key (sigma[K^0]):
     */
     for (i = 0, pos = 0; i < 4; i++, pos += 4) {
       state[i] =
          (plaintext[pos    ] << 24) ^
          (plaintext[pos + 1] << 16) ^
          (plaintext[pos + 2] <<  8) ^
          (plaintext[pos + 3]      ) ^
          roundKey[0][i];
     }

     /*
      * R - 1 full rounds:
      */
     for (r = 1; r < R; r++) {
       inter[0] =
          T0[(state[0] >> 24) & 0xff] ^
          T1[(state[1] >> 24) & 0xff] ^
          T2[(state[2] >> 24) & 0xff] ^
          T3[(state[3] >> 24) & 0xff] ^
          roundKey[r][0];
       inter[1] =
          T0[(state[0] >> 16) & 0xff] ^
          T1[(state[1] >> 16) & 0xff] ^
          T2[(state[2] >> 16) & 0xff] ^
          T3[(state[3] >> 16) & 0xff] ^
          roundKey[r][1];
       inter[2] =
          T0[(state[0] >>  8) & 0xff] ^
          T1[(state[1] >>  8) & 0xff] ^
          T2[(state[2] >>  8) & 0xff] ^
          T3[(state[3] >>  8) & 0xff] ^
          roundKey[r][2];
       inter[3] =
          T0[(state[0]      ) & 0xff] ^
          T1[(state[1]      ) & 0xff] ^
          T2[(state[2]      ) & 0xff] ^
          T3[(state[3]      ) & 0xff] ^
          roundKey[r][3];
       state[0] = inter[0];
       state[1] = inter[1];
       state[2] = inter[2];
       state[3] = inter[3];
     }

     /*
     * last round:
     */
    inter[0] =
       (T0[(state[0] >> 24) & 0xff] & 0xff000000U) ^
```

```
      (T1[(state[1] >> 24) & 0xff] & 0x00ff0000U) ^
      (T2[(state[2] >> 24) & 0xff] & 0x0000ff00U) ^
      (T3[(state[3] >> 24) & 0xff] & 0x000000ffU) ^
      roundKey[R][0];
   inter[1] =
      (T0[(state[0] >> 16) & 0xff] & 0xff000000U) ^
      (T1[(state[1] >> 16) & 0xff] & 0x00ff0000U) ^
      (T2[(state[2] >> 16) & 0xff] & 0x0000ff00U) ^
      (T3[(state[3] >> 16) & 0xff] & 0x000000ffU) ^
      roundKey[R][1];
   inter[2] =
      (T0[(state[0] >>  8) & 0xff] & 0xff000000U) ^
      (T1[(state[1] >>  8) & 0xff] & 0x00ff0000U) ^
      (T2[(state[2] >>  8) & 0xff] & 0x0000ff00U) ^
      (T3[(state[3] >>  8) & 0xff] & 0x000000ffU) ^
      roundKey[R][2];
   inter[3] =
      (T0[(state[0]      ) & 0xff] & 0xff000000U) ^
      (T1[(state[1]      ) & 0xff] & 0x00ff0000U) ^
      (T2[(state[2]      ) & 0xff] & 0x0000ff00U) ^
      (T3[(state[3]      ) & 0xff] & 0x000000ffU) ^
      roundKey[R][3];

   /*
    * map cipher state to ciphertext block (mu^{-1}):
    */
    for (i = 0, pos = 0; i < 4; i++, pos += 4) {
        ulong32 w = inter[i];
        ciphertext[pos    ] = (unsigned char)(w >> 24);
        ciphertext[pos + 1] = (unsigned char)(w >> 16);
        ciphertext[pos + 2] = (unsigned char)(w >>  8);
        ciphertext[pos + 3] = (unsigned char)(w      );
    }
}
```

5.3.3.2 void anubis_done (symmetric_key * *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 1521 of file anubis.c.

```
{
}
```

5.3.3.3 int anubis_ecb_decrypt (const unsigned char * *ct*, unsigned char * *pt*, symmetric_key * *skey*)

Decrypts a block of text with Anubis.

Parameters:

ct The input ciphertext (16 bytes)

pt The output plaintext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 1155 of file anubis.c.

References anubis_crypt(), CRYPT_OK, and LTC_ARGCHK.

Referenced by anubis_test().

```
{
   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);
   anubis_crypt(ct, pt, skey->anubis.roundKeyDec, skey->anubis.R);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.3.3.4 int anubis_ecb_encrypt (const unsigned char * *pt*, unsigned char * *ct*, symmetric_key * *skey*)

Encrypts a block of text with Anubis.

Parameters:

pt The input plaintext (16 bytes)

ct The output ciphertext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 1139 of file anubis.c.

References anubis_crypt(), CRYPT_OK, and LTC_ARGCHK.

Referenced by anubis_test().

```
{
   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);
   anubis_crypt(pt, ct, skey->anubis.roundKeyEnc, skey->anubis.R);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.3.3.5 int anubis_keysize (int * *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 1530 of file anubis.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize >= 40) {
      *keysize = 40;
   } else if (*keysize >= 36) {
      *keysize = 36;
   } else if (*keysize >= 32) {
      *keysize = 32;
   } else if (*keysize >= 28) {
      *keysize = 28;
   } else if (*keysize >= 24) {
      *keysize = 24;
   } else if (*keysize >= 20) {
      *keysize = 20;
   } else if (*keysize >= 16) {
      *keysize = 16;
   } else {
      return CRYPT_INVALID_KEYSIZE;
   }
   return CRYPT_OK;
}
```

5.3.3.6 int anubis_setup (const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_key ∗ *skey*)

Initialize the Anubis block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 897 of file anubis.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, LTC_ARGCHK, MAX_N, N, and R.

Referenced by anubis_test().

```
{
   int N, R, i, pos, r;
   ulong32 kappa[MAX_N];
   ulong32 inter[MAX_N];
   ulong32 v, K0, K1, K2, K3;

   LTC_ARGCHK(key  != NULL);
   LTC_ARGCHK(skey != NULL);
```

```
    /* Valid sizes (in bytes) are 16, 20, 24, 28, 32, 36, and 40. */
    if ((keylen & 3) || (keylen < 16) || (keylen > 40)) {
       return CRYPT_INVALID_KEYSIZE;
    }
    skey->anubis.keyBits = keylen*8;

    /*
     * determine the N length parameter:
     * (N.B. it is assumed that the key length is valid!)
     */
    N = skey->anubis.keyBits >> 5;

    /*
     * determine number of rounds from key size:
     */
    skey->anubis.R = R = 8 + N;

    if (num_rounds != 0 && num_rounds != skey->anubis.R) {
       return CRYPT_INVALID_ROUNDS;
    }

     /*
     * map cipher key to initial key state (mu):
     */
     for (i = 0, pos = 0; i < N; i++, pos += 4) {
       kappa[i] =
          (key[pos    ] << 24) ^
          (key[pos + 1] << 16) ^
          (key[pos + 2] <<  8) ^
          (key[pos + 3]      );
     }

    /*
     * generate R + 1 round keys:
     */
    for (r = 0; r <= R; r++) {
       /*
        * generate r-th round key K^r:
        */
       K0 = T4[(kappa[N - 1] >> 24) & 0xff];
       K1 = T4[(kappa[N - 1] >> 16) & 0xff];
       K2 = T4[(kappa[N - 1] >>  8) & 0xff];
       K3 = T4[(kappa[N - 1]      ) & 0xff];
       for (i = N - 2; i >= 0; i--) {
          K0 = T4[(kappa[i] >> 24)  & 0xff] ^
             (T5[(K0 >> 24) & 0xff] & 0xff000000U) ^
             (T5[(K0 >> 16) & 0xff] & 0x00ff0000U) ^
             (T5[(K0 >>  8) & 0xff] & 0x0000ff00U) ^
             (T5[(K0      ) & 0xff] & 0x000000ffU);
          K1 = T4[(kappa[i] >> 16) & 0xff] ^
             (T5[(K1 >> 24) & 0xff] & 0xff000000U) ^
             (T5[(K1 >> 16) & 0xff] & 0x00ff0000U) ^
             (T5[(K1 >>  8) & 0xff] & 0x0000ff00U) ^
             (T5[(K1      ) & 0xff] & 0x000000ffU);
          K2 = T4[(kappa[i] >>  8) & 0xff] ^
             (T5[(K2 >> 24) & 0xff] & 0xff000000U) ^
             (T5[(K2 >> 16) & 0xff] & 0x00ff0000U) ^
             (T5[(K2 >>  8) & 0xff] & 0x0000ff00U) ^
             (T5[(K2      ) & 0xff] & 0x000000ffU);
          K3 = T4[(kappa[i]      ) & 0xff] ^
             (T5[(K3 >> 24) & 0xff] & 0xff000000U) ^
             (T5[(K3 >> 16) & 0xff] & 0x00ff0000U) ^
             (T5[(K3 >>  8) & 0xff] & 0x0000ff00U) ^
             (T5[(K3      ) & 0xff] & 0x000000ffU);
       }
       /*
```

```
     -- this is the code to use with the large U tables:
     K0 = K1 = K2 = K3 = 0;
     for (i = 0; i < N; i++) {
        K0 ^= U[i][(kappa[i] >> 24) & 0xff];
        K1 ^= U[i][(kappa[i] >> 16) & 0xff];
        K2 ^= U[i][(kappa[i] >>  8) & 0xff];
        K3 ^= U[i][(kappa[i]      ) & 0xff];
     }
     */
     skey->anubis.roundKeyEnc[r][0] = K0;
     skey->anubis.roundKeyEnc[r][1] = K1;
     skey->anubis.roundKeyEnc[r][2] = K2;
     skey->anubis.roundKeyEnc[r][3] = K3;

     /*
      * compute kappa^{r+1} from kappa^r:
      */
     if (r == R) {
        break;
     }
     for (i = 0; i < N; i++) {
        int j = i;
        inter[i]  = T0[(kappa[j--] >> 24) & 0xff]; if (j < 0) j = N - 1;
        inter[i] ^= T1[(kappa[j--] >> 16) & 0xff]; if (j < 0) j = N - 1;
        inter[i] ^= T2[(kappa[j--] >>  8) & 0xff]; if (j < 0) j = N - 1;
        inter[i] ^= T3[(kappa[j  ]      ) & 0xff];
      }
      kappa[0] = inter[0] ^ rc[r];
      for (i = 1; i < N; i++) {
         kappa[i] = inter[i];
      }
   }

   /*
    * generate inverse key schedule: K'^0 = K^R, K'^R = K^0, K'^r = theta(K^{R-r}):
    */
   for (i = 0; i < 4; i++) {
      skey->anubis.roundKeyDec[0][i] = skey->anubis.roundKeyEnc[R][i];
      skey->anubis.roundKeyDec[R][i] = skey->anubis.roundKeyEnc[0][i];
   }
   for (r = 1; r < R; r++) {
      for (i = 0; i < 4; i++) {
         v = skey->anubis.roundKeyEnc[R - r][i];
         skey->anubis.roundKeyDec[r][i] =
            T0[T4[(v >> 24) & 0xff] & 0xff] ^
            T1[T4[(v >> 16) & 0xff] & 0xff] ^
            T2[T4[(v >>  8) & 0xff] & 0xff] ^
            T3[T4[(v      ) & 0xff] & 0xff];
      }
   }

   return CRYPT_OK;
}
```

5.3.3.7 int anubis_test (void)

Performs a self-test of the Anubis block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 1168 of file anubis.c.

References anubis_ecb_decrypt(), anubis_ecb_encrypt(), anubis_setup(), CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, and XMEMCMP.

```
{
#if !defined(LTC_TEST)
  return CRYPT_NOP;
#else
  static const struct test {
     int keylen;
     unsigned char pt[16], ct[16], key[40];
  } tests[] = {
#ifndef ANUBIS_TWEAK
  /**** ORIGINAL ANUBIS ****/
  /* 128 bit keys */
{
   16,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0xF0, 0x68, 0x60, 0xFC, 0x67, 0x30, 0xE8, 0x18,
     0xF1, 0x32, 0xC7, 0x8A, 0xF4, 0x13, 0x2A, 0xFE },
   { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
   16,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0xA8, 0x66, 0x84, 0x80, 0x07, 0x74, 0x5C, 0x89,
     0xFC, 0x5E, 0xB5, 0xBA, 0xD4, 0xFE, 0x32, 0x6D },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
},

   /* 160-bit keys */
{
   20,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0xBD, 0x5E, 0x32, 0xBE, 0x51, 0x67, 0xA8, 0xE2,
     0x72, 0xD7, 0x95, 0x0F, 0x83, 0xC6, 0x8C, 0x31 },
   { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00 }
}, {
   20,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x4C, 0x1F, 0x86, 0x2E, 0x11, 0xEB, 0xCE, 0xEB,
     0xFE, 0xB9, 0x73, 0xC9, 0xDF, 0xEF, 0x7A, 0xDB },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x01 }
},

  /* 192-bit keys */
{
   24,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x17, 0xAC, 0x57, 0x44, 0x9D, 0x59, 0x61, 0x66,
     0xD0, 0xC7, 0x9E, 0x04, 0x7C, 0xC7, 0x58, 0xF0 },
   { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
   24,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
```

```
    { 0x71, 0x52, 0xB4, 0xEB, 0x1D, 0xAA, 0x36, 0xFD,
      0x57, 0x14, 0x5F, 0x57, 0x04, 0x9F, 0x70, 0x74 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
},

  /* 224-bit keys */
{
    28,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xA2, 0xF0, 0xA6, 0xB9, 0x17, 0x93, 0x2A, 0x3B,
      0xEF, 0x08, 0xE8, 0x7A, 0x58, 0xD6, 0xF8, 0x53 },
    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00 }
}, {
    28,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xF0, 0xCA, 0xFC, 0x78, 0x8B, 0x4B, 0x4E, 0x53,
      0x8B, 0xC4, 0x32, 0x6A, 0xF5, 0xB9, 0x1B, 0x5F },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x01 }
},

  /* 256-bit keys */
{
    32,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xE0, 0x86, 0xAC, 0x45, 0x6B, 0x3C, 0xE5, 0x13,
      0xED, 0xF5, 0xDF, 0xDD, 0xD6, 0x3B, 0x71, 0x93 },
    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
    32,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x50, 0x01, 0xB9, 0xF5, 0x21, 0xC1, 0xC1, 0x29,
      0x00, 0xD5, 0xEC, 0x98, 0x2B, 0x9E, 0xE8, 0x21 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
},

  /* 288-bit keys */
{
    36,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xE8, 0xF4, 0xAF, 0x2B, 0x21, 0xA0, 0x87, 0x9B,
      0x41, 0x95, 0xB9, 0x71, 0x75, 0x79, 0x04, 0x7C },
    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00 }
}, {
    36,
```

```
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xE6, 0xA6, 0xA5, 0xBC, 0x8B, 0x63, 0x6F, 0xE2,
      0xBD, 0xA7, 0xA7, 0x53, 0xAB, 0x40, 0x22, 0xE0 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x01 }
},

  /* 320-bit keys */
{
    40,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x17, 0x04, 0xD7, 0x2C, 0xC6, 0x85, 0x76, 0x02,
      0x4B, 0xCC, 0x39, 0x80, 0xD8, 0x22, 0xEA, 0xA4 },
    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
    40,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x7A, 0x41, 0xE6, 0x7D, 0x4F, 0xD8, 0x64, 0xF0,
      0x44, 0xA8, 0x3C, 0x73, 0x81, 0x7E, 0x53, 0xD8 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
}
#else
  /**** Tweaked ANUBIS ****/
  /* 128 bit keys */
{
    16,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xB8, 0x35, 0xBD, 0xC3, 0x34, 0x82, 0x9D, 0x83,
      0x71, 0xBF, 0xA3, 0x71, 0xE4, 0xB3, 0xC4, 0xFD },
    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
    16,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xE6, 0x14, 0x1E, 0xAF, 0xEB, 0xE0, 0x59, 0x3C,
      0x48, 0xE1, 0xCD, 0xF2, 0x1B, 0xBA, 0xA1, 0x89 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
},

    /* 160-bit keys */
{
    20,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x97, 0x59, 0x79, 0x4B, 0x5C, 0xA0, 0x70, 0x73,
      0x24, 0xEF, 0xB3, 0x58, 0x67, 0xCA, 0xD4, 0xB3 },
    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00 }
}, {
```

```
    20,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xB8, 0x0D, 0xFB, 0x9B, 0xE4, 0xA1, 0x58, 0x87,
      0xB3, 0x76, 0xD5, 0x02, 0x18, 0x95, 0xC1, 0x2E },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x01 }
},

  /* 192-bit keys */
{
    24,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x7D, 0x62, 0x3B, 0x52, 0xC7, 0x4C, 0x64, 0xD8,
      0xEB, 0xC7, 0x2D, 0x57, 0x97, 0x85, 0x43, 0x8F },
    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
    24,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xB1, 0x0A, 0x59, 0xDD, 0x5D, 0x5D, 0x8D, 0x67,
      0xEC, 0xEE, 0x4A, 0xC4, 0xBE, 0x4F, 0xA8, 0x4F },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
},

  /* 224-bit keys */
{
    28,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x68, 0x9E, 0x05, 0x94, 0x6A, 0x94, 0x43, 0x8F,
      0xE7, 0x8E, 0x37, 0x3D, 0x24, 0x97, 0x92, 0xF5 },
    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00 }
}, {
    28,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0xDD, 0xB7, 0xB0, 0xB4, 0xE9, 0xB4, 0x9B, 0x9C,
      0x38, 0x20, 0x25, 0x0B, 0x47, 0xC2, 0x1F, 0x89 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x01 }
},

  /* 256-bit keys */
{
    32,
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x96, 0x00, 0xF0, 0x76, 0x91, 0x69, 0x29, 0x87,
      0xF5, 0xE5, 0x97, 0xDB, 0xDB, 0xAF, 0x1B, 0x0A },
    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
    32,
```

```
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x69, 0x9C, 0xAF, 0xDD, 0x94, 0xC7, 0xBC, 0x60,
     0x44, 0xFE, 0x02, 0x05, 0x8A, 0x6E, 0xEF, 0xBD },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
},

  /* 288-bit keys */
{
   36,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x0F, 0xC7, 0xA2, 0xC0, 0x11, 0x17, 0xAC, 0x43,
     0x52, 0x5E, 0xDF, 0x6C, 0xF3, 0x96, 0x33, 0x6C },
   { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00 }
}, {
   36,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0xAD, 0x08, 0x4F, 0xED, 0x55, 0xA6, 0x94, 0x3E,
     0x7E, 0x5E, 0xED, 0x05, 0xA1, 0x9D, 0x41, 0xB4 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x01 }
},

  /* 320-bit keys */
{
   40,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0xFE, 0xE2, 0x0E, 0x2A, 0x9D, 0xC5, 0x83, 0xBA,
     0xA3, 0xA6, 0xD6, 0xA6, 0xF2, 0xE8, 0x06, 0xA5 },
   { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
   40,
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x86, 0x3D, 0xCC, 0x4A, 0x60, 0x34, 0x9C, 0x28,
     0xA7, 0xDA, 0xA4, 0x3B, 0x0A, 0xD7, 0xFD, 0xC7 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
}
#endif
};
   int x, y;
   unsigned char buf[2][16];
   symmetric_key skey;

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       anubis_setup(tests[x].key, tests[x].keylen, 0, &skey);
```

```
       anubis_ecb_encrypt(tests[x].pt, buf[0], &skey);
       anubis_ecb_decrypt(buf[0], buf[1], &skey);
       if (XMEMCMP(buf[0], tests[x].ct, 16) || XMEMCMP(buf[1], tests[x].pt, 16)) {
          return CRYPT_FAIL_TESTVECTOR;
       }

       for (y = 0; y < 1000; y++) anubis_ecb_encrypt(buf[0], buf[0], &skey);
       for (y = 0; y < 1000; y++) anubis_ecb_decrypt(buf[0], buf[0], &skey);
       if (XMEMCMP(buf[0], tests[x].ct, 16)) {
          return CRYPT_FAIL_TESTVECTOR;
       }

   }
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.3.4 Variable Documentation

5.3.4.1 const struct ltc_cipher_descriptor anubis_desc

Initial value:

```
 {
   "anubis",
   19,
   16, 40, 16, 12,
   &anubis_setup,
   &anubis_ecb_encrypt,
   &anubis_ecb_decrypt,
   &anubis_test,
   &anubis_done,
   &anubis_keysize,
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 22 of file anubis.c.

Referenced by yarrow_start().

5.3.4.2 const ulong32 rc[ ] `[static]`

Initial value:

```
 {
    0xba542f74U, 0x53d3d24dU, 0x50ac8dbfU, 0x70529a4cU,
    0xead597d1U, 0x33515ba6U, 0xde48a899U, 0xdb32b7fcU,
    0xe39e919bU, 0xe2bb416eU, 0xa5cb6b95U, 0xa1f3b102U,
    0xccc41d14U, 0xc363da5dU, 0x5fdc7dcdU, 0x7f5a6c5cU,
    0xf726ffedU, 0xe89d6f8eU, 0x19a0f089U,
}
```

The round constants.

Definition at line 458 of file anubis.c.

5.3.4.3 const ulong32 T0[256] [static]

Definition at line 53 of file anubis.c.

Referenced by khazad_crypt(), and khazad_setup().

5.3.4.4 const ulong32 T1[256] [static]

Definition at line 120 of file anubis.c.

Referenced by khazad_crypt(), and khazad_setup().

5.3.4.5 const ulong32 T2[256] [static]

Definition at line 187 of file anubis.c.

Referenced by khazad_crypt(), khazad_setup(), and rounds().

5.3.4.6 const ulong32 T3[256] [static]

Definition at line 254 of file anubis.c.

Referenced by khazad_crypt(), and khazad_setup().

5.3.4.7 const ulong32 T4[256] [static]

Definition at line 321 of file anubis.c.

Referenced by khazad_crypt(), and khazad_setup().

5.3.4.8 const ulong32 T5[256] [static]

Definition at line 388 of file anubis.c.

Referenced by khazad_crypt(), and khazad_setup().

5.4 ciphers/blowfish.c File Reference

5.4.1 Detailed Description

Implementation of the Blowfish block cipher, Tom St Denis.

Definition in file blowfish.c.

`#include "tomcrypt.h"`

Include dependency graph for blowfish.c:

Defines

- #define F(x) ((S1[byte(x,3)] + S2[byte(x,2)]) ^ S3[byte(x,1)]) + S4[byte(x,0)]

Functions

- int blowfish_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the Blowfish block cipher.

- int blowfish_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with Blowfish.

- int blowfish_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with Blowfish.

- int blowfish_test (void)

  Performs a self-test of the Blowfish block cipher.

- void blowfish_done (symmetric_key ∗skey)

  Terminate the context.

- int blowfish_keysize (int ∗keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor blowfish_desc
- static const ulong32 ORIG_P [16+2]
- static const ulong32 ORIG_S [4][256]

5.4.2 Define Documentation

5.4.2.1 #define F(x) ((S1[byte(x,3)] + S2[byte(x,2)]) ^ S3[byte(x,1)]) + S4[byte(x,0)]

Definition at line 378 of file blowfish.c.

Referenced by blowfish_ecb_decrypt(), blowfish_ecb_encrypt(), and rounds().

5.4.3 Function Documentation

5.4.3.1 void blowfish_done (symmetric_key ∗ *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 568 of file blowfish.c.

```
569 {
570 }
```

5.4.3.2 int blowfish_ecb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, symmetric_key ∗ *skey*)

Decrypts a block of text with Blowfish.

Parameters:

ct The input ciphertext (8 bytes)

pt The output plaintext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 455 of file blowfish.c.

References F, LTC_ARGCHK, R, S1, S2, S3, and S4.

```
457 {
458    ulong32 L, R;
459    int r;
460 #ifndef __GNUC__
461    ulong32 *S1, *S2, *S3, *S4;
462 #endif
463 
464     LTC_ARGCHK(pt   != NULL);
465     LTC_ARGCHK(ct   != NULL);
466     LTC_ARGCHK(skey != NULL);
467 
468 #ifndef __GNUC__
469     S1 = skey->blowfish.S[0];
470     S2 = skey->blowfish.S[1];
471     S3 = skey->blowfish.S[2];
472     S4 = skey->blowfish.S[3];
473 #endif
474 
475    /* load it */
476    LOAD32H(R, &ct[0]);
477    LOAD32H(L, &ct[4]);
478 
479    /* undo last keying */
480    R ^= skey->blowfish.K[17];
481    L ^= skey->blowfish.K[16];
482 
```

```
   /* do 16 rounds */
   for (r = 15; r > 0; ) {
      L ^= F(R); R ^= skey->blowfish.K[r--];
      R ^= F(L); L ^= skey->blowfish.K[r--];
      L ^= F(R); R ^= skey->blowfish.K[r--];
      R ^= F(L); L ^= skey->blowfish.K[r--];
   }

   /* store */
   STORE32H(L, &pt[0]);
   STORE32H(R, &pt[4]);
   return CRYPT_OK;
}
```

5.4.3.3 int blowfish_ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with Blowfish.

Parameters:

pt The input plaintext (8 bytes)

ct The output ciphertext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 393 of file blowfish.c.

References F, LTC_ARGCHK, R, S1, S2, S3, and S4.

```
{
   ulong32 L, R;
   int r;
#ifndef __GNUC__
   ulong32 *S1, *S2, *S3, *S4;
#endif

    LTC_ARGCHK(pt   != NULL);
    LTC_ARGCHK(ct   != NULL);
    LTC_ARGCHK(skey != NULL);

#ifndef __GNUC__
    S1 = skey->blowfish.S[0];
    S2 = skey->blowfish.S[1];
    S3 = skey->blowfish.S[2];
    S4 = skey->blowfish.S[3];
#endif

   /* load it */
   LOAD32H(L, &pt[0]);
   LOAD32H(R, &pt[4]);

   /* do 16 rounds */
   for (r = 0; r < 16; ) {
      L ^= skey->blowfish.K[r++];  R ^= F(L);
      R ^= skey->blowfish.K[r++];  L ^= F(R);
      L ^= skey->blowfish.K[r++];  R ^= F(L);
      R ^= skey->blowfish.K[r++];  L ^= F(R);
   }
```

```

   /* last keying */
   R ^= skey->blowfish.K[17];
   L ^= skey->blowfish.K[16];

   /* store */
   STORE32H(R, &ct[0]);
   STORE32H(L, &ct[4]);

   return CRYPT_OK;
}
```

5.4.3.4 int blowfish_keysize (int * *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 577 of file blowfish.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);

   if (*keysize < 8) {
      return CRYPT_INVALID_KEYSIZE;
   } else if (*keysize > 56) {
      *keysize = 56;
   }
   return CRYPT_OK;
}
```

5.4.3.5 int blowfish_setup (const unsigned char * *key*, int *keylen*, int *num_rounds*, symmetric_key * *skey*)

Initialize the Blowfish block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 308 of file blowfish.c.

References B, CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, and LTC_ARGCHK.

Referenced by blowfish_test().

```
{
   ulong32 x, y, z, A;
   unsigned char B[8];

   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(skey != NULL);

   /* check key length */
   if (keylen < 8 || keylen > 56) {
      return CRYPT_INVALID_KEYSIZE;
   }

   /* check rounds */
   if (num_rounds != 0 && num_rounds != 16) {
      return CRYPT_INVALID_ROUNDS;
   }

   /* load in key bytes (Supplied by David Hopwood) */
   for (x = y = 0; x < 18; x++) {
       A = 0;
       for (z = 0; z < 4; z++) {
           A = (A << 8) | ((ulong32)key[y++] & 255);
           if (y == (ulong32)keylen) {
              y = 0;
           }
       }
       skey->blowfish.K[x] = ORIG_P[x] ^ A;
   }

   /* copy sboxes */
   for (x = 0; x < 4; x++) {
       for (y = 0; y < 256; y++) {
           skey->blowfish.S[x][y] = ORIG_S[x][y];
       }
   }

   /* encrypt K array */
   for (x = 0; x < 8; x++) {
       B[x] = 0;
   }

   for (x = 0; x < 18; x += 2) {
       /* encrypt it */
       blowfish_ecb_encrypt(B, B, skey);
       /* copy it */
       LOAD32H(skey->blowfish.K[x], &B[0]);
       LOAD32H(skey->blowfish.K[x+1], &B[4]);
   }

   /* encrypt S array */
   for (x = 0; x < 4; x++) {
       for (y = 0; y < 256; y += 2) {
          /* encrypt it */
          blowfish_ecb_encrypt(B, B, skey);
          /* copy it */
          LOAD32H(skey->blowfish.S[x][y], &B[0]);
          LOAD32H(skey->blowfish.S[x][y+1], &B[4]);
       }
   }

#ifdef LTC_CLEAN_STACK
   zeromem(B, sizeof(B));
#endif

   return CRYPT_OK;
}
```

5.4.3.6 int blowfish_test (void)

Performs a self-test of the Blowfish block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 511 of file blowfish.c.

References blowfish_setup(), CRYPT_NOP, and CRYPT_OK.

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   int err;
   symmetric_key key;
   static const struct {
          unsigned char key[8], pt[8], ct[8];
   } tests[] = {
       {
           { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
           { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
           { 0x4E, 0xF9, 0x97, 0x45, 0x61, 0x98, 0xDD, 0x78}
       },
       {
           { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
           { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
           { 0x51, 0x86, 0x6F, 0xD5, 0xB8, 0x5E, 0xCB, 0x8A}
       },
       {
           { 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00},
           { 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01},
           { 0x7D, 0x85, 0x6F, 0x9A, 0x61, 0x30, 0x63, 0xF2}
       }
   };
   unsigned char tmp[2][8];
   int x, y;

   for (x = 0; x < (int)(sizeof(tests) / sizeof(tests[0])); x++) {
      /* setup key */
      if ((err = blowfish_setup(tests[x].key, 8, 16, &key)) != CRYPT_OK) {
         return err;
      }

      /* encrypt and decrypt */
      blowfish_ecb_encrypt(tests[x].pt, tmp[0], &key);
      blowfish_ecb_decrypt(tmp[0], tmp[1], &key);

      /* compare */
      if ((XMEMCMP(tmp[0], tests[x].ct, 8) != 0) || (XMEMCMP(tmp[1], tests[x].pt, 8) != 0)) {
         return CRYPT_FAIL_TESTVECTOR;
      }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 8; y++) tmp[0][y] = 0;
      for (y = 0; y < 1000; y++) blowfish_ecb_encrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 1000; y++) blowfish_ecb_decrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 8; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
   }
   return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.4.4 Variable Documentation

5.4.4.1 const struct ltc_cipher_descriptor blowfish_desc

Initial value:

```
{
    "blowfish",
    0,
    8, 56, 8, 16,
    &blowfish_setup,
    &blowfish_ecb_encrypt,
    &blowfish_ecb_decrypt,
    &blowfish_test,
    &blowfish_done,
    &blowfish_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 19 of file blowfish.c.

Referenced by yarrow_start().

5.4.4.2 const ulong32 ORIG_P[16+2] [static]

Initial value:

```
 {
        0x243F6A88UL, 0x85A308D3UL, 0x13198A2EUL, 0x03707344UL,
        0xA4093822UL, 0x299F31D0UL, 0x082EFA98UL, 0xEC4E6C89UL,
        0x452821E6UL, 0x38D01377UL, 0xBE5466CFUL, 0x34E90C6CUL,
        0xC0AC29B7UL, 0xC97C50DDUL, 0x3F84D5B5UL, 0xB5470917UL,
        0x9216D5D9UL, 0x8979FB1BUL
}
```

Definition at line 33 of file blowfish.c.

5.4.4.3 const ulong32 ORIG_S[4][256] [static]

Definition at line 41 of file blowfish.c.

5.5 ciphers/cast5.c File Reference

5.5.1 Detailed Description

Implementation of CAST5 (RFC 2144) by Tom St Denis.

Definition in file cast5.c.

```
#include "tomcrypt.h"
```

Include dependency graph for cast5.c:

Defines

- #define GB(x, i) (((x[(15-i)>>2])>>(unsigned)(8*((15-i)&3)))&255)
- #define INLINE

Functions

- int cast5_setup (const unsigned char *key, int keylen, int num_rounds, symmetric_key *skey)

  Initialize the CAST5 block cipher.
- static INLINE ulong32 FI (ulong32 R, ulong32 Km, ulong32 Kr)
- static INLINE ulong32 FII (ulong32 R, ulong32 Km, ulong32 Kr)
- static INLINE ulong32 FIII (ulong32 R, ulong32 Km, ulong32 Kr)
- int cast5_ecb_encrypt (const unsigned char *pt, unsigned char *ct, symmetric_key *skey)

  Encrypts a block of text with CAST5.
- int cast5_ecb_decrypt (const unsigned char *ct, unsigned char *pt, symmetric_key *skey)

  Decrypts a block of text with CAST5.
- int cast5_test (void)

  Performs a self-test of the CAST5 block cipher.
- void cast5_done (symmetric_key *skey)

  Terminate the context.
- int cast5_keysize (int *keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor cast5_desc
- static const ulong32 S1 [256]
- static const ulong32 S2 [256]
- static const ulong32 S3 [256]
- static const ulong32 S4 [256]
- static const ulong32 S5 [256]
- static const ulong32 S6 [256]
- static const ulong32 S7 [256]
- static const ulong32 S8 [256]

5.5.2 Define Documentation

5.5.2.1 #define GB(x, i) (((x[(15-i)>>2])>>(unsigned)(8*((15-i)&3)))&255)

Definition at line 397 of file cast5.c.

5.5.2.2 #define INLINE

Definition at line 505 of file cast5.c.

5.5.3 Function Documentation

5.5.3.1 void cast5_done (symmetric_key * *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 696 of file cast5.c.

```
{
}
```

5.5.3.2 int cast5_ecb_decrypt (const unsigned char * *ct*, unsigned char * *pt*, symmetric_key * *skey*)

Decrypts a block of text with CAST5.

Parameters:

ct The input ciphertext (8 bytes)

pt The output plaintext (8 bytes)

skey The key as scheduled

Definition at line 594 of file cast5.c.

References CRYPT_OK, FI(), FII(), FIII(), LTC_ARGCHK, and R.

```
{
   ulong32 R, L;

   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);

   LOAD32H(R,&ct[0]);
   LOAD32H(L,&ct[4]);
   if (skey->cast5.keylen > 10) {
      R ^= FI(L, skey->cast5.K[15], skey->cast5.K[31]);
      L ^= FIII(R, skey->cast5.K[14], skey->cast5.K[30]);
      R ^= FII(L, skey->cast5.K[13], skey->cast5.K[29]);
      L ^= FI(R, skey->cast5.K[12], skey->cast5.K[28]);
   }
   R ^= FIII(L, skey->cast5.K[11], skey->cast5.K[27]);
```

```
612    L ^= FII(R, skey->cast5.K[10], skey->cast5.K[26]);
613    R ^= FI(L, skey->cast5.K[9], skey->cast5.K[25]);
614    L ^= FIII(R, skey->cast5.K[8], skey->cast5.K[24]);
615    R ^= FII(L, skey->cast5.K[7], skey->cast5.K[23]);
616    L ^= FI(R, skey->cast5.K[6], skey->cast5.K[22]);
617    R ^= FIII(L, skey->cast5.K[5], skey->cast5.K[21]);
618    L ^= FII(R, skey->cast5.K[4], skey->cast5.K[20]);
619    R ^= FI(L, skey->cast5.K[3], skey->cast5.K[19]);
620    L ^= FIII(R, skey->cast5.K[2], skey->cast5.K[18]);
621    R ^= FII(L, skey->cast5.K[1], skey->cast5.K[17]);
622    L ^= FI(R, skey->cast5.K[0], skey->cast5.K[16]);
623    STORE32H(L,&pt[0]);
624    STORE32H(R,&pt[4]);
625
626    return CRYPT_OK;
627 }
```

Here is the call graph for this function:

5.5.3.3 int cast5_ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with CAST5.

Parameters:

pt The input plaintext (8 bytes)

ct The output ciphertext (8 bytes)

skey The key as scheduled

Definition at line 541 of file cast5.c.

References CRYPT_OK, FI(), FII(), FIII(), LTC_ARGCHK, and R.

```
543 {
544    ulong32 R, L;
545
546    LTC_ARGCHK(pt   != NULL);
547    LTC_ARGCHK(ct   != NULL);
548    LTC_ARGCHK(skey != NULL);
549
550    LOAD32H(L,&pt[0]);
551    LOAD32H(R,&pt[4]);
552    L ^= FI(R, skey->cast5.K[0], skey->cast5.K[16]);
553    R ^= FII(L, skey->cast5.K[1], skey->cast5.K[17]);
554    L ^= FIII(R, skey->cast5.K[2], skey->cast5.K[18]);
555    R ^= FI(L, skey->cast5.K[3], skey->cast5.K[19]);
556    L ^= FII(R, skey->cast5.K[4], skey->cast5.K[20]);
557    R ^= FIII(L, skey->cast5.K[5], skey->cast5.K[21]);
558    L ^= FI(R, skey->cast5.K[6], skey->cast5.K[22]);
559    R ^= FII(L, skey->cast5.K[7], skey->cast5.K[23]);
560    L ^= FIII(R, skey->cast5.K[8], skey->cast5.K[24]);
561    R ^= FI(L, skey->cast5.K[9], skey->cast5.K[25]);
562    L ^= FII(R, skey->cast5.K[10], skey->cast5.K[26]);
563    R ^= FIII(L, skey->cast5.K[11], skey->cast5.K[27]);
564    if (skey->cast5.keylen > 10) {
565       L ^= FI(R, skey->cast5.K[12], skey->cast5.K[28]);
566       R ^= FII(L, skey->cast5.K[13], skey->cast5.K[29]);
567       L ^= FIII(R, skey->cast5.K[14], skey->cast5.K[30]);
568       R ^= FI(L, skey->cast5.K[15], skey->cast5.K[31]);
569    }
570    STORE32H(R,&ct[0]);
571    STORE32H(L,&ct[4]);
572    return CRYPT_OK;
573 }
```

Here is the call graph for this function:

5.5.3.4 int cast5_keysize (int * *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 705 of file cast5.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize < 5) {
      return CRYPT_INVALID_KEYSIZE;
   } else if (*keysize > 16) {
      *keysize = 16;
   }
   return CRYPT_OK;
}
```

5.5.3.5 int cast5_setup (const unsigned char * *key*, int *keylen*, int *num_rounds*, symmetric_key * *skey*)

Initialize the CAST5 block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 411 of file cast5.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, LTC_ARGCHK, XMEMCPY, and zeromem().

Referenced by cast5_test().

```
{
   ulong32 x[4], z[4];
   unsigned char buf[16];
   int y, i;

   LTC_ARGCHK(key != NULL);
```

```
   LTC_ARGCHK(skey != NULL);

   if (num_rounds != 12 && num_rounds != 16 && num_rounds != 0) {
      return CRYPT_INVALID_ROUNDS;
   }

   if (num_rounds == 12 && keylen > 10) {
      return CRYPT_INVALID_ROUNDS;
   }

   if (keylen < 5 || keylen > 16) {
      return CRYPT_INVALID_KEYSIZE;
   }

   /* extend the key as required */
   zeromem(buf, sizeof(buf));
   XMEMCPY(buf, key, (size_t)keylen);

   /* load and start the awful looking network */
   for (y = 0; y < 4; y++) {
       LOAD32H(x[3-y],buf+4*y);
   }

   for (i = y = 0; y < 2; y++) {
        z[3] = x[3] ^ S5[GB(x, 0xD)] ^ S6[GB(x, 0xF)] ^ S7[GB(x, 0xC)] ^ S8[GB(x, 0xE)] ^ S7[GB(x, 0x8
        z[2] = x[1] ^ S5[GB(z, 0x0)] ^ S6[GB(z, 0x2)] ^ S7[GB(z, 0x1)] ^ S8[GB(z, 0x3)] ^ S8[GB(x, 0xA
        z[1] = x[0] ^ S5[GB(z, 0x7)] ^ S6[GB(z, 0x6)] ^ S7[GB(z, 0x5)] ^ S8[GB(z, 0x4)] ^ S5[GB(x, 0x9
        z[0] = x[2] ^ S5[GB(z, 0xA)] ^ S6[GB(z, 0x9)] ^ S7[GB(z, 0xb)] ^ S8[GB(z, 0x8)] ^ S6[GB(x, 0xB
        skey->cast5.K[i++] = S5[GB(z, 0x8)] ^ S6[GB(z, 0x9)] ^ S7[GB(z, 0x7)] ^ S8[GB(z, 0x6)] ^ S5[GB
        skey->cast5.K[i++] = S5[GB(z, 0xA)] ^ S6[GB(z, 0xB)] ^ S7[GB(z, 0x5)] ^ S8[GB(z, 0x4)] ^ S6[GB
        skey->cast5.K[i++] = S5[GB(z, 0xC)] ^ S6[GB(z, 0xd)] ^ S7[GB(z, 0x3)] ^ S8[GB(z, 0x2)] ^ S7[GB
        skey->cast5.K[i++] = S5[GB(z, 0xE)] ^ S6[GB(z, 0xF)] ^ S7[GB(z, 0x1)] ^ S8[GB(z, 0x0)] ^ S8[GB

        x[3] = z[1] ^ S5[GB(z, 0x5)] ^ S6[GB(z, 0x7)] ^ S7[GB(z, 0x4)] ^ S8[GB(z, 0x6)] ^ S7[GB(z, 0x0
        x[2] = z[3] ^ S5[GB(x, 0x0)] ^ S6[GB(x, 0x2)] ^ S7[GB(x, 0x1)] ^ S8[GB(x, 0x3)] ^ S8[GB(z, 0x2
        x[1] = z[2] ^ S5[GB(x, 0x7)] ^ S6[GB(x, 0x6)] ^ S7[GB(x, 0x5)] ^ S8[GB(x, 0x4)] ^ S5[GB(z, 0x1
        x[0] = z[0] ^ S5[GB(x, 0xA)] ^ S6[GB(x, 0x9)] ^ S7[GB(x, 0xb)] ^ S8[GB(x, 0x8)] ^ S6[GB(z, 0x3
        skey->cast5.K[i++] = S5[GB(x, 0x3)] ^ S6[GB(x, 0x2)] ^ S7[GB(x, 0xc)] ^ S8[GB(x, 0xd)] ^ S5[GB
        skey->cast5.K[i++] = S5[GB(x, 0x1)] ^ S6[GB(x, 0x0)] ^ S7[GB(x, 0xe)] ^ S8[GB(x, 0xf)] ^ S6[GB
        skey->cast5.K[i++] = S5[GB(x, 0x7)] ^ S6[GB(x, 0x6)] ^ S7[GB(x, 0x8)] ^ S8[GB(x, 0x9)] ^ S7[GB
        skey->cast5.K[i++] = S5[GB(x, 0x5)] ^ S6[GB(x, 0x4)] ^ S7[GB(x, 0xa)] ^ S8[GB(x, 0xb)] ^ S8[GB

        /* second half */
        z[3] = x[3] ^ S5[GB(x, 0xD)] ^ S6[GB(x, 0xF)] ^ S7[GB(x, 0xC)] ^ S8[GB(x, 0xE)] ^ S7[GB(x, 0x8
        z[2] = x[1] ^ S5[GB(z, 0x0)] ^ S6[GB(z, 0x2)] ^ S7[GB(z, 0x1)] ^ S8[GB(z, 0x3)] ^ S8[GB(x, 0xA
        z[1] = x[0] ^ S5[GB(z, 0x7)] ^ S6[GB(z, 0x6)] ^ S7[GB(z, 0x5)] ^ S8[GB(z, 0x4)] ^ S5[GB(x, 0x9
        z[0] = x[2] ^ S5[GB(z, 0xA)] ^ S6[GB(z, 0x9)] ^ S7[GB(z, 0xb)] ^ S8[GB(z, 0x8)] ^ S6[GB(x, 0xB
        skey->cast5.K[i++] = S5[GB(z, 0x3)] ^ S6[GB(z, 0x2)] ^ S7[GB(z, 0xc)] ^ S8[GB(z, 0xd)] ^ S5[GB
        skey->cast5.K[i++] = S5[GB(z, 0x1)] ^ S6[GB(z, 0x0)] ^ S7[GB(z, 0xe)] ^ S8[GB(z, 0xf)] ^ S6[GB
        skey->cast5.K[i++] = S5[GB(z, 0x7)] ^ S6[GB(z, 0x6)] ^ S7[GB(z, 0x8)] ^ S8[GB(z, 0x9)] ^ S7[GB
        skey->cast5.K[i++] = S5[GB(z, 0x5)] ^ S6[GB(z, 0x4)] ^ S7[GB(z, 0xa)] ^ S8[GB(z, 0xb)] ^ S8[GB

        x[3] = z[1] ^ S5[GB(z, 0x5)] ^ S6[GB(z, 0x7)] ^ S7[GB(z, 0x4)] ^ S8[GB(z, 0x6)] ^ S7[GB(z, 0x0
        x[2] = z[3] ^ S5[GB(x, 0x0)] ^ S6[GB(x, 0x2)] ^ S7[GB(x, 0x1)] ^ S8[GB(x, 0x3)] ^ S8[GB(z, 0x2
        x[1] = z[2] ^ S5[GB(x, 0x7)] ^ S6[GB(x, 0x6)] ^ S7[GB(x, 0x5)] ^ S8[GB(x, 0x4)] ^ S5[GB(z, 0x1
        x[0] = z[0] ^ S5[GB(x, 0xA)] ^ S6[GB(x, 0x9)] ^ S7[GB(x, 0xb)] ^ S8[GB(x, 0x8)] ^ S6[GB(z, 0x3
        skey->cast5.K[i++] = S5[GB(x, 0x8)] ^ S6[GB(x, 0x9)] ^ S7[GB(x, 0x7)] ^ S8[GB(x, 0x6)] ^ S5[GB
        skey->cast5.K[i++] = S5[GB(x, 0xa)] ^ S6[GB(x, 0xb)] ^ S7[GB(x, 0x5)] ^ S8[GB(x, 0x4)] ^ S6[GB
        skey->cast5.K[i++] = S5[GB(x, 0xc)] ^ S6[GB(x, 0xd)] ^ S7[GB(x, 0x3)] ^ S8[GB(x, 0x2)] ^ S7[GB
        skey->cast5.K[i++] = S5[GB(x, 0xe)] ^ S6[GB(x, 0xf)] ^ S7[GB(x, 0x1)] ^ S8[GB(x, 0x0)] ^ S8[GB
   }

   skey->cast5.keylen = keylen;

#ifdef LTC_CLEAN_STACK
   zeromem(buf, sizeof(buf));
   zeromem(x, sizeof(x));
```

```
   zeromem(z, sizeof(z));
#endif

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.5.3.6 int cast5_test (void)

Performs a self-test of the CAST5 block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 642 of file cast5.c.

References cast5_setup(), CRYPT_NOP, and CRYPT_OK.

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   static const struct {
       int keylen;
       unsigned char key[16];
       unsigned char pt[8];
       unsigned char ct[8];
   } tests[] = {
     { 16,
       {0x01, 0x23, 0x45, 0x67, 0x12, 0x34, 0x56, 0x78, 0x23, 0x45, 0x67, 0x89, 0x34, 0x56, 0x78, 0x9A
       {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF},
       {0x23, 0x8B, 0x4F, 0xE5, 0x84, 0x7E, 0x44, 0xB2}
     },
     { 10,
       {0x01, 0x23, 0x45, 0x67, 0x12, 0x34, 0x56, 0x78, 0x23, 0x45, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
       {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF},
       {0xEB, 0x6A, 0x71, 0x1A, 0x2C, 0x02, 0x27, 0x1B},
     },
     { 5,
       {0x01, 0x23, 0x45, 0x67, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
       {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF},
       {0x7A, 0xC8, 0x16, 0xD1, 0x6E, 0x9B, 0x30, 0x2E}
     }
   };
   int i, y, err;
   symmetric_key key;
   unsigned char tmp[2][8];

   for (i = 0; i < (int)(sizeof(tests) / sizeof(tests[0])); i++) {
       if ((err = cast5_setup(tests[i].key, tests[i].keylen, 0, &key)) != CRYPT_OK) {
          return err;
       }
       cast5_ecb_encrypt(tests[i].pt, tmp[0], &key);
       cast5_ecb_decrypt(tmp[0], tmp[1], &key);
       if ((XMEMCMP(tmp[0], tests[i].ct, 8) != 0) || (XMEMCMP(tmp[1], tests[i].pt, 8) != 0)) {
          return CRYPT_FAIL_TESTVECTOR;
       }
      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 8; y++) tmp[0][y] = 0;
      for (y = 0; y < 1000; y++) cast5_ecb_encrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 1000; y++) cast5_ecb_decrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 8; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
```

```

   }
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.5.3.7 static INLINE ulong32 FI (ulong32 *R*, ulong32 *Km*, ulong32 *Kr*) [static]

Definition at line 508 of file cast5.c.

References byte, I, ROL, S1, S2, S3, and S4.

Referenced by cast5_ecb_decrypt(), cast5_ecb_encrypt(), and FO().

```
{
   ulong32 I;
   I = (Km + R);
   I = ROL(I, Kr);
   return ((S1[byte(I, 3)] ^ S2[byte(I,2)]) - S3[byte(I,1)]) + S4[byte(I,0)];
}
```

5.5.3.8 static INLINE ulong32 FII (ulong32 *R*, ulong32 *Km*, ulong32 *Kr*) [static]

Definition at line 516 of file cast5.c.

References byte, I, ROL, S1, S2, S3, and S4.

Referenced by cast5_ecb_decrypt(), and cast5_ecb_encrypt().

```
{
   ulong32 I;
   I = (Km ^ R);
   I = ROL(I, Kr);
   return ((S1[byte(I, 3)] - S2[byte(I,2)]) + S3[byte(I,1)]) ^ S4[byte(I,0)];
}
```

5.5.3.9 static INLINE ulong32 FIII (ulong32 *R*, ulong32 *Km*, ulong32 *Kr*) [static]

Definition at line 524 of file cast5.c.

References byte, I, ROL, S1, S2, S3, and S4.

Referenced by cast5_ecb_decrypt(), and cast5_ecb_encrypt().

```
{
   ulong32 I;
   I = (Km - R);
   I = ROL(I, Kr);
   return ((S1[byte(I, 3)] + S2[byte(I,2)]) ^ S3[byte(I,1)]) - S4[byte(I,0)];
}
```

5.5.4 Variable Documentation

5.5.4.1 const struct ltc_cipher_descriptor cast5_desc

Initial value:

```
 {
   "cast5",
   15,
   5, 16, 8, 16,
   &cast5_setup,
   &cast5_ecb_encrypt,
   &cast5_ecb_decrypt,
   &cast5_test,
   &cast5_done,
   &cast5_keysize,
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 20 of file cast5.c.

Referenced by yarrow_start().

5.5.4.2 const ulong32 S1[256] `[static]`

Definition at line 33 of file cast5.c.

Referenced by blowfish_ecb_decrypt(), blowfish_ecb_encrypt(), FI(), FII(), FIII(), twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.5.4.3 const ulong32 S2[256] `[static]`

Definition at line 78 of file cast5.c.

Referenced by blowfish_ecb_decrypt(), blowfish_ecb_encrypt(), FI(), FII(), FIII(), twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.5.4.4 const ulong32 S3[256] `[static]`

Definition at line 123 of file cast5.c.

Referenced by blowfish_ecb_decrypt(), blowfish_ecb_encrypt(), FI(), FII(), FIII(), twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.5.4.5 const ulong32 S4[256] `[static]`

Definition at line 168 of file cast5.c.

Referenced by blowfish_ecb_decrypt(), blowfish_ecb_encrypt(), FI(), FII(), FIII(), twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.5.4.6 const ulong32 S5[256] `[static]`

Definition at line 213 of file cast5.c.

5.5.4.7 const ulong32 S6[256] `[static]`

Definition at line 258 of file cast5.c.

5.5.4.8 const ulong32 S7[256] [static]

Definition at line 303 of file cast5.c.

Referenced by FI().

5.5.4.9 const ulong32 S8[256] [static]

Definition at line 348 of file cast5.c.

5.6 ciphers/des.c File Reference

5.6.1 Detailed Description

DES code submitted by Dobes Vandermeer.

Definition in file des.c.

```
#include "tomcrypt.h"
```

Include dependency graph for des.c:

Defines

- #define EN0 0
- #define DE1 1

Functions

- static void cookey (const ulong32 ∗raw1, ulong32 ∗keyout)
- static void deskey (const unsigned char ∗key, short edf, ulong32 ∗keyout)
- static void desfunc (ulong32 ∗block, const ulong32 ∗keys)
- int des_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the DES block cipher.

- int des3_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the 3DES-EDE block cipher.

- int des_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with DES.

- int des_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with DES.

- int des3_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with 3DES-EDE.

- int des3_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with 3DES-EDE.

- int des_test (void)

  Performs a self-test of the DES block cipher.

- int des3_test (void)
- void des_done (symmetric_key ∗skey)

  Terminate the context.

- void des3_done (symmetric_key ∗skey)

  Terminate the context.

- int des_keysize (int ∗keysize)

Gets suitable key size.

- int des3_keysize (int ∗keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor des_desc
- const struct ltc_cipher_descriptor des3_desc
- static const ulong32 bytebit [8]
- static const ulong32 bigbyte [24]
- static const unsigned char pc1 [56]
- static const unsigned char totrot [16]
- static const unsigned char pc2 [48]
- static const ulong32 SP1 [64]
- static const ulong32 SP2 [64]
- static const ulong32 SP3 [64]
- static const ulong32 SP4 [64]
- static const ulong32 SP5 [64]
- static const ulong32 SP6 [64]
- static const ulong32 SP7 [64]
- static const ulong32 SP8 [64]
- static const ulong64 des_ip [8][256]
- static const ulong64 des_fp [8][256]

5.6.2 Define Documentation

5.6.2.1 #define DE1 1

Definition at line 21 of file des.c.

Referenced by des3_setup(), and des_setup().

5.6.2.2 #define EN0 0

Definition at line 20 of file des.c.

Referenced by des3_setup(), and des_setup().

5.6.3 Function Documentation

5.6.3.1 static void cookey (const ulong32 ∗ *raw1*, ulong32 ∗ *keyout*) [static]

Definition at line 1366 of file des.c.

```
{
    ulong32 *cook;
    const ulong32 *raw0;
    ulong32 dough[32];
    int i;
```

```

    cook = dough;
    for(i=0; i < 16; i++, raw1++)
    {
        raw0 = raw1++;
        *cook    = (*raw0 & 0x00fc0000L) << 6;
        *cook   |= (*raw0 & 0x00000fc0L) << 10;
        *cook   |= (*raw1 & 0x00fc0000L) >> 10;
        *cook++ |= (*raw1 & 0x00000fc0L) >> 6;
        *cook    = (*raw0 & 0x0003f000L) << 12;
        *cook   |= (*raw0 & 0x0000003fL) << 16;
        *cook   |= (*raw1 & 0x0003f000L) >> 4;
        *cook++ |= (*raw1 & 0x0000003fL);
    }

    XMEMCPY(keyout, dough, sizeof dough);
}
```

5.6.3.2 void des3_done (symmetric_key * *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 1862 of file des.c.

```
{
}
```

5.6.3.3 int des3_ecb_decrypt (const unsigned char * *ct*, unsigned char * *pt*, symmetric_key * *skey*)

Decrypts a block of text with 3DES-EDE.

Parameters:

ct The input ciphertext (8 bytes)

pt The output plaintext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 1653 of file des.c.

References CRYPT_OK, desfunc(), and LTC_ARGCHK.

```
{
    ulong32 work[2];
    LTC_ARGCHK(pt   != NULL);
    LTC_ARGCHK(ct   != NULL);
    LTC_ARGCHK(skey != NULL);
    LOAD32H(work[0], ct+0);
    LOAD32H(work[1], ct+4);
    desfunc(work, skey->des3.dk[0]);
    desfunc(work, skey->des3.dk[1]);
```

```
    desfunc(work, skey->des3.dk[2]);
    STORE32H(work[0],pt+0);
    STORE32H(work[1],pt+4);
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.6.3.4 int des3_ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with 3DES-EDE.

Parameters:

pt The input plaintext (8 bytes)

ct The output ciphertext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 1629 of file des.c.

References CRYPT_OK, desfunc(), and LTC_ARGCHK.

```
{
    ulong32 work[2];

    LTC_ARGCHK(pt   != NULL);
    LTC_ARGCHK(ct   != NULL);
    LTC_ARGCHK(skey != NULL);
    LOAD32H(work[0], pt+0);
    LOAD32H(work[1], pt+4);
    desfunc(work, skey->des3.ek[0]);
    desfunc(work, skey->des3.ek[1]);
    desfunc(work, skey->des3.ek[2]);
    STORE32H(work[0],ct+0);
    STORE32H(work[1],ct+4);
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.6.3.5 int des3_keysize (int ∗ *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 1887 of file des.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
    LTC_ARGCHK(keysize != NULL);
    if(*keysize < 24) {
        return CRYPT_INVALID_KEYSIZE;
    }
    *keysize = 24;
    return CRYPT_OK;
}
```

5.6.3.6 int des3_setup (const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_key ∗ *skey*)

Initialize the 3DES-EDE block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 1556 of file des.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, CRYPT_OK, DE1, deskey(), EN0, and LTC_ARGCHK.

```
{
    LTC_ARGCHK(key != NULL);
    LTC_ARGCHK(skey != NULL);

    if(num_rounds != 0 && num_rounds != 16) {
        return CRYPT_INVALID_ROUNDS;
    }

    if (keylen != 24) {
        return CRYPT_INVALID_KEYSIZE;
    }

    deskey(key,    EN0, skey->des3.ek[0]);
    deskey(key+8,  DE1, skey->des3.ek[1]);
    deskey(key+16, EN0, skey->des3.ek[2]);

    deskey(key,    DE1, skey->des3.dk[2]);
    deskey(key+8,  EN0, skey->des3.dk[1]);
    deskey(key+16, DE1, skey->des3.dk[0]);

    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.6.3.7 int des3_test (void)

Definition at line 1816 of file des.c.

References CRYPT_NOP, CRYPT_OK, and des_test().

```
1817 {
1818  #ifndef LTC_TEST
1819     return CRYPT_NOP;
1820  #else
1821    unsigned char key[24], pt[8], ct[8], tmp[8];
1822    symmetric_key skey;
1823    int x, err;
1824
1825    if ((err = des_test()) != CRYPT_OK) {
1826       return err;
1827    }
1828
1829    for (x = 0; x < 8; x++) {
1830        pt[x] = x;
1831    }
1832
1833    for (x = 0; x < 24; x++) {
1834        key[x] = x;
1835    }
1836
1837    if ((err = des3_setup(key, 24, 0, &skey)) != CRYPT_OK) {
1838       return err;
1839    }
1840
1841    des3_ecb_encrypt(pt, ct, &skey);
1842    des3_ecb_decrypt(ct, tmp, &skey);
1843
1844    if (XMEMCMP(pt, tmp, 8) != 0) {
1845       return CRYPT_FAIL_TESTVECTOR;
1846    }
1847
1848    return CRYPT_OK;
1849  #endif
1850 }
```

Here is the call graph for this function:

5.6.3.8 void des_done (symmetric_key ∗ *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 1855 of file des.c.

```
1856 {
1857 }
```

5.6.3.9 int des_ecb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, symmetric_key ∗ *skey*)

Decrypts a block of text with DES.

Parameters:

ct The input ciphertext (8 bytes)

pt The output plaintext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 1608 of file des.c.

References CRYPT_OK, desfunc(), and LTC_ARGCHK.

```
{
    ulong32 work[2];
    LTC_ARGCHK(pt   != NULL);
    LTC_ARGCHK(ct   != NULL);
    LTC_ARGCHK(skey != NULL);
    LOAD32H(work[0], ct+0);
    LOAD32H(work[1], ct+4);
    desfunc(work, skey->des.dk);
    STORE32H(work[0],pt+0);
    STORE32H(work[1],pt+4);
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.6.3.10 int des_ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with DES.

Parameters:

pt The input plaintext (8 bytes)

ct The output ciphertext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 1587 of file des.c.

References CRYPT_OK, desfunc(), and LTC_ARGCHK.

```
{
    ulong32 work[2];
    LTC_ARGCHK(pt   != NULL);
    LTC_ARGCHK(ct   != NULL);
    LTC_ARGCHK(skey != NULL);
    LOAD32H(work[0], pt+0);
    LOAD32H(work[1], pt+4);
    desfunc(work, skey->des.ek);
    STORE32H(work[0],ct+0);
    STORE32H(work[1],ct+4);
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.6.3.11 int des_keysize (int ∗ *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 1872 of file des.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
    LTC_ARGCHK(keysize != NULL);
    if(*keysize < 8) {
        return CRYPT_INVALID_KEYSIZE;
    }
    *keysize = 8;
    return CRYPT_OK;
}
```

5.6.3.12 int des_setup (const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_key ∗ *skey*)

Initialize the DES block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 1529 of file des.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, CRYPT_OK, DE1, deskey(), EN0, and LTC_ARGCHK.

Referenced by des_test().

```
{
    LTC_ARGCHK(key != NULL);
    LTC_ARGCHK(skey != NULL);

    if (num_rounds != 0 && num_rounds != 16) {
        return CRYPT_INVALID_ROUNDS;
    }

    if (keylen != 8) {
        return CRYPT_INVALID_KEYSIZE;
    }

    deskey(key, EN0, skey->des.ek);
    deskey(key, DE1, skey->des.dk);

    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.6.3.13 int des_test (void)

Performs a self-test of the DES block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 1673 of file des.c.

References CRYPT_NOP, CRYPT_OK, and des_setup().

Referenced by des3_test().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
    int err;
    static const struct des_test_case {
        int num, mode; /* mode 1 = encrypt */
        unsigned char key[8], txt[8], out[8];
    } cases[] = {
        { 1, 1,     { 0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A },
                    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02 } },
        { 2, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x95, 0xF8, 0xA5, 0xE5, 0xDD, 0x31, 0xD9, 0x00 },
                    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
        { 3, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0xDD, 0x7F, 0x12, 0x1C, 0xA5, 0x01, 0x56, 0x19 },
                    { 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
        { 4, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x2E, 0x86, 0x53, 0x10, 0x4F, 0x38, 0x34, 0xEA },
                    { 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
        { 5, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x4B, 0xD3, 0x88, 0xFF, 0x6C, 0xD8, 0x1D, 0x4F },
                    { 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
        { 6, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x20, 0xB9, 0xE7, 0x67, 0xB2, 0xFB, 0x14, 0x56 },
                    { 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
        { 7, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x55, 0x57, 0x93, 0x80, 0xD7, 0x71, 0x38, 0xEF },
                    { 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
        { 8, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x6C, 0xC5, 0xDE, 0xFA, 0xAF, 0x04, 0x51, 0x2F },
                    { 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
        { 9, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x0D, 0x9F, 0x27, 0x9B, 0xA5, 0xD8, 0x72, 0x60 },
                    { 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
        {10, 1,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0xD9, 0x03, 0x1B, 0x02, 0x71, 0xBD, 0x5A, 0x0A },
                    { 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },

        { 1, 0,     { 0x10, 0x31, 0x6E, 0x02, 0x8C, 0x8F, 0x3B, 0x4A },
                    { 0x82, 0xDC, 0xBA, 0xFB, 0xDE, 0xAB, 0x66, 0x02 },
                    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 } },
        { 2, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0x95, 0xF8, 0xA5, 0xE5, 0xDD, 0x31, 0xD9, 0x00 } },
        { 3, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0xDD, 0x7F, 0x12, 0x1C, 0xA5, 0x01, 0x56, 0x19 } },
        { 4, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0x2E, 0x86, 0x53, 0x10, 0x4F, 0x38, 0x34, 0xEA } },
```

```
        { 5, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0x4B, 0xD3, 0x88, 0xFF, 0x6C, 0xD8, 0x1D, 0x4F } },
        { 6, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0x20, 0xB9, 0xE7, 0x67, 0xB2, 0xFB, 0x14, 0x56 } },
        { 7, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0x55, 0x57, 0x93, 0x80, 0xD7, 0x71, 0x38, 0xEF } },
        { 8, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0x6C, 0xC5, 0xDE, 0xFA, 0xAF, 0x04, 0x51, 0x2F } },
        { 9, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0x0D, 0x9F, 0x27, 0x9B, 0xA5, 0xD8, 0x72, 0x60 } },
        {10, 0,     { 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01 },
                    { 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
                    { 0xD9, 0x03, 0x1B, 0x02, 0x71, 0xBD, 0x5A, 0x0A } }

        /*** more test cases you could add if you are not convinced (the above test cases aren't real

                key              plaintext        ciphertext
                0000000000000000 0000000000000000 8CA64DE9C1B123A7
                FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF 7359B2163E4EDC58
                3000000000000000 1000000000000001 958E6E627A05557B
                1111111111111111 1111111111111111 F40379AB9E0EC533
                0123456789ABCDEF 1111111111111111 17668DFC7292532D
                1111111111111111 0123456789ABCDEF 8A5AE1F81AB8F2DD
                0000000000000000 0000000000000000 8CA64DE9C1B123A7
                FEDCBA9876543210 0123456789ABCDEF ED39D950FA74BCC4
                7CA110454A1A6E57 01A1D6D039776742 690F5B0D9A26939B
                0131D9619DC1376E 5CD54CA83DEF57DA 7A389D10354BD271
                07A1133E4A0B2686 0248D43806F67172 868EBB51CAB4599A
                3849674C2602319E 51454B582DDF440A 7178876E01F19B2A
                04B915BA43FEB5B6 42FD443059577FA2 AF37FB421F8C4095
                0113B970FD34F2CE 059B5E0851CF143A 86A560F10EC6D85B
                0170F175468FB5E6 0756D8E0774761D2 0CD3DA020021DC09
                43297FAD38E373FE 762514B829BF486A EA676B2CB7DB2B7A
                07A7137045DA2A16 3BDD119049372802 DFD64A815CAF1A0F
                04689104C2FD3B2F 26955F6835AF609A 5C513C9C4886C088
                37D06BB516CB7546 164D5E404F275232 0A2AEEAE3FF4AB77
                1F08260D1AC2465E 6B056E18759F5CCA EF1BF03E5DFA575A
                584023641ABA6176 004BD6EF09176062 88BF0DB6D70DEE56
                025816164629B007 480D39006EE762F2 A1F9915541020B56
                49793EBC79B3258F 437540C8698F3CFA 6FBF1CAFCFFD0556
                4FB05E1515AB73A7 072D43A077075292 2F22E49BAB7CA1AC
                49E95D6D4CA229BF 02FE55778117F12A 5A6B612CC26CCE4A
                018310DC409B26D6 1D9D5C5018F728C2 5F4C038ED12B2E41
                1C587F1C13924FEF 305532286D6F295A 63FAC0D034D9F793
                0101010101010101 0123456789ABCDEF 617B3A0CE8F07100
                1F1F1F1F0E0E0E0E 0123456789ABCDEF DB958605F8C8C606
                E0FEE0FEF1FEF1FE 0123456789ABCDEF EDBFD1C66C29CCC7
                0000000000000000 FFFFFFFFFFFFFFFF 355550B2150E2451
                FFFFFFFFFFFFFFFF 0000000000000000 CAAAAF4DEAF1DBAE
                0123456789ABCDEF 0000000000000000 D5D44FF720683D0D
                FEDCBA9876543210 FFFFFFFFFFFFFFFF 2A2BB008DF97C2F2

            http://www.ecs.soton.ac.uk/~prw99r/ez438/vectors.txt
        ***/
    };
    int i, y;
    unsigned char tmp[8];
    symmetric_key des;

    for(i=0; i < (int)(sizeof(cases)/sizeof(cases[0])); i++)
    {
        if ((err = des_setup(cases[i].key, 8, 0, &des)) != CRYPT_OK) {
```

```
            return err;
         }
         if (cases[i].mode != 0) {
            des_ecb_encrypt(cases[i].txt, tmp, &des);
         } else {
            des_ecb_decrypt(cases[i].txt, tmp, &des);
         }

         if (XMEMCMP(cases[i].out, tmp, sizeof(tmp)) != 0) {
            return CRYPT_FAIL_TESTVECTOR;
         }

       /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started
       for (y = 0; y < 8; y++) tmp[y] = 0;
       for (y = 0; y < 1000; y++) des_ecb_encrypt(tmp, tmp, &des);
       for (y = 0; y < 1000; y++) des_ecb_decrypt(tmp, tmp, &des);
       for (y = 0; y < 8; y++) if (tmp[y] != 0) return CRYPT_FAIL_TESTVECTOR;
}

    return CRYPT_OK;
  #endif
}
```

Here is the call graph for this function:

5.6.3.14 static void desfunc (ulong32 ∗ *block*, const ulong32 ∗ *keys*) [static]

Definition at line 1400 of file des.c.

References byte, des_ip, ROLc, RORc, SP1, SP2, SP3, SP4, SP5, SP6, SP7, and SP8.

Referenced by des3_ecb_decrypt(), des3_ecb_encrypt(), des_ecb_decrypt(), and des_ecb_encrypt().

```
{
    ulong32 work, right, leftt;
    int cur_round;

    leftt = block[0];
    right = block[1];

#ifdef LTC_SMALL_CODE
    work = ((leftt >> 4)  ^ right) & 0x0f0f0f0fL;
    right ^= work;
    leftt ^= (work << 4);

    work = ((leftt >> 16) ^ right) & 0x0000ffffL;
    right ^= work;
    leftt ^= (work << 16);

    work = ((right >> 2)  ^ leftt) & 0x33333333L;
    leftt ^= work;
    right ^= (work << 2);

    work = ((right >> 8)  ^ leftt) & 0x00ff00ffL;
    leftt ^= work;
    right ^= (work << 8);

    right = ROLc(right, 1);
    work = (leftt ^ right) & 0xaaaaaaaaL;

    leftt ^= work;
    right ^= work;
    leftt = ROLc(leftt, 1);
#else
   {
```

```
      ulong64 tmp;
      tmp = des_ip[0][byte(leftt, 0)] ^
            des_ip[1][byte(leftt, 1)] ^
            des_ip[2][byte(leftt, 2)] ^
            des_ip[3][byte(leftt, 3)] ^
            des_ip[4][byte(right, 0)] ^
            des_ip[5][byte(right, 1)] ^
            des_ip[6][byte(right, 2)] ^
            des_ip[7][byte(right, 3)];
      leftt = (ulong32)(tmp >> 32);
      right = (ulong32)(tmp & 0xFFFFFFFFUL);
   }
#endif

    for (cur_round = 0; cur_round < 8; cur_round++) {
        work  = RORc(right, 4) ^ *keys++;
        leftt ^= SP7[work        & 0x3fL]
              ^ SP5[(work >>  8) & 0x3fL]
              ^ SP3[(work >> 16) & 0x3fL]
              ^ SP1[(work >> 24) & 0x3fL];
        work  = right ^ *keys++;
        leftt ^= SP8[ work        & 0x3fL]
              ^  SP6[(work >>  8) & 0x3fL]
              ^  SP4[(work >> 16) & 0x3fL]
              ^  SP2[(work >> 24) & 0x3fL];

        work = RORc(leftt, 4) ^ *keys++;
        right ^= SP7[ work        & 0x3fL]
              ^  SP5[(work >>  8) & 0x3fL]
              ^  SP3[(work >> 16) & 0x3fL]
              ^  SP1[(work >> 24) & 0x3fL];
        work  = leftt ^ *keys++;
        right ^= SP8[ work        & 0x3fL]
              ^  SP6[(work >>  8) & 0x3fL]
              ^  SP4[(work >> 16) & 0x3fL]
              ^  SP2[(work >> 24) & 0x3fL];
    }

#ifdef LTC_SMALL_CODE
    right = RORc(right, 1);
    work = (leftt ^ right) & 0xaaaaaaaaL;
    leftt ^= work;
    right ^= work;
    leftt = RORc(leftt, 1);
    work = ((leftt >> 8) ^ right) & 0x00ff00ffL;
    right ^= work;
    leftt ^= (work << 8);
    /* -- */
    work = ((leftt >> 2) ^ right) & 0x33333333L;
    right ^= work;
    leftt ^= (work << 2);
    work = ((right >> 16) ^ leftt) & 0x0000ffffL;
    leftt ^= work;
    right ^= (work << 16);
    work = ((right >> 4) ^ leftt) & 0x0f0f0f0fL;
    leftt ^= work;
    right ^= (work << 4);
#else
   {
      ulong64 tmp;
      tmp = des_fp[0][byte(leftt, 0)] ^
            des_fp[1][byte(leftt, 1)] ^
            des_fp[2][byte(leftt, 2)] ^
            des_fp[3][byte(leftt, 3)] ^
            des_fp[4][byte(right, 0)] ^
            des_fp[5][byte(right, 1)] ^
            des_fp[6][byte(right, 2)] ^
```

```
            des_fp[7][byte(right, 3)];
      leftt = (ulong32)(tmp >> 32);
      right = (ulong32)(tmp & 0xFFFFFFFFUL);
   }
#endif

    block[0] = right;
    block[1] = leftt;
}
```

5.6.3.15 static void deskey (const unsigned char ∗ *key*, short *edf*, ulong32 ∗ *keyout*) [static]

Definition at line 1306 of file des.c.

References bytebit, and pc1.

Referenced by des3_setup(), and des_setup().

```
{
    ulong32 i, j, l, m, n, kn[32];
    unsigned char pc1m[56], pcr[56];

    for (j=0; j < 56; j++) {
        l = (ulong32)pc1[j];
        m = l & 7;
        pc1m[j] = (unsigned char)((key[l >> 3U] & bytebit[m]) == bytebit[m] ? 1 : 0);
    }

    for (i=0; i < 16; i++) {
        if (edf == DE1) {
           m = (15 - i) << 1;
        } else {
           m = i << 1;
        }
        n = m + 1;
        kn[m] = kn[n] = 0L;
        for (j=0; j < 28; j++) {
            l = j + (ulong32)totrot[i];
            if (l < 28) {
               pcr[j] = pc1m[l];
            } else {
               pcr[j] = pc1m[l - 28];
            }
        }
        for (/*j = 28*/; j < 56; j++) {
            l = j + (ulong32)totrot[i];
            if (l < 56) {
               pcr[j] = pc1m[l];
            } else {
               pcr[j] = pc1m[l - 28];
            }
        }
        for (j=0; j < 24; j++)  {
            if ((int)pcr[(int)pc2[j]] != 0) {
               kn[m] |= bigbyte[j];
            }
            if ((int)pcr[(int)pc2[j+24]] != 0) {
               kn[n] |= bigbyte[j];
            }
        }
    }

    cookey(kn, keyout);
}
```

5.6.4 Variable Documentation

5.6.4.1 const ulong32 bigbyte[24] [static]

Initial value:

```
{
    0x800000UL,  0x400000UL,  0x200000UL,  0x100000UL,
    0x80000UL,   0x40000UL,   0x20000UL,   0x10000UL,
    0x8000UL,    0x4000UL,    0x2000UL,    0x1000UL,
    0x800UL,     0x400UL,     0x200UL,     0x100UL,
    0x80UL,      0x40UL,      0x20UL,      0x10UL,
    0x8UL,       0x4UL,       0x2UL,       0x1L
}
```

Definition at line 56 of file des.c.

5.6.4.2 const ulong32 bytebit[8] [static]

Initial value:

```
{
    0200, 0100, 040, 020, 010, 04, 02, 01
}
```

Definition at line 51 of file des.c.

Referenced by deskey().

5.6.4.3 const struct ltc_cipher_descriptor des3_desc

Initial value:

```
{
    "3des",
    14,
    24, 24, 8, 16,
    &des3_setup,
    &des3_ecb_encrypt,
    &des3_ecb_decrypt,
    &des3_test,
    &des3_done,
    &des3_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 37 of file des.c.

Referenced by yarrow_start().

5.6.4.4 const struct ltc_cipher_descriptor des_desc

Initial value:

```
{
    "des",
```

```
    13,
    8, 8, 8, 16,
    &des_setup,
    &des_ecb_encrypt,
    &des_ecb_decrypt,
    &des_test,
    &des_done,
    &des_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 23 of file des.c.

5.6.4.5 const ulong64 des_fp[8][256] [static]

Definition at line 775 of file des.c.

5.6.4.6 const ulong64 des_ip[8][256] [static]

Definition at line 252 of file des.c.

Referenced by desfunc().

5.6.4.7 const unsigned char pc1[56] [static]

Initial value:

```
 {
    56, 48, 40, 32, 24, 16,  8,  0, 57, 49, 41, 33, 25, 17,
     9,  1, 58, 50, 42, 34, 26, 18, 10,  2, 59, 51, 43, 35,
    62, 54, 46, 38, 30, 22, 14,  6, 61, 53, 45, 37, 29, 21,
    13,  5, 60, 52, 44, 36, 28, 20, 12,  4, 27, 19, 11,  3
}
```

Definition at line 68 of file des.c.

Referenced by deskey().

5.6.4.8 const unsigned char pc2[48] [static]

Initial value:

```
 {
    13, 16, 10, 23,  0,  4,      2, 27, 14,  5, 20,  9,
    22, 18, 11,  3, 25,  7,     15,  6, 26, 19, 12,  1,
    40, 51, 30, 36, 46, 54,     29, 39, 50, 44, 32, 47,
    43, 48, 38, 55, 33, 52,     45, 41, 49, 35, 28, 31
}
```

Definition at line 82 of file des.c.

5.6.4.9 const ulong32 SP1[64] [static]

Initial value:

```
{
    0x01010400UL, 0x00000000UL, 0x00010000UL, 0x01010404UL,
    0x01010004UL, 0x00010404UL, 0x00000004UL, 0x00010000UL,
    0x00000400UL, 0x01010400UL, 0x01010404UL, 0x00000400UL,
    0x01000404UL, 0x01010004UL, 0x01000000UL, 0x00000004UL,
    0x00000404UL, 0x01000400UL, 0x01000400UL, 0x00010400UL,
    0x00010400UL, 0x01010000UL, 0x01010000UL, 0x01000404UL,
    0x00010004UL, 0x01000004UL, 0x01000004UL, 0x00010004UL,
    0x00000000UL, 0x00000404UL, 0x00010404UL, 0x01000000UL,
    0x00010000UL, 0x01010404UL, 0x00000004UL, 0x01010000UL,
    0x01010400UL, 0x01000000UL, 0x01000000UL, 0x00000400UL,
    0x01010004UL, 0x00010000UL, 0x00010400UL, 0x01000004UL,
    0x00000400UL, 0x00000004UL, 0x01000404UL, 0x00010404UL,
    0x01010404UL, 0x00010004UL, 0x01010000UL, 0x01000404UL,
    0x01000004UL, 0x00000404UL, 0x00010404UL, 0x01010400UL,
    0x00000404UL, 0x01000400UL, 0x01000400UL, 0x00000000UL,
    0x00010004UL, 0x00010400UL, 0x00000000UL, 0x01010004UL
}
```

Definition at line 90 of file des.c.

Referenced by desfunc().

5.6.4.10 const ulong32 SP2[64] [static]

Initial value:

```
{
    0x80108020UL, 0x80008000UL, 0x00008000UL, 0x00108020UL,
    0x00100000UL, 0x00000020UL, 0x80100020UL, 0x80008020UL,
    0x80000020UL, 0x80108020UL, 0x80108000UL, 0x80000000UL,
    0x80008000UL, 0x00100000UL, 0x00000020UL, 0x80100020UL,
    0x00108000UL, 0x00100020UL, 0x80008020UL, 0x00000000UL,
    0x80000000UL, 0x00008000UL, 0x00108020UL, 0x80100000UL,
    0x00100020UL, 0x80000020UL, 0x00000000UL, 0x00108000UL,
    0x00008020UL, 0x80108000UL, 0x80100000UL, 0x00008020UL,
    0x00000000UL, 0x00108020UL, 0x80100020UL, 0x00100000UL,
    0x80008020UL, 0x80100000UL, 0x80108000UL, 0x00008000UL,
    0x80100000UL, 0x80008000UL, 0x00000020UL, 0x80108020UL,
    0x00108020UL, 0x00000020UL, 0x00008000UL, 0x80000000UL,
    0x00008020UL, 0x80108000UL, 0x00100000UL, 0x80000020UL,
    0x00100020UL, 0x80008020UL, 0x80000020UL, 0x00100020UL,
    0x00108000UL, 0x00000000UL, 0x80008000UL, 0x00008020UL,
    0x80000000UL, 0x80100020UL, 0x80108020UL, 0x00108000UL
}
```

Definition at line 110 of file des.c.

Referenced by desfunc().

5.6.4.11 const ulong32 SP3[64] [static]

Initial value:

```
{
    0x00000208UL, 0x08020200UL, 0x00000000UL, 0x08020008UL,
    0x08000200UL, 0x00000000UL, 0x00020208UL, 0x08000200UL,
    0x00020008UL, 0x08000008UL, 0x08000008UL, 0x00020000UL,
    0x08020208UL, 0x00020008UL, 0x08020000UL, 0x00000208UL,
    0x08000000UL, 0x00000008UL, 0x08020200UL, 0x00000200UL,
    0x00020200UL, 0x08020000UL, 0x08020008UL, 0x00020208UL,
```

```
    0x08000208UL, 0x00020200UL, 0x00020000UL, 0x08000208UL,
    0x00000008UL, 0x08020208UL, 0x00000200UL, 0x08000000UL,
    0x08020200UL, 0x08000000UL, 0x00020008UL, 0x00000208UL,
    0x00020000UL, 0x08020200UL, 0x08000200UL, 0x00000000UL,
    0x00000200UL, 0x00020008UL, 0x08020208UL, 0x08000200UL,
    0x08000008UL, 0x00000200UL, 0x00000000UL, 0x08020008UL,
    0x08000208UL, 0x00020000UL, 0x08000000UL, 0x08020208UL,
    0x00000008UL, 0x00020208UL, 0x00020200UL, 0x08000008UL,
    0x08020000UL, 0x08000208UL, 0x00000208UL, 0x08020000UL,
    0x00020208UL, 0x00000008UL, 0x08020008UL, 0x00020200UL
}
```

Definition at line 130 of file des.c.

Referenced by desfunc().

5.6.4.12 const ulong32 SP4[64] `[static]`

Initial value:

```
{
    0x00802001UL, 0x00002081UL, 0x00002081UL, 0x00000080UL,
    0x00802080UL, 0x00800081UL, 0x00800001UL, 0x00002001UL,
    0x00000000UL, 0x00802000UL, 0x00802000UL, 0x00802081UL,
    0x00000081UL, 0x00000000UL, 0x00800080UL, 0x00800001UL,
    0x00000001UL, 0x00002000UL, 0x00800000UL, 0x00802001UL,
    0x00000080UL, 0x00800000UL, 0x00002001UL, 0x00002080UL,
    0x00800081UL, 0x00000001UL, 0x00002080UL, 0x00800080UL,
    0x00002000UL, 0x00802080UL, 0x00802081UL, 0x00000081UL,
    0x00800080UL, 0x00800001UL, 0x00802000UL, 0x00802081UL,
    0x00000081UL, 0x00000000UL, 0x00000000UL, 0x00802000UL,
    0x00002080UL, 0x00800080UL, 0x00800081UL, 0x00000001UL,
    0x00802001UL, 0x00002081UL, 0x00002081UL, 0x00000080UL,
    0x00802081UL, 0x00000081UL, 0x00000001UL, 0x00002000UL,
    0x00800001UL, 0x00002001UL, 0x00802080UL, 0x00800081UL,
    0x00002001UL, 0x00002080UL, 0x00800000UL, 0x00802001UL,
    0x00000080UL, 0x00800000UL, 0x00002000UL, 0x00802080UL
}
```

Definition at line 150 of file des.c.

Referenced by desfunc().

5.6.4.13 const ulong32 SP5[64] `[static]`

Initial value:

```
{
    0x00000100UL, 0x02080100UL, 0x02080000UL, 0x42000100UL,
    0x00080000UL, 0x00000100UL, 0x40000000UL, 0x02080000UL,
    0x40080100UL, 0x00080000UL, 0x02000100UL, 0x40080100UL,
    0x42000100UL, 0x42080000UL, 0x00080100UL, 0x40000000UL,
    0x02000000UL, 0x40080000UL, 0x40080000UL, 0x00000000UL,
    0x40000100UL, 0x42080100UL, 0x42080100UL, 0x02000100UL,
    0x42080000UL, 0x40000100UL, 0x00000000UL, 0x42000000UL,
    0x02080100UL, 0x02000000UL, 0x42000000UL, 0x00080100UL,
    0x00080000UL, 0x42000100UL, 0x00000100UL, 0x02000000UL,
    0x40000000UL, 0x02080000UL, 0x42000100UL, 0x40080100UL,
    0x02000100UL, 0x40000000UL, 0x42080000UL, 0x02080100UL,
    0x40080100UL, 0x00000100UL, 0x02000000UL, 0x42080000UL,
    0x42080100UL, 0x00080100UL, 0x42000000UL, 0x42080100UL,
```

```
    0x02080000UL, 0x00000000UL, 0x40080000UL, 0x42000000UL,
    0x00080100UL, 0x02000100UL, 0x40000100UL, 0x00080000UL,
    0x00000000UL, 0x40080000UL, 0x02080100UL, 0x40000100UL
}
```

Definition at line 170 of file des.c.

Referenced by desfunc().

5.6.4.14 const ulong32 SP6[64] [static]

Initial value:

```
{
    0x20000010UL, 0x20400000UL, 0x00004000UL, 0x20404010UL,
    0x20400000UL, 0x00000010UL, 0x20404010UL, 0x00400000UL,
    0x20004000UL, 0x00404010UL, 0x00400000UL, 0x20000010UL,
    0x00400010UL, 0x20004000UL, 0x20000000UL, 0x00004010UL,
    0x00000000UL, 0x00400010UL, 0x20004010UL, 0x00004000UL,
    0x00404000UL, 0x20004010UL, 0x00000010UL, 0x20400010UL,
    0x20400010UL, 0x00000000UL, 0x00404010UL, 0x20404000UL,
    0x00004010UL, 0x00404000UL, 0x20404000UL, 0x20000000UL,
    0x20004000UL, 0x00000010UL, 0x20400010UL, 0x00404000UL,
    0x20404010UL, 0x00400000UL, 0x00004010UL, 0x20000010UL,
    0x00400000UL, 0x20004000UL, 0x20000000UL, 0x00004010UL,
    0x20000010UL, 0x20404010UL, 0x00404000UL, 0x20400000UL,
    0x00404010UL, 0x20404000UL, 0x00000000UL, 0x20400010UL,
    0x00000010UL, 0x00004000UL, 0x20400000UL, 0x00404010UL,
    0x00004000UL, 0x00400010UL, 0x20004010UL, 0x00000000UL,
    0x20404000UL, 0x20000000UL, 0x00400010UL, 0x20004010UL
}
```

Definition at line 190 of file des.c.

Referenced by desfunc().

5.6.4.15 const ulong32 SP7[64] [static]

Initial value:

```
{
    0x00200000UL, 0x04200002UL, 0x04000802UL, 0x00000000UL,
    0x00000800UL, 0x04000802UL, 0x00200802UL, 0x04200800UL,
    0x04200802UL, 0x00200000UL, 0x00000000UL, 0x04000002UL,
    0x00000002UL, 0x04000000UL, 0x04200002UL, 0x00000802UL,
    0x04000800UL, 0x00200802UL, 0x00200002UL, 0x04000800UL,
    0x04000002UL, 0x04200000UL, 0x04200800UL, 0x00200002UL,
    0x04200000UL, 0x00000800UL, 0x00000802UL, 0x04200802UL,
    0x00200800UL, 0x00000002UL, 0x04000000UL, 0x00200800UL,
    0x04000000UL, 0x00200800UL, 0x00200000UL, 0x04000802UL,
    0x04000802UL, 0x04200002UL, 0x04200002UL, 0x00000002UL,
    0x00200002UL, 0x04000000UL, 0x04000800UL, 0x00200000UL,
    0x04200800UL, 0x00000802UL, 0x00200802UL, 0x04200800UL,
    0x00000802UL, 0x04000002UL, 0x04200802UL, 0x04200000UL,
    0x00200800UL, 0x00000000UL, 0x00000002UL, 0x04200802UL,
    0x00000000UL, 0x00200802UL, 0x04200000UL, 0x00000800UL,
    0x04000002UL, 0x04000800UL, 0x00000800UL, 0x00200002UL
}
```

Definition at line 210 of file des.c.

Referenced by desfunc().

5.6.4.16 const ulong32 SP8[64] [static]

Initial value:

```
{
    0x10001040UL, 0x00001000UL, 0x00040000UL, 0x10041040UL,
    0x10000000UL, 0x10001040UL, 0x00000040UL, 0x10000000UL,
    0x00040040UL, 0x10040000UL, 0x10041040UL, 0x00041000UL,
    0x10041000UL, 0x00041040UL, 0x00001000UL, 0x00000040UL,
    0x10040000UL, 0x10000040UL, 0x10001000UL, 0x00001040UL,
    0x00041000UL, 0x00040040UL, 0x10040040UL, 0x10041000UL,
    0x00001040UL, 0x00000000UL, 0x00000000UL, 0x10040040UL,
    0x10000040UL, 0x10001000UL, 0x00041040UL, 0x00040000UL,
    0x00041040UL, 0x00040000UL, 0x10041000UL, 0x00001000UL,
    0x00000040UL, 0x10040040UL, 0x00001000UL, 0x00041040UL,
    0x10001000UL, 0x00000040UL, 0x10000040UL, 0x10040000UL,
    0x10040040UL, 0x10000000UL, 0x00040000UL, 0x10001040UL,
    0x00000000UL, 0x10041040UL, 0x00040040UL, 0x10000040UL,
    0x10040000UL, 0x10001000UL, 0x10001040UL, 0x00000000UL,
    0x10041040UL, 0x00041000UL, 0x00041000UL, 0x00001040UL,
    0x00001040UL, 0x00040040UL, 0x10000000UL, 0x10041000UL
}
```

Definition at line 230 of file des.c.

Referenced by desfunc().

5.6.4.17 const unsigned char totrot[16] [static]

Initial value:

```
 {
    1,   2,  4,  6,
    8,  10, 12, 14,
    15, 17, 19, 21,
    23, 25, 27, 28
}
```

Definition at line 75 of file des.c.

5.7 ciphers/kasumi.c File Reference

5.7.1 Detailed Description

Implementation of the 3GPP Kasumi block cipher Derived from the 3GPP standard source code.

Definition in file kasumi.c.

`#include "tomcrypt.h"`

Include dependency graph for kasumi.c:

Defines

- #define ROL16(x, y) ((((x)<<(y)) | ((x)>>(16-(y)))) & 0xFFFF)

Typedefs

- typedef unsigned u16

Functions

- static u16 FI (u16 in, u16 subkey)
- static ulong32 FO (ulong32 in, int round_no, symmetric_key ∗key)
- static ulong32 FL (ulong32 in, int round_no, symmetric_key ∗key)
- int kasumi_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)
- int kasumi_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)
- int kasumi_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)
- void kasumi_done (symmetric_key ∗skey)
- int kasumi_keysize (int ∗keysize)
- int kasumi_test (void)

Variables

- const struct ltc_cipher_descriptor kasumi_desc

5.7.2 Define Documentation

5.7.2.1 #define ROL16(x, y) ((((x)<<(y)) | ((x)>>(16-(y)))) & 0xFFFF)

Definition at line 24 of file kasumi.c.

Referenced by FL().

5.7.3 Typedef Documentation

5.7.3.1 typedef unsigned u16

Definition at line 22 of file kasumi.c.

5.7.4 Function Documentation

5.7.4.1 static u16 FI (u16 *in*, u16 *subkey*) [static]

Definition at line 39 of file kasumi.c.

References S7.

```
{
   u16 nine, seven;
   static const u16 S7[128] = {
      54, 50, 62, 56, 22, 34, 94, 96, 38, 6, 63, 93, 2, 18,123, 33,
      55,113, 39,114, 21, 67, 65, 12, 47, 73, 46, 27, 25,111,124, 81,
      53, 9,121, 79, 52, 60, 58, 48,101,127, 40,120,104, 70, 71, 43,
      20,122, 72, 61, 23,109, 13,100, 77, 1, 16, 7, 82, 10,105, 98,
      117,116, 76, 11, 89,106, 0,125,118, 99, 86, 69, 30, 57,126, 87,
      112, 51, 17, 5, 95, 14, 90, 84, 91, 8, 35,103, 32, 97, 28, 66,
      102, 31, 26, 45, 75, 4, 85, 92, 37, 74, 80, 49, 68, 29,115, 44,
      64,107,108, 24,110, 83, 36, 78, 42, 19, 15, 41, 88,119, 59, 3 };
  static const u16 S9[512] = {
      167,239,161,379,391,334, 9,338, 38,226, 48,358,452,385, 90,397,
      183,253,147,331,415,340, 51,362,306,500,262, 82,216,159,356,177,
      175,241,489, 37,206, 17, 0,333, 44,254,378, 58,143,220, 81,400,
       95, 3,315,245, 54,235,218,405,472,264,172,494,371,290,399, 76,
      165,197,395,121,257,480,423,212,240, 28,462,176,406,507,288,223,
      501,407,249,265, 89,186,221,428,164, 74,440,196,458,421,350,163,
      232,158,134,354, 13,250,491,142,191, 69,193,425,152,227,366,135,
      344,300,276,242,437,320,113,278, 11,243, 87,317, 36, 93,496, 27,
      487,446,482, 41, 68,156,457,131,326,403,339, 20, 39,115,442,124,
      475,384,508, 53,112,170,479,151,126,169, 73,268,279,321,168,364,
      363,292, 46,499,393,327,324, 24,456,267,157,460,488,426,309,229,
      439,506,208,271,349,401,434,236, 16,209,359, 52, 56,120,199,277,
      465,416,252,287,246, 6, 83,305,420,345,153,502, 65, 61,244,282,
      173,222,418, 67,386,368,261,101,476,291,195,430, 49, 79,166,330,
      280,383,373,128,382,408,155,495,367,388,274,107,459,417, 62,454,
      132,225,203,316,234, 14,301, 91,503,286,424,211,347,307,140,374,
       35,103,125,427, 19,214,453,146,498,314,444,230,256,329,198,285,
       50,116, 78,410, 10,205,510,171,231, 45,139,467, 29, 86,505, 32,
       72, 26,342,150,313,490,431,238,411,325,149,473, 40,119,174,355,
      185,233,389, 71,448,273,372, 55,110,178,322, 12,469,392,369,190,
        1,109,375,137,181, 88, 75,308,260,484, 98,272,370,275,412,111,
      336,318, 4,504,492,259,304, 77,337,435, 21,357,303,332,483, 18,
       47, 85, 25,497,474,289,100,269,296,478,270,106, 31,104,433, 84,
      414,486,394, 96, 99,154,511,148,413,361,409,255,162,215,302,201,
      266,351,343,144,441,365,108,298,251, 34,182,509,138,210,335,133,
      311,352,328,141,396,346,123,319,450,281,429,228,443,481, 92,404,
      485,422,248,297, 23,213,130,466, 22,217,283, 70,294,360,419,127,
      312,377, 7,468,194, 2,117,295,463,258,224,447,247,187, 80,398,
      284,353,105,390,299,471,470,184, 57,200,348, 63,204,188, 33,451,
       97, 30,310,219, 94,160,129,493, 64,179,263,102,189,207,114,402,
      438,477,387,122,192, 42,381, 5,145,118,180,449,293,323,136,380,
       43, 66, 60,455,341,445,202,432, 8,237, 15,376,436,464, 59,461};

  /* The sixteen bit input is split into two unequal halves, *
   * nine bits and seven bits - as is the subkey            */

  nine  = (u16)(in>>7)&0x1FF;
  seven = (u16)(in&0x7F);

  /* Now run the various operations */
  nine   = (u16)(S9[nine] ^ seven);
  seven  = (u16)(S7[seven] ^ (nine & 0x7F));
  seven ^= (subkey>>9);
  nine  ^= (subkey&0x1FF);
  nine   = (u16)(S9[nine] ^ seven);
  seven  = (u16)(S7[seven] ^ (nine & 0x7F));
```

```
98   return (u16)(seven<<9) + nine;
99 }
```

5.7.4.2 static ulong32 FL (ulong32 *in*, int *round_no*, symmetric_key ∗ *key*) [static]

Definition at line 125 of file kasumi.c.

References ROL16.

Referenced by kasumi_ecb_decrypt(), and kasumi_ecb_encrypt().

```
126 {
127     u16 l, r, a, b;
128     /* split out the left and right halves */
129     l = (u16)(in>>16);
130     r = (u16)(in)&0xFFFF;
131     /* do the FL() operations           */
132     a = (u16) (l & key->kasumi.KLi1[round_no]);
133     r ^= ROL16(a,1);
134     b = (u16)(r | key->kasumi.KLi2[round_no]);
135     l ^= ROL16(b,1);
136     /* put the two halves back together */
137
138     return (((ulong32)l)<<16) + r;
139 }
```

5.7.4.3 static ulong32 FO (ulong32 *in*, int *round_no*, symmetric_key ∗ *key*) [static]

Definition at line 101 of file kasumi.c.

References FI().

Referenced by kasumi_ecb_decrypt(), and kasumi_ecb_encrypt().

```
102 {
103    u16 left, right;
104
105   /* Split the input into two 16-bit words */
106   left = (u16)(in>>16);
107   right = (u16) in&0xFFFF;
108
109   /* Now apply the same basic transformation three times */
110   left ^= key->kasumi.KOi1[round_no];
111   left = FI( left, key->kasumi.KIi1[round_no] );
112   left ^= right;
113
114   right ^= key->kasumi.KOi2[round_no];
115   right = FI( right, key->kasumi.KIi2[round_no] );
116   right ^= left;
117
118   left ^= key->kasumi.KOi3[round_no];
119   left = FI( left, key->kasumi.KIi3[round_no] );
120   left ^= right;
121
122   return (((ulong32)right)<<16)+left;
123 }
```

Here is the call graph for this function:

5.7.4.4 void kasumi_done (symmetric_key * *skey*)

Definition at line 237 of file kasumi.c.

```
238 {
239 }
```

5.7.4.5 int kasumi_ecb_decrypt (const unsigned char * *ct*, unsigned char * *pt*, symmetric_key * *skey*)

Definition at line 168 of file kasumi.c.

References FL(), FO(), and LTC_ARGCHK.

```
169 {
170     ulong32 left, right, temp;
171     int n;
172 
173     LTC_ARGCHK(pt   != NULL);
174     LTC_ARGCHK(ct   != NULL);
175     LTC_ARGCHK(skey != NULL);
176 
177     LOAD32H(left, ct);
178     LOAD32H(right, ct+4);
179 
180     for (n = 7; n >= 0; ) {
181         temp = FO(right, n,   skey);
182         temp = FL(temp,  n--, skey);
183         left ^= temp;
184         temp = FL(left,  n,   skey);
185         temp = FO(temp,  n--, skey);
186         right ^= temp;
187     }
188 
189     STORE32H(left, pt);
190     STORE32H(right, pt+4);
191 
192     return CRYPT_OK;
193 }
```

Here is the call graph for this function:

5.7.4.6 int kasumi_ecb_encrypt (const unsigned char * *pt*, unsigned char * *ct*, symmetric_key * *skey*)

Definition at line 141 of file kasumi.c.

References FL(), FO(), and LTC_ARGCHK.

```
142 {
143     ulong32 left, right, temp;
144     int n;
145 
146     LTC_ARGCHK(pt   != NULL);
147     LTC_ARGCHK(ct   != NULL);
148     LTC_ARGCHK(skey != NULL);
149 
150     LOAD32H(left, pt);
151     LOAD32H(right, pt+4);
```

```

    for (n = 0; n <= 7; ) {
        temp = FL(left,  n,   skey);
        temp = FO(temp,  n++, skey);
        right ^= temp;
        temp = FO(right, n,   skey);
        temp = FL(temp,  n++, skey);
        left ^= temp;
    }

    STORE32H(left, ct);
    STORE32H(right, ct+4);

    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.7.4.7 int kasumi_keysize (int ∗ *keysize*)

Definition at line 241 of file kasumi.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize >= 16) {
      *keysize = 16;
      return CRYPT_OK;
   } else {
      return CRYPT_INVALID_KEYSIZE;
   }
}
```

5.7.4.8 int kasumi_setup (const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_key ∗ *skey*)

Definition at line 195 of file kasumi.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, and LTC_ARGCHK.

Referenced by kasumi_test().

```
{
    static const u16 C[8] = { 0x0123,0x4567,0x89AB,0xCDEF, 0xFEDC,0xBA98,0x7654,0x3210 };
    u16 ukey[8], Kprime[8];
    int n;

    LTC_ARGCHK(key  != NULL);
    LTC_ARGCHK(skey != NULL);

    if (keylen != 16) {
       return CRYPT_INVALID_KEYSIZE;
    }

    if (num_rounds != 0 && num_rounds != 8) {
       return CRYPT_INVALID_ROUNDS;
    }

    /* Start by ensuring the subkeys are endian correct on a 16-bit basis */
    for (n = 0; n < 8; n++ ) {
```

```
        ukey[n] = (((u16)key[2*n]) << 8) | key[2*n+1];
    }

    /* Now build the K’[] keys */
    for (n = 0; n < 8; n++) {
        Kprime[n] = ukey[n] ^ C[n];
    }

    /* Finally construct the various sub keys */
    for(n = 0; n < 8; n++) {
        skey->kasumi.KLi1[n] = ROL16(ukey[n],1);
        skey->kasumi.KLi2[n] = Kprime[(n+2)&0x7];
        skey->kasumi.KOi1[n] = ROL16(ukey[(n+1)&0x7],5);
        skey->kasumi.KOi2[n] = ROL16(ukey[(n+5)&0x7],8);
        skey->kasumi.KOi3[n] = ROL16(ukey[(n+6)&0x7],13);
        skey->kasumi.KIi1[n] = Kprime[(n+4)&0x7];
        skey->kasumi.KIi2[n] = Kprime[(n+3)&0x7];
        skey->kasumi.KIi3[n] = Kprime[(n+7)&0x7];
    }

    return CRYPT_OK;
}
```

5.7.4.9 int kasumi_test (void)

Definition at line 252 of file kasumi.c.

References CRYPT_NOP, CRYPT_OK, and kasumi_setup().

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   static const struct {
      unsigned char key[16], pt[8], ct[8];
   } tests[] = {

{
   { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x4B, 0x58, 0xA7, 0x71, 0xAF, 0xC7, 0xE5, 0xE8 }
},

{
   { 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x7E, 0xEF, 0x11, 0x3C, 0x95, 0xBB, 0x5A, 0x77 }
},

{
   { 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x5F, 0x14, 0x06, 0x86, 0xD7, 0xAD, 0x5A, 0x39 },
},

{
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x2E, 0x14, 0x91, 0xCF, 0x70, 0xAA, 0x46, 0x5D }
},

{
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
```

```
   { 0xB5, 0x45, 0x86, 0xF4, 0xAB, 0x9A, 0xE5, 0x46 }
},

};
   unsigned char buf[2][8];
   symmetric_key key;
   int err, x;

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       if ((err = kasumi_setup(tests[x].key, 16, 0, &key)) != CRYPT_OK) {
          return err;
       }
       if ((err = kasumi_ecb_encrypt(tests[x].pt, buf[0], &key)) != CRYPT_OK) {
          return err;
       }
       if ((err = kasumi_ecb_decrypt(tests[x].ct, buf[1], &key)) != CRYPT_OK) {
          return err;
       }
       if (XMEMCMP(tests[x].pt, buf[1], 8) || XMEMCMP(tests[x].ct, buf[0], 8)) {
          return CRYPT_FAIL_TESTVECTOR;
       }
   }
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.7.5 Variable Documentation

5.7.5.1 const struct ltc_cipher_descriptor kasumi_desc

Initial value:

```
 {
   "kasumi",
   21,
   16, 16, 8, 8,
   &kasumi_setup,
   &kasumi_ecb_encrypt,
   &kasumi_ecb_decrypt,
   &kasumi_test,
   &kasumi_done,
   &kasumi_keysize,
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 26 of file kasumi.c.

5.8 ciphers/khazad.c File Reference

5.8.1 Detailed Description

Khazad implementation derived from public domain source Authors: Paulo S.L.M.

Barreto and Vincent Rijmen.

Definition in file khazad.c.

```
#include "tomcrypt.h"
```

Include dependency graph for khazad.c:

Defines

- #define R 8
- #define KEYSIZE 128
- #define KEYSIZEB (KEYSIZE/8)
- #define BLOCKSIZE 64
- #define BLOCKSIZEB (BLOCKSIZE/8)

Functions

- int khazad_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the Khazad block cipher.

- static void khazad_crypt (const unsigned char ∗plaintext, unsigned char ∗ciphertext, const ulong64 ∗roundKey)
- int khazad_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with Khazad.

- int khazad_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with Khazad.

- int khazad_test (void)

  Performs a self-test of the Khazad block cipher.

- void khazad_done (symmetric_key ∗skey)

  Terminate the context.

- int khazad_keysize (int ∗keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor khazad_desc
- static const ulong64 T0 [256]
- static const ulong64 T1 [256]
- static const ulong64 T2 [256]

- static const ulong64 T3 [256]
- static const ulong64 T4 [256]
- static const ulong64 T5 [256]
- static const ulong64 T6 [256]
- static const ulong64 T7 [256]
- static const ulong64 c [R+1]

5.8.2 Define Documentation

5.8.2.1 #define BLOCKSIZE 64

Definition at line 37 of file khazad.c.

5.8.2.2 #define BLOCKSIZEB (BLOCKSIZE/8)

Definition at line 38 of file khazad.c.

5.8.2.3 #define KEYSIZE 128

Definition at line 35 of file khazad.c.

5.8.2.4 #define KEYSIZEB (KEYSIZE/8)

Definition at line 36 of file khazad.c.

5.8.2.5 #define R 8

Definition at line 34 of file khazad.c.

Referenced by anubis_setup(), blowfish_ecb_decrypt(), blowfish_ecb_encrypt(), cast5_ecb_decrypt(), cast5_ecb_encrypt(), and khazad_setup().

5.8.3 Function Documentation

5.8.3.1 static void khazad_crypt (const unsigned char ∗ *plaintext*, unsigned char ∗ *ciphertext*, const ulong64 ∗ *roundKey*) [static]

Definition at line 677 of file khazad.c.

References T0, T1, T2, T3, T4, T5, T6, and T7.

Referenced by khazad_ecb_decrypt(), and khazad_ecb_encrypt().

```
                                                                {
   int      r;
   ulong64 state;
    /*
    * map plaintext block to cipher state (mu)
    * and add initial round key (sigma[K^0]):
    */
   state =
      ((ulong64)plaintext[0] << 56) ^
```

```
      ((ulong64)plaintext[1] << 48) ^
      ((ulong64)plaintext[2] << 40) ^
      ((ulong64)plaintext[3] << 32) ^
      ((ulong64)plaintext[4] << 24) ^
      ((ulong64)plaintext[5] << 16) ^
      ((ulong64)plaintext[6] <<  8) ^
      ((ulong64)plaintext[7]      ) ^
      roundKey[0];

    /*
    * R - 1 full rounds:
    */
    for (r = 1; r < R; r++) {
      state =
         T0[(int)(state >> 56)       ] ^
         T1[(int)(state >> 48) & 0xff] ^
         T2[(int)(state >> 40) & 0xff] ^
         T3[(int)(state >> 32) & 0xff] ^
         T4[(int)(state >> 24) & 0xff] ^
         T5[(int)(state >> 16) & 0xff] ^
         T6[(int)(state >>  8) & 0xff] ^
         T7[(int)(state      ) & 0xff] ^
         roundKey[r];
    }

    /*
    * last round:
    */
   state =
      (T0[(int)(state >> 56)       ] & CONST64(0xff00000000000000)) ^
      (T1[(int)(state >> 48) & 0xff] & CONST64(0x00ff000000000000)) ^
      (T2[(int)(state >> 40) & 0xff] & CONST64(0x0000ff0000000000)) ^
      (T3[(int)(state >> 32) & 0xff] & CONST64(0x000000ff00000000)) ^
      (T4[(int)(state >> 24) & 0xff] & CONST64(0x00000000ff000000)) ^
      (T5[(int)(state >> 16) & 0xff] & CONST64(0x0000000000ff0000)) ^
      (T6[(int)(state >>  8) & 0xff] & CONST64(0x000000000000ff00)) ^
      (T7[(int)(state      ) & 0xff] & CONST64(0x00000000000000ff)) ^
      roundKey[R];

   /*
    * map cipher state to ciphertext block (mu^{-1}):
    */
   ciphertext[0] = (unsigned char)(state >> 56);
   ciphertext[1] = (unsigned char)(state >> 48);
   ciphertext[2] = (unsigned char)(state >> 40);
   ciphertext[3] = (unsigned char)(state >> 32);
   ciphertext[4] = (unsigned char)(state >> 24);
   ciphertext[5] = (unsigned char)(state >> 16);
   ciphertext[6] = (unsigned char)(state >>  8);
   ciphertext[7] = (unsigned char)(state      );
}
```

5.8.3.2 void khazad_done (symmetric_key ∗ *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 831 of file khazad.c.

```
{
}
```

5.8.3.3 int khazad_ecb_decrypt (const unsigned char * *ct*, unsigned char * *pt*, symmetric_key * *skey*)

Decrypts a block of text with Khazad.

Parameters:

ct The input ciphertext (8 bytes)

pt The output plaintext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 762 of file khazad.c.

References CRYPT_OK, khazad_crypt(), and LTC_ARGCHK.

Referenced by khazad_test().

```
{
   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);
   khazad_crypt(ct, pt, skey->khazad.roundKeyDec);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.8.3.4 int khazad_ecb_encrypt (const unsigned char * *pt*, unsigned char * *ct*, symmetric_key * *skey*)

Encrypts a block of text with Khazad.

Parameters:

pt The input plaintext (8 bytes)

ct The output ciphertext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 746 of file khazad.c.

References CRYPT_OK, khazad_crypt(), and LTC_ARGCHK.

Referenced by khazad_test().

```
{
   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);
   khazad_crypt(pt, ct, skey->khazad.roundKeyEnc);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.8.3.5 int khazad_keysize (int * *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 840 of file khazad.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize >= 16) {
      *keysize = 16;
      return CRYPT_OK;
   } else {
      return CRYPT_INVALID_KEYSIZE;
   }
}
```

5.8.3.6 int khazad_setup (const unsigned char * *key*, int *keylen*, int *num_rounds*, symmetric_key * *skey*)

Initialize the Khazad block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 596 of file khazad.c.

References c, CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, LTC_ARGCHK, R, S, T0, T1, T2, T3, T4, T5, T6, and T7.

Referenced by khazad_test().

```
{
   int               r;
   const ulong64    *S;
   ulong64           K2, K1;

   LTC_ARGCHK(key  != NULL);
   LTC_ARGCHK(skey != NULL);
   if (keylen != 16) {
      return CRYPT_INVALID_KEYSIZE;
   }
```

```
   if (num_rounds != 8 && num_rounds != 0) {
      return CRYPT_INVALID_ROUNDS;
   }

   /* use 7th table */
   S = T7;

    /*
    * map unsigned char array cipher key to initial key state (mu):
    */
   K2 =
      ((ulong64)key[ 0] << 56) ^
      ((ulong64)key[ 1] << 48) ^
      ((ulong64)key[ 2] << 40) ^
      ((ulong64)key[ 3] << 32) ^
      ((ulong64)key[ 4] << 24) ^
      ((ulong64)key[ 5] << 16) ^
      ((ulong64)key[ 6] <<  8) ^
      ((ulong64)key[ 7]      );
   K1 =
      ((ulong64)key[ 8] << 56) ^
      ((ulong64)key[ 9] << 48) ^
      ((ulong64)key[10] << 40) ^
      ((ulong64)key[11] << 32) ^
      ((ulong64)key[12] << 24) ^
      ((ulong64)key[13] << 16) ^
      ((ulong64)key[14] <<  8) ^
      ((ulong64)key[15]      );

   /*
    * compute the round keys:
    */
   for (r = 0; r <= R; r++) {
      /*
       * K[r] = rho(c[r], K1) ^ K2;
       */
      skey->khazad.roundKeyEnc[r] =
         T0[(int)(K1 >> 56)       ] ^
         T1[(int)(K1 >> 48) & 0xff] ^
         T2[(int)(K1 >> 40) & 0xff] ^
         T3[(int)(K1 >> 32) & 0xff] ^
         T4[(int)(K1 >> 24) & 0xff] ^
         T5[(int)(K1 >> 16) & 0xff] ^
         T6[(int)(K1 >>  8) & 0xff] ^
         T7[(int)(K1      ) & 0xff] ^
         c[r] ^ K2;
      K2 = K1; K1 = skey->khazad.roundKeyEnc[r];
   }
   /*
    * compute the inverse key schedule:
    * K'^0 = K^R, K'^R = K^0, K'^r = theta(K^{R-r})
    */
   skey->khazad.roundKeyDec[0] = skey->khazad.roundKeyEnc[R];
   for (r = 1; r < R; r++) {
      K1 = skey->khazad.roundKeyEnc[R - r];
      skey->khazad.roundKeyDec[r] =
         T0[(int)S[(int)(K1 >> 56)       ] & 0xff] ^
         T1[(int)S[(int)(K1 >> 48) & 0xff] & 0xff] ^
         T2[(int)S[(int)(K1 >> 40) & 0xff] & 0xff] ^
         T3[(int)S[(int)(K1 >> 32) & 0xff] & 0xff] ^
         T4[(int)S[(int)(K1 >> 24) & 0xff] & 0xff] ^
         T5[(int)S[(int)(K1 >> 16) & 0xff] & 0xff] ^
         T6[(int)S[(int)(K1 >>  8) & 0xff] & 0xff] ^
         T7[(int)S[(int)(K1      ) & 0xff] & 0xff];
   }
   skey->khazad.roundKeyDec[R] = skey->khazad.roundKeyEnc[0];

```

```
   return CRYPT_OK;
}
```

5.8.3.7 int khazad_test (void)

Performs a self-test of the Khazad block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 775 of file khazad.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, khazad_ecb_decrypt(), khazad_ecb_encrypt(), khazad_setup(), and XMEMCMP.

```
{
#ifndef LTC_TEST
  return CRYPT_NOP;
#else
  static const struct test {
     unsigned char pt[8], ct[8], key[16];
  } tests[] = {
{
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x49, 0xA4, 0xCE, 0x32, 0xAC, 0x19, 0x0E, 0x3F },
   { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x64, 0x5D, 0x77, 0x3E, 0x40, 0xAB, 0xDD, 0x53 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 }
}, {
   { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
   { 0x9E, 0x39, 0x98, 0x64, 0xF7, 0x8E, 0xCA, 0x02 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}, {
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
   { 0xA9, 0xDF, 0x3D, 0x2C, 0x64, 0xD3, 0xEA, 0x28 },
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
}
};
   int x, y;
   unsigned char buf[2][8];
   symmetric_key skey;

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       khazad_setup(tests[x].key, 16, 0, &skey);
       khazad_ecb_encrypt(tests[x].pt, buf[0], &skey);
       khazad_ecb_decrypt(buf[0], buf[1], &skey);
       if (XMEMCMP(buf[0], tests[x].ct, 8) || XMEMCMP(buf[1], tests[x].pt, 8)) {
          return CRYPT_FAIL_TESTVECTOR;
       }

       for (y = 0; y < 1000; y++) khazad_ecb_encrypt(buf[0], buf[0], &skey);
       for (y = 0; y < 1000; y++) khazad_ecb_decrypt(buf[0], buf[0], &skey);
       if (XMEMCMP(buf[0], tests[x].ct, 8)) {
          return CRYPT_FAIL_TESTVECTOR;
       }

```

```
   }
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.8.4 Variable Documentation

5.8.4.1 const ulong64 c[R+1] [static]

Initial value:

```
 {
    CONST64(0xba542f7453d3d24d),
    CONST64(0x50ac8dbf70529a4c),
    CONST64(0xead597d133515ba6),
    CONST64(0xde48a899db32b7fc),
    CONST64(0xe39e919be2bb416e),
    CONST64(0xa5cb6b95a1f3b102),
    CONST64(0xccc41d14c363da5d),
    CONST64(0x5fdc7dcd7f5a6c5c),
    CONST64(0xf726ffede89d6f8e),
}
```

Definition at line 576 of file khazad.c.

Referenced by base64_decode(), der_object_identifier_bits(), khazad_setup(), md4_compress(), md5_-compress(), noekeon_ecb_decrypt(), noekeon_ecb_encrypt(), ocb_ntz(), pmac_ntz(), rc6_ecb_decrypt(), rc6_ecb_encrypt(), safer_ecb_decrypt(), safer_ecb_encrypt(), sha1_compress(), sober128_add_entropy(), sober128_read(), sober128_start(), tiger_compress(), twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.8.4.2 const struct ltc_cipher_descriptor khazad_desc

Initial value:

```
 {
   "khazad",
   18,
   16, 16, 8, 8,
   &khazad_setup,
   &khazad_ecb_encrypt,
   &khazad_ecb_decrypt,
   &khazad_test,
   &khazad_done,
   &khazad_keysize,
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 21 of file khazad.c.

Referenced by yarrow_start().

5.8.4.3 const ulong64 T0[256] [static]

Definition at line 40 of file khazad.c.

5.8.4.4 const ulong64 T1[256] [static]

Definition at line 107 of file khazad.c.

5.8.4.5 const ulong64 T2[256] [static]

Definition at line 174 of file khazad.c.

5.8.4.6 const ulong64 T3[256] [static]

Definition at line 241 of file khazad.c.

5.8.4.7 const ulong64 T4[256] [static]

Definition at line 308 of file khazad.c.

5.8.4.8 const ulong64 T5[256] [static]

Definition at line 375 of file khazad.c.

5.8.4.9 const ulong64 T6[256] [static]

Definition at line 442 of file khazad.c.

Referenced by khazad_crypt(), and khazad_setup().

5.8.4.10 const ulong64 T7[256] [static]

Definition at line 509 of file khazad.c.

Referenced by khazad_crypt(), and khazad_setup().

5.9 ciphers/kseed.c File Reference

5.9.1 Detailed Description

seed implementation of SEED derived from RFC4269 Tom St Denis

Definition in file kseed.c.

```
#include "tomcrypt.h"
```

Include dependency graph for kseed.c:

Defines

- #define G(x) (SS3[((x)>>24)&255] ^ SS2[((x)>>16)&255] ^ SS1[((x)>>8)&255] ^ SS0[(x)&255])
- #define F(L1, L2, R1, R2, K1, K2)

Functions

- int kseed_setup (const unsigned char *key, int keylen, int num_rounds, symmetric_key *skey)

  Initialize the SEED block cipher.
- static void rounds (ulong32 *P, ulong32 *K)
- int kseed_ecb_encrypt (const unsigned char *pt, unsigned char *ct, symmetric_key *skey)

  Encrypts a block of text with SEED.
- int kseed_ecb_decrypt (const unsigned char *ct, unsigned char *pt, symmetric_key *skey)

  Decrypts a block of text with SEED.
- void kseed_done (symmetric_key *skey)

  Terminate the context.
- int kseed_test (void)

  Performs a self-test of the SEED block cipher.
- int kseed_keysize (int *keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor kseed_desc
- static const ulong32 SS0 [256]
- static const ulong32 SS1 [256]
- static const ulong32 SS2 [256]
- static const ulong32 SS3 [256]
- static const ulong32 KCi [16]

5.9.2 Define Documentation

5.9.2.1 #define F(L1, L2, R1, R2, K1, K2)

Value:

```
T2 = G((R1 ^ K1) ^ (R2 ^ K2)); \
   T = G( G(T2 + (R1 ^ K1)) + T2); \
   L2 ^= T; \
   L1 ^= (T + G(T2 + (R1 ^ K1))); \
```

Definition at line 188 of file kseed.c.

5.9.2.2 #define G(x) (SS3[((x)>>24)&255] ^ SS2[((x)>>16)&255] ^ SS1[((x)>>8)&255] ^ SS0[(x)&255])

Definition at line 186 of file kseed.c.

Referenced by ecc_test(), and kseed_setup().

5.9.3 Function Documentation

5.9.3.1 void kseed_done (symmetric_key ∗ *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 299 of file kseed.c.

```
300 {
301 }
```

5.9.3.2 int kseed_ecb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, symmetric_key ∗ *skey*)

Decrypts a block of text with SEED.

Parameters:

ct The input ciphertext (16 bytes)

pt The output plaintext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 281 of file kseed.c.

References CRYPT_OK, and rounds().

Referenced by kseed_test().

```
{
  ulong32 P[4];
  LOAD32H(P[0], ct);
  LOAD32H(P[1], ct+4);
  LOAD32H(P[2], ct+8);
  LOAD32H(P[3], ct+12);
  rounds(P, skey->kseed.dK);
  STORE32H(P[2], pt);
  STORE32H(P[3], pt+4);
  STORE32H(P[0], pt+8);
  STORE32H(P[1], pt+12);
  return CRYPT_OK;
}
```

Here is the call graph for this function:

5.9.3.3 int kseed_ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with SEED.

Parameters:

pt The input plaintext (16 bytes)

ct The output ciphertext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 259 of file kseed.c.

References CRYPT_OK, and rounds().

Referenced by kseed_test().

```
{
  ulong32 P[4];
  LOAD32H(P[0], pt);
  LOAD32H(P[1], pt+4);
  LOAD32H(P[2], pt+8);
  LOAD32H(P[3], pt+12);
  rounds(P, skey->kseed.K);
  STORE32H(P[2], ct);
  STORE32H(P[3], ct+4);
  STORE32H(P[0], ct+8);
  STORE32H(P[1], ct+12);
  return CRYPT_OK;
}
```

Here is the call graph for this function:

5.9.3.4 int kseed_keysize (int ∗ *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 361 of file kseed.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize >= 16) {
      *keysize = 16;
   } else {
      return CRYPT_INVALID_KEYSIZE;
   }
   return CRYPT_OK;
}
```

5.9.3.5 int kseed_setup (const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_key ∗ *skey*)

Initialize the SEED block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 202 of file kseed.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, G, and KCi.

Referenced by kseed_test().

```
{
    int     i;
    ulong32 tmp, k1, k2, k3, k4;

    if (keylen != 16) {
       return CRYPT_INVALID_KEYSIZE;
    }

    if (num_rounds != 16 && num_rounds != 0) {
       return CRYPT_INVALID_ROUNDS;
    }

    /* load key */
    LOAD32H(k1, key);
    LOAD32H(k2, key+4);
    LOAD32H(k3, key+8);
    LOAD32H(k4, key+12);

    for (i = 0; i < 16; i++) {
       skey->kseed.K[2*i+0] = G(k1 + k3 - KCi[i]);
       skey->kseed.K[2*i+1] = G(k2 - k4 + KCi[i]);
```

```
        if (i&1) {
           tmp = k3;
           k3 = ((k3 << 8) | (k4 >> 24)) & 0xFFFFFFFF;
           k4 = ((k4 << 8) | (tmp >> 24)) & 0xFFFFFFFF;
        } else {
           tmp = k1;
           k1 = ((k1 >> 8) | (k2 << 24)) & 0xFFFFFFFF;
           k2 = ((k2 >> 8) | (tmp << 24)) & 0xFFFFFFFF;
       }
       /* reverse keys for decrypt */
       skey->kseed.dK[2*(15-i)+0] = skey->kseed.K[2*i+0];
       skey->kseed.dK[2*(15-i)+1] = skey->kseed.K[2*i+1];
    }

    return CRYPT_OK;
}
```

5.9.3.6 int kseed_test (void)

Performs a self-test of the SEED block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 307 of file kseed.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, kseed_ecb_decrypt(), kseed_ecb_encrypt(), kseed_setup(), and XMEMCMP.

```
{
#if !defined(LTC_TEST)
  return CRYPT_NOP;
#else
  static const struct test {
     unsigned char pt[16], ct[16], key[16];
  } tests[] = {

{
  { 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F },
  { 0x5E,0xBA,0xC6,0xE0,0x05,0x4E,0x16,0x68,0x19,0xAF,0xF1,0xCC,0x6D,0x34,0x6C,0xDB },
  { 0 },
},

{
  { 0 },
  { 0xC1,0x1F,0x22,0xF2,0x01,0x40,0x50,0x50,0x84,0x48,0x35,0x97,0xE4,0x37,0x0F,0x43 },
  { 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F },
},

{
  { 0x83,0xA2,0xF8,0xA2,0x88,0x64,0x1F,0xB9,0xA4,0xE9,0xA5,0xCC,0x2F,0x13,0x1C,0x7D },
  { 0xEE,0x54,0xD1,0x3E,0xBC,0xAE,0x70,0x6D,0x22,0x6B,0xC3,0x14,0x2C,0xD4,0x0D,0x4A },
  { 0x47,0x06,0x48,0x08,0x51,0xE6,0x1B,0xE8,0x5D,0x74,0xBF,0xB3,0xFD,0x95,0x61,0x85 },
},

{
  { 0xB4,0x1E,0x6B,0xE2,0xEB,0xA8,0x4A,0x14,0x8E,0x2E,0xED,0x84,0x59,0x3C,0x5E,0xC7 },
  { 0x9B,0x9B,0x7B,0xFC,0xD1,0x81,0x3C,0xB9,0x5D,0x0B,0x36,0x18,0xF4,0x0F,0x51,0x22 },
  { 0x28,0xDB,0xC3,0xBC,0x49,0xFF,0xD8,0x7D,0xCF,0xA5,0x09,0xB1,0x1D,0x42,0x2B,0xE7 },
}
};
   int x;
```

```
    unsigned char buf[2][16];
    symmetric_key skey;

    for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
        kseed_setup(tests[x].key, 16, 0, &skey);
        kseed_ecb_encrypt(tests[x].pt, buf[0], &skey);
        kseed_ecb_decrypt(buf[0], buf[1], &skey);
        if (XMEMCMP(buf[0], tests[x].ct, 16) || XMEMCMP(buf[1], tests[x].pt, 16)) {
           return CRYPT_FAIL_TESTVECTOR;
        }
    }
    return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.9.3.7 static void rounds (ulong32 ∗ *P*, ulong32 ∗ *K*) [static]

Definition at line 241 of file kseed.c.

References F, and T2.

Referenced by kseed_ecb_decrypt(), kseed_ecb_encrypt(), and saferp_setup().

```
{
    ulong32 T, T2;
    int     i;
    for (i = 0; i < 16; i += 2) {
      F(P[0], P[1], P[2], P[3], K[0], K[1]);
      F(P[2], P[3], P[0], P[1], K[2], K[3]);
      K += 4;
    }
}
```

5.9.4 Variable Documentation

5.9.4.1 const ulong32 KCi[16] [static]

Initial value:

```
 {
0x9E3779B9,0x3C6EF373,
0x78DDE6E6,0xF1BBCDCC,
0xE3779B99,0xC6EF3733,
0x8DDE6E67,0x1BBCDCCF,
0x3779B99E,0x6EF3733C,
0xDDE6E678,0xBBCDCCF1,
0x779B99E3,0xEF3733C6,
0xDE6E678D,0xBCDCCF1B
}
```

Definition at line 175 of file kseed.c.

Referenced by kseed_setup().

5.9.4.2 const struct ltc_cipher_descriptor kseed_desc

Initial value:

```
{
   "seed",
   20,
   16, 16, 16, 16,
   &kseed_setup,
   &kseed_ecb_encrypt,
   &kseed_ecb_decrypt,
   &kseed_test,
   &kseed_done,
   &kseed_keysize,
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 22 of file kseed.c.

Referenced by yarrow_start().

5.9.4.3 const ulong32 SS0[256] `[static]`

Definition at line 35 of file kseed.c.

5.9.4.4 const ulong32 SS1[256] `[static]`

Definition at line 70 of file kseed.c.

5.9.4.5 const ulong32 SS2[256] `[static]`

Definition at line 105 of file kseed.c.

5.9.4.6 const ulong32 SS3[256] `[static]`

Definition at line 140 of file kseed.c.

5.10 ciphers/noekeon.c File Reference

5.10.1 Detailed Description

Implementation of the Noekeon block cipher by Tom St Denis.

Definition in file noekeon.c.

```
#include "tomcrypt.h"
```

Include dependency graph for noekeon.c:

Defines

- #define kTHETA(a, b, c, d)
- #define THETA(k, a, b, c, d)
- #define GAMMA(a, b, c, d)
- #define PI1(a, b, c, d) a = ROLc(a, 1); c = ROLc(c, 5); d = ROLc(d, 2);
- #define PI2(a, b, c, d) a = RORc(a, 1); c = RORc(c, 5); d = RORc(d, 2);
- #define ROUND(i)
- #define ROUND(i)

Functions

- int noekeon_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the Noekeon block cipher.

- int noekeon_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with Noekeon.

- int noekeon_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with Noekeon.

- int noekeon_test (void)

  Performs a self-test of the Noekeon block cipher.

- void noekeon_done (symmetric_key ∗skey)

  Terminate the context.

- int noekeon_keysize (int ∗keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor noekeon_desc
- static const ulong32 RC [ ]

5.10.2 Define Documentation

5.10.2.1 #define GAMMA(a, b, c, d)

Value:

```
b ^= ~(d|c);              \
    a ^= c&b;                 \
    temp = d; d = a; a = temp;\
    c ^= a ^ b ^ d;           \
    b ^= ~(d|c);              \
    a ^= c&b;
```

Definition at line 53 of file noekeon.c.

5.10.2.2 #define kTHETA(a, b, c, d)

Value:

```
temp = a^c; temp = temp ^ ROLc(temp, 8) ^ RORc(temp, 8); \
    b ^= temp; d ^= temp;                                  \
    temp = b^d; temp = temp ^ ROLc(temp, 8) ^ RORc(temp, 8); \
    a ^= temp; c ^= temp;
```

Definition at line 41 of file noekeon.c.

Referenced by noekeon_setup().

5.10.2.3 #define PI1(a, b, c, d) a = ROLc(a, 1); c = ROLc(c, 5); d = ROLc(d, 2);

Definition at line 61 of file noekeon.c.

5.10.2.4 #define PI2(a, b, c, d) a = RORc(a, 1); c = RORc(c, 5); d = RORc(d, 2);

Definition at line 64 of file noekeon.c.

5.10.2.5 #define ROUND(i)

Value:

```
THETA(skey->noekeon.dK, a,b,c,d); \
       a ^= RC[i]; \
       PI1(a,b,c,d); \
       GAMMA(a,b,c,d); \
       PI2(a,b,c,d);
```

5.10.2.6 #define ROUND(i)

Value:

```
a ^= RC[i]; \
       THETA(skey->noekeon.K, a,b,c,d); \
       PI1(a,b,c,d); \
       GAMMA(a,b,c,d); \
       PI2(a,b,c,d);
```

Referenced by noekeon_ecb_decrypt(), and noekeon_ecb_encrypt().

5.10.2.7 #define THETA(k, a, b, c, d)

Value:

```
temp = a^c; temp = temp ^ ROLc(temp, 8) ^ RORc(temp, 8); \
    b ^= temp ^ k[1]; d ^= temp ^ k[3];                  \
    temp = b^d; temp = temp ^ ROLc(temp, 8) ^ RORc(temp, 8); \
    a ^= temp ^ k[0]; c ^= temp ^ k[2];
```

Definition at line 47 of file noekeon.c.

5.10.3 Function Documentation

5.10.3.1 void noekeon_done (symmetric_key * *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 278 of file noekeon.c.

```
279 {
280 }
```

5.10.3.2 int noekeon_ecb_decrypt (const unsigned char * *ct*, unsigned char * *pt*, symmetric_key * *skey*)

Decrypts a block of text with Noekeon.

Parameters:

ct The input ciphertext (16 bytes)

pt The output plaintext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 169 of file noekeon.c.

References c, LTC_ARGCHK, and ROUND.

```
{
   ulong32 a,b,c,d, temp;
   int r;

   LTC_ARGCHK(skey != NULL);
   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);

   LOAD32H(a,&ct[0]); LOAD32H(b,&ct[4]);
   LOAD32H(c,&ct[8]); LOAD32H(d,&ct[12]);


#define ROUND(i) \
       THETA(skey->noekeon.dK, a,b,c,d); \
       a ^= RC[i]; \
       PI1(a,b,c,d); \
       GAMMA(a,b,c,d); \
       PI2(a,b,c,d);

   for (r = 16; r > 0; --r) {
       ROUND(r);
   }

#undef ROUND

   THETA(skey->noekeon.dK, a,b,c,d);
   a ^= RC[0];
   STORE32H(a,&pt[0]); STORE32H(b, &pt[4]);
   STORE32H(c,&pt[8]); STORE32H(d, &pt[12]);
   return CRYPT_OK;
}
```

5.10.3.3 int noekeon_ecb_encrypt (const unsigned char * *pt*, unsigned char * *ct*, symmetric_key * *skey*)

Encrypts a block of text with Noekeon.

Parameters:

pt The input plaintext (16 bytes)

ct The output ciphertext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 115 of file noekeon.c.

References c, LTC_ARGCHK, and ROUND.

```
{
   ulong32 a,b,c,d,temp;
   int r;

   LTC_ARGCHK(skey != NULL);
   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);

   LOAD32H(a,&pt[0]); LOAD32H(b,&pt[4]);
   LOAD32H(c,&pt[8]); LOAD32H(d,&pt[12]);

```

```
#define ROUND(i) \
       a ^= RC[i]; \
       THETA(skey->noekeon.K, a,b,c,d); \
       PI1(a,b,c,d); \
       GAMMA(a,b,c,d); \
       PI2(a,b,c,d);

   for (r = 0; r < 16; ++r) {
       ROUND(r);
   }

#undef ROUND

   a ^= RC[16];
   THETA(skey->noekeon.K, a, b, c, d);

   STORE32H(a,&ct[0]); STORE32H(b,&ct[4]);
   STORE32H(c,&ct[8]); STORE32H(d,&ct[12]);

   return CRYPT_OK;
}
```

5.10.3.4 int noekeon_keysize (int * *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 287 of file noekeon.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize < 16) {
      return CRYPT_INVALID_KEYSIZE;
   } else {
      *keysize = 16;
      return CRYPT_OK;
   }
}
```

5.10.3.5 int noekeon_setup (const unsigned char * *key*, int *keylen*, int *num_rounds*, symmetric_key * *skey*)

Initialize the Noekeon block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 75 of file noekeon.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, CRYPT_OK, kTHETA, and LTC_ARGCHK.

Referenced by noekeon_test().

```
{
   ulong32 temp;

   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(skey != NULL);

   if (keylen != 16) {
      return CRYPT_INVALID_KEYSIZE;
   }

   if (num_rounds != 16 && num_rounds != 0) {
      return CRYPT_INVALID_ROUNDS;
   }

   LOAD32H(skey->noekeon.K[0],&key[0]);
   LOAD32H(skey->noekeon.K[1],&key[4]);
   LOAD32H(skey->noekeon.K[2],&key[8]);
   LOAD32H(skey->noekeon.K[3],&key[12]);

   LOAD32H(skey->noekeon.dK[0],&key[0]);
   LOAD32H(skey->noekeon.dK[1],&key[4]);
   LOAD32H(skey->noekeon.dK[2],&key[8]);
   LOAD32H(skey->noekeon.dK[3],&key[12]);

   kTHETA(skey->noekeon.dK[0], skey->noekeon.dK[1], skey->noekeon.dK[2], skey->noekeon.dK[3]);

   return CRYPT_OK;
}
```

5.10.3.6 int noekeon_test (void)

Performs a self-test of the Noekeon block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 216 of file noekeon.c.

References CRYPT_NOP, CRYPT_OK, noekeon_setup(), and zeromem().

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
static const struct {
    int keylen;
    unsigned char key[16], pt[16], ct[16];
} tests[] = {
```

```
   {
      16,
      { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 },
      { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 },
      { 0x18, 0xa6, 0xec, 0xe5, 0x28, 0xaa, 0x79, 0x73,
        0x28, 0xb2, 0xc0, 0x91, 0xa0, 0x2f, 0x54, 0xc5}
   }
 };
 symmetric_key key;
 unsigned char tmp[2][16];
 int err, i, y;

 for (i = 0; i < (int)(sizeof(tests)/sizeof(tests[0])); i++) {
    zeromem(&key, sizeof(key));
    if ((err = noekeon_setup(tests[i].key, tests[i].keylen, 0, &key)) != CRYPT_OK) {
       return err;
    }

    noekeon_ecb_encrypt(tests[i].pt, tmp[0], &key);
    noekeon_ecb_decrypt(tmp[0], tmp[1], &key);
    if (XMEMCMP(tmp[0], tests[i].ct, 16) || XMEMCMP(tmp[1], tests[i].pt, 16)) {
#if 0
       printf("\n\nTest %d failed\n", i);
       if (XMEMCMP(tmp[0], tests[i].ct, 16)) {
          printf("CT: ");
          for (i = 0; i < 16; i++) {
             printf("%02x ", tmp[0][i]);
          }
          printf("\n");
       } else {
          printf("PT: ");
          for (i = 0; i < 16; i++) {
             printf("%02x ", tmp[1][i]);
          }
          printf("\n");
       }
#endif
        return CRYPT_FAIL_TESTVECTOR;
    }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 16; y++) tmp[0][y] = 0;
      for (y = 0; y < 1000; y++) noekeon_ecb_encrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 1000; y++) noekeon_ecb_decrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 16; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
 }
 return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.10.4 Variable Documentation

5.10.4.1 const struct ltc_cipher_descriptor noekeon_desc

Initial value:

```
{
    "noekeon",
    16,
    16, 16, 16, 16,
    &noekeon_setup,
    &noekeon_ecb_encrypt,
```

```
    &noekeon_ecb_decrypt,
    &noekeon_test,
    &noekeon_done,
    &noekeon_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 19 of file noekeon.c.

Referenced by yarrow_start().

5.10.4.2 const ulong32 RC[ ] [static]

Initial value:

```
 {
   0x00000080UL, 0x0000001bUL, 0x00000036UL, 0x0000006cUL,
   0x000000d8UL, 0x000000abUL, 0x0000004dUL, 0x0000009aUL,
   0x0000002fUL, 0x0000005eUL, 0x000000bcUL, 0x00000063UL,
   0x000000c6UL, 0x00000097UL, 0x00000035UL, 0x0000006aUL,
   0x000000d4UL
}
```

Definition at line 33 of file noekeon.c.

5.11 ciphers/rc2.c File Reference

5.11.1 Detailed Description

Implementation of RC2.

Definition in file rc2.c.

```
#include <tomcrypt.h>
```

Include dependency graph for rc2.c:

Functions

- int rc2_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)
  Initialize the RC2 block cipher.
- int rc2_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)
  Encrypts a block of text with RC2.
- int rc2_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)
  Decrypts a block of text with RC2.
- int rc2_test (void)
  Performs a self-test of the RC2 block cipher.
- void rc2_done (symmetric_key ∗skey)
  Terminate the context.
- int rc2_keysize (int ∗keysize)
  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor rc2_desc
- static const unsigned char permute [256]

5.11.2 Function Documentation

5.11.2.1 void rc2_done (symmetric_key ∗ *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 335 of file rc2.c.

```
{
}
```

5.11.2.2 int rc2_ecb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, symmetric_key ∗ *skey*)

Decrypts a block of text with RC2.

Parameters:

ct The input ciphertext (8 bytes)

pt The output plaintext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 213 of file rc2.c.

References LTC_ARGCHK.

```
{
    unsigned x76, x54, x32, x10;
    unsigned *xkey;
    int i;

    LTC_ARGCHK(pt  != NULL);
    LTC_ARGCHK(ct != NULL);
    LTC_ARGCHK(skey   != NULL);

    xkey = skey->rc2.xkey;

    x76 = ((unsigned)ct[7] << 8) + (unsigned)ct[6];
    x54 = ((unsigned)ct[5] << 8) + (unsigned)ct[4];
    x32 = ((unsigned)ct[3] << 8) + (unsigned)ct[2];
    x10 = ((unsigned)ct[1] << 8) + (unsigned)ct[0];

    for (i = 15; i >= 0; i--) {
        if (i == 4 || i == 10) {
            x76 = (x76 - xkey[x54 & 63]) & 0xFFFF;
            x54 = (x54 - xkey[x32 & 63]) & 0xFFFF;
            x32 = (x32 - xkey[x10 & 63]) & 0xFFFF;
            x10 = (x10 - xkey[x76 & 63]) & 0xFFFF;
        }

        x76 = ((x76 << 11) | (x76 >> 5));
        x76 = (x76 - ((x10 & ~x54) + (x32 & x54) + xkey[4*i+3])) & 0xFFFF;

        x54 = ((x54 << 13) | (x54 >> 3));
        x54 = (x54 - ((x76 & ~x32) + (x10 & x32) + xkey[4*i+2])) & 0xFFFF;

        x32 = ((x32 << 14) | (x32 >> 2));
        x32 = (x32 - ((x54 & ~x10) + (x76 & x10) + xkey[4*i+1])) & 0xFFFF;

        x10 = ((x10 << 15) | (x10 >> 1));
        x10 = (x10 - ((x32 & ~x76) + (x54 & x76) + xkey[4*i+0])) & 0xFFFF;
    }

    pt[0] = (unsigned char)x10;
    pt[1] = (unsigned char)(x10 >> 8);
    pt[2] = (unsigned char)x32;
    pt[3] = (unsigned char)(x32 >> 8);
    pt[4] = (unsigned char)x54;
    pt[5] = (unsigned char)(x54 >> 8);
    pt[6] = (unsigned char)x76;
    pt[7] = (unsigned char)(x76 >> 8);

    return CRYPT_OK;
}
```

5.11.2.3 int rc2_ecb_encrypt (const unsigned char * *pt*, unsigned char * *ct*, symmetric_key * *skey*)

Encrypts a block of text with RC2.

Parameters:

pt The input plaintext (8 bytes)

ct The output ciphertext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 135 of file rc2.c.

References LTC_ARGCHK.

```
{
    unsigned *xkey;
    unsigned x76, x54, x32, x10, i;

    LTC_ARGCHK(pt  != NULL);
    LTC_ARGCHK(ct != NULL);
    LTC_ARGCHK(skey   != NULL);

    xkey = skey->rc2.xkey;

    x76 = ((unsigned)pt[7] << 8) + (unsigned)pt[6];
    x54 = ((unsigned)pt[5] << 8) + (unsigned)pt[4];
    x32 = ((unsigned)pt[3] << 8) + (unsigned)pt[2];
    x10 = ((unsigned)pt[1] << 8) + (unsigned)pt[0];

    for (i = 0; i < 16; i++) {
        x10 = (x10 + (x32 & ~x76) + (x54 & x76) + xkey[4*i+0]) & 0xFFFF;
        x10 = ((x10 << 1) | (x10 >> 15));

        x32 = (x32 + (x54 & ~x10) + (x76 & x10) + xkey[4*i+1]) & 0xFFFF;
        x32 = ((x32 << 2) | (x32 >> 14));

        x54 = (x54 + (x76 & ~x32) + (x10 & x32) + xkey[4*i+2]) & 0xFFFF;
        x54 = ((x54 << 3) | (x54 >> 13));

        x76 = (x76 + (x10 & ~x54) + (x32 & x54) + xkey[4*i+3]) & 0xFFFF;
        x76 = ((x76 << 5) | (x76 >> 11));

        if (i == 4 || i == 10) {
            x10 = (x10 + xkey[x76 & 63]) & 0xFFFF;
            x32 = (x32 + xkey[x10 & 63]) & 0xFFFF;
            x54 = (x54 + xkey[x32 & 63]) & 0xFFFF;
            x76 = (x76 + xkey[x54 & 63]) & 0xFFFF;
        }
    }

    ct[0] = (unsigned char)x10;
    ct[1] = (unsigned char)(x10 >> 8);
    ct[2] = (unsigned char)x32;
    ct[3] = (unsigned char)(x32 >> 8);
    ct[4] = (unsigned char)x54;
    ct[5] = (unsigned char)(x54 >> 8);
    ct[6] = (unsigned char)x76;
    ct[7] = (unsigned char)(x76 >> 8);

    return CRYPT_OK;
}
```

5.11.2.4 int rc2_keysize (int * *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 344 of file rc2.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize < 8) {
       return CRYPT_INVALID_KEYSIZE;
   } else if (*keysize > 128) {
       *keysize = 128;
   }
   return CRYPT_OK;
}
```

5.11.2.5 int rc2_setup (const unsigned char * *key*, int *keylen*, int *num_rounds*, symmetric_key * *skey*)

Initialize the RC2 block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 70 of file rc2.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, and LTC_ARGCHK.

Referenced by rc2_test().

```
{
   unsigned *xkey = skey->rc2.xkey;
   unsigned char tmp[128];
   unsigned T8, TM;
   int i, bits;

   LTC_ARGCHK(key  != NULL);
   LTC_ARGCHK(skey != NULL);

   if (keylen < 8 || keylen > 128) {
      return CRYPT_INVALID_KEYSIZE;
```

```
   }

   if (num_rounds != 0 && num_rounds != 16) {
      return CRYPT_INVALID_ROUNDS;
   }

   for (i = 0; i < keylen; i++) {
       tmp[i] = key[i] & 255;
   }

    /* Phase 1: Expand input key to 128 bytes */
    if (keylen < 128) {
        for (i = keylen; i < 128; i++) {
            tmp[i] = permute[(tmp[i - 1] + tmp[i - keylen]) & 255];
        }
    }

    /* Phase 2 - reduce effective key size to "bits" */
    bits = keylen<<3;
    T8   = (unsigned)(bits+7)>>3;
    TM   = (255 >> (unsigned)(7 & -bits));
    tmp[128 - T8] = permute[tmp[128 - T8] & TM];
    for (i = 127 - T8; i >= 0; i--) {
        tmp[i] = permute[tmp[i + 1] ^ tmp[i + T8]];
    }

    /* Phase 3 - copy to xkey in little-endian order */
    for (i = 0; i < 64; i++) {
        xkey[i] =  (unsigned)tmp[2*i] + ((unsigned)tmp[2*i+1] << 8);
    }

#ifdef LTC_CLEAN_STACK
    zeromem(tmp, sizeof(tmp));
#endif

    return CRYPT_OK;
}
```

5.11.2.6 int rc2_test (void)

Performs a self-test of the RC2 block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 281 of file rc2.c.

References CRYPT_NOP, CRYPT_OK, rc2_setup(), and zeromem().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   static const struct {
        int keylen;
        unsigned char key[16], pt[8], ct[8];
   } tests[] = {

   { 8,
     { 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
       0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
     { 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
     { 0x30, 0x64, 0x9e, 0xdf, 0x9b, 0xe7, 0xd2, 0xc2 }
```

```

   },
   { 16,
     { 0x88, 0xbc, 0xa9, 0x0e, 0x90, 0x87, 0x5a, 0x7f,
       0x0f, 0x79, 0xc3, 0x84, 0x62, 0x7b, 0xaf, 0xb2 },
     { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
     { 0x22, 0x69, 0x55, 0x2a, 0xb0, 0xf8, 0x5c, 0xa6 }
   }
  };
    int x, y, err;
    symmetric_key skey;
    unsigned char tmp[2][8];

    for (x = 0; x < (int)(sizeof(tests) / sizeof(tests[0])); x++) {
        zeromem(tmp, sizeof(tmp));
        if ((err = rc2_setup(tests[x].key, tests[x].keylen, 0, &skey)) != CRYPT_OK) {
           return err;
        }

        rc2_ecb_encrypt(tests[x].pt, tmp[0], &skey);
        rc2_ecb_decrypt(tmp[0], tmp[1], &skey);

        if (XMEMCMP(tmp[0], tests[x].ct, 8) != 0 || XMEMCMP(tmp[1], tests[x].pt, 8) != 0) {
           return CRYPT_FAIL_TESTVECTOR;
        }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 8; y++) tmp[0][y] = 0;
      for (y = 0; y < 1000; y++) rc2_ecb_encrypt(tmp[0], tmp[0], &skey);
      for (y = 0; y < 1000; y++) rc2_ecb_decrypt(tmp[0], tmp[0], &skey);
      for (y = 0; y < 8; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
    }
    return CRYPT_OK;
   #endif
}
```

Here is the call graph for this function:

5.11.3 Variable Documentation

5.11.3.1 const unsigned char permute[256] [static]

Initial value:

```
{
       217,120,249,196, 25,221,181,237, 40,233,253,121, 74,160,216,157,
       198,126, 55,131, 43,118, 83,142, 98, 76,100,136, 68,139,251,162,
        23,154, 89,245,135,179, 79, 19, 97, 69,109,141,  9,129,125, 50,
       189,143, 64,235,134,183,123, 11,240,149, 33, 34, 92,107, 78,130,
        84,214,101,147,206, 96,178, 28,115, 86,192, 20,167,140,241,220,
        18,117,202, 31, 59,190,228,209, 66, 61,212, 48,163, 60,182, 38,
       111,191, 14,218, 70,105,  7, 87, 39,242, 29,155,188,148, 67,  3,
       248, 17,199,246,144,239, 62,231,  6,195,213, 47,200,102, 30,215,
         8,232,234,222,128, 82,238,247,132,170,114,172, 53, 77,106, 42,
       150, 26,210,113, 90, 21, 73,116, 75,159,208, 94,  4, 24,164,236,
       194,224, 65,110, 15, 81,203,204, 36,145,175, 80,161,244,112, 57,
       153,124, 58,133, 35,184,180,122,252,  2, 54, 91, 37, 85,151, 49,
        45, 93,250,152,227,138,146,174,  5,223, 41, 16,103,108,186,201,
       211,  0,230,207,225,158,168, 44, 99, 22,  1, 63, 88,226,137,169,
        13, 56, 52, 27,171, 51,255,176,187, 72, 12, 95,185,177,205, 46,
       197,243,219, 71,229,165,156,119, 10,166, 32,104,254,127,193,173
}
```

Definition at line 43 of file rc2.c.

5.11.3.2 const struct ltc_cipher_descriptor rc2_desc

Initial value:

```
 {
   "rc2",
   12, 8, 128, 8, 16,
   &rc2_setup,
   &rc2_ecb_encrypt,
   &rc2_ecb_decrypt,
   &rc2_test,
   &rc2_done,
   &rc2_keysize,
   NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 30 of file rc2.c.

Referenced by yarrow_start().

5.12 ciphers/rc5.c File Reference

5.12.1 Detailed Description

RC5 code by Tom St Denis.

Definition in file rc5.c.

`#include "tomcrypt.h"`

Include dependency graph for rc5.c:

Functions

- int rc5_setup (const unsigned char *key, int keylen, int num_rounds, symmetric_key *skey)
  Initialize the RC5 block cipher.
- int rc5_ecb_encrypt (const unsigned char *pt, unsigned char *ct, symmetric_key *skey)
  Encrypts a block of text with RC5.
- int rc5_ecb_decrypt (const unsigned char *ct, unsigned char *pt, symmetric_key *skey)
  Decrypts a block of text with RC5.
- int rc5_test (void)
  Performs a self-test of the RC5 block cipher.
- void rc5_done (symmetric_key *skey)
  Terminate the context.
- int rc5_keysize (int *keysize)
  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor rc5_desc
- static const ulong32 stab [50]

5.12.2 Function Documentation

5.12.2.1 void rc5_done (symmetric_key * *skey*)

Terminate the context.

Parameters:
skey The scheduled key

Definition at line 295 of file rc5.c.

```
296 {
297 }
```

5.12.2.2 int rc5_ecb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, symmetric_key ∗ *skey*)

Decrypts a block of text with RC5.

Parameters:

ct The input ciphertext (8 bytes)

pt The output plaintext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 186 of file rc5.c.

References B, CRYPT_OK, K, LTC_ARGCHK, and ROR.

```
{
   ulong32 A, B, *K;
   int r;
   LTC_ARGCHK(skey != NULL);
   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);

   LOAD32L(A, &ct[0]);
   LOAD32L(B, &ct[4]);
   K = skey->rc5.K + (skey->rc5.rounds << 1);

   if ((skey->rc5.rounds & 1) == 0) {
       K -= 2;
       for (r = skey->rc5.rounds - 1; r >= 0; r -= 2) {
          B = ROR(B - K[3], A) ^ A;
          A = ROR(A - K[2], B) ^ B;
          B = ROR(B - K[1], A) ^ A;
          A = ROR(A - K[0], B) ^ B;
          K -= 4;
        }
   } else {
      for (r = skey->rc5.rounds - 1; r >= 0; r--) {
          B = ROR(B - K[1], A) ^ A;
          A = ROR(A - K[0], B) ^ B;
          K -= 2;
      }
   }
   A -= skey->rc5.K[0];
   B -= skey->rc5.K[1];
   STORE32L(A, &pt[0]);
   STORE32L(B, &pt[4]);

   return CRYPT_OK;
}
```

5.12.2.3 int rc5_ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with RC5.

Parameters:

pt The input plaintext (8 bytes)

ct The output ciphertext (8 bytes)

skey The key as scheduled

Returns:
CRYPT_OK if successful

Definition at line 131 of file rc5.c.

References B, K, LTC_ARGCHK, and ROL.

```
{
   ulong32 A, B, *K;
   int r;
   LTC_ARGCHK(skey != NULL);
   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);

   LOAD32L(A, &pt[0]);
   LOAD32L(B, &pt[4]);
   A += skey->rc5.K[0];
   B += skey->rc5.K[1];
   K  = skey->rc5.K + 2;

   if ((skey->rc5.rounds & 1) == 0) {
      for (r = 0; r < skey->rc5.rounds; r += 2) {
          A = ROL(A ^ B, B) + K[0];
          B = ROL(B ^ A, A) + K[1];
          A = ROL(A ^ B, B) + K[2];
          B = ROL(B ^ A, A) + K[3];
          K += 4;
      }
   } else {
      for (r = 0; r < skey->rc5.rounds; r++) {
          A = ROL(A ^ B, B) + K[0];
          B = ROL(B ^ A, A) + K[1];
          K += 2;
      }
   }
   STORE32L(A, &ct[0]);
   STORE32L(B, &ct[4]);

   return CRYPT_OK;
}
```

5.12.2.4 int rc5_keysize (int ∗ *keysize*)

Gets suitable key size.

Parameters:
keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:
CRYPT_OK if the input key size is acceptable.

Definition at line 304 of file rc5.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
```

```
   LTC_ARGCHK(keysize != NULL);
   if (*keysize < 8) {
      return CRYPT_INVALID_KEYSIZE;
   } else if (*keysize > 128) {
      *keysize = 128;
   }
   return CRYPT_OK;
}
```

5.12.2.5 int rc5_setup (const unsigned char * *key*, int *keylen*, int *num_rounds*, symmetric_key * *skey*)

Initialize the RC5 block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 56 of file rc5.c.

References B, BSWAP, CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, ltc_cipher_descriptor::default_rounds, LTC_ARGCHK, rc5_desc, and S.

Referenced by rc5_test().

```
{
   ulong32 L[64], *S, A, B, i, j, v, s, t, l;

   LTC_ARGCHK(skey != NULL);
   LTC_ARGCHK(key  != NULL);

   /* test parameters */
   if (num_rounds == 0) {
      num_rounds = rc5_desc.default_rounds;
   }

   if (num_rounds < 12 || num_rounds > 24) {
      return CRYPT_INVALID_ROUNDS;
   }

   /* key must be between 64 and 1024 bits */
   if (keylen < 8 || keylen > 128) {
      return CRYPT_INVALID_KEYSIZE;
   }

   skey->rc5.rounds = num_rounds;
   S = skey->rc5.K;

   /* copy the key into the L array */
   for (A = i = j = 0; i < (ulong32)keylen; ) {
       A = (A << 8) | ((ulong32)(key[i++] & 255));
       if ((i & 3) == 0) {
          L[j++] = BSWAP(A);
          A = 0;
```

```
        }
    }

    if ((keylen & 3) != 0) {
       A <<= (ulong32)((8 * (4 - (keylen&3))));
       L[j++] = BSWAP(A);
    }

    /* setup the S array */
    t = (ulong32)(2 * (num_rounds + 1));
    XMEMCPY(S, stab, t * sizeof(*S));

    /* mix buffer */
    s = 3 * MAX(t, j);
    l = j;
    for (A = B = i = j = v = 0; v < s; v++) {
        A = S[i] = ROLc(S[i] + A + B, 3);
        B = L[j] = ROL(L[j] + A + B, (A+B));
        if (++i == t) { i = 0; }
        if (++j == l) { j = 0; }
    }
    return CRYPT_OK;
}
```

5.12.2.6 int rc5_test (void)

Performs a self-test of the RC5 block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 236 of file rc5.c.

References CRYPT_NOP, CRYPT_OK, and rc5_setup().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   static const struct {
       unsigned char key[16], pt[8], ct[8];
   } tests[] = {
   {
       { 0x91, 0x5f, 0x46, 0x19, 0xbe, 0x41, 0xb2, 0x51,
         0x63, 0x55, 0xa5, 0x01, 0x10, 0xa9, 0xce, 0x91 },
       { 0x21, 0xa5, 0xdb, 0xee, 0x15, 0x4b, 0x8f, 0x6d },
       { 0xf7, 0xc0, 0x13, 0xac, 0x5b, 0x2b, 0x89, 0x52 }
   },
   {
       { 0x78, 0x33, 0x48, 0xe7, 0x5a, 0xeb, 0x0f, 0x2f,
         0xd7, 0xb1, 0x69, 0xbb, 0x8d, 0xc1, 0x67, 0x87 },
       { 0xF7, 0xC0, 0x13, 0xAC, 0x5B, 0x2B, 0x89, 0x52 },
       { 0x2F, 0x42, 0xB3, 0xB7, 0x03, 0x69, 0xFC, 0x92 }
   },
   {
       { 0xDC, 0x49, 0xdb, 0x13, 0x75, 0xa5, 0x58, 0x4f,
         0x64, 0x85, 0xb4, 0x13, 0xb5, 0xf1, 0x2b, 0xaf },
       { 0x2F, 0x42, 0xB3, 0xB7, 0x03, 0x69, 0xFC, 0x92 },
       { 0x65, 0xc1, 0x78, 0xb2, 0x84, 0xd1, 0x97, 0xcc }
   }
   };
   unsigned char tmp[2][8];
   int x, y, err;
```

```
   symmetric_key key;

   for (x = 0; x < (int)(sizeof(tests) / sizeof(tests[0])); x++) {
      /* setup key */
      if ((err = rc5_setup(tests[x].key, 16, 12, &key)) != CRYPT_OK) {
         return err;
      }

      /* encrypt and decrypt */
      rc5_ecb_encrypt(tests[x].pt, tmp[0], &key);
      rc5_ecb_decrypt(tmp[0], tmp[1], &key);

      /* compare */
      if (XMEMCMP(tmp[0], tests[x].ct, 8) != 0 || XMEMCMP(tmp[1], tests[x].pt, 8) != 0) {
         return CRYPT_FAIL_TESTVECTOR;
      }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 8; y++) tmp[0][y] = 0;
      for (y = 0; y < 1000; y++) rc5_ecb_encrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 1000; y++) rc5_ecb_decrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 8; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
   }
   return CRYPT_OK;
  #endif
}
```

Here is the call graph for this function:

5.12.3 Variable Documentation

5.12.3.1 const struct ltc_cipher_descriptor rc5_desc

Initial value:

```
{
    "rc5",
    2,
    8, 128, 8, 12,
    &rc5_setup,
    &rc5_ecb_encrypt,
    &rc5_ecb_decrypt,
    &rc5_test,
    &rc5_done,
    &rc5_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 21 of file rc5.c.

Referenced by rc5_setup(), and yarrow_start().

5.12.3.2 const ulong32 stab[50] [static]

Initial value:

```
 {
0xb7e15163UL, 0x5618cb1cUL, 0xf45044d5UL, 0x9287be8eUL, 0x30bf3847UL, 0xcef6b200UL, 0x6d2e2bb9UL, 0x0b65a5
0xa99d1f2bUL, 0x47d498e4UL, 0xe60c129dUL, 0x84438c56UL, 0x227b060fUL, 0xc0b27fc8UL, 0x5ee9f981UL, 0xfd2173
0x9b58ecf3UL, 0x399066acUL, 0xd7c7e065UL, 0x75ff5a1eUL, 0x1436d3d7UL, 0xb26e4d90UL, 0x50a5c749UL, 0xeedd41
```

```
0x8d14babbUL, 0x2b4c3474UL, 0xc983ae2dUL, 0x67bb27e6UL, 0x05f2a19fUL, 0xa42a1b58UL, 0x42619511UL, 0xe0990e
0x7ed08883UL, 0x1d08023cUL, 0xbb3f7bf5UL, 0x5976f5aeUL, 0xf7ae6f67UL, 0x95e5e920UL, 0x341d62d9UL, 0xd254dc
0x708c564bUL, 0x0ec3d004UL, 0xacfb49bdUL, 0x4b32c376UL, 0xe96a3d2fUL, 0x87a1b6e8UL, 0x25d930a1UL, 0xc410aa
0x62482413UL, 0x007f9dccUL
}
```

Definition at line 35 of file rc5.c.

5.13 ciphers/rc6.c File Reference

5.13.1 Detailed Description

RC6 code by Tom St Denis.

Definition in file rc6.c.

`#include "tomcrypt.h"`

Include dependency graph for rc6.c:

Defines

- #define RND(a, b, c, d)
- #define RND(a, b, c, d)

Functions

- int rc6_setup (const unsigned char *key, int keylen, int num_rounds, symmetric_key *skey)

    Initialize the RC6 block cipher.

- int rc6_ecb_encrypt (const unsigned char *pt, unsigned char *ct, symmetric_key *skey)

    Encrypts a block of text with RC6.

- int rc6_ecb_decrypt (const unsigned char *ct, unsigned char *pt, symmetric_key *skey)

    Decrypts a block of text with RC6.

- int rc6_test (void)

    Performs a self-test of the RC6 block cipher.

- void rc6_done (symmetric_key *skey)

    Terminate the context.

- int rc6_keysize (int *keysize)

    Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor rc6_desc
- static const ulong32 stab [44]

5.13.2 Define Documentation

5.13.2.1 #define RND(a, b, c, d)

Value:

```
t = (b * (b + b + 1)); t = ROLc(t, 5); \
       u = (d * (d + d + 1)); u = ROLc(u, 5); \
       c = ROR(c - K[1], t) ^ u; \
       a = ROR(a - K[0], u) ^ t; K -= 2;
```

5.13.2.2 #define RND(a, b, c, d)

Value:

```
t = (b * (b + b + 1)); t = ROLc(t, 5); \
       u = (d * (d + d + 1)); u = ROLc(u, 5); \
       a = ROL(a^t,u) + K[0];                \
       c = ROL(c^u,t) + K[1]; K += 2;
```

Referenced by rc6_ecb_decrypt(), and rc6_ecb_encrypt().

5.13.3 Function Documentation

5.13.3.1 void rc6_done (symmetric_key ∗ *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 322 of file rc6.c.

```
323 {
324 }
```

5.13.3.2 int rc6_ecb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, symmetric_key ∗ *skey*)

Decrypts a block of text with RC6.

Parameters:

ct The input ciphertext (16 bytes)

pt The output plaintext (16 bytes)

skey The key as scheduled

Definition at line 179 of file rc6.c.

References c, K, LTC_ARGCHK, and RND.

```
181 {
182    ulong32 a,b,c,d,t,u, *K;
183    int r;
184
185    LTC_ARGCHK(skey != NULL);
186    LTC_ARGCHK(pt   != NULL);
187    LTC_ARGCHK(ct   != NULL);
188
189    LOAD32L(a,&ct[0]);LOAD32L(b,&ct[4]);LOAD32L(c,&ct[8]);LOAD32L(d,&ct[12]);
```

```
190    a -= skey->rc6.K[42];
191    c -= skey->rc6.K[43];
192
193 #define RND(a,b,c,d) \
194        t = (b * (b + b + 1)); t = ROLc(t, 5); \
195        u = (d * (d + d + 1)); u = ROLc(u, 5); \
196        c = ROR(c - K[1], t) ^ u; \
197        a = ROR(a - K[0], u) ^ t; K -= 2;
198
199    K = skey->rc6.K + 40;
200
201    for (r = 0; r < 20; r += 4) {
202        RND(d,a,b,c);
203        RND(c,d,a,b);
204        RND(b,c,d,a);
205        RND(a,b,c,d);
206    }
207
208 #undef RND
209
210    b -= skey->rc6.K[0];
211    d -= skey->rc6.K[1];
212    STORE32L(a,&pt[0]);STORE32L(b,&pt[4]);STORE32L(c,&pt[8]);STORE32L(d,&pt[12]);
213
214    return CRYPT_OK;
215 }
```

5.13.3.3 int rc6_ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with RC6.

Parameters:

pt The input plaintext (16 bytes)

ct The output ciphertext (16 bytes)

skey The key as scheduled

Definition at line 125 of file rc6.c.

References c, K, LTC_ARGCHK, and RND.

```
127 {
128    ulong32 a,b,c,d,t,u, *K;
129    int r;
130
131    LTC_ARGCHK(skey != NULL);
132    LTC_ARGCHK(pt   != NULL);
133    LTC_ARGCHK(ct   != NULL);
134    LOAD32L(a,&pt[0]);LOAD32L(b,&pt[4]);LOAD32L(c,&pt[8]);LOAD32L(d,&pt[12]);
135
136    b += skey->rc6.K[0];
137    d += skey->rc6.K[1];
138
139 #define RND(a,b,c,d) \
140        t = (b * (b + b + 1)); t = ROLc(t, 5); \
141        u = (d * (d + d + 1)); u = ROLc(u, 5); \
142        a = ROL(a^t,u) + K[0];                 \
143        c = ROL(c^u,t) + K[1]; K += 2;
144
145    K = skey->rc6.K + 2;
146    for (r = 0; r < 20; r += 4) {
147        RND(a,b,c,d);
```

```
        RND(b,c,d,a);
        RND(c,d,a,b);
        RND(d,a,b,c);
    }

#undef RND

    a += skey->rc6.K[42];
    c += skey->rc6.K[43];
    STORE32L(a,&ct[0]);STORE32L(b,&ct[4]);STORE32L(c,&ct[8]);STORE32L(d,&ct[12]);
    return CRYPT_OK;
}
```

5.13.3.4 int rc6_keysize (int * *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 331 of file rc6.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
    LTC_ARGCHK(keysize != NULL);
    if (*keysize < 8) {
       return CRYPT_INVALID_KEYSIZE;
    } else if (*keysize > 128) {
       *keysize = 128;
    }
    return CRYPT_OK;
}
```

5.13.3.5 int rc6_setup (const unsigned char * *key*, int *keylen*, int *num_rounds*, symmetric_key * *skey*)

Initialize the RC6 block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 53 of file rc6.c.

References B, BSWAP, CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, LTC_ARGCHK, and S.

Referenced by rc6_test().

```
{
    ulong32 L[64], S[50], A, B, i, j, v, s, l;

    LTC_ARGCHK(key != NULL);
    LTC_ARGCHK(skey != NULL);

    /* test parameters */
    if (num_rounds != 0 && num_rounds != 20) {
       return CRYPT_INVALID_ROUNDS;
    }

    /* key must be between 64 and 1024 bits */
    if (keylen < 8 || keylen > 128) {
       return CRYPT_INVALID_KEYSIZE;
    }

    /* copy the key into the L array */
    for (A = i = j = 0; i < (ulong32)keylen; ) {
        A = (A << 8) | ((ulong32)(key[i++] & 255));
        if (!(i & 3)) {
           L[j++] = BSWAP(A);
           A = 0;
        }
    }

    /* handle odd sized keys */
    if (keylen & 3) {
       A <<= (8 * (4 - (keylen&3)));
       L[j++] = BSWAP(A);
    }

    /* setup the S array */
    XMEMCPY(S, stab, 44 * sizeof(stab[0]));

    /* mix buffer */
    s = 3 * MAX(44, j);
    l = j;
    for (A = B = i = j = v = 0; v < s; v++) {
        A = S[i] = ROLc(S[i] + A + B, 3);
        B = L[j] = ROL(L[j] + A + B, (A+B));
        if (++i == 44) { i = 0; }
        if (++j == l)  { j = 0; }
    }

    /* copy to key */
    for (i = 0; i < 44; i++) {
        skey->rc6.K[i] = S[i];
    }
    return CRYPT_OK;
}
```

5.13.3.6 int rc6_test (void)

Performs a self-test of the RC6 block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 230 of file rc6.c.

References CRYPT_NOP, CRYPT_OK, and rc6_setup().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   static const struct {
       int keylen;
       unsigned char key[32], pt[16], ct[16];
   } tests[] = {
   {
       16,
       { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
         0x01, 0x12, 0x23, 0x34, 0x45, 0x56, 0x67, 0x78,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
       { 0x02, 0x13, 0x24, 0x35, 0x46, 0x57, 0x68, 0x79,
         0x8a, 0x9b, 0xac, 0xbd, 0xce, 0xdf, 0xe0, 0xf1 },
       { 0x52, 0x4e, 0x19, 0x2f, 0x47, 0x15, 0xc6, 0x23,
         0x1f, 0x51, 0xf6, 0x36, 0x7e, 0xa4, 0x3f, 0x18 }
   },
   {
       24,
       { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
         0x01, 0x12, 0x23, 0x34, 0x45, 0x56, 0x67, 0x78,
         0x89, 0x9a, 0xab, 0xbc, 0xcd, 0xde, 0xef, 0xf0,
         0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
       { 0x02, 0x13, 0x24, 0x35, 0x46, 0x57, 0x68, 0x79,
         0x8a, 0x9b, 0xac, 0xbd, 0xce, 0xdf, 0xe0, 0xf1 },
       { 0x68, 0x83, 0x29, 0xd0, 0x19, 0xe5, 0x05, 0x04,
         0x1e, 0x52, 0xe9, 0x2a, 0xf9, 0x52, 0x91, 0xd4 }
   },
   {
       32,
       { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef,
         0x01, 0x12, 0x23, 0x34, 0x45, 0x56, 0x67, 0x78,
         0x89, 0x9a, 0xab, 0xbc, 0xcd, 0xde, 0xef, 0xf0,
         0x10, 0x32, 0x54, 0x76, 0x98, 0xba, 0xdc, 0xfe },
       { 0x02, 0x13, 0x24, 0x35, 0x46, 0x57, 0x68, 0x79,
         0x8a, 0x9b, 0xac, 0xbd, 0xce, 0xdf, 0xe0, 0xf1 },
       { 0xc8, 0x24, 0x18, 0x16, 0xf0, 0xd7, 0xe4, 0x89,
         0x20, 0xad, 0x16, 0xa1, 0x67, 0x4e, 0x5d, 0x48 }
   }
   };
   unsigned char tmp[2][16];
   int x, y, err;
   symmetric_key key;

   for (x  = 0; x < (int)(sizeof(tests) / sizeof(tests[0])); x++) {
      /* setup key */
      if ((err = rc6_setup(tests[x].key, tests[x].keylen, 0, &key)) != CRYPT_OK) {
         return err;
      }

      /* encrypt and decrypt */
      rc6_ecb_encrypt(tests[x].pt, tmp[0], &key);
      rc6_ecb_decrypt(tmp[0], tmp[1], &key);

      /* compare */
      if (XMEMCMP(tmp[0], tests[x].ct, 16) || XMEMCMP(tmp[1], tests[x].pt, 16)) {
#if 0
         printf("\n\nFailed test %d\n", x);
         if (XMEMCMP(tmp[0], tests[x].ct, 16)) {
            printf("Ciphertext:  ");
            for (y = 0; y < 16; y++) printf("%02x ", tmp[0][y]);
```

```
            printf("\nExpected  :  ");
            for (y = 0; y < 16; y++) printf("%02x ", tests[x].ct[y]);
            printf("\n");
         }
         if (XMEMCMP(tmp[1], tests[x].pt, 16)) {
            printf("Plaintext:  ");
            for (y = 0; y < 16; y++) printf("%02x ", tmp[0][y]);
            printf("\nExpected :  ");
            for (y = 0; y < 16; y++) printf("%02x ", tests[x].pt[y]);
            printf("\n");
         }
#endif
         return CRYPT_FAIL_TESTVECTOR;
      }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 16; y++) tmp[0][y] = 0;
      for (y = 0; y < 1000; y++) rc6_ecb_encrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 1000; y++) rc6_ecb_decrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 16; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
   }
   return CRYPT_OK;
  #endif
}
```

Here is the call graph for this function:

5.13.4 Variable Documentation

5.13.4.1 const struct ltc_cipher_descriptor rc6_desc

Initial value:

```
{
    "rc6",
    3,
    8, 128, 16, 20,
    &rc6_setup,
    &rc6_ecb_encrypt,
    &rc6_ecb_decrypt,
    &rc6_test,
    &rc6_done,
    &rc6_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 20 of file rc6.c.

Referenced by yarrow_start().

5.13.4.2 const ulong32 stab[44] [static]

Initial value:

```
 {
0xb7e15163UL, 0x5618cb1cUL, 0xf45044d5UL, 0x9287be8eUL, 0x30bf3847UL, 0xcef6b200UL, 0x6d2e2bb9UL, 0x0b65a5
0xa99d1f2bUL, 0x47d498e4UL, 0xe60c129dUL, 0x84438c56UL, 0x227b060fUL, 0xc0b27fc8UL, 0x5ee9f981UL, 0xfd2173
0x9b58ecf3UL, 0x399066acUL, 0xd7c7e065UL, 0x75ff5a1eUL, 0x1436d3d7UL, 0xb26e4d90UL, 0x50a5c749UL, 0xeedd41
0x8d14babbUL, 0x2b4c3474UL, 0xc983ae2dUL, 0x67bb27e6UL, 0x05f2a19fUL, 0xa42a1b58UL, 0x42619511UL, 0xe0990e
0x7ed08883UL, 0x1d08023cUL, 0xbb3f7bf5UL, 0x5976f5aeUL, 0xf7ae6f67UL, 0x95e5e920UL, 0x341d62d9UL, 0xd254dc
0x708c564bUL, 0x0ec3d004UL, 0xacfb49bdUL, 0x4b32c376UL }
```

Definition at line 34 of file rc6.c.

5.14 ciphers/safer/safer.c File Reference

```
#include <tomcrypt.h>
```

Include dependency graph for safer.c:

Defines

- #define ROL8(x, n)
- #define EXP(x) safer_ebox[(x) & 0xFF]
- #define LOG(x) safer_lbox[(x) & 0xFF]
- #define PHT(x, y) { y += x; x += y; }
- #define IPHT(x, y) { x -= y; y -= x; }

Functions

- static void Safer_Expand_Userkey (const unsigned char ∗userkey_1, const unsigned char ∗userkey_-2, unsigned int nof_rounds, int strengthened, safer_key_t key)
- int safer_k64_setup (const unsigned char ∗key, int keylen, int numrounds, symmetric_key ∗skey)
- int safer_sk64_setup (const unsigned char ∗key, int keylen, int numrounds, symmetric_key ∗skey)
- int safer_k128_setup (const unsigned char ∗key, int keylen, int numrounds, symmetric_key ∗skey)
- int safer_sk128_setup (const unsigned char ∗key, int keylen, int numrounds, symmetric_key ∗skey)
- int safer_ecb_encrypt (const unsigned char ∗block_in, unsigned char ∗block_out, symmetric_key ∗skey)
- int safer_ecb_decrypt (const unsigned char ∗block_in, unsigned char ∗block_out, symmetric_key ∗skey)
- int safer_64_keysize (int ∗keysize)
- int safer_128_keysize (int ∗keysize)
- int safer_k64_test (void)
- int safer_sk64_test (void)
- void safer_done (symmetric_key ∗skey)

  Terminate the context.

- int safer_sk128_test (void)

Variables

- const struct ltc_cipher_descriptor safer_k64_desc
- const struct ltc_cipher_descriptor safer_sk64_desc
- const struct ltc_cipher_descriptor safer_k128_desc
- const struct ltc_cipher_descriptor safer_sk128_desc
- const unsigned char safer_ebox [ ]
- const unsigned char safer_lbox [ ]

5.14.1 Define Documentation

5.14.1.1 #define EXP(x) safer_ebox[(x) & 0xFF]

Definition at line 92 of file safer.c.

Referenced by safer_ecb_decrypt(), and safer_ecb_encrypt().

5.14.1.2 #define IPHT(x, y) { x -= y; y -= x; }

Definition at line 95 of file safer.c.

Referenced by safer_ecb_decrypt().

5.14.1.3 #define LOG(x) safer_lbox[(x) & 0xFF]

Definition at line 93 of file safer.c.

Referenced by safer_ecb_decrypt(), and safer_ecb_encrypt().

5.14.1.4 #define PHT(x, y) { y += x; x += y; }

Definition at line 94 of file safer.c.

Referenced by safer_ecb_encrypt().

5.14.1.5 #define ROL8(x, n)

Value:

```
((unsigned char)((unsigned int)(x) << (n)\
                                     |(unsigned int)((x) & 0xFF) >> (8 - (n))))
```

Definition at line 90 of file safer.c.

Referenced by Safer_Expand_Userkey().

5.14.2 Function Documentation

5.14.2.1 int safer_128_keysize (int * *keysize*)

Definition at line 368 of file safer.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize < 16) {
      return CRYPT_INVALID_KEYSIZE;
   } else {
      *keysize = 16;
      return CRYPT_OK;
   }
}
```

5.14.2.2 int safer_64_keysize (int * *keysize*)

Definition at line 357 of file safer.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize < 8) {
      return CRYPT_INVALID_KEYSIZE;
   } else {
      *keysize = 8;
      return CRYPT_OK;
   }
}
```

5.14.2.3 void safer_done (symmetric_key * *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 446 of file safer.c.

```
{
}
```

5.14.2.4 int safer_ecb_decrypt (const unsigned char * *block_in*, unsigned char * *block_out*, symmetric_key * *skey*)

Definition at line 307 of file safer.c.

References c, CRYPT_OK, EXP, IPHT, LOG, and LTC_ARGCHK.

Referenced by safer_k64_test(), safer_sk128_test(), and safer_sk64_test().

```
{   unsigned char a, b, c, d, e, f, g, h, t;
    unsigned int round;
    unsigned char *key;

    LTC_ARGCHK(block_in != NULL);
    LTC_ARGCHK(block_out != NULL);
    LTC_ARGCHK(skey != NULL);

    key = skey->safer.key;
    a = block_in[0]; b = block_in[1]; c = block_in[2]; d = block_in[3];
    e = block_in[4]; f = block_in[5]; g = block_in[6]; h = block_in[7];
    if (SAFER_MAX_NOF_ROUNDS < (round = *key)) round = SAFER_MAX_NOF_ROUNDS;
    key += SAFER_BLOCK_LEN * (1 + 2 * round);
    h ^= *key; g -= *--key; f -= *--key; e ^= *--key;
    d ^= *--key; c -= *--key; b -= *--key; a ^= *--key;
    while (round--)
    {
        t = e; e = b; b = c; c = t; t = f; f = d; d = g; g = t;
        IPHT(a, e); IPHT(b, f); IPHT(c, g); IPHT(d, h);
        IPHT(a, c); IPHT(e, g); IPHT(b, d); IPHT(f, h);
        IPHT(a, b); IPHT(c, d); IPHT(e, f); IPHT(g, h);
        h -= *--key; g ^= *--key; f ^= *--key; e -= *--key;
        d -= *--key; c ^= *--key; b ^= *--key; a -= *--key;
        h = LOG(h) ^ *--key; g = EXP(g) - *--key;
        f = EXP(f) - *--key; e = LOG(e) ^ *--key;
        d = LOG(d) ^ *--key; c = EXP(c) - *--key;
        b = EXP(b) - *--key; a = LOG(a) ^ *--key;
    }
```

```
    block_out[0] = a & 0xFF; block_out[1] = b & 0xFF;
    block_out[2] = c & 0xFF; block_out[3] = d & 0xFF;
    block_out[4] = e & 0xFF; block_out[5] = f & 0xFF;
    block_out[6] = g & 0xFF; block_out[7] = h & 0xFF;
    return CRYPT_OK;
}
```

5.14.2.5 int safer_ecb_encrypt (const unsigned char ∗ *block_in*, unsigned char ∗ *block_out*, symmetric_key ∗ *skey*)

Definition at line 253 of file safer.c.

References c, CRYPT_OK, EXP, LOG, LTC_ARGCHK, and PHT.

Referenced by safer_k64_test(), safer_sk128_test(), and safer_sk64_test().

```
{   unsigned char a, b, c, d, e, f, g, h, t;
    unsigned int round;
    unsigned char *key;

    LTC_ARGCHK(block_in != NULL);
    LTC_ARGCHK(block_out != NULL);
    LTC_ARGCHK(skey != NULL);

    key = skey->safer.key;
    a = block_in[0]; b = block_in[1]; c = block_in[2]; d = block_in[3];
    e = block_in[4]; f = block_in[5]; g = block_in[6]; h = block_in[7];
    if (SAFER_MAX_NOF_ROUNDS < (round = *key)) round = SAFER_MAX_NOF_ROUNDS;
    while(round-- > 0)
    {
        a ^= *++key; b += *++key; c += *++key; d ^= *++key;
        e ^= *++key; f += *++key; g += *++key; h ^= *++key;
        a = EXP(a) + *++key; b = LOG(b) ^ *++key;
        c = LOG(c) ^ *++key; d = EXP(d) + *++key;
        e = EXP(e) + *++key; f = LOG(f) ^ *++key;
        g = LOG(g) ^ *++key; h = EXP(h) + *++key;
        PHT(a, b); PHT(c, d); PHT(e, f); PHT(g, h);
        PHT(a, c); PHT(e, g); PHT(b, d); PHT(f, h);
        PHT(a, e); PHT(b, f); PHT(c, g); PHT(d, h);
        t = b; b = e; e = c; c = t; t = d; d = f; f = g; g = t;
    }
    a ^= *++key; b += *++key; c += *++key; d ^= *++key;
    e ^= *++key; f += *++key; g += *++key; h ^= *++key;
    block_out[0] = a & 0xFF; block_out[1] = b & 0xFF;
    block_out[2] = c & 0xFF; block_out[3] = d & 0xFF;
    block_out[4] = e & 0xFF; block_out[5] = f & 0xFF;
    block_out[6] = g & 0xFF; block_out[7] = h & 0xFF;
    return CRYPT_OK;
}
```

5.14.2.6 static void Safer_Expand_Userkey (const unsigned char ∗ *userkey_1*, const unsigned char ∗ *userkey_2*, unsigned int *nof_rounds*, int *strengthened*, safer_key_t *key*) `[static]`

Definition at line 107 of file safer.c.

References ROL8.

Referenced by safer_k128_setup(), safer_k64_setup(), safer_sk128_setup(), and safer_sk64_setup().

```
{   unsigned int i, j, k;
    unsigned char ka[SAFER_BLOCK_LEN + 1];
```

```
    unsigned char kb[SAFER_BLOCK_LEN + 1];

    if (SAFER_MAX_NOF_ROUNDS < nof_rounds)
        nof_rounds = SAFER_MAX_NOF_ROUNDS;
    *key++ = (unsigned char)nof_rounds;
    ka[SAFER_BLOCK_LEN] = (unsigned char)0;
    kb[SAFER_BLOCK_LEN] = (unsigned char)0;
    k = 0;
    for (j = 0; j < SAFER_BLOCK_LEN; j++) {
        ka[j] = ROL8(userkey_1[j], 5);
        ka[SAFER_BLOCK_LEN] ^= ka[j];
        kb[j] = *key++ = userkey_2[j];
        kb[SAFER_BLOCK_LEN] ^= kb[j];
    }
    for (i = 1; i <= nof_rounds; i++) {
        for (j = 0; j < SAFER_BLOCK_LEN + 1; j++) {
            ka[j] = ROL8(ka[j], 6);
            kb[j] = ROL8(kb[j], 6);
        }
        if (strengthened) {
           k = 2 * i - 1;
           while (k >= (SAFER_BLOCK_LEN + 1)) { k -= SAFER_BLOCK_LEN + 1; }
        }
        for (j = 0; j < SAFER_BLOCK_LEN; j++) {
            if (strengthened) {
                *key++ = (ka[k]
                                + safer_ebox[(int)safer_ebox[(int)((18 * i + j + 1)&0xFF)]]) & 0xFF;
                if (++k == (SAFER_BLOCK_LEN + 1)) { k = 0; }
            } else {
                *key++ = (ka[j] + safer_ebox[(int)safer_ebox[(int)((18 * i + j + 1)&0xFF)]]) & 0xFF;
            }
        }
        if (strengthened) {
           k = 2 * i;
           while (k >= (SAFER_BLOCK_LEN + 1)) { k -= SAFER_BLOCK_LEN + 1; }
        }
        for (j = 0; j < SAFER_BLOCK_LEN; j++) {
            if (strengthened) {
                *key++ = (kb[k]
                                + safer_ebox[(int)safer_ebox[(int)((18 * i + j + 10)&0xFF)]]) & 0xFF;
                if (++k == (SAFER_BLOCK_LEN + 1)) { k = 0; }
            } else {
                *key++ = (kb[j] + safer_ebox[(int)safer_ebox[(int)((18 * i + j + 10)&0xFF)]]) & 0xFF;
            }
        }
    }

#ifdef LTC_CLEAN_STACK
    zeromem(ka, sizeof(ka));
    zeromem(kb, sizeof(kb));
#endif
}
```

5.14.2.7 int safer_k128_setup (const unsigned char ∗ *key*, int *keylen*, int *numrounds*, symmetric_key ∗ *skey*)

Definition at line 214 of file safer.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, CRYPT_OK, LTC_ARGCHK, and Safer_Expand_Userkey().

```
{
   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(skey != NULL);
```

```

   if (numrounds != 0 && (numrounds < 6 || numrounds > SAFER_MAX_NOF_ROUNDS)) {
      return CRYPT_INVALID_ROUNDS;
   }

   if (keylen != 16) {
      return CRYPT_INVALID_KEYSIZE;
   }

   Safer_Expand_Userkey(key, key+8, (unsigned int)(numrounds != 0 ?numrounds:SAFER_K128_DEFAULT_NOF_RO
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.14.2.8 int safer_k64_setup (const unsigned char ∗ *key*, int *keylen*, int *numrounds*, symmetric_key ∗ *skey*)

Definition at line 180 of file safer.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, CRYPT_OK, LTC_ARGCHK, and Safer_Expand_Userkey().

Referenced by safer_k64_test().

```
{
   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(skey != NULL);

   if (numrounds != 0 && (numrounds < 6 || numrounds > SAFER_MAX_NOF_ROUNDS)) {
      return CRYPT_INVALID_ROUNDS;
   }

   if (keylen != 8) {
      return CRYPT_INVALID_KEYSIZE;
   }

   Safer_Expand_Userkey(key, key, (unsigned int)(numrounds != 0 ?numrounds:SAFER_K64_DEFAULT_NOF_ROUND
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.14.2.9 int safer_k64_test (void)

Definition at line 379 of file safer.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, CRYPT_OK, safer_ecb_decrypt(), safer_ecb_encrypt(), safer_k64_setup(), and XMEMCMP.

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   static const unsigned char k64_pt[]  = { 1, 2, 3, 4, 5, 6, 7, 8 },
                              k64_key[] = { 8, 7, 6, 5, 4, 3, 2, 1 },
                              k64_ct[]  = { 200, 242, 156, 221, 135, 120, 62, 217 };

   symmetric_key skey;
   unsigned char buf[2][8];
   int err;
```

```
391
392    /* test K64 */
393    if ((err = safer_k64_setup(k64_key, 8, 6, &skey)) != CRYPT_OK) {
394       return err;
395    }
396    safer_ecb_encrypt(k64_pt, buf[0], &skey);
397    safer_ecb_decrypt(buf[0], buf[1], &skey);
398
399    if (XMEMCMP(buf[0], k64_ct, 8) != 0 || XMEMCMP(buf[1], k64_pt, 8) != 0) {
400       return CRYPT_FAIL_TESTVECTOR;
401    }
402
403    return CRYPT_OK;
404  #endif
405 }
```

Here is the call graph for this function:

5.14.2.10 int safer_sk128_setup (const unsigned char ∗ *key*, int *keylen*, int *numrounds*, symmetric_key ∗ *skey*)

Definition at line 231 of file safer.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, CRYPT_OK, LTC_ARGCHK, and Safer_Expand_Userkey().

Referenced by safer_sk128_test().

```
232 {
233    LTC_ARGCHK(key != NULL);
234    LTC_ARGCHK(skey != NULL);
235
236    if (numrounds != 0 && (numrounds < 6 || numrounds > SAFER_MAX_NOF_ROUNDS)) {
237       return CRYPT_INVALID_ROUNDS;
238    }
239
240    if (keylen != 16) {
241       return CRYPT_INVALID_KEYSIZE;
242    }
243
244    Safer_Expand_Userkey(key, key+8, (unsigned int)(numrounds != 0?numrounds:SAFER_SK128_DEFAULT_NOF_RC
245    return CRYPT_OK;
246 }
```

Here is the call graph for this function:

5.14.2.11 int safer_sk128_test (void)

Definition at line 450 of file safer.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, CRYPT_OK, safer_ecb_decrypt(), safer_ecb_-encrypt(), safer_sk128_setup(), and XMEMCMP.

```
451 {
452  #ifndef LTC_TEST
453     return CRYPT_NOP;
454  #else
455    static const unsigned char sk128_pt[]  = { 1, 2, 3, 4, 5, 6, 7, 8 },
456                               sk128_key[] = { 1, 2, 3, 4, 5, 6, 7, 8,
457                                               0, 0, 0, 0, 0, 0, 0, 0 },
458                               sk128_ct[]  = { 255, 120, 17, 228, 179, 167, 46, 113 };
```

```

   symmetric_key skey;
   unsigned char buf[2][8];
   int err, y;

   /* test SK128 */
   if ((err = safer_sk128_setup(sk128_key, 16, 0, &skey)) != CRYPT_OK) {
      return err;
   }
   safer_ecb_encrypt(sk128_pt, buf[0], &skey);
   safer_ecb_decrypt(buf[0], buf[1], &skey);

   if (XMEMCMP(buf[0], sk128_ct, 8) != 0 || XMEMCMP(buf[1], sk128_pt, 8) != 0) {
      return CRYPT_FAIL_TESTVECTOR;
   }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 8; y++) buf[0][y] = 0;
      for (y = 0; y < 1000; y++) safer_ecb_encrypt(buf[0], buf[0], &skey);
      for (y = 0; y < 1000; y++) safer_ecb_decrypt(buf[0], buf[0], &skey);
      for (y = 0; y < 8; y++) if (buf[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
  return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.14.2.12 int safer_sk64_setup (const unsigned char * *key*, int *keylen*, int *numrounds*, symmetric_key * *skey*)

Definition at line 197 of file safer.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, CRYPT_OK, LTC_ARGCHK, and Safer_Expand_Userkey().

Referenced by safer_sk64_test().

```
{
   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(skey != NULL);

   if (numrounds != 0 && (numrounds < 6 || numrounds > SAFER_MAX_NOF_ROUNDS)) {
      return CRYPT_INVALID_ROUNDS;
   }

   if (keylen != 8) {
      return CRYPT_INVALID_KEYSIZE;
   }

   Safer_Expand_Userkey(key, key, (unsigned int)(numrounds != 0 ?numrounds:SAFER_SK64_DEFAULT_NOF_ROUN
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.14.2.13 int safer_sk64_test (void)

Definition at line 408 of file safer.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, CRYPT_OK, safer_ecb_decrypt(), safer_ecb_encrypt(), safer_sk64_setup(), and XMEMCMP.

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   static const unsigned char sk64_pt[]  = { 1, 2, 3, 4, 5, 6, 7, 8 },
                              sk64_key[] = { 1, 2, 3, 4, 5, 6, 7, 8 },
                              sk64_ct[]  = { 95, 206, 155, 162, 5, 132, 56, 199 };

   symmetric_key skey;
   unsigned char buf[2][8];
   int err, y;

   /* test SK64 */
   if ((err = safer_sk64_setup(sk64_key, 8, 6, &skey)) != CRYPT_OK) {
      return err;
   }

   safer_ecb_encrypt(sk64_pt, buf[0], &skey);
   safer_ecb_decrypt(buf[0], buf[1], &skey);

   if (XMEMCMP(buf[0], sk64_ct, 8) != 0 || XMEMCMP(buf[1], sk64_pt, 8) != 0) {
      return CRYPT_FAIL_TESTVECTOR;
   }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 8; y++) buf[0][y] = 0;
      for (y = 0; y < 1000; y++) safer_ecb_encrypt(buf[0], buf[0], &skey);
      for (y = 0; y < 1000; y++) safer_ecb_decrypt(buf[0], buf[0], &skey);
      for (y = 0; y < 8; y++) if (buf[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;

   return CRYPT_OK;
  #endif
}
```

Here is the call graph for this function:

5.14.3 Variable Documentation

5.14.3.1 const unsigned char safer_ebox[ ]

Definition at line 23 of file safer_tab.c.

5.14.3.2 const struct ltc_cipher_descriptor safer_k128_desc

Initial value:

```
{
  "safer-k128",
  10, 16, 16, 8, SAFER_K128_DEFAULT_NOF_ROUNDS,
  &safer_k128_setup,
  &safer_ecb_encrypt,
  &safer_ecb_decrypt,
  &safer_sk128_test,
  &safer_done,
  &safer_128_keysize,
  NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
  }
```

Definition at line 60 of file safer.c.

5.14.3.3 const struct ltc_cipher_descriptor safer_k64_desc

Initial value:

```
{
  "safer-k64",
  8, 8, 8, 8, SAFER_K64_DEFAULT_NOF_ROUNDS,
  &safer_k64_setup,
  &safer_ecb_encrypt,
  &safer_ecb_decrypt,
  &safer_k64_test,
  &safer_done,
  &safer_64_keysize,
  NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
  }
```

Definition at line 36 of file safer.c.

5.14.3.4 const unsigned char safer_lbox[ ]

Definition at line 43 of file safer_tab.c.

5.14.3.5 const struct ltc_cipher_descriptor safer_sk128_desc

Initial value:

```
{
  "safer-sk128",
  11, 16, 16, 8, SAFER_SK128_DEFAULT_NOF_ROUNDS,
  &safer_sk128_setup,
  &safer_ecb_encrypt,
  &safer_ecb_decrypt,
  &safer_sk128_test,
  &safer_done,
  &safer_128_keysize,
  NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
  }
```

Definition at line 72 of file safer.c.

Referenced by yarrow_start().

5.14.3.6 const struct ltc_cipher_descriptor safer_sk64_desc

Initial value:

```
{
  "safer-sk64",
  9, 8, 8, 8, SAFER_SK64_DEFAULT_NOF_ROUNDS,
  &safer_sk64_setup,
  &safer_ecb_encrypt,
  &safer_ecb_decrypt,
  &safer_sk64_test,
  &safer_done,
  &safer_64_keysize,
  NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
  }
```

Definition at line 48 of file safer.c.

5.15 ciphers/safer/safer_tab.c File Reference

5.15.1 Detailed Description

Tables for SAFER block ciphers.

Definition in file safer_tab.c.

#include "tomcrypt.h"

Include dependency graph for safer_tab.c:

Variables

- const unsigned char safer_ebox [256]
- const unsigned char safer_lbox [256]

5.15.2 Variable Documentation

5.15.2.1 const unsigned char safer_ebox[256]

Initial value:

```
 {
  1,  45, 226, 147, 190,  69,  21, 174, 120,   3, 135, 164, 184,  56, 207,  63,
  8, 103,   9, 148, 235,  38, 168, 107, 189,  24,  52,  27, 187, 191, 114, 247,
 64,  53,  72, 156,  81,  47,  59,  85, 227, 192, 159, 216, 211, 243, 141, 177,
255, 167,  62, 220, 134, 119, 215, 166,  17, 251, 244, 186, 146, 145, 100, 131,
241,  51, 239, 218,  44, 181, 178,  43, 136, 209, 153, 203, 140, 132,  29,  20,
129, 151, 113, 202,  95, 163, 139,  87,  60, 130, 196,  82,  92,  28, 232, 160,
  4, 180, 133,  74, 246,  19,  84, 182, 223,  12,  26, 142, 222, 224,  57, 252,
 32, 155,  36,  78, 169, 152, 158, 171, 242,  96, 208, 108, 234, 250, 199, 217,
  0, 212,  31, 110,  67, 188, 236,  83, 137, 254, 122,  93,  73, 201,  50, 194,
249, 154, 248, 109,  22, 219,  89, 150,  68, 233, 205, 230,  70,  66, 143,  10,
193, 204, 185, 101, 176, 210, 198, 172,  30,  65,  98,  41,  46,  14, 116,  80,
  2,  90, 195,  37, 123, 138,  42,  91, 240,   6,  13,  71, 111, 112, 157, 126,
 16, 206,  18,  39, 213,  76,  79, 214, 121,  48, 104,  54, 117, 125, 228, 237,
128, 106, 144,  55, 162,  94, 118, 170, 197, 127,  61, 175, 165, 229,  25,  97,
253,  77, 124, 183,  11, 238, 173,  75,  34, 245, 231, 115,  35,  33, 200,   5,
225, 102, 221, 179,  88, 105,  99,  86,  15, 161,  49, 149,  23,   7,  58,  40
}
```

Definition at line 23 of file safer_tab.c.

5.15.2.2 const unsigned char safer_lbox[256]

Initial value:

```
 {
128,   0, 176,   9,  96, 239, 185, 253,  16,  18, 159, 228, 105, 186, 173, 248,
192,  56, 194, 101,  79,   6, 148, 252,  25, 222, 106,  27,  93,  78, 168, 130,
112, 237, 232, 236, 114, 179,  21, 195, 255, 171, 182,  71,  68,   1, 172,  37,
201, 250, 142,  65,  26,  33, 203, 211,  13, 110, 254,  38,  88, 218,  50,  15,
 32, 169, 157, 132, 152,   5, 156, 187,  34, 140,  99, 231, 197, 225, 115, 198,
175,  36,  91, 135, 102,  39, 247,  87, 244, 150, 177, 183,  92, 139, 213,  84,
121, 223, 170, 246,  62, 163, 241,  17, 202, 245, 209,  23, 123, 147, 131, 188,
189,  82,  30, 235, 174, 204, 214,  53,   8, 200, 138, 180, 226, 205, 191, 217,
```

```
208,  80,  89,  63,  77,  98,  52,  10,  72, 136, 181,  86,  76,  46, 107, 158,
210,  61,  60,   3,  19, 251, 151,  81, 117,  74, 145, 113,  35, 190, 118,  42,
 95, 249, 212,  85,  11, 220,  55,  49,  22, 116, 215, 119, 167, 230,   7, 219,
164,  47,  70, 243,  97,  69, 103, 227,  12, 162,  59,  28, 133,  24,   4,  29,
 41, 160, 143, 178,  90, 216, 166, 126, 238, 141,  83,  75, 161, 154, 193,  14,
122,  73, 165,  44, 129, 196, 199,  54,  43, 127,  67, 149,  51, 242, 108, 104,
109, 240,   2,  40, 206, 221, 155, 234,  94, 153, 124,  20, 134, 207, 229,  66,
184,  64, 120,  45,  58, 233, 100,  31, 146, 144, 125,  57, 111, 224, 137,  48
}
```

Definition at line 43 of file safer_tab.c.

5.16 ciphers/safer/saferp.c File Reference

5.16.1 Detailed Description

SAFER+ Implementation by Tom St Denis.

Definition in file saferp.c.

`#include "tomcrypt.h"`

Include dependency graph for saferp.c:

Defines

- #define ROUND(b, i)
- #define iROUND(b, i)
- #define PHT(b)
- #define iPHT(b)
- #define SHUF(b, b2)
- #define iSHUF(b, b2)
- #define LT(b, b2)
- #define iLT(b, b2)

Functions

- int saferp_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the SAFER+ block cipher.

- int saferp_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with SAFER+.

- int saferp_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with SAFER+.

- int saferp_test (void)

  Performs a self-test of the SAFER+ block cipher.

- void saferp_done (symmetric_key ∗skey)

  Terminate the context.

- int saferp_keysize (int ∗keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor saferp_desc
- const unsigned char safer_ebox [ ]
- const unsigned char safer_lbox [ ]
- static const unsigned char safer_bias [33][16]

5.16.2 Define Documentation

5.16.2.1 #define iLT(b, b2)

Value:

```
iPHT(b);                  \
    iSHUF(b, b2); iPHT(b2);    \
    iSHUF(b2, b); iPHT(b);     \
    iSHUF(b, b2); iPHT(b2);
```

Definition at line 131 of file saferp.c.

Referenced by saferp_ecb_decrypt().

5.16.2.2 #define iPHT(b)

Value:

```
b[15] = (b[15] - (b[14] = (b[14] - b[15]) & 255)) & 255;  \
    b[13] = (b[13] - (b[12] = (b[12] - b[13]) & 255)) & 255;  \
    b[11] = (b[11] - (b[10] = (b[10] - b[11]) & 255)) & 255;  \
    b[9]  = (b[9] - (b[8] = (b[8] - b[9]) & 255)) & 255;      \
    b[7]  = (b[7] - (b[6] = (b[6] - b[7]) & 255)) & 255;      \
    b[5]  = (b[5] - (b[4] = (b[4] - b[5]) & 255)) & 255;      \
    b[3]  = (b[3] - (b[2] = (b[2] - b[3]) & 255)) & 255;      \
    b[1]  = (b[1] - (b[0] = (b[0] - b[1]) & 255)) & 255;      \
```

Definition at line 96 of file saferp.c.

5.16.2.3 #define iROUND(b, i)

Value:

```
b[0]  = safer_lbox[(b[0] - skey->saferp.K[i+1][0]) & 255] ^ skey->saferp.K[i][0];           \
    b[1]  = (safer_ebox[(b[1] ^ skey->saferp.K[i+1][1]) & 255] - skey->saferp.K[i][1]) & 255;    \
    b[2]  = (safer_ebox[(b[2] ^ skey->saferp.K[i+1][2]) & 255] - skey->saferp.K[i][2]) & 255;    \
    b[3]  = safer_lbox[(b[3] - skey->saferp.K[i+1][3]) & 255] ^ skey->saferp.K[i][3];            \
    b[4]  = safer_lbox[(b[4] - skey->saferp.K[i+1][4]) & 255] ^ skey->saferp.K[i][4];            \
    b[5]  = (safer_ebox[(b[5] ^ skey->saferp.K[i+1][5]) & 255] - skey->saferp.K[i][5]) & 255;    \
    b[6]  = (safer_ebox[(b[6] ^ skey->saferp.K[i+1][6]) & 255] - skey->saferp.K[i][6]) & 255;    \
    b[7]  = safer_lbox[(b[7] - skey->saferp.K[i+1][7]) & 255] ^ skey->saferp.K[i][7];            \
    b[8]  = safer_lbox[(b[8] - skey->saferp.K[i+1][8]) & 255] ^ skey->saferp.K[i][8];            \
    b[9]  = (safer_ebox[(b[9] ^ skey->saferp.K[i+1][9]) & 255] - skey->saferp.K[i][9]) & 255;    \
    b[10] = (safer_ebox[(b[10] ^ skey->saferp.K[i+1][10]) & 255] - skey->saferp.K[i][10]) & 255; \
    b[11] = safer_lbox[(b[11] - skey->saferp.K[i+1][11]) & 255] ^ skey->saferp.K[i][11];         \
    b[12] = safer_lbox[(b[12] - skey->saferp.K[i+1][12]) & 255] ^ skey->saferp.K[i][12];         \
    b[13] = (safer_ebox[(b[13] ^ skey->saferp.K[i+1][13]) & 255] - skey->saferp.K[i][13]) & 255; \
    b[14] = (safer_ebox[(b[14] ^ skey->saferp.K[i+1][14]) & 255] - skey->saferp.K[i][14]) & 255; \
    b[15] = safer_lbox[(b[15] - skey->saferp.K[i+1][15]) & 255] ^ skey->saferp.K[i][15];
```

Definition at line 66 of file saferp.c.

Referenced by saferp_ecb_decrypt().

5.16.2.4 #define iSHUF(b, b2)

Value:

```
b2[0] = b[12]; b2[1] = b[5]; b2[2] = b[4]; b2[3] = b[15];      \
    b2[4] = b[14]; b2[5] = b[7]; b2[6] = b[6]; b2[7] = b[13];      \
    b2[8] = b[0]; b2[9] = b[9]; b2[10] = b[8]; b2[11] = b[1];      \
    b2[12] = b[2]; b2[13] = b[11]; b2[14] = b[10]; b2[15] = b[3];
```

Definition at line 114 of file saferp.c.

5.16.2.5 #define LT(b, b2)

Value:

```
PHT(b);  SHUF(b, b2);     \
    PHT(b2); SHUF(b2, b);     \
    PHT(b);  SHUF(b, b2);     \
    PHT(b2);
```

Definition at line 124 of file saferp.c.

5.16.2.6 #define PHT(b)

Value:

```
b[0]  = (b[0] + (b[1] = (b[0] + b[1]) & 255)) & 255;     \
    b[2]  = (b[2] + (b[3] = (b[3] + b[2]) & 255)) & 255;     \
    b[4]  = (b[4] + (b[5] = (b[5] + b[4]) & 255)) & 255;     \
    b[6]  = (b[6] + (b[7] = (b[7] + b[6]) & 255)) & 255;     \
    b[8]  = (b[8] + (b[9] = (b[9] + b[8]) & 255)) & 255;     \
    b[10] = (b[10] + (b[11] = (b[11] + b[10]) & 255)) & 255; \
    b[12] = (b[12] + (b[13] = (b[13] + b[12]) & 255)) & 255; \
    b[14] = (b[14] + (b[15] = (b[15] + b[14]) & 255)) & 255;
```

Definition at line 85 of file saferp.c.

5.16.2.7 #define ROUND(b, i)

Value:

```
b[0]  = (safer_ebox[(b[0] ^ skey->saferp.K[i][0]) & 255] + skey->saferp.K[i+1][0]) & 255;    \
    b[1]  = safer_lbox[(b[1] + skey->saferp.K[i][1]) & 255] ^ skey->saferp.K[i+1][1];            \
    b[2]  = safer_lbox[(b[2] + skey->saferp.K[i][2]) & 255] ^ skey->saferp.K[i+1][2];            \
    b[3]  = (safer_ebox[(b[3] ^ skey->saferp.K[i][3]) & 255] + skey->saferp.K[i+1][3]) & 255;    \
    b[4]  = (safer_ebox[(b[4] ^ skey->saferp.K[i][4]) & 255] + skey->saferp.K[i+1][4]) & 255;    \
    b[5]  = safer_lbox[(b[5] + skey->saferp.K[i][5]) & 255] ^ skey->saferp.K[i+1][5];            \
    b[6]  = safer_lbox[(b[6] + skey->saferp.K[i][6]) & 255] ^ skey->saferp.K[i+1][6];            \
    b[7]  = (safer_ebox[(b[7] ^ skey->saferp.K[i][7]) & 255] + skey->saferp.K[i+1][7]) & 255;    \
    b[8]  = (safer_ebox[(b[8] ^ skey->saferp.K[i][8]) & 255] + skey->saferp.K[i+1][8]) & 255;    \
    b[9]  = safer_lbox[(b[9] + skey->saferp.K[i][9]) & 255] ^ skey->saferp.K[i+1][9];            \
    b[10] = safer_lbox[(b[10] + skey->saferp.K[i][10]) & 255] ^ skey->saferp.K[i+1][10];         \
    b[11] = (safer_ebox[(b[11] ^ skey->saferp.K[i][11]) & 255] + skey->saferp.K[i+1][11]) & 255; \
    b[12] = (safer_ebox[(b[12] ^ skey->saferp.K[i][12]) & 255] + skey->saferp.K[i+1][12]) & 255; \
    b[13] = safer_lbox[(b[13] + skey->saferp.K[i][13]) & 255] ^ skey->saferp.K[i+1][13];         \
    b[14] = safer_lbox[(b[14] + skey->saferp.K[i][14]) & 255] ^ skey->saferp.K[i+1][14];         \
    b[15] = (safer_ebox[(b[15] ^ skey->saferp.K[i][15]) & 255] + skey->saferp.K[i+1][15]) & 255;
```

Definition at line 47 of file saferp.c.

5.16.2.8 #define SHUF(b, b2)

Value:

```
b2[0] = b[8]; b2[1] = b[11]; b2[2] = b[12]; b2[3] = b[15];   \
    b2[4] = b[2]; b2[5] = b[1]; b2[6] = b[6]; b2[7] = b[5];      \
    b2[8] = b[10]; b2[9] = b[9]; b2[10] = b[14]; b2[11] = b[13]; \
    b2[12] = b[0]; b2[13] = b[7]; b2[14] = b[4]; b2[15] = b[3];
```

Definition at line 107 of file saferp.c.

5.16.3 Function Documentation

5.16.3.1 void saferp_done (symmetric_key * *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 528 of file saferp.c.

```
{
}
```

5.16.3.2 int saferp_ecb_decrypt (const unsigned char * *ct*, unsigned char * *pt*, symmetric_key * *skey*)

Decrypts a block of text with SAFER+.

Parameters:

ct The input ciphertext (16 bytes)

pt The output plaintext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 398 of file saferp.c.

References iLT, iROUND, and LTC_ARGCHK.

```
{
   unsigned char b[16];
   int x;

   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);

   /* do eight rounds */
   b[0] = ct[0] ^ skey->saferp.K[skey->saferp.rounds*2][0];
   b[1] = (ct[1] - skey->saferp.K[skey->saferp.rounds*2][1]) & 255;
```

```
   b[2] = (ct[2] - skey->saferp.K[skey->saferp.rounds*2][2]) & 255;
   b[3] = ct[3] ^ skey->saferp.K[skey->saferp.rounds*2][3];
   b[4] = ct[4] ^ skey->saferp.K[skey->saferp.rounds*2][4];
   b[5] = (ct[5] - skey->saferp.K[skey->saferp.rounds*2][5]) & 255;
   b[6] = (ct[6] - skey->saferp.K[skey->saferp.rounds*2][6]) & 255;
   b[7] = ct[7] ^ skey->saferp.K[skey->saferp.rounds*2][7];
   b[8] = ct[8] ^ skey->saferp.K[skey->saferp.rounds*2][8];
   b[9] = (ct[9] - skey->saferp.K[skey->saferp.rounds*2][9]) & 255;
   b[10] = (ct[10] - skey->saferp.K[skey->saferp.rounds*2][10]) & 255;
   b[11] = ct[11] ^ skey->saferp.K[skey->saferp.rounds*2][11];
   b[12] = ct[12] ^ skey->saferp.K[skey->saferp.rounds*2][12];
   b[13] = (ct[13] - skey->saferp.K[skey->saferp.rounds*2][13]) & 255;
   b[14] = (ct[14] - skey->saferp.K[skey->saferp.rounds*2][14]) & 255;
   b[15] = ct[15] ^ skey->saferp.K[skey->saferp.rounds*2][15];
   /* 256-bit key? */
   if (skey->saferp.rounds > 12) {
      iLT(b, pt); iROUND(pt, 30);
      iLT(pt, b); iROUND(b, 28);
      iLT(b, pt); iROUND(pt, 26);
      iLT(pt, b); iROUND(b, 24);
   }
   /* 192-bit key? */
   if (skey->saferp.rounds > 8) {
      iLT(b, pt); iROUND(pt, 22);
      iLT(pt, b); iROUND(b, 20);
      iLT(b, pt); iROUND(pt, 18);
      iLT(pt, b); iROUND(b, 16);
   }
   iLT(b, pt); iROUND(pt, 14);
   iLT(pt, b); iROUND(b, 12);
   iLT(b, pt); iROUND(pt,10);
   iLT(pt, b); iROUND(b, 8);
   iLT(b, pt); iROUND(pt,6);
   iLT(pt, b); iROUND(b, 4);
   iLT(b, pt); iROUND(pt,2);
   iLT(pt, b); iROUND(b, 0);
   for (x = 0; x < 16; x++) {
       pt[x] = b[x];
   }
#ifdef LTC_CLEAN_STACK
   zeromem(b, sizeof(b));
#endif
   return CRYPT_OK;
}
```

5.16.3.3 int saferp_ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with SAFER+.

Parameters:

pt The input plaintext (16 bytes)

ct The output ciphertext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 334 of file saferp.c.

References LTC_ARGCHK.

```
{
   unsigned char b[16];
   int x;

   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);

   /* do eight rounds */
   for (x = 0; x < 16; x++) {
       b[x] = pt[x];
   }
   ROUND(b,  0);  LT(b, ct);
   ROUND(ct, 2);  LT(ct, b);
   ROUND(b,  4);  LT(b, ct);
   ROUND(ct, 6);  LT(ct, b);
   ROUND(b,  8);  LT(b, ct);
   ROUND(ct, 10); LT(ct, b);
   ROUND(b,  12); LT(b, ct);
   ROUND(ct, 14); LT(ct, b);
   /* 192-bit key? */
   if (skey->saferp.rounds > 8) {
      ROUND(b, 16);  LT(b, ct);
      ROUND(ct, 18); LT(ct, b);
      ROUND(b, 20);  LT(b, ct);
      ROUND(ct, 22); LT(ct, b);
   }
   /* 256-bit key? */
   if (skey->saferp.rounds > 12) {
      ROUND(b, 24);  LT(b, ct);
      ROUND(ct, 26); LT(ct, b);
      ROUND(b, 28);  LT(b, ct);
      ROUND(ct, 30); LT(ct, b);
   }
   ct[0] = b[0] ^ skey->saferp.K[skey->saferp.rounds*2][0];
   ct[1] = (b[1] + skey->saferp.K[skey->saferp.rounds*2][1]) & 255;
   ct[2] = (b[2] + skey->saferp.K[skey->saferp.rounds*2][2]) & 255;
   ct[3] = b[3] ^ skey->saferp.K[skey->saferp.rounds*2][3];
   ct[4] = b[4] ^ skey->saferp.K[skey->saferp.rounds*2][4];
   ct[5] = (b[5] + skey->saferp.K[skey->saferp.rounds*2][5]) & 255;
   ct[6] = (b[6] + skey->saferp.K[skey->saferp.rounds*2][6]) & 255;
   ct[7] = b[7] ^ skey->saferp.K[skey->saferp.rounds*2][7];
   ct[8] = b[8] ^ skey->saferp.K[skey->saferp.rounds*2][8];
   ct[9] = (b[9] + skey->saferp.K[skey->saferp.rounds*2][9]) & 255;
   ct[10] = (b[10] + skey->saferp.K[skey->saferp.rounds*2][10]) & 255;
   ct[11] = b[11] ^ skey->saferp.K[skey->saferp.rounds*2][11];
   ct[12] = b[12] ^ skey->saferp.K[skey->saferp.rounds*2][12];
   ct[13] = (b[13] + skey->saferp.K[skey->saferp.rounds*2][13]) & 255;
   ct[14] = (b[14] + skey->saferp.K[skey->saferp.rounds*2][14]) & 255;
   ct[15] = b[15] ^ skey->saferp.K[skey->saferp.rounds*2][15];
#ifdef LTC_CLEAN_STACK
   zeromem(b, sizeof(b));
#endif
   return CRYPT_OK;
}
```

5.16.3.4 int saferp_keysize (int ∗ *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 537 of file saferp.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);

   if (*keysize < 16)
      return CRYPT_INVALID_KEYSIZE;
   if (*keysize < 24) {
      *keysize = 16;
   } else if (*keysize < 32) {
      *keysize = 24;
   } else {
      *keysize = 32;
   }
   return CRYPT_OK;
}
```

5.16.3.5 int saferp_setup (const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_key ∗ *skey*)

Initialize the SAFER+ block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 216 of file saferp.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, LTC_ARGCHK, and rounds().

Referenced by saferp_test().

```
{
   unsigned x, y, z;
   unsigned char t[33];
   static const int rounds[3] = { 8, 12, 16 };

   LTC_ARGCHK(key  != NULL);
   LTC_ARGCHK(skey != NULL);

   /* check arguments */
   if (keylen != 16 && keylen != 24 && keylen != 32) {
      return CRYPT_INVALID_KEYSIZE;
   }

   /* Is the number of rounds valid?  Either use zero for default or
    * 8,12,16 rounds for 16,24,32 byte keys
    */
```

```
   if (num_rounds != 0 && num_rounds != rounds[(keylen/8)-2]) {
      return CRYPT_INVALID_ROUNDS;
   }

   /* 128 bit key version */
   if (keylen == 16) {
       /* copy key into t */
       for (x = y = 0; x < 16; x++) {
           t[x] = key[x];
           y ^= key[x];
       }
       t[16] = y;

       /* make round keys */
       for (x = 0; x < 16; x++) {
           skey->saferp.K[0][x] = t[x];
       }

       /* make the 16 other keys as a transformation of the first key */
       for (x = 1; x < 17; x++) {
           /* rotate 3 bits each */
           for (y = 0; y < 17; y++) {
               t[y] = ((t[y]<<3)|(t[y]>>5)) & 255;
           }

           /* select and add */
           z = x;
           for (y = 0; y < 16; y++) {
               skey->saferp.K[x][y] = (t[z] + safer_bias[x-1][y]) & 255;
               if (++z == 17) { z = 0; }
           }
       }
       skey->saferp.rounds = 8;
   } else if (keylen == 24) {
       /* copy key into t */
       for (x = y = 0; x < 24; x++) {
           t[x] = key[x];
           y ^= key[x];
       }
       t[24] = y;

       /* make round keys */
       for (x = 0; x < 16; x++) {
           skey->saferp.K[0][x] = t[x];
       }

       for (x = 1; x < 25; x++) {
           /* rotate 3 bits each */
           for (y = 0; y < 25; y++) {
               t[y] = ((t[y]<<3)|(t[y]>>5)) & 255;
           }

           /* select and add */
           z = x;
           for (y = 0; y < 16; y++) {
               skey->saferp.K[x][y] = (t[z] + safer_bias[x-1][y]) & 255;
               if (++z == 25) { z = 0; }
           }
       }
       skey->saferp.rounds = 12;
   } else {
       /* copy key into t */
       for (x = y = 0; x < 32; x++) {
           t[x] = key[x];
           y ^= key[x];
       }
       t[32] = y;
```

```

       /* make round keys */
       for (x = 0; x < 16; x++) {
           skey->saferp.K[0][x] = t[x];
       }

       for (x = 1; x < 33; x++) {
           /* rotate 3 bits each */
           for (y = 0; y < 33; y++) {
               t[y] = ((t[y]<<3)|(t[y]>>5)) & 255;
           }

           /* select and add */
           z = x;
           for (y = 0; y < 16; y++) {
               skey->saferp.K[x][y] = (t[z] + safer_bias[x-1][y]) & 255;
               if (++z == 33) { z = 0; }
           }
       }
       skey->saferp.rounds = 16;
   }
#ifdef LTC_CLEAN_STACK
   zeromem(t, sizeof(t));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.16.3.6 int saferp_test (void)

Performs a self-test of the SAFER+ block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 459 of file saferp.c.

References CRYPT_NOP, CRYPT_OK, and saferp_setup().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   static const struct {
       int keylen;
       unsigned char key[32], pt[16], ct[16];
   } tests[] = {
       {
           16,
           { 41, 35, 190, 132, 225, 108, 214, 174,
             82, 144, 73, 241, 241, 187, 233, 235 },
           { 179, 166, 219, 60, 135, 12, 62, 153,
             36, 94, 13, 28, 6, 183, 71, 222 },
           { 224, 31, 182, 10, 12, 255, 84, 70,
             127, 13, 89, 249, 9, 57, 165, 220 }
       }, {
           24,
           { 72, 211, 143, 117, 230, 217, 29, 42,
             229, 192, 247, 43, 120, 129, 135, 68,
             14, 95, 80, 0, 212, 97, 141, 190 },
           { 123, 5, 21, 7, 59, 51, 130, 31,
             24, 112, 146, 218, 100, 84, 206, 177 },
```

```
          { 92, 136, 4, 63, 57, 95, 100, 0,
            150, 130, 130, 16, 193, 111, 219, 133 }
      }, {
          32,
          { 243, 168, 141, 254, 190, 242, 235, 113,
            255, 160, 208, 59, 117, 6, 140, 126,
            135, 120, 115, 77, 208, 190, 130, 190,
            219, 194, 70, 65, 43, 140, 250, 48 },
          { 127, 112, 240, 167, 84, 134, 50, 149,
            170, 91, 104, 19, 11, 230, 252, 245 },
          { 88, 11, 25, 36, 172, 229, 202, 213,
            170, 65, 105, 153, 220, 104, 153, 138 }
      }
   };

  unsigned char tmp[2][16];
  symmetric_key skey;
  int err, i, y;

  for (i = 0; i < (int)(sizeof(tests) / sizeof(tests[0])); i++) {
     if ((err = saferp_setup(tests[i].key, tests[i].keylen, 0, &skey)) != CRYPT_OK)  {
        return err;
     }
     saferp_ecb_encrypt(tests[i].pt, tmp[0], &skey);
     saferp_ecb_decrypt(tmp[0], tmp[1], &skey);

     /* compare */
     if (XMEMCMP(tmp[0], tests[i].ct, 16) || XMEMCMP(tmp[1], tests[i].pt, 16)) {
        return CRYPT_FAIL_TESTVECTOR;
     }

     /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
     for (y = 0; y < 16; y++) tmp[0][y] = 0;
     for (y = 0; y < 1000; y++) saferp_ecb_encrypt(tmp[0], tmp[0], &skey);
     for (y = 0; y < 1000; y++) saferp_ecb_decrypt(tmp[0], tmp[0], &skey);
     for (y = 0; y < 16; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
  }

  return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.16.4 Variable Documentation

5.16.4.1 const unsigned char safer_bias[33][16] [static]

Definition at line 174 of file saferp.c.

5.16.4.2 const unsigned char safer_ebox[ ]

Definition at line 23 of file safer_tab.c.

5.16.4.3 const unsigned char safer_lbox[ ]

Definition at line 43 of file safer_tab.c.

5.16.4.4 const struct ltc_cipher_descriptor saferp_desc

Initial value:

```
{
    "safer+",
    4,
    16, 32, 16, 8,
    &saferp_setup,
    &saferp_ecb_encrypt,
    &saferp_ecb_decrypt,
    &saferp_test,
    &saferp_done,
    &saferp_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 20 of file saferp.c.

Referenced by yarrow_start().

5.17 ciphers/skipjack.c File Reference

5.17.1 Detailed Description

Skipjack Implementation by Tom St Denis.

Definition in file skipjack.c.

```
#include "tomcrypt.h"
```

Include dependency graph for skipjack.c:

Defines

- #define RULE_A
- #define RULE_B
- #define RULE_A1
- #define RULE_B1

Functions

- int skipjack_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the Skipjack block cipher.

- static unsigned g_func (unsigned w, int ∗kp, unsigned char ∗key)
- static unsigned ig_func (unsigned w, int ∗kp, unsigned char ∗key)
- int skipjack_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with Skipjack.

- int skipjack_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with Skipjack.

- int skipjack_test (void)

  Performs a self-test of the Skipjack block cipher.

- void skipjack_done (symmetric_key ∗skey)

  Terminate the context.

- int skipjack_keysize (int ∗keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor skipjack_desc
- static const unsigned char sbox [256]
- static const int keystep [ ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 }
- static const int ikeystep [ ] = { 9, 0, 1, 2, 3, 4, 5, 6, 7, 8 }

5.17.2 Define Documentation

5.17.2.1 #define RULE_A

Value:

```
tmp = g_func(w1, &kp, skey->skipjack.key);      \
   w1  = tmp ^ w4 ^ x;                            \
   w4  = w3; w3 = w2;                             \
   w2  = tmp;
```

Definition at line 90 of file skipjack.c.

Referenced by skipjack_ecb_encrypt().

5.17.2.2 #define RULE_A1

Value:

```
tmp = w1 ^ w2 ^ x;                             \
   w1  = ig_func(w2, &kp, skey->skipjack.key);     \
   w2  = w3; w3 = w4; w4 = tmp;
```

Definition at line 102 of file skipjack.c.

5.17.2.3 #define RULE_B

Value:

```
tmp  = g_func(w1, &kp, skey->skipjack.key);     \
   tmp1 = w4; w4  = w3;                           \
   w3   = w1 ^ w2 ^ x;                            \
   w1   = tmp1; w2 = tmp;
```

Definition at line 96 of file skipjack.c.

5.17.2.4 #define RULE_B1

Value:

```
tmp = ig_func(w2, &kp, skey->skipjack.key);     \
   w2  = tmp ^ w3 ^ x;                            \
   w3  = w4; w4 = w1; w1 = tmp;
```

Definition at line 107 of file skipjack.c.

Referenced by skipjack_ecb_decrypt().

5.17.3 Function Documentation

5.17.3.1 static unsigned g_func (unsigned *w*, int ∗ *kp*, unsigned char ∗ *key*) `[static]`

Definition at line 112 of file skipjack.c.

References keystep, and sbox.

```
{
   unsigned char g1,g2;

   g1 = (w >> 8) & 255; g2 = w & 255;
   g1 ^= sbox[g2^key[*kp]]; *kp = keystep[*kp];
   g2 ^= sbox[g1^key[*kp]]; *kp = keystep[*kp];
   g1 ^= sbox[g2^key[*kp]]; *kp = keystep[*kp];
   g2 ^= sbox[g1^key[*kp]]; *kp = keystep[*kp];
   return ((unsigned)g1<<8)|(unsigned)g2;
}
```

5.17.3.2 static unsigned ig_func (unsigned *w*, int ∗ *kp*, unsigned char ∗ *key*) [static]

Definition at line 124 of file skipjack.c.

References ikeystep, and sbox.

```
{
   unsigned char g1,g2;

   g1 = (w >> 8) & 255; g2 = w & 255;
   *kp = ikeystep[*kp]; g2 ^= sbox[g1^key[*kp]];
   *kp = ikeystep[*kp]; g1 ^= sbox[g2^key[*kp]];
   *kp = ikeystep[*kp]; g2 ^= sbox[g1^key[*kp]];
   *kp = ikeystep[*kp]; g1 ^= sbox[g2^key[*kp]];
   return ((unsigned)g1<<8)|(unsigned)g2;
}
```

5.17.3.3 void skipjack_done (symmetric_key ∗ *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 319 of file skipjack.c.

```
{
}
```

5.17.3.4 int skipjack_ecb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, symmetric_key ∗ *skey*)

Decrypts a block of text with Skipjack.

Parameters:

ct The input ciphertext (8 bytes)

pt The output plaintext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 210 of file skipjack.c.

References LTC_ARGCHK, and RULE_B1.

```
{
   unsigned w1,w2,w3,w4,tmp;
   int x, kp;

   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);

   /* load block */
   w1 = ((unsigned)ct[0]<<8)|ct[1];
   w2 = ((unsigned)ct[2]<<8)|ct[3];
   w3 = ((unsigned)ct[4]<<8)|ct[5];
   w4 = ((unsigned)ct[6]<<8)|ct[7];

   /* 8 rounds of RULE B^-1

      Note the value "kp = 8" comes from "kp = (32 * 4) mod 10" where 32*4 is 128 which mod 10 is 8
    */
   for (x = 32, kp = 8; x > 24; x--) {
       RULE_B1;
   }

   /* 8 rounds of RULE A^-1 */
   for (; x > 16; x--) {
       RULE_A1;
   }


   /* 8 rounds of RULE B^-1 */
   for (; x > 8; x--) {
       RULE_B1;
   }

   /* 8 rounds of RULE A^-1 */
   for (; x > 0; x--) {
       RULE_A1;
   }

   /* store block */
   pt[0] = (w1>>8)&255; pt[1] = w1&255;
   pt[2] = (w2>>8)&255; pt[3] = w2&255;
   pt[4] = (w3>>8)&255; pt[5] = w3&255;
   pt[6] = (w4>>8)&255; pt[7] = w4&255;

   return CRYPT_OK;
}
```

5.17.3.5 int skipjack_ecb_encrypt (const unsigned char * *pt*, unsigned char * *ct*, symmetric_key * *skey*)

Encrypts a block of text with Skipjack.

Parameters:

pt The input plaintext (8 bytes)

ct The output ciphertext (8 bytes)

skey The key as scheduled

Returns:
CRYPT_OK if successful

Definition at line 146 of file skipjack.c.

References LTC_ARGCHK, and RULE_A.

```
{
   unsigned w1,w2,w3,w4,tmp,tmp1;
   int x, kp;

   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);

   /* load block */
   w1 = ((unsigned)pt[0]<<8)|pt[1];
   w2 = ((unsigned)pt[2]<<8)|pt[3];
   w3 = ((unsigned)pt[4]<<8)|pt[5];
   w4 = ((unsigned)pt[6]<<8)|pt[7];

   /* 8 rounds of RULE A */
   for (x = 1, kp = 0; x < 9; x++) {
       RULE_A;
   }

   /* 8 rounds of RULE B */
   for (; x < 17; x++) {
       RULE_B;
   }

   /* 8 rounds of RULE A */
   for (; x < 25; x++) {
       RULE_A;
   }

   /* 8 rounds of RULE B */
   for (; x < 33; x++) {
       RULE_B;
   }

   /* store block */
   ct[0] = (w1>>8)&255; ct[1] = w1&255;
   ct[2] = (w2>>8)&255; ct[3] = w2&255;
   ct[4] = (w3>>8)&255; ct[5] = w3&255;
   ct[6] = (w4>>8)&255; ct[7] = w4&255;

   return CRYPT_OK;
}
```

5.17.3.6 int skipjack_keysize (int ∗ *keysize*)

Gets suitable key size.

Parameters:
keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:
CRYPT_OK if the input key size is acceptable.

Definition at line 328 of file skipjack.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize < 10) {
      return CRYPT_INVALID_KEYSIZE;
   } else if (*keysize > 10) {
      *keysize = 10;
   }
   return CRYPT_OK;
}
```

5.17.3.7 int skipjack_setup (const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_key ∗ *skey*)

Initialize the Skipjack block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 67 of file skipjack.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, and LTC_ARGCHK.

Referenced by skipjack_test().

```
{
   int x;

   LTC_ARGCHK(key  != NULL);
   LTC_ARGCHK(skey != NULL);

   if (keylen != 10) {
      return CRYPT_INVALID_KEYSIZE;
   }

   if (num_rounds != 32 && num_rounds != 0) {
      return CRYPT_INVALID_ROUNDS;
   }

   /* make sure the key is in range for platforms where CHAR_BIT != 8 */
   for (x = 0; x < 10; x++) {
       skey->skipjack.key[x] = key[x] & 255;
   }

   return CRYPT_OK;
}
```

5.17.3.8 int skipjack_test (void)

Performs a self-test of the Skipjack block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 272 of file skipjack.c.

References CRYPT_NOP, CRYPT_OK, and skipjack_setup().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   static const struct {
       unsigned char key[10], pt[8], ct[8];
   } tests[] = {
   {
       { 0x00, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11 },
       { 0x33, 0x22, 0x11, 0x00, 0xdd, 0xcc, 0xbb, 0xaa },
       { 0x25, 0x87, 0xca, 0xe2, 0x7a, 0x12, 0xd3, 0x00 }
   }
   };
   unsigned char buf[2][8];
   int x, y, err;
   symmetric_key key;

   for (x = 0; x < (int)(sizeof(tests) / sizeof(tests[0])); x++) {
      /* setup key */
      if ((err = skipjack_setup(tests[x].key, 10, 0, &key)) != CRYPT_OK) {
         return err;
      }

      /* encrypt and decrypt */
      skipjack_ecb_encrypt(tests[x].pt, buf[0], &key);
      skipjack_ecb_decrypt(buf[0], buf[1], &key);

      /* compare */
      if (XMEMCMP(buf[0], tests[x].ct, 8) != 0 || XMEMCMP(buf[1], tests[x].pt, 8) != 0) {
         return CRYPT_FAIL_TESTVECTOR;
      }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 8; y++) buf[0][y] = 0;
      for (y = 0; y < 1000; y++) skipjack_ecb_encrypt(buf[0], buf[0], &key);
      for (y = 0; y < 1000; y++) skipjack_ecb_decrypt(buf[0], buf[0], &key);
      for (y = 0; y < 8; y++) if (buf[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
   }

   return CRYPT_OK;
  #endif
}
```

Here is the call graph for this function:

5.17.4 Variable Documentation

5.17.4.1 const int ikeystep[ ] = { 9, 0, 1, 2, 3, 4, 5, 6, 7, 8 } [static]

Definition at line 57 of file skipjack.c.

Referenced by ig_func().

5.17.4.2 const int keystep[ ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 } [static]

Definition at line 54 of file skipjack.c.

Referenced by g_func().

5.17.4.3 const unsigned char sbox[256] [static]

Initial value:

```
 {
   0xa3,0xd7,0x09,0x83,0xf8,0x48,0xf6,0xf4,0xb3,0x21,0x15,0x78,0x99,0xb1,0xaf,0xf9,
   0xe7,0x2d,0x4d,0x8a,0xce,0x4c,0xca,0x2e,0x52,0x95,0xd9,0x1e,0x4e,0x38,0x44,0x28,
   0x0a,0xdf,0x02,0xa0,0x17,0xf1,0x60,0x68,0x12,0xb7,0x7a,0xc3,0xe9,0xfa,0x3d,0x53,
   0x96,0x84,0x6b,0xba,0xf2,0x63,0x9a,0x19,0x7c,0xae,0xe5,0xf5,0xf7,0x16,0x6a,0xa2,
   0x39,0xb6,0x7b,0x0f,0xc1,0x93,0x81,0x1b,0xee,0xb4,0x1a,0xea,0xd0,0x91,0x2f,0xb8,
   0x55,0xb9,0xda,0x85,0x3f,0x41,0xbf,0xe0,0x5a,0x58,0x80,0x5f,0x66,0x0b,0xd8,0x90,
   0x35,0xd5,0xc0,0xa7,0x33,0x06,0x65,0x69,0x45,0x00,0x94,0x56,0x6d,0x98,0x9b,0x76,
   0x97,0xfc,0xb2,0xc2,0xb0,0xfe,0xdb,0x20,0xe1,0xeb,0xd6,0xe4,0xdd,0x47,0x4a,0x1d,
   0x42,0xed,0x9e,0x6e,0x49,0x3c,0xcd,0x43,0x27,0xd2,0x07,0xd4,0xde,0xc7,0x67,0x18,
   0x89,0xcb,0x30,0x1f,0x8d,0xc6,0x8f,0xaa,0xc8,0x74,0xdc,0xc9,0x5d,0x5c,0x31,0xa4,
   0x70,0x88,0x61,0x2c,0x9f,0x0d,0x2b,0x87,0x50,0x82,0x54,0x64,0x26,0x7d,0x03,0x40,
   0x34,0x4b,0x1c,0x73,0xd1,0xc4,0xfd,0x3b,0xcc,0xfb,0x7f,0xab,0xe6,0x3e,0x5b,0xa5,
   0xad,0x04,0x23,0x9c,0x14,0x51,0x22,0xf0,0x29,0x79,0x71,0x7e,0xff,0x8c,0x0e,0xe2,
   0x0c,0xef,0xbc,0x72,0x75,0x6f,0x37,0xa1,0xec,0xd3,0x8e,0x62,0x8b,0x86,0x10,0xe8,
   0x08,0x77,0x11,0xbe,0x92,0x4f,0x24,0xc5,0x32,0x36,0x9d,0xcf,0xf3,0xa6,0xbb,0xac,
   0x5e,0x6c,0xa9,0x13,0x57,0x25,0xb5,0xe3,0xbd,0xa8,0x3a,0x01,0x05,0x59,0x2a,0x46
}
```

Definition at line 34 of file skipjack.c.

5.17.4.4 const struct ltc_cipher_descriptor skipjack_desc

Initial value:

```
{
    "skipjack",
    17,
    10, 10, 8, 32,
    &skipjack_setup,
    &skipjack_ecb_encrypt,
    &skipjack_ecb_decrypt,
    &skipjack_test,
    &skipjack_done,
    &skipjack_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 20 of file skipjack.c.

5.18 ciphers/twofish/twofish.c File Reference

5.18.1 Detailed Description

Implementation of Twofish by Tom St Denis.

Definition in file twofish.c.

`#include "tomcrypt.h"`

`#include "twofish_tab.c"`

Include dependency graph for twofish.c:

Defines

- #define MDS_POLY 0x169
- #define RS_POLY 0x14D
- #define sbox(i, x) ((ulong32)SBOX[i][(x)&255])
- #define mds_column_mult(x, i) mds_tab[i][x]
- #define g_func(x, dum) (S1[byte(x,0)] ^ S2[byte(x,1)] ^ S3[byte(x,2)] ^ S4[byte(x,3)])
- #define g1_func(x, dum) (S2[byte(x,0)] ^ S3[byte(x,1)] ^ S4[byte(x,2)] ^ S1[byte(x,3)])

Functions

- static ulong32 gf_mult (ulong32 a, ulong32 b, ulong32 p)
- static void mds_mult (const unsigned char ∗in, unsigned char ∗out)
- static void rs_mult (const unsigned char ∗in, unsigned char ∗out)
- static void h_func (const unsigned char ∗in, unsigned char ∗out, unsigned char ∗M, int k, int offset)
- int twofish_setup (const unsigned char ∗key, int keylen, int num_rounds, symmetric_key ∗skey)

  Initialize the Twofish block cipher.
- int twofish_ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, symmetric_key ∗skey)

  Encrypts a block of text with Twofish.
- int twofish_ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, symmetric_key ∗skey)

  Decrypts a block of text with Twofish.
- int twofish_test (void)

  Performs a self-test of the Twofish block cipher.
- void twofish_done (symmetric_key ∗skey)

  Terminate the context.
- int twofish_keysize (int ∗keysize)

  Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor twofish_desc
- static const unsigned char MDS [4][4]
- static const unsigned char RS [4][8]
- static const unsigned char qord [4][5]

5.18.2 Define Documentation

5.18.2.1 #define g1_func(x, dum) (S2[byte(x,0)] ^ S3[byte(x,1)] ^ S4[byte(x,2)] ^ S1[byte(x,3)])

Definition at line 286 of file twofish.c.

Referenced by twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.18.2.2 #define g_func(x, dum) (S1[byte(x,0)] ^ S2[byte(x,1)] ^ S3[byte(x,2)] ^ S4[byte(x,3)])

Definition at line 285 of file twofish.c.

Referenced by twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.18.2.3 #define mds_column_mult(x, i) mds_tab[i][x]

Definition at line 205 of file twofish.c.

Referenced by mds_mult().

5.18.2.4 #define MDS_POLY 0x169

Definition at line 42 of file twofish.c.

5.18.2.5 #define RS_POLY 0x14D

Definition at line 43 of file twofish.c.

Referenced by rs_mult().

5.18.2.6 #define sbox(i, x) ((ulong32)SBOX[i][(x)&255])

Definition at line 73 of file twofish.c.

Referenced by g_func(), and ig_func().

5.18.3 Function Documentation

5.18.3.1 static ulong32 gf_mult (ulong32 *a*, ulong32 *b*, ulong32 *p*) [static]

Definition at line 146 of file twofish.c.

References B.

Referenced by rs_mult().

```
{
   ulong32 result, B[2], P[2];

   P[1] = p;
   B[1] = b;
   result = P[0] = B[0] = 0;

   /* unrolled branchless GF multiplier */
   result ^= B[a&1]; a >>= 1;  B[1] = P[B[1]>>7] ^ (B[1] << 1);
   result ^= B[a&1]; a >>= 1;  B[1] = P[B[1]>>7] ^ (B[1] << 1);
   result ^= B[a&1]; a >>= 1;  B[1] = P[B[1]>>7] ^ (B[1] << 1);
   result ^= B[a&1]; a >>= 1;  B[1] = P[B[1]>>7] ^ (B[1] << 1);
   result ^= B[a&1]; a >>= 1;  B[1] = P[B[1]>>7] ^ (B[1] << 1);
   result ^= B[a&1]; a >>= 1;  B[1] = P[B[1]>>7] ^ (B[1] << 1);
   result ^= B[a&1]; a >>= 1;  B[1] = P[B[1]>>7] ^ (B[1] << 1);
   result ^= B[a&1];

   return result;
}
```

5.18.3.2 static void h_func (const unsigned char ∗ *in*, unsigned char ∗ *out*, unsigned char ∗ *M*, int *k*, int *offset*) [static]

Definition at line 247 of file twofish.c.

```
{
  int x;
  unsigned char y[4];
  for (x = 0; x < 4; x++) {
      y[x] = in[x];
 }
  switch (k) {
     case 4:
            y[0] = (unsigned char)(sbox(1, (ulong32)y[0]) ^ M[4 * (6 + offset) + 0]);
            y[1] = (unsigned char)(sbox(0, (ulong32)y[1]) ^ M[4 * (6 + offset) + 1]);
            y[2] = (unsigned char)(sbox(0, (ulong32)y[2]) ^ M[4 * (6 + offset) + 2]);
            y[3] = (unsigned char)(sbox(1, (ulong32)y[3]) ^ M[4 * (6 + offset) + 3]);
     case 3:
            y[0] = (unsigned char)(sbox(1, (ulong32)y[0]) ^ M[4 * (4 + offset) + 0]);
            y[1] = (unsigned char)(sbox(1, (ulong32)y[1]) ^ M[4 * (4 + offset) + 1]);
            y[2] = (unsigned char)(sbox(0, (ulong32)y[2]) ^ M[4 * (4 + offset) + 2]);
            y[3] = (unsigned char)(sbox(0, (ulong32)y[3]) ^ M[4 * (4 + offset) + 3]);
     case 2:
            y[0] = (unsigned char)(sbox(1, sbox(0, sbox(0, (ulong32)y[0]) ^ M[4 * (2 + offset) + 0]) ^
            y[1] = (unsigned char)(sbox(0, sbox(0, sbox(1, (ulong32)y[1]) ^ M[4 * (2 + offset) + 1]) ^
            y[2] = (unsigned char)(sbox(1, sbox(1, sbox(0, (ulong32)y[2]) ^ M[4 * (2 + offset) + 2]) ^
            y[3] = (unsigned char)(sbox(0, sbox(1, sbox(1, (ulong32)y[3]) ^ M[4 * (2 + offset) + 3]) ^
  }
  mds_mult(y, out);
}
```

5.18.3.3 static void mds_mult (const unsigned char ∗ *in*, unsigned char ∗ *out*) [static]

Definition at line 210 of file twofish.c.

References mds_column_mult.

```
{
  int x;
  ulong32 tmp;
  for (tmp = x = 0; x < 4; x++) {
```

```
        tmp ^= mds_column_mult(in[x], x);
    }
    STORE32L(tmp, out);
}
```

5.18.3.4 static void rs_mult (const unsigned char ∗ *in*, unsigned char ∗ *out*) [static]

Definition at line 233 of file twofish.c.

References gf_mult(), RS, and RS_POLY.

```
{
    int x, y;
    for (x = 0; x < 4; x++) {
        out[x] = 0;
        for (y = 0; y < 8; y++) {
            out[x] ^= gf_mult(in[y], RS[x][y], RS_POLY);
        }
    }
}
```

Here is the call graph for this function:

5.18.3.5 void twofish_done (symmetric_key ∗ *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 683 of file twofish.c.

```
{
}
```

5.18.3.6 int twofish_ecb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, symmetric_key ∗ *skey*)

Decrypts a block of text with Twofish.

Parameters:

ct The input ciphertext (16 bytes)

pt The output plaintext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 546 of file twofish.c.

References c, g1_func, g_func, LTC_ARGCHK, ROLc, RORc, S1, S2, S3, S4, t1, and t2.

```
{
    ulong32 a,b,c,d,ta,tb,tc,td,t1,t2, *k;
    int r;
#if !defined(TWOFISH_SMALL) && !defined(__GNUC__)
    ulong32 *S1, *S2, *S3, *S4;
#endif

    LTC_ARGCHK(pt   != NULL);
    LTC_ARGCHK(ct   != NULL);
    LTC_ARGCHK(skey != NULL);

#if !defined(TWOFISH_SMALL) && !defined(__GNUC__)
    S1 = skey->twofish.S[0];
    S2 = skey->twofish.S[1];
    S3 = skey->twofish.S[2];
    S4 = skey->twofish.S[3];
#endif

    /* load input */
    LOAD32L(ta,&ct[0]); LOAD32L(tb,&ct[4]);
    LOAD32L(tc,&ct[8]); LOAD32L(td,&ct[12]);

    /* undo undo final swap */
    a = tc ^ skey->twofish.K[6];
    b = td ^ skey->twofish.K[7];
    c = ta ^ skey->twofish.K[4];
    d = tb ^ skey->twofish.K[5];

    k = skey->twofish.K + 36;
    for (r = 8; r != 0; --r) {
        t2 = g1_func(d, skey);
        t1 = g_func(c, skey) + t2;
        a = ROLc(a, 1) ^ (t1 + k[2]);
        b = RORc(b ^ (t2 + t1 + k[3]), 1);

        t2 = g1_func(b, skey);
        t1 = g_func(a, skey) + t2;
        c = ROLc(c, 1) ^ (t1 + k[0]);
        d = RORc(d ^ (t2 +  t1 + k[1]), 1);
        k -= 4;
    }

    /* pre-white */
    a ^= skey->twofish.K[0];
    b ^= skey->twofish.K[1];
    c ^= skey->twofish.K[2];
    d ^= skey->twofish.K[3];

    /* store */
    STORE32L(a, &pt[0]); STORE32L(b, &pt[4]);
    STORE32L(c, &pt[8]); STORE32L(d, &pt[12]);
    return CRYPT_OK;
}
```

5.18.3.7 int twofish_ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, symmetric_key ∗ *skey*)

Encrypts a block of text with Twofish.

Parameters:

pt The input plaintext (16 bytes)

ct The output ciphertext (16 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 473 of file twofish.c.

References c, g1_func, g_func, LTC_ARGCHK, ROLc, RORc, S1, S2, S3, S4, t1, and t2.

```
{
    ulong32 a,b,c,d,ta,tb,tc,td,t1,t2, *k;
    int r;
#if !defined(TWOFISH_SMALL) && !defined(__GNUC__)
    ulong32 *S1, *S2, *S3, *S4;
#endif

    LTC_ARGCHK(pt   != NULL);
    LTC_ARGCHK(ct   != NULL);
    LTC_ARGCHK(skey != NULL);

#if !defined(TWOFISH_SMALL) && !defined(__GNUC__)
    S1 = skey->twofish.S[0];
    S2 = skey->twofish.S[1];
    S3 = skey->twofish.S[2];
    S4 = skey->twofish.S[3];
#endif

    LOAD32L(a,&pt[0]); LOAD32L(b,&pt[4]);
    LOAD32L(c,&pt[8]); LOAD32L(d,&pt[12]);
    a ^= skey->twofish.K[0];
    b ^= skey->twofish.K[1];
    c ^= skey->twofish.K[2];
    d ^= skey->twofish.K[3];

    k  = skey->twofish.K + 8;
    for (r = 8; r != 0; --r) {
        t2 = g1_func(b, skey);
        t1 = g_func(a, skey) + t2;
        c  = RORc(c ^ (t1 + k[0]), 1);
        d  = ROLc(d, 1) ^ (t2 + t1 + k[1]);

        t2 = g1_func(d, skey);
        t1 = g_func(c, skey) + t2;
        a  = RORc(a ^ (t1 + k[2]), 1);
        b  = ROLc(b, 1) ^ (t2 + t1 + k[3]);
        k += 4;
   }

    /* output with "undo last swap" */
    ta = c ^ skey->twofish.K[4];
    tb = d ^ skey->twofish.K[5];
    tc = a ^ skey->twofish.K[6];
    td = b ^ skey->twofish.K[7];

    /* store output */
    STORE32L(ta,&ct[0]); STORE32L(tb,&ct[4]);
    STORE32L(tc,&ct[8]); STORE32L(td,&ct[12]);

    return CRYPT_OK;
}
```

5.18.3.8 int twofish_keysize (int ∗ *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 692 of file twofish.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize);
   if (*keysize < 16)
      return CRYPT_INVALID_KEYSIZE;
   if (*keysize < 24) {
      *keysize = 16;
      return CRYPT_OK;
   } else if (*keysize < 32) {
      *keysize = 24;
      return CRYPT_OK;
   } else {
      *keysize = 32;
      return CRYPT_OK;
   }
}
```

5.18.3.9 int twofish_setup (const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_key ∗ *skey*)

Initialize the Twofish block cipher.

Parameters:

key The symmetric key you wish to pass

keylen The key length in bytes

num_rounds The number of rounds desired (0 for default)

skey The key in as scheduled by this function.

Returns:

CRYPT_OK if successful

Definition at line 346 of file twofish.c.

References B, CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, LTC_ARGCHK, and S.

Referenced by twofish_test().

```
{
#ifndef TWOFISH_SMALL
   unsigned char S[4*4], tmpx0, tmpx1;
#endif
   int k, x, y;
   unsigned char tmp[4], tmp2[4], M[8*4];
   ulong32 A, B;

   LTC_ARGCHK(key  != NULL);
   LTC_ARGCHK(skey != NULL);
```

```

   /* invalid arguments? */
   if (num_rounds != 16 && num_rounds != 0) {
      return CRYPT_INVALID_ROUNDS;
   }

   if (keylen != 16 && keylen != 24 && keylen != 32) {
      return CRYPT_INVALID_KEYSIZE;
   }

   /* k = keysize/64 [but since our keysize is in bytes...] */
   k = keylen / 8;

   /* copy the key into M */
   for (x = 0; x < keylen; x++) {
       M[x] = key[x] & 255;
   }

   /* create the S[..] words */
#ifndef TWOFISH_SMALL
   for (x = 0; x < k; x++) {
       rs_mult(M+(x*8), S+(x*4));
   }
#else
   for (x = 0; x < k; x++) {
       rs_mult(M+(x*8), skey->twofish.S+(x*4));
   }
#endif

   /* make subkeys */
   for (x = 0; x < 20; x++) {
       /* A = h(p * 2x, Me) */
       for (y = 0; y < 4; y++) {
           tmp[y] = x+x;
       }
       h_func(tmp, tmp2, M, k, 0);
       LOAD32L(A, tmp2);

       /* B = ROL(h(p * (2x + 1), Mo), 8) */
       for (y = 0; y < 4; y++) {
           tmp[y] = (unsigned char)(x+x+1);
       }
       h_func(tmp, tmp2, M, k, 1);
       LOAD32L(B, tmp2);
       B = ROLc(B, 8);

       /* K[2i]   = A + B */
       skey->twofish.K[x+x] = (A + B) & 0xFFFFFFFFUL;

       /* K[2i+1] = (A + 2B) <<< 9 */
       skey->twofish.K[x+x+1] = ROLc(B + B + A, 9);
   }

#ifndef TWOFISH_SMALL
   /* make the sboxes (large ram variant) */
   if (k == 2) {
        for (x = 0; x < 256; x++) {
           tmpx0 = sbox(0, x);
           tmpx1 = sbox(1, x);
           skey->twofish.S[0][x] = mds_column_mult(sbox(1, (sbox(0, tmpx0 ^ S[0]) ^ S[4])),0);
           skey->twofish.S[1][x] = mds_column_mult(sbox(0, (sbox(0, tmpx1 ^ S[1]) ^ S[5])),1);
           skey->twofish.S[2][x] = mds_column_mult(sbox(1, (sbox(1, tmpx0 ^ S[2]) ^ S[6])),2);
           skey->twofish.S[3][x] = mds_column_mult(sbox(0, (sbox(1, tmpx1 ^ S[3]) ^ S[7])),3);
        }
   } else if (k == 3) {
        for (x = 0; x < 256; x++) {
           tmpx0 = sbox(0, x);
```

```
            tmpx1 = sbox(1, x);
            skey->twofish.S[0][x] = mds_column_mult(sbox(1, (sbox(0, sbox(0, tmpx1 ^ S[0]) ^ S[4]) ^ S[
            skey->twofish.S[1][x] = mds_column_mult(sbox(0, (sbox(0, sbox(1, tmpx1 ^ S[1]) ^ S[5]) ^ S[
            skey->twofish.S[2][x] = mds_column_mult(sbox(1, (sbox(1, sbox(0, tmpx0 ^ S[2]) ^ S[6]) ^ S[
            skey->twofish.S[3][x] = mds_column_mult(sbox(0, (sbox(1, sbox(1, tmpx0 ^ S[3]) ^ S[7]) ^ S[
        }
    } else {
        for (x = 0; x < 256; x++) {
            tmpx0 = sbox(0, x);
            tmpx1 = sbox(1, x);
            skey->twofish.S[0][x] = mds_column_mult(sbox(1, (sbox(0, sbox(0, sbox(1, tmpx1 ^ S[0]) ^ S[
            skey->twofish.S[1][x] = mds_column_mult(sbox(0, (sbox(0, sbox(1, sbox(1, tmpx0 ^ S[1]) ^ S[
            skey->twofish.S[2][x] = mds_column_mult(sbox(1, (sbox(1, sbox(0, sbox(0, tmpx0 ^ S[2]) ^ S[
            skey->twofish.S[3][x] = mds_column_mult(sbox(0, (sbox(1, sbox(1, sbox(0, tmpx1 ^ S[3]) ^ S[
        }
    }
#else
    /* where to start in the sbox layers */
    /* small ram variant */
    switch (k) {
          case 4 : skey->twofish.start = 0; break;
          case 3 : skey->twofish.start = 1; break;
          default: skey->twofish.start = 2; break;
    }
#endif
    return CRYPT_OK;
}
```

5.18.3.10 int twofish_test (void)

Performs a self-test of the Twofish block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 615 of file twofish.c.

References CRYPT_NOP, CRYPT_OK, and twofish_setup().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
 static const struct {
     int keylen;
     unsigned char key[32], pt[16], ct[16];
 } tests[] = {
   { 16,
     { 0x9F, 0x58, 0x9F, 0x5C, 0xF6, 0x12, 0x2C, 0x32,
       0xB6, 0xBF, 0xEC, 0x2F, 0x2A, 0xE8, 0xC3, 0x5A },
     { 0xD4, 0x91, 0xDB, 0x16, 0xE7, 0xB1, 0xC3, 0x9E,
       0x86, 0xCB, 0x08, 0x6B, 0x78, 0x9F, 0x54, 0x19 },
     { 0x01, 0x9F, 0x98, 0x09, 0xDE, 0x17, 0x11, 0x85,
       0x8F, 0xAA, 0xC3, 0xA3, 0xBA, 0x20, 0xFB, 0xC3 }
   }, {
     24,
     { 0x88, 0xB2, 0xB2, 0x70, 0x6B, 0x10, 0x5E, 0x36,
       0xB4, 0x46, 0xBB, 0x6D, 0x73, 0x1A, 0x1E, 0x88,
       0xEF, 0xA7, 0x1F, 0x78, 0x89, 0x65, 0xBD, 0x44 },
     { 0x39, 0xDA, 0x69, 0xD6, 0xBA, 0x49, 0x97, 0xD5,
       0x85, 0xB6, 0xDC, 0x07, 0x3C, 0xA3, 0x41, 0xB2 },
     { 0x18, 0x2B, 0x02, 0xD8, 0x14, 0x97, 0xEA, 0x45,
       0xF9, 0xDA, 0xAC, 0xDC, 0x29, 0x19, 0x3A, 0x65 }
```

```
   }, {
     32,
     { 0xD4, 0x3B, 0xB7, 0x55, 0x6E, 0xA3, 0x2E, 0x46,
       0xF2, 0xA2, 0x82, 0xB7, 0xD4, 0x5B, 0x4E, 0x0D,
       0x57, 0xFF, 0x73, 0x9D, 0x4D, 0xC9, 0x2C, 0x1B,
       0xD7, 0xFC, 0x01, 0x70, 0x0C, 0xC8, 0x21, 0x6F },
     { 0x90, 0xAF, 0xE9, 0x1B, 0xB2, 0x88, 0x54, 0x4F,
       0x2C, 0x32, 0xDC, 0x23, 0x9B, 0x26, 0x35, 0xE6 },
     { 0x6C, 0xB4, 0x56, 0x1C, 0x40, 0xBF, 0x0A, 0x97,
       0x05, 0x93, 0x1C, 0xB6, 0xD4, 0x08, 0xE7, 0xFA }
   }
};


 symmetric_key key;
 unsigned char tmp[2][16];
 int err, i, y;

 for (i = 0; i < (int)(sizeof(tests)/sizeof(tests[0])); i++) {
    if ((err = twofish_setup(tests[i].key, tests[i].keylen, 0, &key)) != CRYPT_OK) {
       return err;
    }
    twofish_ecb_encrypt(tests[i].pt, tmp[0], &key);
    twofish_ecb_decrypt(tmp[0], tmp[1], &key);
    if (XMEMCMP(tmp[0], tests[i].ct, 16) != 0 || XMEMCMP(tmp[1], tests[i].pt, 16) != 0) {
#if 0
       printf("Twofish failed test %d, %d, %d\n", i, XMEMCMP(tmp[0], tests[i].ct, 16), XMEMCMP(tmp[1],
#endif
       return CRYPT_FAIL_TESTVECTOR;
    }
      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 16; y++) tmp[0][y] = 0;
      for (y = 0; y < 1000; y++) twofish_ecb_encrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 1000; y++) twofish_ecb_decrypt(tmp[0], tmp[0], &key);
      for (y = 0; y < 16; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;
 }
 return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.18.4 Variable Documentation

5.18.4.1 const unsigned char MDS[4][4] `[static]`

Initial value:

```
 {
    { 0x01, 0xEF, 0x5B, 0x5B },
    { 0x5B, 0xEF, 0xEF, 0x01 },
    { 0xEF, 0x5B, 0x01, 0xEF },
    { 0xEF, 0x01, 0xEF, 0x5B }
}
```

Definition at line 46 of file twofish.c.

5.18.4.2 const unsigned char qord[4][5] `[static]`

Initial value:

```
 {
   { 1, 1, 0, 0, 1 },
   { 0, 1, 1, 0, 0 },
   { 0, 0, 0, 1, 1 },
   { 1, 0, 1, 1, 0 }
}
```

Definition at line 62 of file twofish.c.

5.18.4.3 const unsigned char RS[4][8] [static]

Initial value:

```
 {
    { 0x01, 0xA4, 0x55, 0x87, 0x5A, 0x58, 0xDB, 0x9E },
    { 0xA4, 0x56, 0x82, 0xF3, 0X1E, 0XC6, 0X68, 0XE5 },
    { 0X02, 0XA1, 0XFC, 0XC1, 0X47, 0XAE, 0X3D, 0X19 },
    { 0XA4, 0X55, 0X87, 0X5A, 0X58, 0XDB, 0X9E, 0X03 }
}
```

Definition at line 54 of file twofish.c.

Referenced by rs_mult().

5.18.4.4 const struct ltc_cipher_descriptor twofish_desc

Initial value:

```
{
    "twofish",
    7,
    16, 32, 16, 16,
    &twofish_setup,
    &twofish_ecb_encrypt,
    &twofish_ecb_decrypt,
    &twofish_test,
    &twofish_done,
    &twofish_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 27 of file twofish.c.

Referenced by yarrow_start().

5.19 ciphers/twofish/twofish_tab.c File Reference

5.19.1 Detailed Description

Twofish tables, Tom St Denis.

Definition in file twofish_tab.c.

This graph shows which files directly or indirectly include this file:

5.20 ciphers/xtea.c File Reference

5.20.1 Detailed Description

Implementation of XTEA, Tom St Denis.

Definition in file xtea.c.

`#include "tomcrypt.h"`

Include dependency graph for xtea.c:

Functions

- int xtea_setup (const unsigned char *key, int keylen, int num_rounds, symmetric_key *skey)
- int xtea_ecb_encrypt (const unsigned char *pt, unsigned char *ct, symmetric_key *skey)

    Encrypts a block of text with XTEA.

- int xtea_ecb_decrypt (const unsigned char *ct, unsigned char *pt, symmetric_key *skey)

    Decrypts a block of text with XTEA.

- int xtea_test (void)

    Performs a self-test of the XTEA block cipher.

- void xtea_done (symmetric_key *skey)

    Terminate the context.

- int xtea_keysize (int *keysize)

    Gets suitable key size.

Variables

- const struct ltc_cipher_descriptor xtea_desc

5.20.2 Function Documentation

5.20.2.1 void xtea_done (symmetric_key * *skey*)

Terminate the context.

Parameters:

skey The scheduled key

Definition at line 184 of file xtea.c.

```
185 {
186 }
```

5.20.2.2 int xtea_ecb_decrypt (const unsigned char * *ct*, unsigned char * *pt*, symmetric_key * *skey*)

Decrypts a block of text with XTEA.

Parameters:

ct The input ciphertext (8 bytes)

pt The output plaintext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 112 of file xtea.c.

References LTC_ARGCHK.

Referenced by xtea_test().

```
{
   unsigned long y, z;
   int r;

   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);

   LOAD32L(y, &ct[0]);
   LOAD32L(z, &ct[4]);
   for (r = 31; r >= 0; r -= 4) {
       z = (z - ((((y<<4)^(y>>5)) + y) ^ skey->xtea.B[r])) & 0xFFFFFFFFUL;
       y = (y - ((((z<<4)^(z>>5)) + z) ^ skey->xtea.A[r])) & 0xFFFFFFFFUL;

       z = (z - ((((y<<4)^(y>>5)) + y) ^ skey->xtea.B[r-1])) & 0xFFFFFFFFUL;
       y = (y - ((((z<<4)^(z>>5)) + z) ^ skey->xtea.A[r-1])) & 0xFFFFFFFFUL;

       z = (z - ((((y<<4)^(y>>5)) + y) ^ skey->xtea.B[r-2])) & 0xFFFFFFFFUL;
       y = (y - ((((z<<4)^(z>>5)) + z) ^ skey->xtea.A[r-2])) & 0xFFFFFFFFUL;

       z = (z - ((((y<<4)^(y>>5)) + y) ^ skey->xtea.B[r-3])) & 0xFFFFFFFFUL;
       y = (y - ((((z<<4)^(z>>5)) + z) ^ skey->xtea.A[r-3])) & 0xFFFFFFFFUL;
   }
   STORE32L(y, &pt[0]);
   STORE32L(z, &pt[4]);
   return CRYPT_OK;
}
```

5.20.2.3 int xtea_ecb_encrypt (const unsigned char * *pt*, unsigned char * *ct*, symmetric_key * *skey*)

Encrypts a block of text with XTEA.

Parameters:

pt The input plaintext (8 bytes)

ct The output ciphertext (8 bytes)

skey The key as scheduled

Returns:

CRYPT_OK if successful

Definition at line 76 of file xtea.c.

References LTC_ARGCHK.

Referenced by xtea_test().

```
{
   unsigned long y, z;
   int r;

   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(skey != NULL);

   LOAD32L(y, &pt[0]);
   LOAD32L(z, &pt[4]);
   for (r = 0; r < 32; r += 4) {
       y = (y + ((((z<<4)^(z>>5)) + z) ^ skey->xtea.A[r])) & 0xFFFFFFFFUL;
       z = (z + ((((y<<4)^(y>>5)) + y) ^ skey->xtea.B[r])) & 0xFFFFFFFFUL;

       y = (y + ((((z<<4)^(z>>5)) + z) ^ skey->xtea.A[r+1])) & 0xFFFFFFFFUL;
       z = (z + ((((y<<4)^(y>>5)) + y) ^ skey->xtea.B[r+1])) & 0xFFFFFFFFUL;

       y = (y + ((((z<<4)^(z>>5)) + z) ^ skey->xtea.A[r+2])) & 0xFFFFFFFFUL;
       z = (z + ((((y<<4)^(y>>5)) + y) ^ skey->xtea.B[r+2])) & 0xFFFFFFFFUL;

       y = (y + ((((z<<4)^(z>>5)) + z) ^ skey->xtea.A[r+3])) & 0xFFFFFFFFUL;
       z = (z + ((((y<<4)^(y>>5)) + y) ^ skey->xtea.B[r+3])) & 0xFFFFFFFFUL;
   }
   STORE32L(y, &ct[0]);
   STORE32L(z, &ct[4]);
   return CRYPT_OK;
}
```

5.20.2.4 int xtea_keysize (int ∗ *keysize*)

Gets suitable key size.

Parameters:

keysize [in/out] The length of the recommended key (in bytes). This function will store the suitable size back in this variable.

Returns:

CRYPT_OK if the input key size is acceptable.

Definition at line 193 of file xtea.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(keysize != NULL);
   if (*keysize < 16) {
      return CRYPT_INVALID_KEYSIZE;
   }
   *keysize = 16;
   return CRYPT_OK;
}
```

5.20.2.5 int xtea_setup (const unsigned char * *key*, int *keylen*, int *num_rounds*, symmetric_key * *skey*)

Definition at line 34 of file xtea.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_INVALID_ROUNDS, K, and LTC_ARGCHK.

Referenced by xtea_test().

```
{
   unsigned long x, sum, K[4];

   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(skey != NULL);

   /* check arguments */
   if (keylen != 16) {
      return CRYPT_INVALID_KEYSIZE;
   }

   if (num_rounds != 0 && num_rounds != 32) {
      return CRYPT_INVALID_ROUNDS;
   }

   /* load key */
   LOAD32L(K[0], key+0);
   LOAD32L(K[1], key+4);
   LOAD32L(K[2], key+8);
   LOAD32L(K[3], key+12);

   for (x = sum = 0; x < 32; x++) {
       skey->xtea.A[x] = (sum + K[sum&3]) & 0xFFFFFFFFUL;
       sum = (sum + 0x9E3779B9UL) & 0xFFFFFFFFUL;
       skey->xtea.B[x] = (sum + K[(sum>>11)&3]) & 0xFFFFFFFFUL;
   }

#ifdef LTC_CLEAN_STACK
   zeromem(&K, sizeof(K));
#endif

   return CRYPT_OK;
}
```

5.20.2.6 int xtea_test (void)

Performs a self-test of the XTEA block cipher.

Returns:

CRYPT_OK if functional, CRYPT_NOP if self-test has been disabled

Definition at line 145 of file xtea.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, CRYPT_OK, XMEMCMP, xtea_ecb_decrypt(), xtea_ecb_encrypt(), and xtea_setup().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
   static const unsigned char key[16] =
      { 0x78, 0x56, 0x34, 0x12, 0xf0, 0xcd, 0xcb, 0x9a,
```

```
        0x48, 0x37, 0x26, 0x15, 0xc0, 0xbf, 0xae, 0x9d };
   static const unsigned char pt[8] =
      { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 };
   static const unsigned char ct[8] =
      { 0x75, 0xd7, 0xc5, 0xbf, 0xcf, 0x58, 0xc9, 0x3f };
   unsigned char tmp[2][8];
   symmetric_key skey;
   int err, y;

   if ((err = xtea_setup(key, 16, 0, &skey)) != CRYPT_OK)  {
      return err;
   }
   xtea_ecb_encrypt(pt, tmp[0], &skey);
   xtea_ecb_decrypt(tmp[0], tmp[1], &skey);

   if (XMEMCMP(tmp[0], ct, 8) != 0 || XMEMCMP(tmp[1], pt, 8) != 0) {
      return CRYPT_FAIL_TESTVECTOR;
   }

      /* now see if we can encrypt all zero bytes 1000 times, decrypt and come back where we started *
      for (y = 0; y < 8; y++) tmp[0][y] = 0;
      for (y = 0; y < 1000; y++) xtea_ecb_encrypt(tmp[0], tmp[0], &skey);
      for (y = 0; y < 1000; y++) xtea_ecb_decrypt(tmp[0], tmp[0], &skey);
      for (y = 0; y < 8; y++) if (tmp[0][y] != 0) return CRYPT_FAIL_TESTVECTOR;

   return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.20.3 Variable Documentation

5.20.3.1 const struct ltc_cipher_descriptor xtea_desc

Initial value:

```
{
    "xtea",
    1,
    16, 16, 8, 32,
    &xtea_setup,
    &xtea_ecb_encrypt,
    &xtea_ecb_decrypt,
    &xtea_test,
    &xtea_done,
    &xtea_keysize,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL
}
```

Definition at line 20 of file xtea.c.

Referenced by yarrow_start().

5.21 encauth/ccm/ccm_memory.c File Reference

5.21.1 Detailed Description

CCM support, process a block of memory, Tom St Denis.

Definition in file ccm_memory.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ccm_memory.c:

Functions

- int ccm_memory (int cipher, const unsigned char *key, unsigned long keylen, symmetric_key *uskey, const unsigned char *nonce, unsigned long noncelen, const unsigned char *header, unsigned long headerlen, unsigned char *pt, unsigned long ptlen, unsigned char *ct, unsigned char *tag, unsigned long *taglen, int direction)

  CCM encrypt/decrypt and produce an authentication tag.

5.21.2 Function Documentation

5.21.2.1 int ccm_memory (int *cipher*, const unsigned char * *key*, unsigned long *keylen*, symmetric_key * *uskey*, const unsigned char * *nonce*, unsigned long *noncelen*, const unsigned char * *header*, unsigned long *headerlen*, unsigned char * *pt*, unsigned long *ptlen*, unsigned char * *ct*, unsigned char * *tag*, unsigned long * *taglen*, int *direction*)

CCM encrypt/decrypt and produce an authentication tag.

Parameters:

cipher The index of the cipher desired

key The secret key to use

keylen The length of the secret key (octets)

uskey A previously scheduled key [optional can be NULL]

nonce The session nonce [use once]

noncelen The length of the nonce

header The header for the session

headerlen The length of the header (octets)

pt [out] The plaintext

ptlen The length of the plaintext (octets)

ct [out] The ciphertext

tag [out] The destination tag

taglen [in/out] The max size and resulting size of the authentication tag

direction Encrypt or Decrypt direction (0 or 1)

Returns:

CRYPT_OK if successful

Definition at line 38 of file ccm_memory.c.

References ltc_cipher_descriptor::accel_ccm_memory, ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_INVALID_CIPHER, CRYPT_MEM, CRYPT_OK, len, LTC_ARGCHK, ltc_cipher_descriptor::setup, XFREE, and XMALLOC.

Referenced by ccm_test().

```
{
   unsigned char  PAD[16], ctr[16], CTRPAD[16], b;
   symmetric_key *skey;
   int            err;
   unsigned long  len, L, x, y, z, CTRlen;

   if (uskey == NULL) {
      LTC_ARGCHK(key    != NULL);
   }
   LTC_ARGCHK(nonce  != NULL);
   if (headerlen > 0) {
      LTC_ARGCHK(header != NULL);
   }
   LTC_ARGCHK(pt     != NULL);
   LTC_ARGCHK(ct     != NULL);
   LTC_ARGCHK(tag    != NULL);
   LTC_ARGCHK(taglen != NULL);

#ifdef LTC_FAST
   if (16 % sizeof(LTC_FAST_TYPE)) {
      return CRYPT_INVALID_ARG;
   }
#endif

   /* check cipher input */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }
   if (cipher_descriptor[cipher].block_length != 16) {
      return CRYPT_INVALID_CIPHER;
   }

   /* make sure the taglen is even and <= 16 */
   *taglen &= ~1;
   if (*taglen > 16) {
      *taglen = 16;
   }

   /* can't use < 4 */
   if (*taglen < 4) {
      return CRYPT_INVALID_ARG;
   }

   /* is there an accelerator? */
   if (cipher_descriptor[cipher].accel_ccm_memory != NULL) {
       return cipher_descriptor[cipher].accel_ccm_memory(
           key,    keylen,
           uskey,
           nonce,  noncelen,
           header, headerlen,
           pt,     ptlen,
           ct,
           tag,    taglen,
           direction);
   }

   /* let's get the L value */
   len = ptlen;
   L   = 0;
```

```
   while (len) {
      ++L;
      len >>= 8;
   }
   if (L <= 1) {
      L = 2;
   }

   /* increase L to match the nonce len */
   noncelen = (noncelen > 13) ? 13 : noncelen;
   if ((15 - noncelen) > L) {
      L = 15 - noncelen;
   }

   /* decrease noncelen to match L */
   if ((noncelen + L) > 15) {
      noncelen = 15 - L;
   }

   /* allocate mem for the symmetric key */
   if (uskey == NULL) {
      skey = XMALLOC(sizeof(*skey));
      if (skey == NULL) {
         return CRYPT_MEM;
      }

      /* initialize the cipher */
      if ((err = cipher_descriptor[cipher].setup(key, keylen, 0, skey)) != CRYPT_OK) {
         XFREE(skey);
         return err;
      }
   } else {
      skey = uskey;
   }

   /* form B_0 == flags | Nonce N | l(m) */
   x = 0;
   PAD[x++] = ((headerlen > 0) ? (1<<6) : 0) |
            (((*taglen - 2)>>1)<<3)        |
            (L-1);

   /* nonce */
   for (y = 0; y < (16 - (L + 1)); y++) {
       PAD[x++] = nonce[y];
   }

   /* store len */
   len = ptlen;

   /* shift len so the upper bytes of len are the contents of the length */
   for (y = L; y < 4; y++) {
       len <<= 8;
   }

   /* store l(m) (only store 32-bits) */
   for (y = 0; L > 4 && (L-y)>4; y++) {
       PAD[x++] = 0;
   }
   for (; y < L; y++) {
       PAD[x++] = (len >> 24) & 255;
       len <<= 8;
   }

   /* encrypt PAD */
   if ((err = cipher_descriptor[cipher].ecb_encrypt(PAD, PAD, skey)) != CRYPT_OK) {
       goto error;
   }
```

```

   /* handle header */
   if (headerlen > 0) {
      x = 0;

      /* store length */
      if (headerlen < ((1UL<<16) - (1UL<<8))) {
         PAD[x++] ^= (headerlen>>8) & 255;
         PAD[x++] ^= headerlen & 255;
      } else {
         PAD[x++] ^= 0xFF;
         PAD[x++] ^= 0xFE;
         PAD[x++] ^= (headerlen>>24) & 255;
         PAD[x++] ^= (headerlen>>16) & 255;
         PAD[x++] ^= (headerlen>>8) & 255;
         PAD[x++] ^= headerlen & 255;
      }

      /* now add the data */
      for (y = 0; y < headerlen; y++) {
          if (x == 16) {
             /* full block so let's encrypt it */
             if ((err = cipher_descriptor[cipher].ecb_encrypt(PAD, PAD, skey)) != CRYPT_OK) {
                goto error;
             }
             x = 0;
          }
          PAD[x++] ^= header[y];
      }

      /* remainder? */
      if (x != 0) {
         if ((err = cipher_descriptor[cipher].ecb_encrypt(PAD, PAD, skey)) != CRYPT_OK) {
            goto error;
         }
      }
   }

   /* setup the ctr counter */
   x = 0;

   /* flags */
   ctr[x++] = L-1;

   /* nonce */
   for (y = 0; y < (16 - (L+1)); ++y) {
      ctr[x++] = nonce[y];
   }
   /* offset */
   while (x < 16) {
      ctr[x++] = 0;
   }

   x      = 0;
   CTRlen = 16;

   /* now handle the PT */
   if (ptlen > 0) {
      y = 0;
#ifdef LTC_FAST
      if (ptlen & ~15)  {
          if (direction == CCM_ENCRYPT) {
             for (; y < (ptlen & ~15); y += 16) {
                /* increment the ctr? */
                for (z = 15; z > 15-L; z--) {
                    ctr[z] = (ctr[z] + 1) & 255;
                    if (ctr[z]) break;
```

```
                }
                if ((err = cipher_descriptor[cipher].ecb_encrypt(ctr, CTRPAD, skey)) != CRYPT_OK) {
                   goto error;
                }

                /* xor the PT against the pad first */
                for (z = 0; z < 16; z += sizeof(LTC_FAST_TYPE)) {
                    *((LTC_FAST_TYPE*)(&PAD[z]))  ^= *((LTC_FAST_TYPE*)(&pt[y+z]));
                    *((LTC_FAST_TYPE*)(&ct[y+z])) = *((LTC_FAST_TYPE*)(&pt[y+z])) ^ *((LTC_FAST_TYPE*)
                }
                if ((err = cipher_descriptor[cipher].ecb_encrypt(PAD, PAD, skey)) != CRYPT_OK) {
                   goto error;
                }
             }
         } else {
             for (; y < (ptlen & ~15); y += 16) {
                /* increment the ctr? */
                for (z = 15; z > 15-L; z--) {
                    ctr[z] = (ctr[z] + 1) & 255;
                    if (ctr[z]) break;
                }
                if ((err = cipher_descriptor[cipher].ecb_encrypt(ctr, CTRPAD, skey)) != CRYPT_OK) {
                   goto error;
                }

                /* xor the PT against the pad last */
                for (z = 0; z < 16; z += sizeof(LTC_FAST_TYPE)) {
                    *((LTC_FAST_TYPE*)(&pt[y+z])) = *((LTC_FAST_TYPE*)(&ct[y+z])) ^ *((LTC_FAST_TYPE*)
                    *((LTC_FAST_TYPE*)(&PAD[z]))  ^= *((LTC_FAST_TYPE*)(&pt[y+z]));
                }
                if ((err = cipher_descriptor[cipher].ecb_encrypt(PAD, PAD, skey)) != CRYPT_OK) {
                   goto error;
                }
             }
         }
     }
#endif

      for (; y < ptlen; y++) {
          /* increment the ctr? */
          if (CTRlen == 16) {
             for (z = 15; z > 15-L; z--) {
                 ctr[z] = (ctr[z] + 1) & 255;
                 if (ctr[z]) break;
             }
             if ((err = cipher_descriptor[cipher].ecb_encrypt(ctr, CTRPAD, skey)) != CRYPT_OK) {
                goto error;
             }
             CTRlen = 0;
          }

          /* if we encrypt we add the bytes to the MAC first */
          if (direction == CCM_ENCRYPT) {
             b     = pt[y];
             ct[y] = b ^ CTRPAD[CTRlen++];
          } else {
             b     = ct[y] ^ CTRPAD[CTRlen++];
             pt[y] = b;
          }

          if (x == 16) {
             if ((err = cipher_descriptor[cipher].ecb_encrypt(PAD, PAD, skey)) != CRYPT_OK) {
                goto error;
             }
             x = 0;
          }
          PAD[x++] ^= b;
```

```
      }

      if (x != 0) {
         if ((err = cipher_descriptor[cipher].ecb_encrypt(PAD, PAD, skey)) != CRYPT_OK) {
            goto error;
         }
      }
   }

   /* setup CTR for the TAG (zero the count) */
   for (y = 15; y > 15 - L; y--) {
      ctr[y] = 0x00;
   }
   if ((err = cipher_descriptor[cipher].ecb_encrypt(ctr, CTRPAD, skey)) != CRYPT_OK) {
      goto error;
   }

   if (skey != uskey) {
      cipher_descriptor[cipher].done(skey);
   }

   /* store the TAG */
   for (x = 0; x < 16 && x < *taglen; x++) {
       tag[x] = PAD[x] ^ CTRPAD[x];
   }
   *taglen = x;

#ifdef LTC_CLEAN_STACK
   zeromem(skey,   sizeof(*skey));
   zeromem(PAD,    sizeof(PAD));
   zeromem(CTRPAD, sizeof(CTRPAD));
#endif
error:
   if (skey != uskey) {
      XFREE(skey);
   }

   return err;
}
```

Here is the call graph for this function:

5.22 encauth/ccm/ccm_test.c File Reference

5.22.1 Detailed Description

CCM support, process a block of memory, Tom St Denis.

Definition in file ccm_test.c.

`#include "tomcrypt.h"`

Include dependency graph for ccm_test.c:

Functions

- int ccm_test (void)

5.22.2 Function Documentation

5.22.2.1 int ccm_test (void)

Definition at line 20 of file ccm_test.c.

References ccm_memory(), cipher_descriptor, CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, CRYPT_-OK, ltc_cipher_descriptor::done, find_cipher(), and XMEMCMP.

```
21 {
22 #ifndef LTC_TEST
23    return CRYPT_NOP;
24 #else
25    static const struct {
26        unsigned char key[16];
27        unsigned char nonce[16];
28        int           noncelen;
29        unsigned char header[64];
30        int           headerlen;
31        unsigned char pt[64];
32        int           ptlen;
33        unsigned char ct[64];
34        unsigned char tag[16];
35        int           taglen;
36    } tests[] = {
37
38 /* 13 byte nonce, 8 byte auth, 23 byte pt */
39 {
40    { 0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7,
41      0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF },
42    { 0x00, 0x00, 0x00, 0x03, 0x02, 0x01, 0x00, 0xA0,
43      0xA1, 0xA2, 0xA3, 0xA4, 0xA5 },
44    13,
45    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07 },
46    8,
47    { 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
48      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
49      0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E },
50    23,
51    { 0x58, 0x8C, 0x97, 0x9A, 0x61, 0xC6, 0x63, 0xD2,
52      0xF0, 0x66, 0xD0, 0xC2, 0xC0, 0xF9, 0x89, 0x80,
53      0x6D, 0x5F, 0x6B, 0x61, 0xDA, 0xC3, 0x84 },
54    { 0x17, 0xe8, 0xd1, 0x2c, 0xfd, 0xf9, 0x26, 0xe0 },
55    8
56 },
```

```

/* 13 byte nonce, 12 byte header, 19 byte pt */
{
   { 0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7,
     0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF },
   { 0x00, 0x00, 0x00, 0x06, 0x05, 0x04, 0x03, 0xA0,
     0xA1, 0xA2, 0xA3, 0xA4, 0xA5 },
   13,
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0A, 0x0B },
   12,
   { 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13,
     0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B,
     0x1C, 0x1D, 0x1E },
   19,
   { 0xA2, 0x8C, 0x68, 0x65, 0x93, 0x9A, 0x9A, 0x79,
     0xFA, 0xAA, 0x5C, 0x4C, 0x2A, 0x9D, 0x4A, 0x91,
     0xCD, 0xAC, 0x8C },
   { 0x96, 0xC8, 0x61, 0xB9, 0xC9, 0xE6, 0x1E, 0xF1 },
   8
},

/* supplied by Brian Gladman */
{
   { 0x40, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47,
     0x48, 0x49, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f },
   { 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16  },
   7,
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07 },
   8,
   { 0x20, 0x21, 0x22, 0x23 },
   4,
   { 0x71, 0x62, 0x01, 0x5b },
   { 0x4d, 0xac, 0x25, 0x5d },
   4
},

{
   { 0xc9, 0x7c, 0x1f, 0x67, 0xce, 0x37, 0x11, 0x85,
     0x51, 0x4a, 0x8a, 0x19, 0xf2, 0xbd, 0xd5, 0x2f },
   { 0x00, 0x50, 0x30, 0xf1, 0x84, 0x44, 0x08, 0xb5,
     0x03, 0x97, 0x76, 0xe7, 0x0c },
   13,
   { 0x08, 0x40, 0x0f, 0xd2, 0xe1, 0x28, 0xa5, 0x7c,
     0x50, 0x30, 0xf1, 0x84, 0x44, 0x08, 0xab, 0xae,
     0xa5, 0xb8, 0xfc, 0xba, 0x00, 0x00 },
   22,
   { 0xf8, 0xba, 0x1a, 0x55, 0xd0, 0x2f, 0x85, 0xae,
     0x96, 0x7b, 0xb6, 0x2f, 0xb6, 0xcd, 0xa8, 0xeb,
     0x7e, 0x78, 0xa0, 0x50 },
   20,
   { 0xf3, 0xd0, 0xa2, 0xfe, 0x9a, 0x3d, 0xbf, 0x23,
     0x42, 0xa6, 0x43, 0xe4, 0x32, 0x46, 0xe8, 0x0c,
     0x3c, 0x04, 0xd0, 0x19 },
   { 0x78, 0x45, 0xce, 0x0b, 0x16, 0xf9, 0x76, 0x23 },
   8
},

};
  unsigned long taglen, x;
  unsigned char buf[64], buf2[64], tag2[16], tag[16];
  int           err, idx;
  symmetric_key skey;

  idx = find_cipher("aes");
  if (idx == -1) {
     idx = find_cipher("rijndael");
```

```
     if (idx == -1) {
        return CRYPT_NOP;
     }
  }

  for (x = 0; x < (sizeof(tests)/sizeof(tests[0])); x++) {
      taglen = tests[x].taglen;
      if ((err = cipher_descriptor[idx].setup(tests[x].key, 16, 0, &skey)) != CRYPT_OK) {
         return err;
      }

      if ((err = ccm_memory(idx,
                            tests[x].key, 16,
                            &skey,
                            tests[x].nonce, tests[x].noncelen,
                            tests[x].header, tests[x].headerlen,
                            (unsigned char*)tests[x].pt, tests[x].ptlen,
                            buf,
                            tag, &taglen, 0)) != CRYPT_OK) {
         return err;
      }

      if (XMEMCMP(buf, tests[x].ct, tests[x].ptlen)) {
         return CRYPT_FAIL_TESTVECTOR;
      }
      if (XMEMCMP(tag, tests[x].tag, tests[x].taglen)) {
         return CRYPT_FAIL_TESTVECTOR;
      }

      if ((err = ccm_memory(idx,
                            tests[x].key, 16,
                            NULL,
                            tests[x].nonce, tests[x].noncelen,
                            tests[x].header, tests[x].headerlen,
                            buf2, tests[x].ptlen,
                            buf,
                            tag2, &taglen, 1    )) != CRYPT_OK) {
         return err;
      }

      if (XMEMCMP(buf2, tests[x].pt, tests[x].ptlen)) {
         return CRYPT_FAIL_TESTVECTOR;
      }
      if (XMEMCMP(tag2, tests[x].tag, tests[x].taglen)) {
         return CRYPT_FAIL_TESTVECTOR;
      }
      cipher_descriptor[idx].done(&skey);
  }
  return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.23 encauth/eax/eax_addheader.c File Reference

5.23.1 Detailed Description

EAX implementation, add meta-data, by Tom St Denis.

Definition in file eax_addheader.c.

`#include "tomcrypt.h"`

Include dependency graph for eax_addheader.c:

Functions

- int eax_addheader (eax_state *eax, const unsigned char *header, unsigned long length)

    add header (metadata) to the stream

5.23.2 Function Documentation

5.23.2.1 int eax_addheader (eax_state * *eax*, const unsigned char * *header*, unsigned long *length*)

add header (metadata) to the stream

Parameters:

eax The current EAX state

header The header (meta-data) data you wish to add to the state

length The length of the header data

Returns:

CRYPT_OK if successful

Definition at line 26 of file eax_addheader.c.

References LTC_ARGCHK, and omac_process().

```
{
   LTC_ARGCHK(eax    != NULL);
   LTC_ARGCHK(header != NULL);
   return omac_process(&eax->headeromac, header, length);
}
```

Here is the call graph for this function:

5.24 encauth/eax/eax_decrypt.c File Reference

5.24.1 Detailed Description

EAX implementation, decrypt block, by Tom St Denis.

Definition in file eax_decrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for eax_decrypt.c:

Functions

- int eax_decrypt (eax_state ∗eax, const unsigned char ∗ct, unsigned char ∗pt, unsigned long length)

  Decrypt data with the EAX protocol.

5.24.2 Function Documentation

5.24.2.1 int eax_decrypt (eax_state ∗ *eax*, const unsigned char ∗ *ct*, unsigned char ∗ *pt*, unsigned long *length*)

Decrypt data with the EAX protocol.

Parameters:

eax The EAX state

ct The ciphertext

pt [out] The plaintext

length The length (octets) of the ciphertext

Returns:

CRYPT_OK if successful

Definition at line 28 of file eax_decrypt.c.

References CRYPT_OK, ctr_decrypt(), LTC_ARGCHK, and omac_process().

Referenced by eax_decrypt_verify_memory().

```
30 {
31    int err;
32
33    LTC_ARGCHK(eax != NULL);
34    LTC_ARGCHK(pt  != NULL);
35    LTC_ARGCHK(ct  != NULL);
36
37    /* omac ciphertext */
38    if ((err = omac_process(&eax->ctomac, ct, length)) != CRYPT_OK) {
39       return err;
40    }
41
42    /* decrypt  */
43    return ctr_decrypt(ct, pt, length, &eax->ctr);
44 }
```

Here is the call graph for this function:

5.25 encauth/eax/eax_decrypt_verify_memory.c File Reference

5.25.1 Detailed Description

EAX implementation, decrypt block of memory, by Tom St Denis.

Definition in file eax_decrypt_verify_memory.c.

`#include "tomcrypt.h"`

Include dependency graph for eax_decrypt_verify_memory.c:

Functions

- int eax_decrypt_verify_memory (int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗nonce, unsigned long noncelen, const unsigned char ∗header, unsigned long headerlen, const unsigned char ∗ct, unsigned long ctlen, unsigned char ∗pt, unsigned char ∗tag, unsigned long taglen, int ∗stat)

  Decrypt a block of memory and verify the provided MAC tag with EAX.

5.25.2 Function Documentation

5.25.2.1 int eax_decrypt_verify_memory (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *nonce*, unsigned long *noncelen*, const unsigned char ∗ *header*, unsigned long *headerlen*, const unsigned char ∗ *ct*, unsigned long *ctlen*, unsigned char ∗ *pt*, unsigned char ∗ *tag*, unsigned long *taglen*, int ∗ *stat*)

Decrypt a block of memory and verify the provided MAC tag with EAX.

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the key (octets)

nonce The nonce data (use once) for the session

noncelen The length of the nonce data.

header The session header data

headerlen The length of the header (octets)

ct The ciphertext

ctlen The length of the ciphertext (octets)

pt [out] The plaintext

tag The authentication tag provided by the encoder

taglen [in/out] The length of the tag (octets)

stat [out] The result of the decryption (1==valid tag, 0==invalid)

Returns:

CRYPT_OK if successful regardless of the resulting tag comparison

Definition at line 37 of file eax_decrypt_verify_memory.c.

References CRYPT_MEM, CRYPT_OK, eax_decrypt(), eax_done(), eax_init(), LTC_ARGCHK, XFREE, XMALLOC, XMEMCMP, and zeromem().

```
{
   int           err;
   eax_state     *eax;
   unsigned char *buf;
   unsigned long  buflen;

   LTC_ARGCHK(stat != NULL);
   LTC_ARGCHK(key  != NULL);
   LTC_ARGCHK(pt   != NULL);
   LTC_ARGCHK(ct   != NULL);
   LTC_ARGCHK(tag  != NULL);

   /* default to zero */
   *stat = 0;

   /* allocate ram */
   buf = XMALLOC(taglen);
   eax = XMALLOC(sizeof(*eax));
   if (eax == NULL || buf == NULL) {
      if (eax != NULL) {
         XFREE(eax);
      }
      if (buf != NULL) {
         XFREE(buf);
      }
      return CRYPT_MEM;
   }

   if ((err = eax_init(eax, cipher, key, keylen, nonce, noncelen, header, headerlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   if ((err = eax_decrypt(eax, ct, pt, ctlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   buflen = taglen;
   if ((err = eax_done(eax, buf, &buflen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* compare tags */
   if (buflen >= taglen && XMEMCMP(buf, tag, taglen) == 0) {
      *stat = 1;
   }

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(buf, taglen);
   zeromem(eax, sizeof(*eax));
#endif

   XFREE(eax);
   XFREE(buf);

    return err;
 }
```

Here is the call graph for this function:

5.26 encauth/eax/eax_done.c File Reference

5.26.1 Detailed Description

EAX implementation, terminate session, by Tom St Denis.

Definition in file eax_done.c.

`#include "tomcrypt.h"`

Include dependency graph for eax_done.c:

Functions

- int eax_done (eax_state *eax, unsigned char *tag, unsigned long *taglen)

    Terminate an EAX session and get the tag.

5.26.2 Function Documentation

5.26.2.1 int eax_done (eax_state * *eax*, unsigned char * *tag*, unsigned long * *taglen*)

Terminate an EAX session and get the tag.

Parameters:

eax The EAX state

tag [out] The destination of the authentication tag

taglen [in/out] The max length and resulting length of the authentication tag

Returns:

CRYPT_OK if successful

Definition at line 27 of file eax_done.c.

References CRYPT_MEM, CRYPT_OK, ctr_done(), len, LTC_ARGCHK, MAXBLOCKSIZE, omac_-done(), XFREE, and XMALLOC.

Referenced by eax_decrypt_verify_memory(), and eax_encrypt_authenticate_memory().

```
28 {
29    int           err;
30    unsigned char *headermac, *ctmac;
31    unsigned long x, len;
32
33    LTC_ARGCHK(eax    != NULL);
34    LTC_ARGCHK(tag    != NULL);
35    LTC_ARGCHK(taglen != NULL);
36
37    /* allocate ram */
38    headermac = XMALLOC(MAXBLOCKSIZE);
39    ctmac     = XMALLOC(MAXBLOCKSIZE);
40
41    if (headermac == NULL || ctmac == NULL) {
42       if (headermac != NULL) {
43          XFREE(headermac);
44       }
```

```
      if (ctmac != NULL) {
         XFREE(ctmac);
      }
      return CRYPT_MEM;
   }

   /* finish ctomac */
   len = MAXBLOCKSIZE;
   if ((err = omac_done(&eax->ctomac, ctmac, &len)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* finish headeromac */

   /* note we specifically don't reset len so the two lens are minimal */

   if ((err = omac_done(&eax->headeromac, headermac, &len)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* terminate the CTR chain */
   if ((err = ctr_done(&eax->ctr)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* compute N xor H xor C */
   for (x = 0; x < len && x < *taglen; x++) {
       tag[x] = eax->N[x] ^ headermac[x] ^ ctmac[x];
   }
   *taglen = x;

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(ctmac,     MAXBLOCKSIZE);
   zeromem(headermac, MAXBLOCKSIZE);
   zeromem(eax,       sizeof(*eax));
#endif

   XFREE(ctmac);
   XFREE(headermac);

   return err;
}
```

Here is the call graph for this function:

5.27 encauth/eax/eax_encrypt.c File Reference

5.27.1 Detailed Description

EAX implementation, encrypt block by Tom St Denis.

Definition in file eax_encrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for eax_encrypt.c:

Functions

- int eax_encrypt (eax_state *eax, const unsigned char *pt, unsigned char *ct, unsigned long length)
  Encrypt with EAX a block of data.

5.27.2 Function Documentation

5.27.2.1 int eax_encrypt (eax_state * *eax*, const unsigned char * *pt*, unsigned char * *ct*, unsigned long *length*)

Encrypt with EAX a block of data.

Parameters:

eax The EAX state

pt The plaintext to encrypt

ct [out] The ciphertext as encrypted

length The length of the plaintext (octets)

Returns:

CRYPT_OK if successful

Definition at line 28 of file eax_encrypt.c.

References CRYPT_OK, ctr_encrypt(), LTC_ARGCHK, and omac_process().

Referenced by eax_encrypt_authenticate_memory().

```
30 {
31    int err;
32 
33    LTC_ARGCHK(eax != NULL);
34    LTC_ARGCHK(pt  != NULL);
35    LTC_ARGCHK(ct  != NULL);
36 
37    /* encrypt */
38    if ((err = ctr_encrypt(pt, ct, length, &eax->ctr)) != CRYPT_OK) {
39       return err;
40    }
41 
42    /* omac ciphertext */
43    return omac_process(&eax->ctomac, ct, length);
44 }
```

Here is the call graph for this function:

5.28 encauth/eax/eax_encrypt_authenticate_memory.c File Reference

5.28.1 Detailed Description

EAX implementation, encrypt a block of memory, by Tom St Denis.

Definition in file eax_encrypt_authenticate_memory.c.

`#include "tomcrypt.h"`

Include dependency graph for eax_encrypt_authenticate_memory.c:

Functions

- int eax_encrypt_authenticate_memory (int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗nonce, unsigned long noncelen, const unsigned char ∗header, unsigned long headerlen, const unsigned char ∗pt, unsigned long ptlen, unsigned char ∗ct, unsigned char ∗tag, unsigned long ∗taglen)

  EAX encrypt and produce an authentication tag.

5.28.2 Function Documentation

5.28.2.1 int eax_encrypt_authenticate_memory (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *nonce*, unsigned long *noncelen*, const unsigned char ∗ *header*, unsigned long *headerlen*, const unsigned char ∗ *pt*, unsigned long *ptlen*, unsigned char ∗ *ct*, unsigned char ∗ *tag*, unsigned long ∗ *taglen*)

EAX encrypt and produce an authentication tag.

Parameters:

cipher The index of the cipher desired

key The secret key to use

keylen The length of the secret key (octets)

nonce The session nonce [use once]

noncelen The length of the nonce

header The header for the session

headerlen The length of the header (octets)

pt The plaintext

ptlen The length of the plaintext (octets)

ct [out] The ciphertext

tag [out] The destination tag

taglen [in/out] The max size and resulting size of the authentication tag

Returns:

CRYPT_OK if successful

Definition at line 36 of file eax_encrypt_authenticate_memory.c.

References CRYPT_OK, eax_done(), eax_encrypt(), eax_init(), LTC_ARGCHK, XFREE, XMALLOC, and zeromem().

Referenced by eax_test().

```
{
   int err;
   eax_state *eax;

   LTC_ARGCHK(key    != NULL);
   LTC_ARGCHK(pt     != NULL);
   LTC_ARGCHK(ct     != NULL);
   LTC_ARGCHK(tag    != NULL);
   LTC_ARGCHK(taglen != NULL);

   eax = XMALLOC(sizeof(*eax));

   if ((err = eax_init(eax, cipher, key, keylen, nonce, noncelen, header, headerlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   if ((err = eax_encrypt(eax, pt, ct, ptlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   if ((err = eax_done(eax, tag, taglen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(eax, sizeof(*eax));
#endif

   XFREE(eax);

   return err;
}
```

Here is the call graph for this function:

5.29 encauth/eax/eax_init.c File Reference

5.29.1 Detailed Description

EAX implementation, initialized EAX state, by Tom St Denis.

Definition in file eax_init.c.

`#include "tomcrypt.h"`

Include dependency graph for eax_init.c:

Functions

- int eax_init (eax_state ∗eax, int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗nonce, unsigned long noncelen, const unsigned char ∗header, unsigned long headerlen)

  Initialized an EAX state.

5.29.2 Function Documentation

5.29.2.1 int eax_init (eax_state ∗ *eax*, int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *nonce*, unsigned long *noncelen*, const unsigned char ∗ *header*, unsigned long *headerlen*)

Initialized an EAX state.

Parameters:

eax [out] The EAX state to initialize

cipher The index of the desired cipher

key The secret key

keylen The length of the secret key (octets)

nonce The use-once nonce for the session

noncelen The length of the nonce (octets)

header The header for the EAX state

headerlen The header length (octets)

Returns:

CRYPT_OK if successful

Definition at line 32 of file eax_init.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_MEM, CRYPT_OK, ctr_start(), len, LTC_ARGCHK, MAXBLOCKSIZE, omac_done(), omac_init(), omac_-process(), XFREE, XMALLOC, and zeromem().

Referenced by eax_decrypt_verify_memory(), and eax_encrypt_authenticate_memory().

```
{
   unsigned char *buf;
   int           err, blklen;
```

```
   omac_state    *omac;
   unsigned long len;


   LTC_ARGCHK(eax   != NULL);
   LTC_ARGCHK(key   != NULL);
   LTC_ARGCHK(nonce != NULL);
   if (headerlen > 0) {
      LTC_ARGCHK(header != NULL);
   }

   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }
   blklen = cipher_descriptor[cipher].block_length;

   /* allocate ram */
   buf  = XMALLOC(MAXBLOCKSIZE);
   omac = XMALLOC(sizeof(*omac));

   if (buf == NULL || omac == NULL) {
      if (buf != NULL) {
         XFREE(buf);
      }
      if (omac != NULL) {
         XFREE(omac);
      }
      return CRYPT_MEM;
   }

   /* N = OMAC_0K(nonce) */
   zeromem(buf, MAXBLOCKSIZE);
   if ((err = omac_init(omac, cipher, key, keylen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* omac the [0]_n */
   if ((err = omac_process(omac, buf, blklen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   /* omac the nonce */
   if ((err = omac_process(omac, nonce, noncelen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   /* store result */
   len = sizeof(eax->N);
   if ((err = omac_done(omac, eax->N, &len)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* H = OMAC_1K(header) */
   zeromem(buf, MAXBLOCKSIZE);
   buf[blklen - 1] = 1;

   if ((err = omac_init(&eax->headeromac, cipher, key, keylen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* omac the [1]_n */
   if ((err = omac_process(&eax->headeromac, buf, blklen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   /* omac the header */
   if (headerlen != 0) {
      if ((err = omac_process(&eax->headeromac, header, headerlen)) != CRYPT_OK) {
          goto LBL_ERR;
      }
```

```
   }

   /* note we don’t finish the headeromac, this allows us to add more header later */

   /* setup the CTR mode */
   if ((err = ctr_start(cipher, eax->N, key, keylen, 0, CTR_COUNTER_BIG_ENDIAN, &eax->ctr)) != CRYPT_C
      goto LBL_ERR;
   }

   /* setup the OMAC for the ciphertext */
   if ((err = omac_init(&eax->ctomac, cipher, key, keylen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* omac [2]_n */
   zeromem(buf, MAXBLOCKSIZE);
   buf[blklen-1] = 2;
   if ((err = omac_process(&eax->ctomac, buf, blklen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(buf,  MAXBLOCKSIZE);
   zeromem(omac, sizeof(*omac));
#endif

   XFREE(omac);
   XFREE(buf);

   return err;
}
```

Here is the call graph for this function:

5.30 encauth/eax/eax_test.c File Reference

5.30.1 Detailed Description

EAX implementation, self-test, by Tom St Denis.

Definition in file eax_test.c.

`#include "tomcrypt.h"`

Include dependency graph for eax_test.c:

Functions

- int eax_test (void)

    Test the EAX implementation.

5.30.2 Function Documentation

5.30.2.1 int eax_test (void)

Test the EAX implementation.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-testing has been disabled

Definition at line 24 of file eax_test.c.

References CRYPT_NOP, CRYPT_OK, eax_encrypt_authenticate_memory(), find_cipher(), len, and MAXBLOCKSIZE.

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   static const struct {
       int               keylen,
                       noncelen,
                      headerlen,
                         msglen;

       unsigned char        key[MAXBLOCKSIZE],
                          nonce[MAXBLOCKSIZE],
                         header[MAXBLOCKSIZE],
                      plaintext[MAXBLOCKSIZE],
                     ciphertext[MAXBLOCKSIZE],
                            tag[MAXBLOCKSIZE];
   } tests[] = {

/* NULL message */
{
   16, 0, 0, 0,
   /* key */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* nonce */
   { 0 },
```

```
   /* header */
   { 0 },
   /* plaintext */
   { 0 },
   /* ciphertext */
   { 0 },
   /* tag */
   { 0x9a, 0xd0, 0x7e, 0x7d, 0xbf, 0xf3, 0x01, 0xf5,
     0x05, 0xde, 0x59, 0x6b, 0x96, 0x15, 0xdf, 0xff }
},

/* test with nonce */
{
   16, 16, 0, 0,
   /* key */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* nonce */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* header */
   { 0 },
   /* plaintext */
   { 0 },
   /* ciphertext */
   { 0 },
   /* tag */
   { 0x1c, 0xe1, 0x0d, 0x3e, 0xff, 0xd4, 0xca, 0xdb,
     0xe2, 0xe4, 0x4b, 0x58, 0xd6, 0x0a, 0xb9, 0xec }
},

/* test with header [no nonce]  */
{
   16, 0, 16, 0,
   /* key */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* nonce */
   { 0 },
   /* header */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* plaintext */
   { 0 },
   /* ciphertext */
   { 0 },
   /* tag */
   { 0x3a, 0x69, 0x8f, 0x7a, 0x27, 0x0e, 0x51, 0xb0,
     0xf6, 0x5b, 0x3d, 0x3e, 0x47, 0x19, 0x3c, 0xff }
},

/* test with header + nonce + plaintext */
{
   16, 16, 16, 32,
   /* key */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* nonce */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* header */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* plaintext */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
     0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
```

```
     0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f },
   /* ciphertext */
   { 0x29, 0xd8, 0x78, 0xd1, 0xa3, 0xbe, 0x85, 0x7b,
     0x6f, 0xb8, 0xc8, 0xea, 0x59, 0x50, 0xa7, 0x78,
     0x33, 0x1f, 0xbf, 0x2c, 0xcf, 0x33, 0x98, 0x6f,
     0x35, 0xe8, 0xcf, 0x12, 0x1d, 0xcb, 0x30, 0xbc },
   /* tag */
   { 0x4f, 0xbe, 0x03, 0x38, 0xbe, 0x1c, 0x8c, 0x7e,
     0x1d, 0x7a, 0xe7, 0xe4, 0x5b, 0x92, 0xc5, 0x87 }
},

/* test with header + nonce + plaintext [not even sizes!] */
{
   16, 15, 14, 29,
   /* key */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* nonce */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e },
   /* header */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d },
   /* plaintext */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
     0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
     0x18, 0x19, 0x1a, 0x1b, 0x1c },
   /* ciphertext */
   { 0xdd, 0x25, 0xc7, 0x54, 0xc5, 0xb1, 0x7c, 0x59,
     0x28, 0xb6, 0x9b, 0x73, 0x15, 0x5f, 0x7b, 0xb8,
     0x88, 0x8f, 0xaf, 0x37, 0x09, 0x1a, 0xd9, 0x2c,
     0x8a, 0x24, 0xdb, 0x86, 0x8b },
   /* tag */
   { 0x0d, 0x1a, 0x14, 0xe5, 0x22, 0x24, 0xff, 0xd2,
     0x3a, 0x05, 0xfa, 0x02, 0xcd, 0xef, 0x52, 0xda }
},

/* Vectors from Brian Gladman */

{
   16, 16, 8, 0,
   /* key */
   { 0x23, 0x39, 0x52, 0xde, 0xe4, 0xd5, 0xed, 0x5f,
     0x9b, 0x9c, 0x6d, 0x6f, 0xf8, 0x0f, 0xf4, 0x78 },
   /* nonce */
   { 0x62, 0xec, 0x67, 0xf9, 0xc3, 0xa4, 0xa4, 0x07,
     0xfc, 0xb2, 0xa8, 0xc4, 0x90, 0x31, 0xa8, 0xb3 },
   /* header */
   { 0x6b, 0xfb, 0x91, 0x4f, 0xd0, 0x7e, 0xae, 0x6b },
   /* PT */
   { 0x00 },
   /* CT */
   { 0x00 },
   /* tag */
   { 0xe0, 0x37, 0x83, 0x0e, 0x83, 0x89, 0xf2, 0x7b,
     0x02, 0x5a, 0x2d, 0x65, 0x27, 0xe7, 0x9d, 0x01 }
},

{
   16, 16, 8, 2,
   /* key */
   { 0x91, 0x94, 0x5d, 0x3f, 0x4d, 0xcb, 0xee, 0x0b,
     0xf4, 0x5e, 0xf5, 0x22, 0x55, 0xf0, 0x95, 0xa4 },
   /* nonce */
   { 0xbe, 0xca, 0xf0, 0x43, 0xb0, 0xa2, 0x3d, 0x84,
     0x31, 0x94, 0xba, 0x97, 0x2c, 0x66, 0xde, 0xbd },
```

```
   /* header */
   { 0xfa, 0x3b, 0xfd, 0x48, 0x06, 0xeb, 0x53, 0xfa },
   /* PT */
   { 0xf7, 0xfb },
   /* CT */
   { 0x19, 0xdd },
   /* tag */
   { 0x5c, 0x4c, 0x93, 0x31, 0x04, 0x9d, 0x0b, 0xda,
     0xb0, 0x27, 0x74, 0x08, 0xf6, 0x79, 0x67, 0xe5 }
},

{
   16, 16, 8, 5,
   /* key */
   { 0x01, 0xf7, 0x4a, 0xd6, 0x40, 0x77, 0xf2, 0xe7,
     0x04, 0xc0, 0xf6, 0x0a, 0xda, 0x3d, 0xd5, 0x23 },
   /* nonce */
   { 0x70, 0xc3, 0xdb, 0x4f, 0x0d, 0x26, 0x36, 0x84,
     0x00, 0xa1, 0x0e, 0xd0, 0x5d, 0x2b, 0xff, 0x5e },
   /* header */
   { 0x23, 0x4a, 0x34, 0x63, 0xc1, 0x26, 0x4a, 0xc6 },
   /* PT */
   { 0x1a, 0x47, 0xcb, 0x49, 0x33 },
   /* CT */
   { 0xd8, 0x51, 0xd5, 0xba, 0xe0 },
   /* Tag */
   { 0x3a, 0x59, 0xf2, 0x38, 0xa2, 0x3e, 0x39, 0x19,
     0x9d, 0xc9, 0x26, 0x66, 0x26, 0xc4, 0x0f, 0x80 }
}

};
   int err, x, idx, res;
   unsigned long len;
   unsigned char outct[MAXBLOCKSIZE], outtag[MAXBLOCKSIZE];

    /* AES can be under rijndael or aes... try to find it */
    if ((idx = find_cipher("aes")) == -1) {
       if ((idx = find_cipher("rijndael")) == -1) {
          return CRYPT_NOP;
       }
    }

    for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
        len = sizeof(outtag);
        if ((err = eax_encrypt_authenticate_memory(idx, tests[x].key, tests[x].keylen,
            tests[x].nonce, tests[x].noncelen, tests[x].header, tests[x].headerlen,
            tests[x].plaintext, tests[x].msglen, outct, outtag, &len)) != CRYPT_OK) {
           return err;
        }
        if (XMEMCMP(outct, tests[x].ciphertext, tests[x].msglen) || XMEMCMP(outtag, tests[x].tag, len)
#if 0
           unsigned long y;
           printf("\n\nFailure: \nCT:\n");
           for (y = 0; y < (unsigned long)tests[x].msglen; ) {
               printf("0x%02x", outct[y]);
               if (y < (unsigned long)(tests[x].msglen-1)) printf(", ");
               if (!(++y % 8)) printf("\n");
           }
           printf("\nTAG:\n");
           for (y = 0; y < len; ) {
               printf("0x%02x", outtag[y]);
               if (y < len-1) printf(", ");
               if (!(++y % 8)) printf("\n");
           }
#endif
           return CRYPT_FAIL_TESTVECTOR;
        }
```

```

        /* test decrypt */
        if ((err = eax_decrypt_verify_memory(idx, tests[x].key, tests[x].keylen,
             tests[x].nonce, tests[x].noncelen, tests[x].header, tests[x].headerlen,
             outct, tests[x].msglen, outct, outtag, len, &res)) != CRYPT_OK) {
            return err;
        }
        if ((res != 1) || XMEMCMP(outct, tests[x].plaintext, tests[x].msglen)) {
#if 0
           unsigned long y;
           printf("\n\nFailure (res == %d): \nPT:\n", res);
           for (y = 0; y < (unsigned long)tests[x].msglen; ) {
               printf("0x%02x", outct[y]);
               if (y < (unsigned long)(tests[x].msglen-1)) printf(", ");
               if (!(++y % 8)) printf("\n");
           }
           printf("\n\n");
#endif
           return CRYPT_FAIL_TESTVECTOR;
        }

     }
     return CRYPT_OK;
#endif /* LTC_TEST */
}
```

Here is the call graph for this function:

5.31 encauth/gcm/gcm_add_aad.c File Reference

5.31.1 Detailed Description

GCM implementation, Add AAD data to the stream, by Tom St Denis.

Definition in file gcm_add_aad.c.

`#include "tomcrypt.h"`

Include dependency graph for gcm_add_aad.c:

Functions

- int gcm_add_aad (gcm_state *gcm, const unsigned char *adata, unsigned long adatalen)

    Add AAD to the GCM state.

5.31.2 Function Documentation

5.31.2.1 int gcm_add_aad (gcm_state * *gcm*, const unsigned char * *adata*, unsigned long *adatalen*)

Add AAD to the GCM state.

Parameters:

gcm The GCM state

adata The additional authentication data to add to the GCM state

adatalen The length of the AAD data.

Returns:

CRYPT_OK on success

Definition at line 27 of file gcm_add_aad.c.

References cipher_is_valid(), CONST64, CRYPT_INVALID_ARG, CRYPT_OK, gcm_mult_h(), LTC_-ARGCHK, XMEMCPY, and zeromem().

Referenced by gcm_memory().

```
29 {
30    unsigned long x;
31    int           err;
32 #ifdef LTC_FAST
33    unsigned long y;
34 #endif
35
36    LTC_ARGCHK(gcm    != NULL);
37    if (adatalen > 0) {
38       LTC_ARGCHK(adata  != NULL);
39    }
40
41    if (gcm->buflen > 16 || gcm->buflen < 0) {
42       return CRYPT_INVALID_ARG;
43    }
44
45    if ((err = cipher_is_valid(gcm->cipher)) != CRYPT_OK) {
```

```
      return err;
   }

   /* in IV mode? */
   if (gcm->mode == GCM_MODE_IV) {
      /* let's process the IV */
      if (gcm->ivmode || gcm->buflen != 12) {
         for (x = 0; x < (unsigned long)gcm->buflen; x++) {
             gcm->X[x] ^= gcm->buf[x];
         }
         if (gcm->buflen) {
            gcm->totlen += gcm->buflen * CONST64(8);
            gcm_mult_h(gcm, gcm->X);
         }

         /* mix in the length */
         zeromem(gcm->buf, 8);
         STORE64H(gcm->totlen, gcm->buf+8);
         for (x = 0; x < 16; x++) {
             gcm->X[x] ^= gcm->buf[x];
         }
         gcm_mult_h(gcm, gcm->X);

         /* copy counter out */
         XMEMCPY(gcm->Y, gcm->X, 16);
         zeromem(gcm->X, 16);
      } else {
         XMEMCPY(gcm->Y, gcm->buf, 12);
         gcm->Y[12] = 0;
         gcm->Y[13] = 0;
         gcm->Y[14] = 0;
         gcm->Y[15] = 1;
      }
      XMEMCPY(gcm->Y_0, gcm->Y, 16);
      zeromem(gcm->buf, 16);
      gcm->buflen = 0;
      gcm->totlen = 0;
      gcm->mode   = GCM_MODE_AAD;
   }

   if (gcm->mode != GCM_MODE_AAD || gcm->buflen >= 16) {
      return CRYPT_INVALID_ARG;
   }

   x = 0;
#ifdef LTC_FAST
   if (gcm->buflen == 0) {
      for (x = 0; x < (adatalen & ~15); x += 16) {
          for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
              *((LTC_FAST_TYPE*)(&gcm->X[y])) ^= *((LTC_FAST_TYPE*)(&adata[x + y]));
          }
          gcm_mult_h(gcm, gcm->X);
          gcm->totlen += 128;
      }
      adata += x;
   }
#endif


   /* start adding AAD data to the state */
   for (; x < adatalen; x++) {
       gcm->X[gcm->buflen++] ^= *adata++;

       if (gcm->buflen == 16) {
         /* GF mult it */
         gcm_mult_h(gcm, gcm->X);
         gcm->buflen = 0;
```

```
            gcm->totlen += 128;
        }
    }

    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.32 encauth/gcm/gcm_add_iv.c File Reference

5.32.1 Detailed Description

GCM implementation, add IV data to the state, by Tom St Denis.

Definition in file gcm_add_iv.c.

`#include "tomcrypt.h"`

Include dependency graph for gcm_add_iv.c:

Functions

- int gcm_add_iv (gcm_state ∗gcm, const unsigned char ∗IV, unsigned long IVlen)

  Add IV data to the GCM state.

5.32.2 Function Documentation

5.32.2.1 int gcm_add_iv (gcm_state ∗ *gcm*, const unsigned char ∗ *IV*, unsigned long *IVlen*)

Add IV data to the GCM state.

Parameters:

gcm The GCM state

IV The initial value data to add

IVlen The length of the IV

Returns:

CRYPT_OK on success

Definition at line 27 of file gcm_add_iv.c.

References cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, gcm_mult_h(), and LTC_-ARGCHK.

Referenced by gcm_memory().

```
{
   unsigned long x, y;
   int           err;

   LTC_ARGCHK(gcm != NULL);
   if (IVlen > 0) {
      LTC_ARGCHK(IV  != NULL);
   }

   /* must be in IV mode */
   if (gcm->mode != GCM_MODE_IV) {
      return CRYPT_INVALID_ARG;
   }

   if (gcm->buflen >= 16 || gcm->buflen < 0) {
      return CRYPT_INVALID_ARG;
   }
```

```

   if ((err = cipher_is_valid(gcm->cipher)) != CRYPT_OK) {
      return err;
   }


   /* trip the ivmode flag */
   if (IVlen + gcm->buflen > 12) {
      gcm->ivmode |= 1;
   }

   x = 0;
#ifdef LTC_FAST
   if (gcm->buflen == 0) {
      for (x = 0; x < (IVlen & ~15); x += 16) {
          for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
              *((LTC_FAST_TYPE*)(&gcm->X[y])) ^= *((LTC_FAST_TYPE*)(&IV[x + y]));
          }
          gcm_mult_h(gcm, gcm->X);
          gcm->totlen += 128;
      }
      IV += x;
   }
#endif

   /* start adding IV data to the state */
   for (; x < IVlen; x++) {
       gcm->buf[gcm->buflen++] = *IV++;

       if (gcm->buflen == 16) {
         /* GF mult it */
         for (y = 0; y < 16; y++) {
             gcm->X[y] ^= gcm->buf[y];
         }
         gcm_mult_h(gcm, gcm->X);
         gcm->buflen = 0;
         gcm->totlen += 128;
      }
   }

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.33 encauth/gcm/gcm_done.c File Reference

5.33.1 Detailed Description

GCM implementation, Terminate the stream, by Tom St Denis.

Definition in file gcm_done.c.

`#include "tomcrypt.h"`

Include dependency graph for gcm_done.c:

Functions

- int gcm_done (gcm_state ∗gcm, unsigned char ∗tag, unsigned long ∗taglen)

  Terminate a GCM stream.

5.33.2 Function Documentation

5.33.2.1 int gcm_done (gcm_state ∗ *gcm*, unsigned char ∗ *tag*, unsigned long ∗ *taglen*)

Terminate a GCM stream.

Parameters:

gcm The GCM state

tag [out] The destination for the MAC tag

taglen [in/out] The length of the MAC tag

Returns:

CRYPT_OK on success

Definition at line 27 of file gcm_done.c.

References cipher_is_valid(), CONST64, CRYPT_INVALID_ARG, CRYPT_OK, gcm_mult_h(), and LTC_ARGCHK.

Referenced by gcm_memory().

```
{
   unsigned long x;
   int err;

   LTC_ARGCHK(gcm     != NULL);
   LTC_ARGCHK(tag     != NULL);
   LTC_ARGCHK(taglen  != NULL);

   if (gcm->buflen > 16 || gcm->buflen < 0) {
      return CRYPT_INVALID_ARG;
   }

   if ((err = cipher_is_valid(gcm->cipher)) != CRYPT_OK) {
      return err;
   }
```

```
   if (gcm->mode != GCM_MODE_TEXT) {
      return CRYPT_INVALID_ARG;
   }

   /* handle remaining ciphertext */
   if (gcm->buflen) {
      gcm->pttotlen += gcm->buflen * CONST64(8);
      gcm_mult_h(gcm, gcm->X);
   }

   /* length */
   STORE64H(gcm->totlen, gcm->buf);
   STORE64H(gcm->pttotlen, gcm->buf+8);
   for (x = 0; x < 16; x++) {
       gcm->X[x] ^= gcm->buf[x];
   }
   gcm_mult_h(gcm, gcm->X);

   /* encrypt original counter */
   if ((err = cipher_descriptor[gcm->cipher].ecb_encrypt(gcm->Y_0, gcm->buf, &gcm->K)) != CRYPT_OK) {
      return err;
   }
   for (x = 0; x < 16 && x < *taglen; x++) {
       tag[x] = gcm->buf[x] ^ gcm->X[x];
   }
   *taglen = x;

   cipher_descriptor[gcm->cipher].done(&gcm->K);

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.34 encauth/gcm/gcm_gf_mult.c File Reference

5.34.1 Detailed Description

GCM implementation, do the GF mult, by Tom St Denis.

Definition in file gcm_gf_mult.c.

`#include "tomcrypt.h"`

Include dependency graph for gcm_gf_mult.c:

Functions

- static void gcm_rightshift (unsigned char ∗a)
- void gcm_gf_mult (const unsigned char ∗a, const unsigned char ∗b, unsigned char ∗c)

    GCM GF multiplier (internal use only) bitserial.

Variables

- const unsigned char gcm_shift_table [256 ∗2]
- static const unsigned char mask [ ] = { 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01 }
- static const unsigned char poly [ ] = { 0x00, 0xE1 }

5.34.2 Function Documentation

5.34.2.1 void gcm_gf_mult (const unsigned char ∗ *a*, const unsigned char ∗ *b*, unsigned char ∗ *c*)

GCM GF multiplier (internal use only) bitserial.

Parameters:

a First value

b Second value

c Destination for a ∗ b

Definition at line 83 of file gcm_gf_mult.c.

References mask, XMEMCPY, and zeromem().

Referenced by gcm_init(), and lrw_start().

```
{
   unsigned char Z[16], V[16];
   unsigned x, y, z;

   zeromem(Z, 16);
   XMEMCPY(V, a, 16);
   for (x = 0; x < 128; x++) {
       if (b[x>>3] & mask[x&7]) {
          for (y = 0; y < 16; y++) {
              Z[y] ^= V[y];
          }
       }
```

```
        z      = V[15] & 0x01;
        gcm_rightshift(V);
        V[0] ^= poly[z];
    }
    XMEMCPY(c, Z, 16);
}
```

Here is the call graph for this function:

5.34.2.2 static void gcm_rightshift (unsigned char * *a*) [static]

Definition at line 63 of file gcm_gf_mult.c.

```
{
    int x;
    for (x = 15; x > 0; x--) {
        a[x] = (a[x]>>1) | ((a[x-1]<<7)&0x80);
    }
    a[0] >>= 1;
}
```

5.34.3 Variable Documentation

5.34.3.1 const unsigned char gcm_shift_table[256 *2]

Definition at line 22 of file gcm_gf_mult.c.

5.34.3.2 const unsigned char mask[ ] = { 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01 } [static]

Definition at line 73 of file gcm_gf_mult.c.

Referenced by der_encode_object_identifier(), gcm_gf_mult(), omac_init(), pkcs_1_oaep_decode(), pkcs_1_oaep_encode(), pkcs_1_pss_decode(), and pkcs_1_pss_encode().

5.34.3.3 const unsigned char poly[ ] = { 0x00, 0xE1 } [static]

Definition at line 74 of file gcm_gf_mult.c.

Referenced by ocb_init(), and pmac_init().

5.35 encauth/gcm/gcm_init.c File Reference

5.35.1 Detailed Description

GCM implementation, initialize state, by Tom St Denis.

Definition in file gcm_init.c.

`#include "tomcrypt.h"`

Include dependency graph for gcm_init.c:

Functions

- int gcm_init (gcm_state *gcm, int cipher, const unsigned char *key, int keylen)

  Initialize a GCM state.

5.35.2 Function Documentation

5.35.2.1 int gcm_init (gcm_state * *gcm*, int *cipher*, const unsigned char * *key*, int *keylen*)

Initialize a GCM state.

Parameters:

gcm The GCM state to initialize

cipher The index of the cipher to use

key The secret key

keylen The length of the secret key

Returns:

CRYPT_OK on success

Definition at line 28 of file gcm_init.c.

References B, cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_INVALID_-CIPHER, CRYPT_OK, ecb_encrypt(), gcm_gf_mult(), LTC_ARGCHK, and zeromem().

Referenced by gcm_memory().

```
30 {
31    int           err;
32    unsigned char B[16];
33 #ifdef GCM_TABLES
34    int           x, y, z, t;
35 #endif
36 
37    LTC_ARGCHK(gcm != NULL);
38    LTC_ARGCHK(key != NULL);
39 
40 #ifdef LTC_FAST
41    if (16 % sizeof(LTC_FAST_TYPE)) {
42       return CRYPT_INVALID_ARG;
43    }
44 #endif
45 
```

```
   /* is cipher valid? */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }
   if (cipher_descriptor[cipher].block_length != 16) {
      return CRYPT_INVALID_CIPHER;
   }

   /* schedule key */
   if ((err = cipher_descriptor[cipher].setup(key, keylen, 0, &gcm->K)) != CRYPT_OK) {
      return err;
   }

   /* H = E(0) */
   zeromem(B, 16);
   if ((err = cipher_descriptor[cipher].ecb_encrypt(B, gcm->H, &gcm->K)) != CRYPT_OK) {
      return err;
   }

   /* setup state */
   zeromem(gcm->buf, sizeof(gcm->buf));
   zeromem(gcm->X,   sizeof(gcm->X));
   gcm->cipher   = cipher;
   gcm->mode     = GCM_MODE_IV;
   gcm->ivmode   = 0;
   gcm->buflen   = 0;
   gcm->totlen   = 0;
   gcm->pttotlen = 0;

#ifdef GCM_TABLES
   /* setup tables */

   /* generate the first table as it has no shifting (from which we make the other tables) */
   zeromem(B, 16);
   for (y = 0; y < 256; y++) {
        B[0] = y;
        gcm_gf_mult(gcm->H, B, &gcm->PC[0][y][0]);
   }

   /* now generate the rest of the tables based the previous table */
   for (x = 1; x < 16; x++) {
      for (y = 0; y < 256; y++) {
         /* now shift it right by 8 bits */
         t = gcm->PC[x-1][y][15];
         for (z = 15; z > 0; z--) {
             gcm->PC[x][y][z] = gcm->PC[x-1][y][z-1];
         }
         gcm->PC[x][y][0] = gcm_shift_table[t<<1];
         gcm->PC[x][y][1] ^= gcm_shift_table[(t<<1)+1];
     }
  }

#endif

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.36 encauth/gcm/gcm_memory.c File Reference

5.36.1 Detailed Description

GCM implementation, process a packet, by Tom St Denis.

Definition in file gcm_memory.c.

```
#include "tomcrypt.h"
```

Include dependency graph for gcm_memory.c:

Functions

- int gcm_memory (int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗IV, unsigned long IVlen, const unsigned char ∗adata, unsigned long adatalen, unsigned char ∗pt, unsigned long ptlen, unsigned char ∗ct, unsigned char ∗tag, unsigned long ∗taglen, int direction)

  Process an entire GCM packet in one call.

5.36.2 Function Documentation

5.36.2.1 int gcm_memory (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *IV*, unsigned long *IVlen*, const unsigned char ∗ *adata*, unsigned long *adatalen*, unsigned char ∗ *pt*, unsigned long *ptlen*, unsigned char ∗ *ct*, unsigned char ∗ *tag*, unsigned long ∗ *taglen*, int *direction*)

Process an entire GCM packet in one call.

Parameters:

cipher Index of cipher to use

key The secret key

keylen The length of the secret key

IV The initial vector

IVlen The length of the initial vector

adata The additional authentication data (header)

adatalen The length of the adata

pt The plaintext

ptlen The length of the plaintext (ciphertext length is the same)

ct The ciphertext

tag [out] The MAC tag

taglen [in/out] The MAC tag length

direction Encrypt or Decrypt mode (GCM_ENCRYPT or GCM_DECRYPT)

Returns:

CRYPT_OK on success

Definition at line 37 of file gcm_memory.c.

References ltc_cipher_descriptor::accel_gcm_memory, cipher_descriptor, cipher_is_valid(), CRYPT_-MEM, CRYPT_OK, gcm_add_aad(), gcm_add_iv(), gcm_done(), gcm_init(), gcm_process(), XFREE, and XMALLOC.

Referenced by gcm_test().

```
{
    void      *orig;
    gcm_state *gcm;
    int        err;

    if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
       return err;
    }

    if (cipher_descriptor[cipher].accel_gcm_memory != NULL) {
       return
         cipher_descriptor[cipher].accel_gcm_memory
                                          (key,   keylen,
                                           IV,    IVlen,
                                           adata, adatalen,
                                           pt,    ptlen,
                                           ct,
                                           tag,   taglen,
                                           direction);
    }



#ifndef GCM_TABLES_SSE2
    orig = gcm = XMALLOC(sizeof(*gcm));
#else
    orig = gcm = XMALLOC(sizeof(*gcm) + 16);
#endif
    if (gcm == NULL) {
        return CRYPT_MEM;
    }

   /* Force GCM to be on a multiple of 16 so we can use 128-bit aligned operations
    * note that we only modify gcm and keep orig intact.  This code is not portable
    * but again it's only for SSE2 anyways, so who cares?
    */
#ifdef GCM_TABLES_SSE2
   if ((unsigned long)gcm & 15) {
      gcm = (gcm_state *)((unsigned long)gcm + (16 - ((unsigned long)gcm & 15)));
   }
#endif

    if ((err = gcm_init(gcm, cipher, key, keylen)) != CRYPT_OK) {
       goto LTC_ERR;
    }
    if ((err = gcm_add_iv(gcm, IV, IVlen)) != CRYPT_OK) {
       goto LTC_ERR;
    }
    if ((err = gcm_add_aad(gcm, adata, adatalen)) != CRYPT_OK) {
       goto LTC_ERR;
    }
    if ((err = gcm_process(gcm, pt, ptlen, ct, direction)) != CRYPT_OK) {
       goto LTC_ERR;
    }
    err = gcm_done(gcm, tag, taglen);
LTC_ERR:
    XFREE(orig);
    return err;
}
```

Here is the call graph for this function:

5.37 encauth/gcm/gcm_mult_h.c File Reference

5.37.1 Detailed Description

GCM implementation, do the GF mult, by Tom St Denis.

Definition in file gcm_mult_h.c.

`#include "tomcrypt.h"`

Include dependency graph for gcm_mult_h.c:

Functions

- void gcm_mult_h (gcm_state ∗gcm, unsigned char ∗I)

  GCM multiply by H.

5.37.2 Function Documentation

5.37.2.1 void gcm_mult_h (gcm_state ∗ *gcm*, unsigned char ∗ *I*)

GCM multiply by H.

Parameters:

gcm The GCM state which holds the H value

I The value to multiply H by

Definition at line 24 of file gcm_mult_h.c.

Referenced by gcm_add_aad(), gcm_add_iv(), gcm_done(), and gcm_process().

```
25 {
26    unsigned char T[16];
27 #ifdef GCM_TABLES
28    int x, y;
29 #ifdef GCM_TABLES_SSE2
30    asm("movdqa (%0),%%xmm0"::"r"(&gcm->PC[0][I[0]][0]));
31    for (x = 1; x < 16; x++) {
32       asm("pxor (%0),%%xmm0"::"r"(&gcm->PC[x][I[x]][0]));
33    }
34    asm("movdqa %%xmm0,(%0)"::"r"(&T));
35 #else
36    XMEMCPY(T, &gcm->PC[0][I[0]][0], 16);
37    for (x = 1; x < 16; x++) {
38 #ifdef LTC_FAST
39        for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
40            *((LTC_FAST_TYPE *)(T + y)) ^= *((LTC_FAST_TYPE *)(&gcm->PC[x][I[x]][y]));
41        }
42 #else
43        for (y = 0; y < 16; y++) {
44            T[y] ^= gcm->PC[x][I[x]][y];
45        }
46 #endif /* LTC_FAST */
47    }
48 #endif /* GCM_TABLES_SSE2 */
49 #else
50    gcm_gf_mult(gcm->H, I, T);
```

```
#endif
   XMEMCPY(I, T, 16);
}
```

5.38 encauth/gcm/gcm_process.c File Reference

5.38.1 Detailed Description

GCM implementation, process message data, by Tom St Denis.

Definition in file gcm_process.c.

`#include "tomcrypt.h"`

Include dependency graph for gcm_process.c:

Functions

- int gcm_process (gcm_state *gcm, unsigned char *pt, unsigned long ptlen, unsigned char *ct, int direction)

  Process plaintext/ciphertext through GCM.

5.38.2 Function Documentation

5.38.2.1 int gcm_process (gcm_state * *gcm*, unsigned char * *pt*, unsigned long *ptlen*, unsigned char * *ct*, int *direction*)

Process plaintext/ciphertext through GCM.

Parameters:

gcm The GCM state

pt The plaintext

ptlen The plaintext length (ciphertext length is the same)

ct The ciphertext

direction Encrypt or Decrypt mode (GCM_ENCRYPT or GCM_DECRYPT)

Returns:

CRYPT_OK on success

Definition at line 29 of file gcm_process.c.

References cipher_is_valid(), CONST64, CRYPT_INVALID_ARG, CRYPT_OK, gcm_mult_h(), and LTC_ARGCHK.

Referenced by gcm_memory().

```
{
   unsigned long x;
   int           y, err;
   unsigned char b;

   LTC_ARGCHK(gcm != NULL);
   if (ptlen > 0) {
      LTC_ARGCHK(pt  != NULL);
      LTC_ARGCHK(ct  != NULL);
   }

```

```
   if (gcm->buflen > 16 || gcm->buflen < 0) {
      return CRYPT_INVALID_ARG;
   }

   if ((err = cipher_is_valid(gcm->cipher)) != CRYPT_OK) {
      return err;
   }

   /* in AAD mode? */
   if (gcm->mode == GCM_MODE_AAD) {
      /* let's process the AAD */
      if (gcm->buflen) {
         gcm->totlen += gcm->buflen * CONST64(8);
         gcm_mult_h(gcm, gcm->X);
      }

      /* increment counter */
      for (y = 15; y >= 0; y--) {
          if (++gcm->Y[y] & 255) { break; }
      }
      /* encrypt the counter */
      if ((err = cipher_descriptor[gcm->cipher].ecb_encrypt(gcm->Y, gcm->buf, &gcm->K)) != CRYPT_OK) {
         return err;
      }

      gcm->buflen = 0;
      gcm->mode   = GCM_MODE_TEXT;
   }

   if (gcm->mode != GCM_MODE_TEXT) {
      return CRYPT_INVALID_ARG;
   }

   x = 0;
#ifdef LTC_FAST
   if (gcm->buflen == 0) {
      if (direction == GCM_ENCRYPT) {
         for (x = 0; x < (ptlen & ~15); x += 16) {
             /* ctr encrypt */
             for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
                 *((LTC_FAST_TYPE*)(&ct[x + y])) = *((LTC_FAST_TYPE*)(&pt[x+y])) ^ *((LTC_FAST_TYPE*)(&
                 *((LTC_FAST_TYPE*)(&gcm->X[y])) ^= *((LTC_FAST_TYPE*)(&ct[x+y]));
             }
             /* GMAC it */
             gcm->pttotlen += 128;
             gcm_mult_h(gcm, gcm->X);
             /* increment counter */
             for (y = 15; y >= 0; y--) {
                 if (++gcm->Y[y] & 255) { break; }
             }
             if ((err = cipher_descriptor[gcm->cipher].ecb_encrypt(gcm->Y, gcm->buf, &gcm->K)) != CRYPT
                return err;
             }
         }
      } else {
         for (x = 0; x < (ptlen & ~15); x += 16) {
              /* ctr encrypt */
              for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
                  *((LTC_FAST_TYPE*)(&gcm->X[y])) ^= *((LTC_FAST_TYPE*)(&ct[x+y]));
                  *((LTC_FAST_TYPE*)(&pt[x + y])) = *((LTC_FAST_TYPE*)(&ct[x+y])) ^ *((LTC_FAST_TYPE*)(
              }
              /* GMAC it */
              gcm->pttotlen += 128;
              gcm_mult_h(gcm, gcm->X);
              /* increment counter */
              for (y = 15; y >= 0; y--) {
                  if (++gcm->Y[y] & 255) { break; }
```

```
             }
             if ((err = cipher_descriptor[gcm->cipher].ecb_encrypt(gcm->Y, gcm->buf, &gcm->K)) != CRYP
                return err;
             }
         }
     }
   }
#endif

   /* process text */
   for (; x < ptlen; x++) {
       if (gcm->buflen == 16) {
          gcm->pttotlen += 128;
          gcm_mult_h(gcm, gcm->X);

          /* increment counter */
          for (y = 15; y >= 0; y--) {
              if (++gcm->Y[y] & 255) { break; }
          }
          if ((err = cipher_descriptor[gcm->cipher].ecb_encrypt(gcm->Y, gcm->buf, &gcm->K)) != CRYPT_O
             return err;
          }
          gcm->buflen = 0;
       }

       if (direction == GCM_ENCRYPT) {
          b = ct[x] = pt[x] ^ gcm->buf[gcm->buflen];
       } else {
          b = ct[x];
          pt[x] = ct[x] ^ gcm->buf[gcm->buflen];
       }
       gcm->X[gcm->buflen++] ^= b;
   }

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.39 encauth/gcm/gcm_reset.c File Reference

5.39.1 Detailed Description

GCM implementation, reset a used state so it can accept IV data, by Tom St Denis.

Definition in file gcm_reset.c.

`#include "tomcrypt.h"`

Include dependency graph for gcm_reset.c:

Functions

- int gcm_reset (gcm_state ∗gcm)

  Reset a GCM state to as if you just called gcm_init().

5.39.2 Function Documentation

5.39.2.1 int gcm_reset (gcm_state ∗ *gcm*)

Reset a GCM state to as if you just called gcm_init().

This saves the initialization time.

Parameters:

gcm The GCM state to reset

Returns:

CRYPT_OK on success

Definition at line 25 of file gcm_reset.c.

References CRYPT_OK, LTC_ARGCHK, and zeromem().

```
26 {
27    LTC_ARGCHK(gcm != NULL);
28
29    zeromem(gcm->buf, sizeof(gcm->buf));
30    zeromem(gcm->X,   sizeof(gcm->X));
31    gcm->mode      = GCM_MODE_IV;
32    gcm->ivmode    = 0;
33    gcm->buflen    = 0;
34    gcm->totlen    = 0;
35    gcm->pttotlen = 0;
36
37    return CRYPT_OK;
38 }
```

Here is the call graph for this function:

5.40 encauth/gcm/gcm_test.c File Reference

5.40.1 Detailed Description

GCM implementation, testing, by Tom St Denis.

Definition in file gcm_test.c.

```
#include "tomcrypt.h"
```

Include dependency graph for gcm_test.c:

Functions

- int gcm_test (void)

  Test the GCM code.

5.40.2 Function Documentation

5.40.2.1 int gcm_test (void)

Test the GCM code.

Returns:

CRYPT_OK on success

Definition at line 24 of file gcm_test.c.

References CRYPT_NOP, CRYPT_OK, find_cipher(), gcm_memory(), and K.

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   static const struct {
       unsigned char K[32];
       int           keylen;
       unsigned char P[64];
       unsigned long ptlen;
         unsigned char A[64];
       unsigned long alen;
       unsigned char IV[64];
       unsigned long IVlen;
       unsigned char C[64];
       unsigned char T[16];
   } tests[] = {

/* test case #1 */
{
  /* key */
  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
  16,

  /* plaintext */
  { 0 },
  0,

```

```
  /* AAD data */
  { 0 },
  0,

  /* IV */
  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00 },
  12,

  /* ciphertext  */
  { 0 },

  /* tag */
  { 0x58, 0xe2, 0xfc, 0xce, 0xfa, 0x7e, 0x30, 0x61,
    0x36, 0x7f, 0x1d, 0x57, 0xa4, 0xe7, 0x45, 0x5a }
},

/* test case #2 */
{
  /* key */
  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
  16,

  /* PT */
  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
  16,

  /* ADATA */
  { 0 },
  0,

  /* IV */
  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00 },
  12,

  /* CT */
  { 0x03, 0x88, 0xda, 0xce, 0x60, 0xb6, 0xa3, 0x92,
    0xf3, 0x28, 0xc2, 0xb9, 0x71, 0xb2, 0xfe, 0x78 },

  /* TAG */
  { 0xab, 0x6e, 0x47, 0xd4, 0x2c, 0xec, 0x13, 0xbd,
    0xf5, 0x3a, 0x67, 0xb2, 0x12, 0x57, 0xbd, 0xdf }
},

/* test case #3 */
{
   /* key */
   { 0xfe, 0xff, 0xe9, 0x92, 0x86, 0x65, 0x73, 0x1c,
     0x6d, 0x6a, 0x8f, 0x94, 0x67, 0x30, 0x83, 0x08, },
   16,

   /* PT */
   { 0xd9, 0x31, 0x32, 0x25, 0xf8, 0x84, 0x06, 0xe5,
     0xa5, 0x59, 0x09, 0xc5, 0xaf, 0xf5, 0x26, 0x9a,
     0x86, 0xa7, 0xa9, 0x53, 0x15, 0x34, 0xf7, 0xda,
     0x2e, 0x4c, 0x30, 0x3d, 0x8a, 0x31, 0x8a, 0x72,
     0x1c, 0x3c, 0x0c, 0x95, 0x95, 0x68, 0x09, 0x53,
     0x2f, 0xcf, 0x0e, 0x24, 0x49, 0xa6, 0xb5, 0x25,
     0xb1, 0x6a, 0xed, 0xf5, 0xaa, 0x0d, 0xe6, 0x57,
     0xba, 0x63, 0x7b, 0x39, 0x1a, 0xaf, 0xd2, 0x55, },
  64,

  /* ADATA */
  { 0 },
```

```
  0,

  /* IV */
  { 0xca, 0xfe, 0xba, 0xbe, 0xfa, 0xce, 0xdb, 0xad,
    0xde, 0xca, 0xf8, 0x88,  },
  12,

  /* CT */
  { 0x42, 0x83, 0x1e, 0xc2, 0x21, 0x77, 0x74, 0x24,
    0x4b, 0x72, 0x21, 0xb7, 0x84, 0xd0, 0xd4, 0x9c,
    0xe3, 0xaa, 0x21, 0x2f, 0x2c, 0x02, 0xa4, 0xe0,
    0x35, 0xc1, 0x7e, 0x23, 0x29, 0xac, 0xa1, 0x2e,
    0x21, 0xd5, 0x14, 0xb2, 0x54, 0x66, 0x93, 0x1c,
    0x7d, 0x8f, 0x6a, 0x5a, 0xac, 0x84, 0xaa, 0x05,
    0x1b, 0xa3, 0x0b, 0x39, 0x6a, 0x0a, 0xac, 0x97,
    0x3d, 0x58, 0xe0, 0x91, 0x47, 0x3f, 0x59, 0x85, },

  /* TAG */
  { 0x4d, 0x5c, 0x2a, 0xf3, 0x27, 0xcd, 0x64, 0xa6,
    0x2c, 0xf3, 0x5a, 0xbd, 0x2b, 0xa6, 0xfa, 0xb4, }
},

/* test case #4 */
{
   /* key */
   { 0xfe, 0xff, 0xe9, 0x92, 0x86, 0x65, 0x73, 0x1c,
     0x6d, 0x6a, 0x8f, 0x94, 0x67, 0x30, 0x83, 0x08, },
   16,

   /* PT */
   { 0xd9, 0x31, 0x32, 0x25, 0xf8, 0x84, 0x06, 0xe5,
     0xa5, 0x59, 0x09, 0xc5, 0xaf, 0xf5, 0x26, 0x9a,
     0x86, 0xa7, 0xa9, 0x53, 0x15, 0x34, 0xf7, 0xda,
     0x2e, 0x4c, 0x30, 0x3d, 0x8a, 0x31, 0x8a, 0x72,
     0x1c, 0x3c, 0x0c, 0x95, 0x95, 0x68, 0x09, 0x53,
     0x2f, 0xcf, 0x0e, 0x24, 0x49, 0xa6, 0xb5, 0x25,
     0xb1, 0x6a, 0xed, 0xf5, 0xaa, 0x0d, 0xe6, 0x57,
     0xba, 0x63, 0x7b, 0x39,  },
   60,

   /* ADATA */
   { 0xfe, 0xed, 0xfa, 0xce, 0xde, 0xad, 0xbe, 0xef,
     0xfe, 0xed, 0xfa, 0xce, 0xde, 0xad, 0xbe, 0xef,
     0xab, 0xad, 0xda, 0xd2,  },
   20,

   /* IV */
   { 0xca, 0xfe, 0xba, 0xbe, 0xfa, 0xce, 0xdb, 0xad,
     0xde, 0xca, 0xf8, 0x88,  },
   12,

   /* CT */
   { 0x42, 0x83, 0x1e, 0xc2, 0x21, 0x77, 0x74, 0x24,
     0x4b, 0x72, 0x21, 0xb7, 0x84, 0xd0, 0xd4, 0x9c,
     0xe3, 0xaa, 0x21, 0x2f, 0x2c, 0x02, 0xa4, 0xe0,
     0x35, 0xc1, 0x7e, 0x23, 0x29, 0xac, 0xa1, 0x2e,
     0x21, 0xd5, 0x14, 0xb2, 0x54, 0x66, 0x93, 0x1c,
     0x7d, 0x8f, 0x6a, 0x5a, 0xac, 0x84, 0xaa, 0x05,
     0x1b, 0xa3, 0x0b, 0x39, 0x6a, 0x0a, 0xac, 0x97,
     0x3d, 0x58, 0xe0, 0x91,  },

   /* TAG */
   { 0x5b, 0xc9, 0x4f, 0xbc, 0x32, 0x21, 0xa5, 0xdb,
     0x94, 0xfa, 0xe9, 0x5a, 0xe7, 0x12, 0x1a, 0x47, }

},

```

```
/* test case #5 */
{
   /* key */
   { 0xfe, 0xff, 0xe9, 0x92, 0x86, 0x65, 0x73, 0x1c,
     0x6d, 0x6a, 0x8f, 0x94, 0x67, 0x30, 0x83, 0x08, },
   16,

   /* PT */
   { 0xd9, 0x31, 0x32, 0x25, 0xf8, 0x84, 0x06, 0xe5,
     0xa5, 0x59, 0x09, 0xc5, 0xaf, 0xf5, 0x26, 0x9a,
     0x86, 0xa7, 0xa9, 0x53, 0x15, 0x34, 0xf7, 0xda,
     0x2e, 0x4c, 0x30, 0x3d, 0x8a, 0x31, 0x8a, 0x72,
     0x1c, 0x3c, 0x0c, 0x95, 0x95, 0x68, 0x09, 0x53,
     0x2f, 0xcf, 0x0e, 0x24, 0x49, 0xa6, 0xb5, 0x25,
     0xb1, 0x6a, 0xed, 0xf5, 0xaa, 0x0d, 0xe6, 0x57,
     0xba, 0x63, 0x7b, 0x39,  },
   60,

   /* ADATA */
   { 0xfe, 0xed, 0xfa, 0xce, 0xde, 0xad, 0xbe, 0xef,
     0xfe, 0xed, 0xfa, 0xce, 0xde, 0xad, 0xbe, 0xef,
     0xab, 0xad, 0xda, 0xd2,  },
   20,

   /* IV */
   { 0xca, 0xfe, 0xba, 0xbe, 0xfa, 0xce, 0xdb, 0xad, },
   8,

   /* CT */
   { 0x61, 0x35, 0x3b, 0x4c, 0x28, 0x06, 0x93, 0x4a,
     0x77, 0x7f, 0xf5, 0x1f, 0xa2, 0x2a, 0x47, 0x55,
     0x69, 0x9b, 0x2a, 0x71, 0x4f, 0xcd, 0xc6, 0xf8,
     0x37, 0x66, 0xe5, 0xf9, 0x7b, 0x6c, 0x74, 0x23,
     0x73, 0x80, 0x69, 0x00, 0xe4, 0x9f, 0x24, 0xb2,
     0x2b, 0x09, 0x75, 0x44, 0xd4, 0x89, 0x6b, 0x42,
     0x49, 0x89, 0xb5, 0xe1, 0xeb, 0xac, 0x0f, 0x07,
     0xc2, 0x3f, 0x45, 0x98,  },

   /* TAG */
   { 0x36, 0x12, 0xd2, 0xe7, 0x9e, 0x3b, 0x07, 0x85,
     0x56, 0x1b, 0xe1, 0x4a, 0xac, 0xa2, 0xfc, 0xcb, }
},

/* test case #6 */
{
   /* key */
   { 0xfe, 0xff, 0xe9, 0x92, 0x86, 0x65, 0x73, 0x1c,
     0x6d, 0x6a, 0x8f, 0x94, 0x67, 0x30, 0x83, 0x08, },
   16,

   /* PT */
   { 0xd9, 0x31, 0x32, 0x25, 0xf8, 0x84, 0x06, 0xe5,
     0xa5, 0x59, 0x09, 0xc5, 0xaf, 0xf5, 0x26, 0x9a,
     0x86, 0xa7, 0xa9, 0x53, 0x15, 0x34, 0xf7, 0xda,
     0x2e, 0x4c, 0x30, 0x3d, 0x8a, 0x31, 0x8a, 0x72,
     0x1c, 0x3c, 0x0c, 0x95, 0x95, 0x68, 0x09, 0x53,
     0x2f, 0xcf, 0x0e, 0x24, 0x49, 0xa6, 0xb5, 0x25,
     0xb1, 0x6a, 0xed, 0xf5, 0xaa, 0x0d, 0xe6, 0x57,
     0xba, 0x63, 0x7b, 0x39,  },
   60,

   /* ADATA */
   { 0xfe, 0xed, 0xfa, 0xce, 0xde, 0xad, 0xbe, 0xef,
     0xfe, 0xed, 0xfa, 0xce, 0xde, 0xad, 0xbe, 0xef,
     0xab, 0xad, 0xda, 0xd2,  },
   20,

```

```
   /* IV */
   { 0x93, 0x13, 0x22, 0x5d, 0xf8, 0x84, 0x06, 0xe5,
     0x55, 0x90, 0x9c, 0x5a, 0xff, 0x52, 0x69, 0xaa,
     0x6a, 0x7a, 0x95, 0x38, 0x53, 0x4f, 0x7d, 0xa1,
     0xe4, 0xc3, 0x03, 0xd2, 0xa3, 0x18, 0xa7, 0x28,
     0xc3, 0xc0, 0xc9, 0x51, 0x56, 0x80, 0x95, 0x39,
     0xfc, 0xf0, 0xe2, 0x42, 0x9a, 0x6b, 0x52, 0x54,
     0x16, 0xae, 0xdb, 0xf5, 0xa0, 0xde, 0x6a, 0x57,
     0xa6, 0x37, 0xb3, 0x9b,  },
   60,

   /* CT */
   { 0x8c, 0xe2, 0x49, 0x98, 0x62, 0x56, 0x15, 0xb6,
     0x03, 0xa0, 0x33, 0xac, 0xa1, 0x3f, 0xb8, 0x94,
     0xbe, 0x91, 0x12, 0xa5, 0xc3, 0xa2, 0x11, 0xa8,
     0xba, 0x26, 0x2a, 0x3c, 0xca, 0x7e, 0x2c, 0xa7,
     0x01, 0xe4, 0xa9, 0xa4, 0xfb, 0xa4, 0x3c, 0x90,
     0xcc, 0xdc, 0xb2, 0x81, 0xd4, 0x8c, 0x7c, 0x6f,
     0xd6, 0x28, 0x75, 0xd2, 0xac, 0xa4, 0x17, 0x03,
     0x4c, 0x34, 0xae, 0xe5,  },

   /* TAG */
   { 0x61, 0x9c, 0xc5, 0xae, 0xff, 0xfe, 0x0b, 0xfa,
     0x46, 0x2a, 0xf4, 0x3c, 0x16, 0x99, 0xd0, 0x50, }
},

#if 0

/* test case #10 */
{
   { 0xdb, 0xbc, 0x85, 0x66, 0xd6, 0xf5, 0xb1, 0x58,
     0xda, 0x99, 0xa2, 0xff, 0x2e, 0x01, 0xdd, 0xa6,
     0x29, 0xb8, 0x9c, 0x34, 0xad, 0x1e, 0x5f, 0xeb,
     0xa7, 0x0e, 0x7a, 0xae, 0x43, 0x28, 0x28, 0x9c },
   32,

   { 0xce, 0x20, 0x27, 0xb4, 0x7a, 0x84, 0x32, 0x52,
     0x01, 0x34, 0x65, 0x83, 0x4d, 0x75, 0xfd, 0x0f },
   16,

   { 0 },
   0,

   { 0xcf, 0xc0, 0x6e, 0x72, 0x2b, 0xe9, 0x87, 0xb3,
     0x76, 0x7f, 0x70, 0xa7, 0xb8, 0x56, 0xb7, 0x74 },
   16,

   { 0x03, 0x30, 0xea, 0x65, 0xb1, 0xf4, 0x8a, 0xd7,
     0x18, 0xc3, 0xf1, 0xf3, 0xdc, 0xef, 0xe4, 0x20 },

   { 0xe9, 0xef, 0xa9, 0x97, 0xd0, 0xae, 0x82, 0x42,
     0x90, 0xbb, 0x5a, 0x66, 0x95, 0xff, 0x2c, 0x7a }
}

#endif


/* rest of test cases are the same except AES key size changes... ignored... */
};
   int           idx, err;
   unsigned long x, y;
   unsigned char out[2][64], T[2][16];

   /* find aes */
   idx = find_cipher("aes");
   if (idx == -1) {
      idx = find_cipher("rijndael");
```

```
      if (idx == -1) {
         return CRYPT_NOP;
      }
   }

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       y = sizeof(T[0]);
       if ((err = gcm_memory(idx, tests[x].K, tests[x].keylen,
                             tests[x].IV, tests[x].IVlen,
                             tests[x].A, tests[x].alen,
                             (unsigned char*)tests[x].P, tests[x].ptlen,
                             out[0], T[0], &y, GCM_ENCRYPT)) != CRYPT_OK) {
          return err;
       }

       if (XMEMCMP(out[0], tests[x].C, tests[x].ptlen)) {
#if 0
          printf("\nCiphertext wrong %lu\n", x);
          for (y = 0; y < tests[x].ptlen; y++) {
              printf("%02x", out[0][y] & 255);
          }
          printf("\n");
#endif
          return CRYPT_FAIL_TESTVECTOR;
       }

       if (XMEMCMP(T[0], tests[x].T, 16)) {
#if 0
          printf("\nTag on plaintext wrong %lu\n", x);
          for (y = 0; y < 16; y++) {
              printf("%02x", T[0][y] & 255);
          }
          printf("\n");
#endif
          return CRYPT_FAIL_TESTVECTOR;
       }

       y = sizeof(T[1]);
       if ((err = gcm_memory(idx, tests[x].K, tests[x].keylen,
                             tests[x].IV, tests[x].IVlen,
                             tests[x].A, tests[x].alen,
                             out[1], tests[x].ptlen,
                             out[0], T[1], &y, GCM_DECRYPT)) != CRYPT_OK) {
          return err;
       }

       if (XMEMCMP(out[1], tests[x].P, tests[x].ptlen)) {
#if 0
          printf("\nplaintext wrong %lu\n", x);
          for (y = 0; y < tests[x].ptlen; y++) {
              printf("%02x", out[0][y] & 255);
          }
          printf("\n");
#endif
          return CRYPT_FAIL_TESTVECTOR;
       }

       if (XMEMCMP(T[1], tests[x].T, 16)) {
#if 0
          printf("\nTag on ciphertext wrong %lu\n", x);
          for (y = 0; y < 16; y++) {
              printf("%02x", T[1][y] & 255);
          }
          printf("\n");
#endif
          return CRYPT_FAIL_TESTVECTOR;
       }
```

```
   }
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.41 encauth/ocb/ocb_decrypt.c File Reference

5.41.1 Detailed Description

OCB implementation, decrypt data, by Tom St Denis.

Definition in file ocb_decrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for ocb_decrypt.c:

Functions

- int ocb_decrypt (ocb_state ∗ocb, const unsigned char ∗ct, unsigned char ∗pt)

  Decrypt a block with OCB.

5.41.2 Function Documentation

5.41.2.1 int ocb_decrypt (ocb_state ∗ *ocb*, const unsigned char ∗ *ct*, unsigned char ∗ *pt*)

Decrypt a block with OCB.

Parameters:

ocb The OCB state

ct The ciphertext (length of the block size of the block cipher)

pt [out] The plaintext (length of ct)

Returns:

CRYPT_OK if successful

Definition at line 27 of file ocb_decrypt.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_decrypt, LTC_ARGCHK, MAXBLOCKSIZE, and ocb_shift_xor().

Referenced by ocb_decrypt_verify_memory().

```
{
   unsigned char Z[MAXBLOCKSIZE], tmp[MAXBLOCKSIZE];
   int err, x;

   LTC_ARGCHK(ocb != NULL);
   LTC_ARGCHK(pt  != NULL);
   LTC_ARGCHK(ct  != NULL);

   /* check if valid cipher */
   if ((err = cipher_is_valid(ocb->cipher)) != CRYPT_OK) {
      return err;
   }
   LTC_ARGCHK(cipher_descriptor[ocb->cipher].ecb_decrypt != NULL);

   /* check length */
   if (ocb->block_len != cipher_descriptor[ocb->cipher].block_length) {
```

```
      return CRYPT_INVALID_ARG;
   }

   /* Get Z[i] value */
   ocb_shift_xor(ocb, Z);

   /* xor ct in, encrypt, xor Z out */
   for (x = 0; x < ocb->block_len; x++) {
       tmp[x] = ct[x] ^ Z[x];
   }
   if ((err = cipher_descriptor[ocb->cipher].ecb_decrypt(tmp, pt, &ocb->key)) != CRYPT_OK) {
      return err;
   }
   for (x = 0; x < ocb->block_len; x++) {
       pt[x] ^= Z[x];
   }

   /* compute checksum */
   for (x = 0; x < ocb->block_len; x++) {
       ocb->checksum[x] ^= pt[x];
   }


#ifdef LTC_CLEAN_STACK
   zeromem(Z, sizeof(Z));
   zeromem(tmp, sizeof(tmp));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.42 encauth/ocb/ocb_decrypt_verify_memory.c File Reference

5.42.1 Detailed Description

OCB implementation, helper to decrypt block of memory, by Tom St Denis.

Definition in file ocb_decrypt_verify_memory.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ocb_decrypt_verify_memory.c:

Functions

- int ocb_decrypt_verify_memory (int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗nonce, const unsigned char ∗ct, unsigned long ctlen, unsigned char ∗pt, const unsigned char ∗tag, unsigned long taglen, int ∗stat)

  Decrypt and compare the tag with OCB.

5.42.2 Function Documentation

5.42.2.1 int ocb_decrypt_verify_memory (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *nonce*, const unsigned char ∗ *ct*, unsigned long *ctlen*, unsigned char ∗ *pt*, const unsigned char ∗ *tag*, unsigned long *taglen*, int ∗ *stat*)

Decrypt and compare the tag with OCB.

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the secret key (octets)

nonce The session nonce (length of the block size of the block cipher)

ct The ciphertext

ctlen The length of the ciphertext (octets)

pt [out] The plaintext

tag The tag to compare against

taglen The length of the tag (octets)

stat [out] The result of the tag comparison (1==valid, 0==invalid)

Returns:

CRYPT_OK if successful regardless of the tag comparison

Definition at line 34 of file ocb_decrypt_verify_memory.c.

References CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, ocb_decrypt(), ocb_done_decrypt(), ocb_init(), XFREE, XMALLOC, and zeromem().

```
{
   int err;
   ocb_state *ocb;
```

```

   LTC_ARGCHK(key    != NULL);
   LTC_ARGCHK(nonce  != NULL);
   LTC_ARGCHK(pt     != NULL);
   LTC_ARGCHK(ct     != NULL);
   LTC_ARGCHK(tag    != NULL);
   LTC_ARGCHK(stat    != NULL);

   /* allocate memory */
   ocb = XMALLOC(sizeof(ocb_state));
   if (ocb == NULL) {
      return CRYPT_MEM;
   }

   if ((err = ocb_init(ocb, cipher, key, keylen, nonce)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   while (ctlen > (unsigned long)ocb->block_len) {
        if ((err = ocb_decrypt(ocb, ct, pt)) != CRYPT_OK) {
            goto LBL_ERR;
        }
        ctlen   -= ocb->block_len;
        pt      += ocb->block_len;
        ct      += ocb->block_len;
   }

   err = ocb_done_decrypt(ocb, ct, ctlen, pt, tag, taglen, stat);
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(ocb, sizeof(ocb_state));
#endif

   XFREE(ocb);

   return err;
}
```

Here is the call graph for this function:

5.43 encauth/ocb/ocb_done_decrypt.c File Reference

5.43.1 Detailed Description

OCB implementation, terminate decryption, by Tom St Denis.

Definition in file ocb_done_decrypt.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ocb_done_decrypt.c:

Functions

- int ocb_done_decrypt (ocb_state ∗ocb, const unsigned char ∗ct, unsigned long ctlen, unsigned char ∗pt, const unsigned char ∗tag, unsigned long taglen, int ∗stat)

  Terminate a decrypting OCB state.

5.43.2 Function Documentation

5.43.2.1 int ocb_done_decrypt (ocb_state ∗ *ocb*, const unsigned char ∗ *ct*, unsigned long *ctlen*, unsigned char ∗ *pt*, const unsigned char ∗ *tag*, unsigned long *taglen*, int ∗ *stat*)

Terminate a decrypting OCB state.

Parameters:

ocb The OCB state

ct The ciphertext (if any)

ctlen The length of the ciphertext (octets)

pt [out] The plaintext

tag The authentication tag (to compare against)

taglen The length of the authentication tag provided

stat [out] The result of the tag comparison

Returns:

CRYPT_OK if the process was successful regardless if the tag is valid

Definition at line 31 of file ocb_done_decrypt.c.

References CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, MAXBLOCKSIZE, s_ocb_done(), XFREE, XMALLOC, XMEMCMP, and zeromem().

Referenced by ocb_decrypt_verify_memory().

```
{
   int err;
   unsigned char *tagbuf;
   unsigned long tagbuflen;

   LTC_ARGCHK(ocb   != NULL);
   LTC_ARGCHK(pt    != NULL);
   LTC_ARGCHK(ct    != NULL);
```

```
   LTC_ARGCHK(tag  != NULL);
   LTC_ARGCHK(stat != NULL);

   /* default to failed */
   *stat = 0;

   /* allocate memory */
   tagbuf = XMALLOC(MAXBLOCKSIZE);
   if (tagbuf == NULL) {
      return CRYPT_MEM;
   }

   tagbuflen = MAXBLOCKSIZE;
   if ((err = s_ocb_done(ocb, ct, ctlen, pt, tagbuf, &tagbuflen, 1)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   if (taglen <= tagbuflen && XMEMCMP(tagbuf, tag, taglen) == 0) {
      *stat = 1;
   }

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(tagbuf, MAXBLOCKSIZE);
#endif

   XFREE(tagbuf);

   return err;
}
```

Here is the call graph for this function:

5.44 encauth/ocb/ocb_done_encrypt.c File Reference

5.44.1 Detailed Description

OCB implementation, terminate encryption, by Tom St Denis.

Definition in file ocb_done_encrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for ocb_done_encrypt.c:

Functions

- int ocb_done_encrypt (ocb_state ∗ocb, const unsigned char ∗pt, unsigned long ptlen, unsigned char ∗ct, unsigned char ∗tag, unsigned long ∗taglen)

  Terminate an encryption OCB state.

5.44.2 Function Documentation

5.44.2.1 int ocb_done_encrypt (ocb_state ∗ *ocb*, const unsigned char ∗ *pt*, unsigned long *ptlen*, unsigned char ∗ *ct*, unsigned char ∗ *tag*, unsigned long ∗ *taglen*)

Terminate an encryption OCB state.

Parameters:

ocb The OCB state

pt Remaining plaintext (if any)

ptlen The length of the plaintext (octets)

ct [out] The ciphertext (if any)

tag [out] The tag for the OCB stream

taglen [in/out] The max size and resulting size of the tag

Returns:

CRYPT_OK if successful

Definition at line 30 of file ocb_done_encrypt.c.

References LTC_ARGCHK, and s_ocb_done().

Referenced by ocb_encrypt_authenticate_memory().

```
{
   LTC_ARGCHK(ocb    != NULL);
   LTC_ARGCHK(pt     != NULL);
   LTC_ARGCHK(ct     != NULL);
   LTC_ARGCHK(tag    != NULL);
   LTC_ARGCHK(taglen != NULL);
   return s_ocb_done(ocb, pt, ptlen, ct, tag, taglen, 0);
}
```

Here is the call graph for this function:

5.45 encauth/ocb/ocb_encrypt.c File Reference

5.45.1 Detailed Description

OCB implementation, encrypt data, by Tom St Denis.

Definition in file ocb_encrypt.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ocb_encrypt.c:

Functions

- int ocb_encrypt (ocb_state ∗ocb, const unsigned char ∗pt, unsigned char ∗ct)

  Encrypt a block of data with OCB.

5.45.2 Function Documentation

5.45.2.1 int ocb_encrypt (ocb_state ∗ *ocb*, const unsigned char ∗ *pt*, unsigned char ∗ *ct*)

Encrypt a block of data with OCB.

Parameters:

ocb The OCB state

pt The plaintext (length of the block size of the block cipher)

ct [out] The ciphertext (same size as the pt)

Returns:

CRYPT_OK if successful

Definition at line 27 of file ocb_encrypt.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, and MAXBLOCKSIZE.

Referenced by ocb_encrypt_authenticate_memory().

```
{
   unsigned char Z[MAXBLOCKSIZE], tmp[MAXBLOCKSIZE];
   int err, x;

   LTC_ARGCHK(ocb != NULL);
   LTC_ARGCHK(pt  != NULL);
   LTC_ARGCHK(ct  != NULL);
   if ((err = cipher_is_valid(ocb->cipher)) != CRYPT_OK) {
      return err;
   }
   if (ocb->block_len != cipher_descriptor[ocb->cipher].block_length) {
      return CRYPT_INVALID_ARG;
   }

   /* compute checksum */
   for (x = 0; x < ocb->block_len; x++) {
       ocb->checksum[x] ^= pt[x];
```

```
   }

   /* Get Z[i] value */
   ocb_shift_xor(ocb, Z);

   /* xor pt in, encrypt, xor Z out */
   for (x = 0; x < ocb->block_len; x++) {
       tmp[x] = pt[x] ^ Z[x];
   }
   if ((err = cipher_descriptor[ocb->cipher].ecb_encrypt(tmp, ct, &ocb->key)) != CRYPT_OK) {
      return err;
   }
   for (x = 0; x < ocb->block_len; x++) {
       ct[x] ^= Z[x];
   }

#ifdef LTC_CLEAN_STACK
   zeromem(Z, sizeof(Z));
   zeromem(tmp, sizeof(tmp));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.46 encauth/ocb/ocb_encrypt_authenticate_memory.c File Reference

5.46.1 Detailed Description

OCB implementation, encrypt block of memory, by Tom St Denis.

Definition in file ocb_encrypt_authenticate_memory.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ocb_encrypt_authenticate_memory.c:

Functions

- int ocb_encrypt_authenticate_memory (int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗nonce, const unsigned char ∗pt, unsigned long ptlen, unsigned char ∗ct, unsigned char ∗tag, unsigned long ∗taglen)

  Encrypt and generate an authentication code for a buffer of memory.

5.46.2 Function Documentation

5.46.2.1 int ocb_encrypt_authenticate_memory (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *nonce*, const unsigned char ∗ *pt*, unsigned long *ptlen*, unsigned char ∗ *ct*, unsigned char ∗ *tag*, unsigned long ∗ *taglen*)

Encrypt and generate an authentication code for a buffer of memory.

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the secret key (octets)

nonce The session nonce (length of the block ciphers block size)

pt The plaintext

ptlen The length of the plaintext (octets)

ct [out] The ciphertext

tag [out] The authentication tag

taglen [in/out] The max size and resulting size of the authentication tag

Returns:

CRYPT_OK if successful

Definition at line 33 of file ocb_encrypt_authenticate_memory.c.

References CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, ocb_done_encrypt(), ocb_encrypt(), ocb_init(), XFREE, XMALLOC, and zeromem().

Referenced by ocb_test().

```
{
   int err;
   ocb_state *ocb;

   LTC_ARGCHK(key    != NULL);
   LTC_ARGCHK(nonce  != NULL);
   LTC_ARGCHK(pt     != NULL);
   LTC_ARGCHK(ct     != NULL);
   LTC_ARGCHK(tag    != NULL);
   LTC_ARGCHK(taglen != NULL);

   /* allocate ram */
   ocb = XMALLOC(sizeof(ocb_state));
   if (ocb == NULL) {
      return CRYPT_MEM;
   }

   if ((err = ocb_init(ocb, cipher, key, keylen, nonce)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   while (ptlen > (unsigned long)ocb->block_len) {
        if ((err = ocb_encrypt(ocb, pt, ct)) != CRYPT_OK) {
           goto LBL_ERR;
        }
        ptlen   -= ocb->block_len;
        pt      += ocb->block_len;
        ct      += ocb->block_len;
   }

   err = ocb_done_encrypt(ocb, pt, ptlen, ct, tag, taglen);
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(ocb, sizeof(ocb_state));
#endif

   XFREE(ocb);

   return err;
}
```

Here is the call graph for this function:

5.47 encauth/ocb/ocb_init.c File Reference

5.47.1 Detailed Description

OCB implementation, initialize state, by Tom St Denis.

Definition in file ocb_init.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ocb_init.c:

Functions

- int ocb_init (ocb_state ∗ocb, int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗nonce)
  Initialize an OCB context.

Variables

- struct {
  int len
  unsigned char poly_div [MAXBLOCKSIZE]
  unsigned char poly_mul [MAXBLOCKSIZE]
  int code
  int value
  } polys [ ]

5.47.2 Function Documentation

5.47.2.1 int ocb_init (ocb_state ∗ *ocb*, int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *nonce*)

Initialize an OCB context.

Parameters:

ocb [out] The destination of the OCB state

cipher The index of the desired cipher

key The secret key

keylen The length of the secret key (octets)

nonce The session nonce (length of the block size of the cipher)

Returns:

CRYPT_OK if successful

Definition at line 47 of file ocb_init.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_OK, len, LTC_ARGCHK, poly, and polys.

Referenced by ocb_decrypt_verify_memory(), and ocb_encrypt_authenticate_memory().

```
{
   int poly, x, y, m, err;

   LTC_ARGCHK(ocb   != NULL);
   LTC_ARGCHK(key   != NULL);
   LTC_ARGCHK(nonce != NULL);

   /* valid cipher? */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

   /* determine which polys to use */
   ocb->block_len = cipher_descriptor[cipher].block_length;
   for (poly = 0; poly < (int)(sizeof(polys)/sizeof(polys[0])); poly++) {
       if (polys[poly].len == ocb->block_len) {
          break;
       }
   }
   if (polys[poly].len != ocb->block_len) {
      return CRYPT_INVALID_ARG;
   }

   /* schedule the key */
   if ((err = cipher_descriptor[cipher].setup(key, keylen, 0, &ocb->key)) != CRYPT_OK) {
      return err;
   }

   /* find L = E[0] */
   zeromem(ocb->L, ocb->block_len);
   if ((err = cipher_descriptor[cipher].ecb_encrypt(ocb->L, ocb->L, &ocb->key)) != CRYPT_OK) {
      return err;
   }

   /* find R = E[N xor L] */
   for (x = 0; x < ocb->block_len; x++) {
       ocb->R[x] = ocb->L[x] ^ nonce[x];
   }
   if ((err = cipher_descriptor[cipher].ecb_encrypt(ocb->R, ocb->R, &ocb->key)) != CRYPT_OK) {
      return err;
   }

   /* find Ls[i] = L << i for i == 0..31 */
   XMEMCPY(ocb->Ls[0], ocb->L, ocb->block_len);
   for (x = 1; x < 32; x++) {
       m = ocb->Ls[x-1][0] >> 7;
       for (y = 0; y < ocb->block_len-1; y++) {
           ocb->Ls[x][y] = ((ocb->Ls[x-1][y] << 1) | (ocb->Ls[x-1][y+1] >> 7)) & 255;
       }
       ocb->Ls[x][ocb->block_len-1] = (ocb->Ls[x-1][ocb->block_len-1] << 1) & 255;

       if (m == 1) {
          for (y = 0; y < ocb->block_len; y++) {
              ocb->Ls[x][y] ^= polys[poly].poly_mul[y];
          }
       }
   }

   /* find Lr = L / x */
   m = ocb->L[ocb->block_len-1] & 1;

   /* shift right */
   for (x = ocb->block_len - 1; x > 0; x--) {
       ocb->Lr[x] = ((ocb->L[x] >> 1) | (ocb->L[x-1] << 7)) & 255;
   }
   ocb->Lr[0] = ocb->L[0] >> 1;

```

```
    if (m == 1) {
       for (x = 0; x < ocb->block_len; x++) {
           ocb->Lr[x] ^= polys[poly].poly_div[x];
       }
    }

    /* set Li, checksum */
    zeromem(ocb->Li,       ocb->block_len);
    zeromem(ocb->checksum, ocb->block_len);

    /* set other params */
    ocb->block_index = 1;
    ocb->cipher      = cipher;

    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.47.3 Variable Documentation

5.47.3.1 int len

Definition at line 21 of file ocb_init.c.

Referenced by ccm_memory(), der_decode_ia5_string(), der_decode_object_identifier(), der_decode_-octet_string(), der_decode_printable_string(), der_decode_sequence_flexi(), der_decode_short_integer(), der_encode_bit_string(), der_encode_ia5_string(), der_encode_octet_string(), der_encode_printable_-string(), der_encode_short_integer(), der_length_integer(), der_length_short_integer(), eax_done(), eax_-init(), eax_test(), ocb_init(), ocb_test(), omac_init(), omac_test(), pmac_init(), pmac_test(), sober128_-test(), and whirlpool_test().

5.47.3.2 unsigned char poly_div[MAXBLOCKSIZE]

Definition at line 22 of file ocb_init.c.

5.47.3.3 unsigned char poly_mul[MAXBLOCKSIZE]

Definition at line 22 of file ocb_init.c.

5.47.3.4 const { ... } polys[ ] `[static]`

Referenced by ocb_init(), and pmac_init().

5.48 encauth/ocb/ocb_ntz.c File Reference

5.48.1 Detailed Description

OCB implementation, internal function, by Tom St Denis.

Definition in file ocb_ntz.c.

`#include "tomcrypt.h"`

Include dependency graph for ocb_ntz.c:

Functions

- int ocb_ntz (unsigned long x)

  Returns the number of leading zero bits [from lsb up].

5.48.2 Function Documentation

5.48.2.1 int ocb_ntz (unsigned long *x*)

Returns the number of leading zero bits [from lsb up].

Parameters:

x The 32-bit value to observe

Returns:

The number of bits [from the lsb up] that are zero

Definition at line 26 of file ocb_ntz.c.

References c.

Referenced by ocb_shift_xor().

```
{
   int c;
   x &= 0xFFFFFFFFUL;
   c = 0;
   while ((x & 1) == 0) {
      ++c;
      x >>= 1;
   }
   return c;
}
```

5.49 encauth/ocb/ocb_shift_xor.c File Reference

5.49.1 Detailed Description

OCB implementation, internal function, by Tom St Denis.

Definition in file ocb_shift_xor.c.

`#include "tomcrypt.h"`

Include dependency graph for ocb_shift_xor.c:

Functions

- void ocb_shift_xor (ocb_state ∗ocb, unsigned char ∗Z)

  Compute the shift/xor for OCB (internal function).

5.49.2 Function Documentation

5.49.2.1 void ocb_shift_xor (ocb_state ∗ *ocb*, unsigned char ∗ *Z*)

Compute the shift/xor for OCB (internal function).

Parameters:

ocb The OCB state

Z The destination of the shift

Definition at line 25 of file ocb_shift_xor.c.

References ocb_ntz().

Referenced by ocb_decrypt(), and s_ocb_done().

```
{
   int x, y;
   y = ocb_ntz(ocb->block_index++);
   for (x = 0; x < ocb->block_len; x++) {
       ocb->Li[x] ^= ocb->Ls[y][x];
       Z[x]        = ocb->Li[x] ^ ocb->R[x];
   }
}
```

Here is the call graph for this function:

5.50 encauth/ocb/ocb_test.c File Reference

5.50.1 Detailed Description

OCB implementation, self-test by Tom St Denis.

Definition in file ocb_test.c.

`#include "tomcrypt.h"`

Include dependency graph for ocb_test.c:

Functions

- int ocb_test (void)

  Test the OCB protocol.

5.50.2 Function Documentation

5.50.2.1 int ocb_test (void)

Test the OCB protocol.

Returns:

CRYPT_OK if successful

Definition at line 24 of file ocb_test.c.

References CRYPT_NOP, CRYPT_OK, find_cipher(), len, MAXBLOCKSIZE, and ocb_encrypt_authenticate_memory().

```
25 {
26 #ifndef LTC_TEST
27    return CRYPT_NOP;
28 #else
29    static const struct {
30          int ptlen;
31          unsigned char key[16], nonce[16], pt[34], ct[34], tag[16];
32    } tests[] = {
33
34    /* OCB-AES-128-0B */
35 {
36    0,
37    /* key */
38    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
39      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
40    /* nonce */
41    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
42      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
43    /* pt */
44    { 0 },
45    /* ct */
46    { 0 },
47    /* tag */
48    { 0x15, 0xd3, 0x7d, 0xd7, 0xc8, 0x90, 0xd5, 0xd6,
49      0xac, 0xab, 0x92, 0x7b, 0xc0, 0xdc, 0x60, 0xee },
50 },
```

```


   /* OCB-AES-128-3B */
{
   3,
   /* key */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* nonce */
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
   /* pt */
   { 0x00, 0x01, 0x02 },
   /* ct */
   { 0xfc, 0xd3, 0x7d },
   /* tag */
   { 0x02, 0x25, 0x47, 0x39, 0xa5, 0xe3, 0x56, 0x5a,
     0xe2, 0xdc, 0xd6, 0x2c, 0x65, 0x97, 0x46, 0xba },
},

   /* OCB-AES-128-16B */
{
   16,
   /* key */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* nonce */
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
   /* pt */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* ct */
   { 0x37, 0xdf, 0x8c, 0xe1, 0x5b, 0x48, 0x9b, 0xf3,
     0x1d, 0x0f, 0xc4, 0x4d, 0xa1, 0xfa, 0xf6, 0xd6 },
   /* tag */
   { 0xdf, 0xb7, 0x63, 0xeb, 0xdb, 0x5f, 0x0e, 0x71,
     0x9c, 0x7b, 0x41, 0x61, 0x80, 0x80, 0x04, 0xdf },
},

   /* OCB-AES-128-20B  */
{
   20,
   /* key */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
   /* nonce */
   { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
   /* pt */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
     0x10, 0x11, 0x12, 0x13 },
   /* ct */
   { 0x01, 0xa0, 0x75, 0xf0, 0xd8, 0x15, 0xb1, 0xa4,
     0xe9, 0xc8, 0x81, 0xa1, 0xbc, 0xff, 0xc3, 0xeb,
     0x70, 0x03, 0xeb, 0x55},
   /* tag */
   { 0x75, 0x30, 0x84, 0x14, 0x4e, 0xb6, 0x3b, 0x77,
     0x0b, 0x06, 0x3c, 0x2e, 0x23, 0xcd, 0xa0, 0xbb },
},

   /* OCB-AES-128-32B  */
{
   32,
   /* key */
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
```

```
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
    /* nonce */
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
    /* pt */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
      0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f },
    /* ct */
    { 0x01, 0xa0, 0x75, 0xf0, 0xd8, 0x15, 0xb1, 0xa4,
      0xe9, 0xc8, 0x81, 0xa1, 0xbc, 0xff, 0xc3, 0xeb,
      0x4a, 0xfc, 0xbb, 0x7f, 0xed, 0xc0, 0x8c, 0xa8,
      0x65, 0x4c, 0x6d, 0x30, 0x4d, 0x16, 0x12, 0xfa },

    /* tag */
    { 0xc1, 0x4c, 0xbf, 0x2c, 0x1a, 0x1f, 0x1c, 0x3c,
      0x13, 0x7e, 0xad, 0xea, 0x1f, 0x2f, 0x2f, 0xcf },
},

    /* OCB-AES-128-34B  */
{
    34,
    /* key */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
    /* nonce */
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01 },
    /* pt */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
      0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
      0x20, 0x21 },
    /* ct */
    { 0x01, 0xa0, 0x75, 0xf0, 0xd8, 0x15, 0xb1, 0xa4,
      0xe9, 0xc8, 0x81, 0xa1, 0xbc, 0xff, 0xc3, 0xeb,
      0xd4, 0x90, 0x3d, 0xd0, 0x02, 0x5b, 0xa4, 0xaa,
      0x83, 0x7c, 0x74, 0xf1, 0x21, 0xb0, 0x26, 0x0f,
      0xa9, 0x5d },

    /* tag */
    { 0xcf, 0x83, 0x41, 0xbb, 0x10, 0x82, 0x0c, 0xcf,
      0x14, 0xbd, 0xec, 0x56, 0xb8, 0xd7, 0xd6, 0xab },
},

};

    int err, x, idx, res;
    unsigned long len;
    unsigned char outct[MAXBLOCKSIZE], outtag[MAXBLOCKSIZE];

     /* AES can be under rijndael or aes... try to find it */
     if ((idx = find_cipher("aes")) == -1) {
        if ((idx = find_cipher("rijndael")) == -1) {
           return CRYPT_NOP;
        }
     }

     for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
         len = sizeof(outtag);
         if ((err = ocb_encrypt_authenticate_memory(idx, tests[x].key, 16,
              tests[x].nonce, tests[x].pt, tests[x].ptlen, outct, outtag, &len)) != CRYPT_OK) {
            return err;
         }

```

```
        if (XMEMCMP(outtag, tests[x].tag, len) || XMEMCMP(outct, tests[x].ct, tests[x].ptlen)) {
#if 0
           unsigned long y;
           printf("\n\nFailure: \nCT:\n");
           for (y = 0; y < (unsigned long)tests[x].ptlen; ) {
               printf("0x%02x", outct[y]);
               if (y < (unsigned long)(tests[x].ptlen-1)) printf(", ");
               if (!(++y % 8)) printf("\n");
           }
           printf("\nTAG:\n");
           for (y = 0; y < len; ) {
               printf("0x%02x", outtag[y]);
               if (y < len-1) printf(", ");
               if (!(++y % 8)) printf("\n");
           }
#endif
           return CRYPT_FAIL_TESTVECTOR;
        }

        if ((err = ocb_decrypt_verify_memory(idx, tests[x].key, 16, tests[x].nonce, outct, tests[x].pt
             outct, tests[x].tag, len, &res)) != CRYPT_OK) {
           return err;
        }
        if ((res != 1) || XMEMCMP(tests[x].pt, outct, tests[x].ptlen)) {
#if 0
           unsigned long y;
           printf("\n\nFailure-decrypt: \nPT:\n");
           for (y = 0; y < (unsigned long)tests[x].ptlen; ) {
               printf("0x%02x", outct[y]);
               if (y < (unsigned long)(tests[x].ptlen-1)) printf(", ");
               if (!(++y % 8)) printf("\n");
           }
           printf("\nres = %d\n\n", res);
#endif
        }
    }
    return CRYPT_OK;
#endif /* LTC_TEST */
}
```

Here is the call graph for this function:

5.51 encauth/ocb/s_ocb_done.c File Reference

5.51.1 Detailed Description

OCB implementation, internal helper, by Tom St Denis.

Definition in file s_ocb_done.c.

```
#include "tomcrypt.h"
```

Include dependency graph for s_ocb_done.c:

Functions

- int s_ocb_done (ocb_state ∗ocb, const unsigned char ∗pt, unsigned long ptlen, unsigned char ∗ct, unsigned char ∗tag, unsigned long ∗taglen, int mode)

  Shared code to finish an OCB stream.

5.51.2 Function Documentation

5.51.2.1 int s_ocb_done (ocb_state ∗ *ocb*, const unsigned char ∗ *pt*, unsigned long *ptlen*, unsigned char ∗ *ct*, unsigned char ∗ *tag*, unsigned long ∗ *taglen*, int *mode*)

Shared code to finish an OCB stream.

Parameters:

ocb The OCB state

pt The remaining plaintext [or input]

ptlen The length of the input (octets)

ct [out] The output buffer

tag [out] The destination for the authentication tag

taglen [in/out] The max size and resulting size of the authentication tag

mode The mode we are terminating, 0==encrypt, 1==decrypt

Returns:

CRYPT_OK if successful

Definition at line 39 of file s_ocb_done.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, MAXBLOCKSIZE, ocb_shift_xor(), XFREE, XMALLOC, and XMEMCPY.

Referenced by ocb_done_decrypt(), and ocb_done_encrypt().

```
{
   unsigned char *Z, *Y, *X;
   int err, x;

   LTC_ARGCHK(ocb    != NULL);
   LTC_ARGCHK(pt     != NULL);
   LTC_ARGCHK(ct     != NULL);
```

```
   LTC_ARGCHK(tag    != NULL);
   LTC_ARGCHK(taglen != NULL);
   if ((err = cipher_is_valid(ocb->cipher)) != CRYPT_OK) {
      return err;
   }
   if (ocb->block_len != cipher_descriptor[ocb->cipher].block_length ||
       (int)ptlen > ocb->block_len || (int)ptlen < 0) {
      return CRYPT_INVALID_ARG;
   }

   /* allocate ram */
   Z = XMALLOC(MAXBLOCKSIZE);
   Y = XMALLOC(MAXBLOCKSIZE);
   X = XMALLOC(MAXBLOCKSIZE);
   if (X == NULL || Y == NULL || Z == NULL) {
      if (X != NULL) {
         XFREE(X);
      }
      if (Y != NULL) {
         XFREE(Y);
      }
      if (Z != NULL) {
         XFREE(Z);
      }
      return CRYPT_MEM;
   }

   /* compute X[m] = len(pt[m]) XOR Lr XOR Z[m] */
   ocb_shift_xor(ocb, X);
   XMEMCPY(Z, X, ocb->block_len);

   X[ocb->block_len-1] ^= (ptlen*8)&255;
   X[ocb->block_len-2] ^= ((ptlen*8)>>8)&255;
   for (x = 0; x < ocb->block_len; x++) {
       X[x] ^= ocb->Lr[x];
   }

   /* Y[m] = E(X[m])) */
   if ((err = cipher_descriptor[ocb->cipher].ecb_encrypt(X, Y, &ocb->key)) != CRYPT_OK) {
      goto error;
   }

   if (mode == 1) {
      /* decrypt mode, so let's xor it first */
      /* xor C[m] into checksum */
      for (x = 0; x < (int)ptlen; x++) {
         ocb->checksum[x] ^= ct[x];
      }
   }

   /* C[m] = P[m] xor Y[m] */
    for (x = 0; x < (int)ptlen; x++) {
        ct[x] = pt[x] ^ Y[x];
    }

    if (mode == 0) {
       /* encrypt mode */
       /* xor C[m] into checksum */
       for (x = 0; x < (int)ptlen; x++) {
           ocb->checksum[x] ^= ct[x];
       }
    }

    /* xor Y[m] and Z[m] into checksum */
    for (x = 0; x < ocb->block_len; x++) {
        ocb->checksum[x] ^= Y[x] ^ Z[x];
    }
```

```

   /* encrypt checksum, er... tag!! */
   if ((err = cipher_descriptor[ocb->cipher].ecb_encrypt(ocb->checksum, X, &ocb->key)) != CRYPT_OK) {
      goto error;
   }
   cipher_descriptor[ocb->cipher].done(&ocb->key);

   /* now store it */
   for (x = 0; x < ocb->block_len && x < (int)*taglen; x++) {
       tag[x] = X[x];
   }
   *taglen = x;

#ifdef LTC_CLEAN_STACK
   zeromem(X, MAXBLOCKSIZE);
   zeromem(Y, MAXBLOCKSIZE);
   zeromem(Z, MAXBLOCKSIZE);
   zeromem(ocb, sizeof(*ocb));
#endif
error:
   XFREE(X);
   XFREE(Y);
   XFREE(Z);

   return err;
}
```

Here is the call graph for this function:

5.52 hashes/chc/chc.c File Reference

5.52.1 Detailed Description

CHC support.

(Tom St Denis)

Definition in file chc.c.

`#include "tomcrypt.h"`

Include dependency graph for chc.c:

Defines

- #define UNDEFED_HASH -17

Functions

- int chc_register (int cipher)

  Initialize the CHC state with a given cipher.

- int chc_init (hash_state ∗md)

  Initialize the hash state.

- static int chc_compress (hash_state ∗md, unsigned char ∗buf)
- int _chc_process (hash_state ∗md, const unsigned char ∗buf, unsigned long len)
- HASH_PROCESS (_chc_process, chc_compress, chc,(unsigned long) cipher_blocksize) int chc_-process(hash_state ∗md

  Process a block of memory though the hash.

Variables

- static int cipher_idx = UNDEFED_HASH
- static int cipher_blocksize
- const struct ltc_hash_descriptor chc_desc
- const unsigned char ∗ in

5.52.2 Define Documentation

5.52.2.1 #define UNDEFED_HASH -17

Definition at line 21 of file chc.c.

5.52.3 Function Documentation

5.52.3.1 int _chc_process (hash_state * *md*, const unsigned char * *buf*, unsigned long *len*)

5.52.3.2 static int chc_compress (hash_state * *md*, unsigned char * *buf*) [static]

Definition at line 132 of file chc.c.

References cipher_blocksize, cipher_descriptor, cipher_idx, CRYPT_MEM, CRYPT_OK, ltc_cipher_-descriptor::ecb_encrypt, MAXBLOCKSIZE, XFREE, XMALLOC, and XMEMCPY.

```
{
   unsigned char  T[2][MAXBLOCKSIZE];
   symmetric_key *key;
   int            err, x;

   if ((key = XMALLOC(sizeof(*key))) == NULL) {
      return CRYPT_MEM;
   }
   if ((err = cipher_descriptor[cipher_idx].setup(md->chc.state, cipher_blocksize, 0, key)) != CRYPT_C
      XFREE(key);
      return err;
   }
   XMEMCPY(T[1], buf, cipher_blocksize);
   cipher_descriptor[cipher_idx].ecb_encrypt(buf, T[0], key);
   for (x = 0; x < cipher_blocksize; x++) {
       md->chc.state[x] ^= T[0][x] ^ T[1][x];
   }
   XFREE(key);
#ifdef LTC_CLEAN_STACK
   zeromem(T, sizeof(T));
   zeromem(&key, sizeof(key));
#endif
   return CRYPT_OK;
}
```

5.52.3.3 int chc_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 87 of file chc.c.

References cipher_blocksize, cipher_descriptor, cipher_idx, cipher_is_valid(), CRYPT_INVALID_-CIPHER, CRYPT_MEM, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, LTC_ARGCHK, MAXBLOCKSIZE, XFREE, XMALLOC, and zeromem().

```
{
   symmetric_key *key;
   unsigned char  buf[MAXBLOCKSIZE];
   int            err;

   LTC_ARGCHK(md != NULL);

```

```
   /* is the cipher valid? */
   if ((err = cipher_is_valid(cipher_idx)) != CRYPT_OK) {
      return err;
   }

   if (cipher_blocksize != cipher_descriptor[cipher_idx].block_length) {
      return CRYPT_INVALID_CIPHER;
   }

   if ((key = XMALLOC(sizeof(*key))) == NULL) {
      return CRYPT_MEM;
   }

   /* zero key and what not */
   zeromem(buf, cipher_blocksize);
   if ((err = cipher_descriptor[cipher_idx].setup(buf, cipher_blocksize, 0, key)) != CRYPT_OK) {
      XFREE(key);
      return err;
   }

   /* encrypt zero block */
   cipher_descriptor[cipher_idx].ecb_encrypt(buf, md->chc.state, key);

   /* zero other members */
   md->chc.length = 0;
   md->chc.curlen = 0;
   zeromem(md->chc.buf, sizeof(md->chc.buf));
   XFREE(key);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.52.3.4 int chc_register (int *cipher*)

Initialize the CHC state with a given cipher.

Parameters:

cipher The index of the cipher you wish to bind

Returns:

CRYPT_OK if successful

Definition at line 42 of file chc.c.

References ltc_cipher_descriptor::block_length, ltc_hash_descriptor::blocksize, cipher_blocksize, cipher_descriptor, cipher_idx, cipher_is_valid(), CRYPT_INVALID_CIPHER, CRYPT_OK, find_hash(), hash_descriptor, hash_is_valid(), and ltc_hash_descriptor::hashsize.

```
{
   int err, kl, idx;

   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

   /* will it be valid? */
   kl = cipher_descriptor[cipher].block_length;

   /* must be >64 bit block */
   if (kl <= 8) {
```

```
      return CRYPT_INVALID_CIPHER;
   }

   /* can we use the ideal keysize? */
   if ((err = cipher_descriptor[cipher].keysize(&kl)) != CRYPT_OK) {
      return err;
   }
   /* we require that key size == block size be a valid choice */
   if (kl != cipher_descriptor[cipher].block_length) {
      return CRYPT_INVALID_CIPHER;
   }

   /* determine if chc_hash has been register_hash'ed already */
   if ((err = hash_is_valid(idx = find_hash("chc_hash"))) != CRYPT_OK) {
      return err;
   }

   /* store into descriptor */
   hash_descriptor[idx].hashsize  =
   hash_descriptor[idx].blocksize = cipher_descriptor[cipher].block_length;

   /* store the idx and block size */
   cipher_idx       = cipher;
   cipher_blocksize = cipher_descriptor[cipher].block_length;
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.52.3.5 HASH_PROCESS (_chc_process, chc_compress, chc, (unsigned long) *cipher_blocksize*)

Process a block of memory though the hash.

Parameters:

md The hash state

in The data to hash

inlen The length of the data (octets)

Returns:

CRYPT_OK if successful

5.52.4 Variable Documentation

5.52.4.1 const struct ltc_hash_descriptor chc_desc

Initial value:

```
 {
   "chc_hash", 12, 0, 0, { 0 }, 0,
   &chc_init,
   &chc_process,
   &chc_done,
   &chc_test,
   NULL
}
```

Definition at line 28 of file chc.c.

5.52.4.2 int cipher_blocksize [static]

Definition at line 24 of file chc.c.

Referenced by chc_compress(), chc_init(), and chc_register().

5.52.4.3 int cipher_idx = UNDEFED_HASH [static]

Definition at line 24 of file chc.c.

Referenced by chc_compress(), chc_init(), and chc_register().

5.52.4.4 const unsigned char* in

Definition at line 169 of file chc.c.

Referenced by f9_file(), hash_file(), hmac_file(), omac_file(), pmac_file(), and xcbc_file().

5.53 hashes/helper/hash_file.c File Reference

5.53.1 Detailed Description

Hash a file, Tom St Denis.

Definition in file hash_file.c.

`#include "tomcrypt.h"`

Include dependency graph for hash_file.c:

Functions

- int hash_file (int hash, const char ∗fname, unsigned char ∗out, unsigned long ∗outlen)

5.53.2 Function Documentation

5.53.2.1 int hash_file (int *hash*, const char ∗ *fname*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Parameters:

hash The index of the hash desired

fname The name of the file you wish to hash

out [out] The destination of the digest

outlen [in/out] The max size and resulting size of the message digest

Returns:

CRYPT_OK if successful

Definition at line 25 of file hash_file.c.

References CRYPT_ERROR, CRYPT_FILE_NOTFOUND, CRYPT_NOP, CRYPT_OK, hash_filehandle(), hash_is_valid(), in, and LTC_ARGCHK.

```
26 {
27 #ifdef LTC_NO_FILE
28     return CRYPT_NOP;
29 #else
30     FILE *in;
31     int err;
32     LTC_ARGCHK(fname  != NULL);
33     LTC_ARGCHK(out    != NULL);
34     LTC_ARGCHK(outlen != NULL);
35 
36     if ((err = hash_is_valid(hash)) != CRYPT_OK) {
37         return err;
38     }
39 
40     in = fopen(fname, "rb");
41     if (in == NULL) {
42        return CRYPT_FILE_NOTFOUND;
43     }
44 
45     err = hash_filehandle(hash, in, out, outlen);
46     if (fclose(in) != 0) {
47        return CRYPT_ERROR;
48     }
```

```

    return err;
#endif
}
```

Here is the call graph for this function:

5.54 hashes/helper/hash_filehandle.c File Reference

5.54.1 Detailed Description

Hash open files, Tom St Denis.

Definition in file hash_filehandle.c.

`#include "tomcrypt.h"`

Include dependency graph for hash_filehandle.c:

Functions

- int hash_filehandle (int hash, FILE ∗in, unsigned char ∗out, unsigned long ∗outlen)

  Hash data from an open file handle.

5.54.2 Function Documentation

5.54.2.1 int hash_filehandle (int *hash*, FILE ∗ *in*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Hash data from an open file handle.

Parameters:

hash The index of the hash you want to use

in The FILE∗ handle of the file you want to hash

out [out] The destination of the digest

outlen [in/out] The max size and resulting size of the digest

Returns:

CRYPT_OK if successful

Definition at line 26 of file hash_filehandle.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_NOP, CRYPT_OK, ltc_hash_descriptor::done, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, and zeromem().

Referenced by hash_file().

```
{
#ifdef LTC_NO_FILE
    return CRYPT_NOP;
#else
    hash_state md;
    unsigned char buf[512];
    size_t x;
    int err;

    LTC_ARGCHK(out    != NULL);
    LTC_ARGCHK(outlen != NULL);
    LTC_ARGCHK(in     != NULL);

    if ((err = hash_is_valid(hash)) != CRYPT_OK) {
        return err;
    }
```

```

    if (*outlen < hash_descriptor[hash].hashsize) {
       *outlen = hash_descriptor[hash].hashsize;
       return CRYPT_BUFFER_OVERFLOW;
    }
    if ((err = hash_descriptor[hash].init(&md)) != CRYPT_OK) {
       return err;
    }

    *outlen = hash_descriptor[hash].hashsize;
    do {
        x = fread(buf, 1, sizeof(buf), in);
        if ((err = hash_descriptor[hash].process(&md, buf, x)) != CRYPT_OK) {
           return err;
        }
    } while (x == sizeof(buf));
    err = hash_descriptor[hash].done(&md, out);

#ifdef LTC_CLEAN_STACK
    zeromem(buf, sizeof(buf));
#endif
    return err;
#endif
}
```

Here is the call graph for this function:

5.55 hashes/helper/hash_memory.c File Reference

5.55.1 Detailed Description

Hash memory helper, Tom St Denis.

Definition in file hash_memory.c.

`#include "tomcrypt.h"`

Include dependency graph for hash_memory.c:

Functions

- int hash_memory (int hash, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Hash a block of memory and store the digest.

5.55.2 Function Documentation

5.55.2.1 int hash_memory (int *hash*, const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Hash a block of memory and store the digest.

Parameters:

hash The index of the hash you wish to use

in The data you wish to hash

inlen The length of the data to hash (octets)

out [out] Where to store the digest

outlen [in/out] Max size and resulting size of the digest

Returns:

CRYPT_OK if successful

Definition at line 27 of file hash_memory.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_MEM, CRYPT_OK, ltc_hash_descriptor::done, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, XFREE, XMALLOC, and zeromem().

Referenced by dsa_decrypt_key(), dsa_encrypt_key(), ecc_decrypt_key(), ecc_encrypt_key(), hmac_init(), pkcs_1_oaep_encode(), and pkcs_5_alg1().

```
{
    hash_state *md;
    int err;

    LTC_ARGCHK(in     != NULL);
    LTC_ARGCHK(out    != NULL);
    LTC_ARGCHK(outlen != NULL);

    if ((err = hash_is_valid(hash)) != CRYPT_OK) {
```

```
        return err;
    }

    if (*outlen < hash_descriptor[hash].hashsize) {
       *outlen = hash_descriptor[hash].hashsize;
       return CRYPT_BUFFER_OVERFLOW;
    }

    md = XMALLOC(sizeof(hash_state));
    if (md == NULL) {
       return CRYPT_MEM;
    }

    if ((err = hash_descriptor[hash].init(md)) != CRYPT_OK) {
       goto LBL_ERR;
    }
    if ((err = hash_descriptor[hash].process(md, in, inlen)) != CRYPT_OK) {
       goto LBL_ERR;
    }
    err = hash_descriptor[hash].done(md, out);
    *outlen = hash_descriptor[hash].hashsize;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
    XFREE(md);

    return err;
}
```

Here is the call graph for this function:

5.56 hashes/helper/hash_memory_multi.c File Reference

5.56.1 Detailed Description

Hash (multiple buffers) memory helper, Tom St Denis.

Definition in file hash_memory_multi.c.

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for hash_memory_multi.c:

Functions

- int hash_memory_multi (int hash, unsigned char ∗out, unsigned long ∗outlen, const unsigned char ∗in, unsigned long inlen,...)

  Hash multiple (non-adjacent) blocks of memory at once.

5.56.2 Function Documentation

5.56.2.1 int hash_memory_multi (int *hash*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, const unsigned char ∗ *in*, unsigned long *inlen*, ...)

Hash multiple (non-adjacent) blocks of memory at once.

Parameters:

hash The index of the hash you wish to use

out [out] Where to store the digest

outlen [in/out] Max size and resulting size of the digest

in The data you wish to hash

inlen The length of the data to hash (octets)

... tuples of (data,len) pairs to hash, terminated with a (NULL,x) (x=don't care)

Returns:

CRYPT_OK if successful

Definition at line 28 of file hash_memory_multi.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_MEM, CRYPT_OK, hash_descriptor, hash_is_-valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, and XMALLOC.

```
{
    hash_state          *md;
    int                  err;
    va_list              args;
    const unsigned char *curptr;
    unsigned long        curlen;

    LTC_ARGCHK(in     != NULL);
    LTC_ARGCHK(out    != NULL);
    LTC_ARGCHK(outlen != NULL);
```

```

   if ((err = hash_is_valid(hash)) != CRYPT_OK) {
       return err;
   }

   if (*outlen < hash_descriptor[hash].hashsize) {
      *outlen = hash_descriptor[hash].hashsize;
      return CRYPT_BUFFER_OVERFLOW;
   }

   md = XMALLOC(sizeof(hash_state));
   if (md == NULL) {
      return CRYPT_MEM;
   }

   if ((err = hash_descriptor[hash].init(md)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   va_start(args, inlen);
   curptr = in;
   curlen = inlen;
   for (;;) {
      /* process buf */
      if ((err = hash_descriptor[hash].process(md, curptr, curlen)) != CRYPT_OK) {
         goto LBL_ERR;
      }
      /* step to next */
      curptr = va_arg(args, const unsigned char*);
      if (curptr == NULL) {
         break;
      }
      curlen = va_arg(args, unsigned long);
   }
   err = hash_descriptor[hash].done(md, out);
   *outlen = hash_descriptor[hash].hashsize;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(md, sizeof(hash_state));
#endif
   XFREE(md);
   va_end(args);
   return err;
}
```

Here is the call graph for this function:

5.57 hashes/md2.c File Reference

`#include "tomcrypt.h"`

Include dependency graph for md2.c:

Functions

- static void md2_update_chksum (hash_state *md)
- static void md2_compress (hash_state *md)
- int md2_init (hash_state *md)

  Initialize the hash state.

- int md2_process (hash_state *md, const unsigned char *in, unsigned long inlen)

  Process a block of memory though the hash.

- int md2_done (hash_state *md, unsigned char *out)

  Terminate the hash to get the digest.

- int md2_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor md2_desc
- static const unsigned char PI_SUBST [256]

5.57.1 Function Documentation

5.57.1.1 static void md2_compress (hash_state * *md*) [static]

Definition at line 74 of file md2.c.

Referenced by md2_done(), and md2_process().

```
{
   int j, k;
   unsigned char t;

   /* copy block */
   for (j = 0; j < 16; j++) {
       md->md2.X[16+j] = md->md2.buf[j];
       md->md2.X[32+j] = md->md2.X[j] ^ md->md2.X[16+j];
   }

   t = (unsigned char)0;

   /* do 18 rounds */
   for (j = 0; j < 18; j++) {
       for (k = 0; k < 48; k++) {
           t = (md->md2.X[k] ^= PI_SUBST[(int)(t & 255)]);
       }
       t = (t + (unsigned char)j) & 255;
   }
}
```

5.57.1.2 int md2_done (hash_state ∗ *md*, unsigned char ∗ *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (16 bytes)

Returns:

CRYPT_OK if successful

Definition at line 151 of file md2.c.

References CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, md2_compress(), md2_update_-chksum(), XMEMCPY, and zeromem().

Referenced by md2_test().

```
{
    unsigned long i, k;

    LTC_ARGCHK(md  != NULL);
    LTC_ARGCHK(out != NULL);

    if (md->md2.curlen >= sizeof(md->md2.buf)) {
       return CRYPT_INVALID_ARG;
    }


    /* pad the message */
    k = 16 - md->md2.curlen;
    for (i = md->md2.curlen; i < 16; i++) {
        md->md2.buf[i] = (unsigned char)k;
    }

    /* hash and update */
    md2_compress(md);
    md2_update_chksum(md);

    /* hash checksum */
    XMEMCPY(md->md2.buf, md->md2.chksum, 16);
    md2_compress(md);

    /* output is lower 16 bytes of X */
    XMEMCPY(out, md->md2.X, 16);

#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.57.1.3 int md2_init (hash_state ∗ *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:
CRYPT_OK if successful

Definition at line 101 of file md2.c.

References CRYPT_OK, LTC_ARGCHK, and zeromem().

Referenced by md2_test().

```
{
   LTC_ARGCHK(md != NULL);

   /* MD2 uses a zero'ed state... */
   zeromem(md->md2.X, sizeof(md->md2.X));
   zeromem(md->md2.chksum, sizeof(md->md2.chksum));
   zeromem(md->md2.buf, sizeof(md->md2.buf));
   md->md2.curlen = 0;
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.57.1.4 int md2_process (hash_state * *md*, const unsigned char * *in*, unsigned long *inlen*)

Process a block of memory though the hash.

Parameters:
md The hash state
in The data to hash
inlen The length of the data (octets)

Returns:
CRYPT_OK if successful

Definition at line 120 of file md2.c.

References CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, md2_compress(), md2_update_-chksum(), MIN, and XMEMCPY.

Referenced by md2_test().

```
{
    unsigned long n;
    LTC_ARGCHK(md != NULL);
    LTC_ARGCHK(in != NULL);
    if (md-> md2 .curlen > sizeof(md-> md2 .buf)) {
       return CRYPT_INVALID_ARG;
    }
    while (inlen > 0) {
        n = MIN(inlen, (16 - md->md2.curlen));
        XMEMCPY(md->md2.buf + md->md2.curlen, in, (size_t)n);
        md->md2.curlen += n;
        in             += n;
        inlen          -= n;

        /* is 16 bytes full? */
        if (md->md2.curlen == 16) {
            md2_compress(md);
            md2_update_chksum(md);
```

```
139             md->md2.curlen = 0;
140          }
141      }
142      return CRYPT_OK;
143 }
```

Here is the call graph for this function:

5.57.1.5 int md2_test (void)

Self-test the hash.

Returns:
CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 190 of file md2.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, md2_done(), md2_init(), md2_process(), and XMEMCMP.

```
191 {
192  #ifndef LTC_TEST
193     return CRYPT_NOP;
194  #else
195    static const struct {
196         char *msg;
197         unsigned char md[16];
198    } tests[] = {
199       { "",
200         {0x83,0x50,0xe5,0xa3,0xe2,0x4c,0x15,0x3d,
201          0xf2,0x27,0x5c,0x9f,0x80,0x69,0x27,0x73
202         }
203       },
204       { "a",
205         {0x32,0xec,0x01,0xec,0x4a,0x6d,0xac,0x72,
206          0xc0,0xab,0x96,0xfb,0x34,0xc0,0xb5,0xd1
207         }
208       },
209       { "message digest",
210         {0xab,0x4f,0x49,0x6b,0xfb,0x2a,0x53,0x0b,
211          0x21,0x9f,0xf3,0x30,0x31,0xfe,0x06,0xb0
212         }
213       },
214       { "abcdefghijklmnopqrstuvwxyz",
215         {0x4e,0x8d,0xdf,0xf3,0x65,0x02,0x92,0xab,
216          0x5a,0x41,0x08,0xc3,0xaa,0x47,0x94,0x0b
217         }
218       },
219       { "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
220         {0xda,0x33,0xde,0xf2,0xa4,0x2d,0xf1,0x39,
221          0x75,0x35,0x28,0x46,0xc3,0x03,0x38,0xcd
222         }
223       },
224       { "12345678901234567890123456789012345678901234567890123456789012345678901234567890",
225         {0xd5,0x97,0x6f,0x79,0xd8,0x3d,0x3a,0x0d,
226          0xc9,0x80,0x6c,0x3c,0x66,0xf3,0xef,0xd8
227         }
228       }
229    };
230    int i;
231    hash_state md;
232    unsigned char buf[16];
233
```

```
    for (i = 0; i < (int)(sizeof(tests) / sizeof(tests[0])); i++) {
        md2_init(&md);
        md2_process(&md, (unsigned char*)tests[i].msg, (unsigned long)strlen(tests[i].msg));
        md2_done(&md, buf);
        if (XMEMCMP(buf, tests[i].md, 16) != 0) {
           return CRYPT_FAIL_TESTVECTOR;
        }
    }
    return CRYPT_OK;
   #endif
}
```

Here is the call graph for this function:

5.57.1.6 static void md2_update_chksum (hash_state ∗ *md*) [static]

Definition at line 60 of file md2.c.

References PI_SUBST.

Referenced by md2_done(), and md2_process().

```
{
   int j;
   unsigned char L;
   L = md->md2.chksum[15];
   for (j = 0; j < 16; j++) {

/* caution, the RFC says its "C[j] = S[M[i*16+j] xor L]" but the reference source code [and test vector
   otherwise.
*/
       L = (md->md2.chksum[j] ^= PI_SUBST[(int)(md->md2.buf[j] ^ L)] & 255);
   }
}
```

5.57.2 Variable Documentation

5.57.2.1 const struct ltc_hash_descriptor md2_desc

Initial value:

```
{
    "md2",
    7,
    16,
    16,


   { 1, 2, 840, 113549, 2, 2,  },
   6,

    &md2_init,
    &md2_process,
    &md2_done,
    &md2_test,
    NULL
}
```

Parameters:

md2.c MD2 (RFC 1319) hash function implementation by Tom St Denis

Definition at line 20 of file md2.c.

Referenced by yarrow_start().

5.57.2.2 const unsigned char PI_SUBST[256] [static]

Initial value:

```
 {
  41, 46, 67, 201, 162, 216, 124, 1, 61, 54, 84, 161, 236, 240, 6,
  19, 98, 167, 5, 243, 192, 199, 115, 140, 152, 147, 43, 217, 188,
  76, 130, 202, 30, 155, 87, 60, 253, 212, 224, 22, 103, 66, 111, 24,
  138, 23, 229, 18, 190, 78, 196, 214, 218, 158, 222, 73, 160, 251,
  245, 142, 187, 47, 238, 122, 169, 104, 121, 145, 21, 178, 7, 63,
  148, 194, 16, 137, 11, 34, 95, 33, 128, 127, 93, 154, 90, 144, 50,
  39, 53, 62, 204, 231, 191, 247, 151, 3, 255, 25, 48, 179, 72, 165,
  181, 209, 215, 94, 146, 42, 172, 86, 170, 198, 79, 184, 56, 210,
  150, 164, 125, 182, 118, 252, 107, 226, 156, 116, 4, 241, 69, 157,
  112, 89, 100, 113, 135, 32, 134, 91, 207, 101, 230, 45, 168, 2, 27,
  96, 37, 173, 174, 176, 185, 246, 28, 70, 97, 105, 52, 64, 126, 15,
  85, 71, 163, 35, 221, 81, 175, 58, 195, 92, 249, 206, 186, 197,
  234, 38, 44, 83, 13, 110, 133, 40, 132, 9, 211, 223, 205, 244, 65,
  129, 77, 82, 106, 220, 55, 200, 108, 193, 171, 250, 36, 225, 123,
  8, 12, 189, 177, 74, 120, 136, 149, 139, 227, 99, 232, 109, 233,
  203, 213, 254, 59, 0, 29, 57, 242, 239, 183, 14, 102, 88, 208, 228,
  166, 119, 114, 248, 235, 117, 75, 10, 49, 68, 80, 180, 143, 237,
  31, 26, 219, 153, 141, 51, 159, 17, 131, 20
}
```

Definition at line 38 of file md2.c.

Referenced by md2_update_chksum().

5.58 hashes/md4.c File Reference

```
#include "tomcrypt.h"
```

Include dependency graph for md4.c:

Defines

- #define S11 3
- #define S12 7
- #define S13 11
- #define S14 19
- #define S21 3
- #define S22 5
- #define S23 9
- #define S24 13
- #define S31 3
- #define S32 9
- #define S33 11
- #define S34 15
- #define F(x, y, z) (z ^ (x & (y ^ z)))
- #define G(x, y, z) ((x & y) | (z & (x | y)))
- #define H(x, y, z) ((x) ^ (y) ^ (z))
- #define ROTATE_LEFT(x, n) ROLc(x, n)
- #define FF(a, b, c, d, x, s)
- #define GG(a, b, c, d, x, s)
- #define HH(a, b, c, d, x, s)

Functions

- static int md4_compress (hash_state ∗md, unsigned char ∗buf)
- int md4_init (hash_state ∗md)

    Initialize the hash state.

- int md4_done (hash_state ∗md, unsigned char ∗out)

    Terminate the hash to get the digest.

- int md4_test (void)

    Self-test the hash.

Variables

- const struct ltc_hash_descriptor md4_desc

5.58.1 Define Documentation

5.58.1.1 #define F(x, y, z) (z ^ (x & (y ^ z)))

Definition at line 52 of file md4.c.

5.58.1.2 #define FF(a, b, c, d, x, s)

Value:

```
{ \
    (a) += F ((b), (c), (d)) + (x); \
    (a) = ROTATE_LEFT ((a), (s)); \
  }
```

Definition at line 62 of file md4.c.

5.58.1.3 #define G(x, y, z) ((x & y) | (z & (x | y)))

Definition at line 53 of file md4.c.

5.58.1.4 #define GG(a, b, c, d, x, s)

Value:

```
{ \
    (a) += G ((b), (c), (d)) + (x) + 0x5a827999UL; \
    (a) = ROTATE_LEFT ((a), (s)); \
  }
```

Definition at line 66 of file md4.c.

Referenced by ecc_test().

5.58.1.5 #define H(x, y, z) ((x) ^ (y) ^ (z))

Definition at line 54 of file md4.c.

5.58.1.6 #define HH(a, b, c, d, x, s)

Value:

```
{ \
    (a) += H ((b), (c), (d)) + (x) + 0x6ed9eba1UL; \
    (a) = ROTATE_LEFT ((a), (s)); \
  }
```

Definition at line 70 of file md4.c.

5.58.1.7 #define ROTATE_LEFT(x, n) ROLc(x, n)

Definition at line 57 of file md4.c.

5.58.1.8 #define S11 3

Definition at line 38 of file md4.c.

5.58.1.9 #define S12 7

Definition at line 39 of file md4.c.

5.58.1.10 #define S13 11

Definition at line 40 of file md4.c.

5.58.1.11 #define S14 19

Definition at line 41 of file md4.c.

5.58.1.12 #define S21 3

Definition at line 42 of file md4.c.

5.58.1.13 #define S22 5

Definition at line 43 of file md4.c.

5.58.1.14 #define S23 9

Definition at line 44 of file md4.c.

5.58.1.15 #define S24 13

Definition at line 45 of file md4.c.

5.58.1.16 #define S31 3

Definition at line 46 of file md4.c.

5.58.1.17 #define S32 9

Definition at line 47 of file md4.c.

5.58.1.18 #define S33 11

Definition at line 48 of file md4.c.

5.58.1.19 #define S34 15

Definition at line 49 of file md4.c.

5.58.2 Function Documentation

5.58.2.1 static int md4_compress (hash_state * *md*, unsigned char * *buf*) [static]

Definition at line 78 of file md4.c.

References c.

Referenced by md4_done().

```
{
    ulong32 x[16], a, b, c, d;
    int i;

    /* copy state */
    a = md->md4.state[0];
    b = md->md4.state[1];
    c = md->md4.state[2];
    d = md->md4.state[3];

    /* copy the state into 512-bits into W[0..15] */
    for (i = 0; i < 16; i++) {
        LOAD32L(x[i], buf + (4*i));
    }

    /* Round 1 */
    FF (a, b, c, d, x[ 0], S11); /* 1 */
    FF (d, a, b, c, x[ 1], S12); /* 2 */
    FF (c, d, a, b, x[ 2], S13); /* 3 */
    FF (b, c, d, a, x[ 3], S14); /* 4 */
    FF (a, b, c, d, x[ 4], S11); /* 5 */
    FF (d, a, b, c, x[ 5], S12); /* 6 */
    FF (c, d, a, b, x[ 6], S13); /* 7 */
    FF (b, c, d, a, x[ 7], S14); /* 8 */
    FF (a, b, c, d, x[ 8], S11); /* 9 */
    FF (d, a, b, c, x[ 9], S12); /* 10 */
    FF (c, d, a, b, x[10], S13); /* 11 */
    FF (b, c, d, a, x[11], S14); /* 12 */
    FF (a, b, c, d, x[12], S11); /* 13 */
    FF (d, a, b, c, x[13], S12); /* 14 */
    FF (c, d, a, b, x[14], S13); /* 15 */
    FF (b, c, d, a, x[15], S14); /* 16 */

    /* Round 2 */
    GG (a, b, c, d, x[ 0], S21); /* 17 */
    GG (d, a, b, c, x[ 4], S22); /* 18 */
    GG (c, d, a, b, x[ 8], S23); /* 19 */
    GG (b, c, d, a, x[12], S24); /* 20 */
    GG (a, b, c, d, x[ 1], S21); /* 21 */
    GG (d, a, b, c, x[ 5], S22); /* 22 */
    GG (c, d, a, b, x[ 9], S23); /* 23 */
    GG (b, c, d, a, x[13], S24); /* 24 */
    GG (a, b, c, d, x[ 2], S21); /* 25 */
    GG (d, a, b, c, x[ 6], S22); /* 26 */
    GG (c, d, a, b, x[10], S23); /* 27 */
    GG (b, c, d, a, x[14], S24); /* 28 */
    GG (a, b, c, d, x[ 3], S21); /* 29 */
    GG (d, a, b, c, x[ 7], S22); /* 30 */
    GG (c, d, a, b, x[11], S23); /* 31 */
    GG (b, c, d, a, x[15], S24); /* 32 */

    /* Round 3 */
    HH (a, b, c, d, x[ 0], S31); /* 33 */
    HH (d, a, b, c, x[ 8], S32); /* 34 */
    HH (c, d, a, b, x[ 4], S33); /* 35 */
    HH (b, c, d, a, x[12], S34); /* 36 */
```

```
    HH (a, b, c, d, x[ 2], S31); /* 37 */
    HH (d, a, b, c, x[10], S32); /* 38 */
    HH (c, d, a, b, x[ 6], S33); /* 39 */
    HH (b, c, d, a, x[14], S34); /* 40 */
    HH (a, b, c, d, x[ 1], S31); /* 41 */
    HH (d, a, b, c, x[ 9], S32); /* 42 */
    HH (c, d, a, b, x[ 5], S33); /* 43 */
    HH (b, c, d, a, x[13], S34); /* 44 */
    HH (a, b, c, d, x[ 3], S31); /* 45 */
    HH (d, a, b, c, x[11], S32); /* 46 */
    HH (c, d, a, b, x[ 7], S33); /* 47 */
    HH (b, c, d, a, x[15], S34); /* 48 */


    /* Update our state */
    md->md4.state[0] = md->md4.state[0] + a;
    md->md4.state[1] = md->md4.state[1] + b;
    md->md4.state[2] = md->md4.state[2] + c;
    md->md4.state[3] = md->md4.state[3] + d;

    return CRYPT_OK;
}
```

5.58.2.2 int md4_done (hash_state ∗ *md*, unsigned char ∗ *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (16 bytes)

Returns:

CRYPT_OK if successful

Definition at line 201 of file md4.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and md4_compress().

Referenced by md4_test().

```
{
    int i;

    LTC_ARGCHK(md  != NULL);
    LTC_ARGCHK(out != NULL);

    if (md->md4.curlen >= sizeof(md->md4.buf)) {
       return CRYPT_INVALID_ARG;
    }

    /* increase the length of the message */
    md->md4.length += md->md4.curlen * 8;

    /* append the '1' bit */
    md->md4.buf[md->md4.curlen++] = (unsigned char)0x80;

    /* if the length is currently above 56 bytes we append zeros
     * then compress.  Then we can fall back to padding zeros and length
     * encoding like normal.
     */
    if (md->md4.curlen > 56) {
```

```
        while (md->md4.curlen < 64) {
            md->md4.buf[md->md4.curlen++] = (unsigned char)0;
        }
        md4_compress(md, md->md4.buf);
        md->md4.curlen = 0;
    }

    /* pad upto 56 bytes of zeroes */
    while (md->md4.curlen < 56) {
        md->md4.buf[md->md4.curlen++] = (unsigned char)0;
    }

    /* store length */
    STORE64L(md->md4.length, md->md4.buf+56);
    md4_compress(md, md->md4.buf);

    /* copy output */
    for (i = 0; i < 4; i++) {
        STORE32L(md->md4.state[i], out+(4*i));
    }
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.58.2.3 int md4_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 174 of file md4.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by md4_test().

```
{
   LTC_ARGCHK(md != NULL);
   md->md4.state[0] = 0x67452301UL;
   md->md4.state[1] = 0xefcdab89UL;
   md->md4.state[2] = 0x98badcfeUL;
   md->md4.state[3] = 0x10325476UL;
   md->md4.length  = 0;
   md->md4.curlen  = 0;
   return CRYPT_OK;
}
```

5.58.2.4 int md4_test (void)

Self-test the hash.

Returns:
CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 253 of file md4.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, md4_done(), md4_init(), and XMEMCMP.

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
    static const struct md4_test_case {
        char *input;
        unsigned char digest[16];
    } cases[] = {
        { "",
          {0x31, 0xd6, 0xcf, 0xe0, 0xd1, 0x6a, 0xe9, 0x31,
           0xb7, 0x3c, 0x59, 0xd7, 0xe0, 0xc0, 0x89, 0xc0} },
        { "a",
          {0xbd, 0xe5, 0x2c, 0xb3, 0x1d, 0xe3, 0x3e, 0x46,
           0x24, 0x5e, 0x05, 0xfb, 0xdb, 0xd6, 0xfb, 0x24} },
        { "abc",
          {0xa4, 0x48, 0x01, 0x7a, 0xaf, 0x21, 0xd8, 0x52,
           0x5f, 0xc1, 0x0a, 0xe8, 0x7a, 0xa6, 0x72, 0x9d} },
        { "message digest",
          {0xd9, 0x13, 0x0a, 0x81, 0x64, 0x54, 0x9f, 0xe8,
           0x18, 0x87, 0x48, 0x06, 0xe1, 0xc7, 0x01, 0x4b} },
        { "abcdefghijklmnopqrstuvwxyz",
          {0xd7, 0x9e, 0x1c, 0x30, 0x8a, 0xa5, 0xbb, 0xcd,
           0xee, 0xa8, 0xed, 0x63, 0xdf, 0x41, 0x2d, 0xa9} },
        { "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
          {0x04, 0x3f, 0x85, 0x82, 0xf2, 0x41, 0xdb, 0x35,
           0x1c, 0xe6, 0x27, 0xe1, 0x53, 0xe7, 0xf0, 0xe4} },
        { "12345678901234567890123456789012345678901234567890123456789012345678901234567890",
          {0xe3, 0x3b, 0x4d, 0xdc, 0x9c, 0x38, 0xf2, 0x19,
           0x9c, 0x3e, 0x7b, 0x16, 0x4f, 0xcc, 0x05, 0x36} },
    };
    int i;
    hash_state md;
    unsigned char digest[16];

    for(i = 0; i < (int)(sizeof(cases) / sizeof(cases[0])); i++) {
        md4_init(&md);
        md4_process(&md, (unsigned char *)cases[i].input, (unsigned long)strlen(cases[i].input));
        md4_done(&md, digest);
        if (XMEMCMP(digest, cases[i].digest, 16) != 0) {
           return CRYPT_FAIL_TESTVECTOR;
        }

    }
    return CRYPT_OK;
  #endif
}
```

Here is the call graph for this function:

5.58.3 Variable Documentation

5.58.3.1 const struct ltc_hash_descriptor md4_desc

Initial value:

```
{
```

```
    "md4",
    6,
    16,
    64,


   { 1, 2, 840, 113549, 2, 4,  },
   6,

    &md4_init,
    &md4_process,
    &md4_done,
    &md4_test,
    NULL
}
```

Parameters:

md4.c Submitted by Dobes Vandermeer (dobes@smartt.com)

Definition at line 20 of file md4.c.

Referenced by yarrow_start().

5.59 hashes/md5.c File Reference

5.59.1 Detailed Description

MD5 hash function by Tom St Denis.

Definition in file md5.c.

```
#include "tomcrypt.h"
```

Include dependency graph for md5.c:

Defines

- #define F(x, y, z) (z ^ (x & (y ^ z)))
- #define G(x, y, z) (y ^ (z & (y ^ x)))
- #define H(x, y, z) (x^y^z)
- #define I(x, y, z) (y^(x|(∼z)))
- #define FF(a, b, c, d, M, s, t) a = (a + F(b,c,d) + M + t); a = ROLc(a, s) + b;
- #define GG(a, b, c, d, M, s, t) a = (a + G(b,c,d) + M + t); a = ROLc(a, s) + b;
- #define HH(a, b, c, d, M, s, t) a = (a + H(b,c,d) + M + t); a = ROLc(a, s) + b;
- #define II(a, b, c, d, M, s, t) a = (a + I(b,c,d) + M + t); a = ROLc(a, s) + b;

Functions

- static int md5_compress (hash_state ∗md, unsigned char ∗buf)
- int md5_init (hash_state ∗md)

  Initialize the hash state.

- int md5_done (hash_state ∗md, unsigned char ∗out)

  Terminate the hash to get the digest.

- int md5_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor md5_desc

5.59.2 Define Documentation

5.59.2.1 #define F(x, y, z) (z ^ (x & (y ^ z)))

Definition at line 39 of file md5.c.

5.59.2.2 #define FF(a, b, c, d, M, s, t) a = (a + F(b,c,d) + M + t); a = ROLc(a, s) + b;

Definition at line 85 of file md5.c.

5.59.2.3 #define G(x, y, z) (y ^ (z & (y ^ x)))

Definition at line 40 of file md5.c.

5.59.2.4 #define GG(a, b, c, d, M, s, t) a = (a + G(b,c,d) + M + t); a = ROLc(a, s) + b;

Definition at line 88 of file md5.c.

5.59.2.5 #define H(x, y, z) (x^y^z)

Definition at line 41 of file md5.c.

5.59.2.6 #define HH(a, b, c, d, M, s, t) a = (a + H(b,c,d) + M + t); a = ROLc(a, s) + b;

Definition at line 91 of file md5.c.

5.59.2.7 #define I(x, y, z) (y^(x|(~z)))

Definition at line 42 of file md5.c.

Referenced by FI(), FII(), and FIII().

5.59.2.8 #define II(a, b, c, d, M, s, t) a = (a + I(b,c,d) + M + t); a = ROLc(a, s) + b;

Definition at line 94 of file md5.c.

5.59.3 Function Documentation

5.59.3.1 static int md5_compress (hash_state * *md*, unsigned char * *buf*) [static]

Definition at line 103 of file md5.c.

References c.

Referenced by md5_done().

```
{
    ulong32 i, W[16], a, b, c, d;
#ifdef LTC_SMALL_CODE
    ulong32 t;
#endif

    /* copy the state into 512-bits into W[0..15] */
    for (i = 0; i < 16; i++) {
        LOAD32L(W[i], buf + (4*i));
    }

    /* copy state */
    a = md->md5.state[0];
    b = md->md5.state[1];
    c = md->md5.state[2];
    d = md->md5.state[3];

#ifdef LTC_SMALL_CODE
```

```
    for (i = 0; i < 16; ++i) {
        FF(a,b,c,d,W[Worder[i]],Rorder[i],Korder[i]);
        t = d; d = c; c = b; b = a; a = t;
    }

    for (; i < 32; ++i) {
        GG(a,b,c,d,W[Worder[i]],Rorder[i],Korder[i]);
        t = d; d = c; c = b; b = a; a = t;
    }

    for (; i < 48; ++i) {
        HH(a,b,c,d,W[Worder[i]],Rorder[i],Korder[i]);
        t = d; d = c; c = b; b = a; a = t;
    }

    for (; i < 64; ++i) {
        II(a,b,c,d,W[Worder[i]],Rorder[i],Korder[i]);
        t = d; d = c; c = b; b = a; a = t;
    }

#else
    FF(a,b,c,d,W[0],7,0xd76aa478UL)
    FF(d,a,b,c,W[1],12,0xe8c7b756UL)
    FF(c,d,a,b,W[2],17,0x242070dbUL)
    FF(b,c,d,a,W[3],22,0xc1bdceeeUL)
    FF(a,b,c,d,W[4],7,0xf57c0fafUL)
    FF(d,a,b,c,W[5],12,0x4787c62aUL)
    FF(c,d,a,b,W[6],17,0xa8304613UL)
    FF(b,c,d,a,W[7],22,0xfd469501UL)
    FF(a,b,c,d,W[8],7,0x698098d8UL)
    FF(d,a,b,c,W[9],12,0x8b44f7afUL)
    FF(c,d,a,b,W[10],17,0xffff5bb1UL)
    FF(b,c,d,a,W[11],22,0x895cd7beUL)
    FF(a,b,c,d,W[12],7,0x6b901122UL)
    FF(d,a,b,c,W[13],12,0xfd987193UL)
    FF(c,d,a,b,W[14],17,0xa679438eUL)
    FF(b,c,d,a,W[15],22,0x49b40821UL)
    GG(a,b,c,d,W[1],5,0xf61e2562UL)
    GG(d,a,b,c,W[6],9,0xc040b340UL)
    GG(c,d,a,b,W[11],14,0x265e5a51UL)
    GG(b,c,d,a,W[0],20,0xe9b6c7aaUL)
    GG(a,b,c,d,W[5],5,0xd62f105dUL)
    GG(d,a,b,c,W[10],9,0x02441453UL)
    GG(c,d,a,b,W[15],14,0xd8a1e681UL)
    GG(b,c,d,a,W[4],20,0xe7d3fbc8UL)
    GG(a,b,c,d,W[9],5,0x21e1cde6UL)
    GG(d,a,b,c,W[14],9,0xc33707d6UL)
    GG(c,d,a,b,W[3],14,0xf4d50d87UL)
    GG(b,c,d,a,W[8],20,0x455a14edUL)
    GG(a,b,c,d,W[13],5,0xa9e3e905UL)
    GG(d,a,b,c,W[2],9,0xfcefa3f8UL)
    GG(c,d,a,b,W[7],14,0x676f02d9UL)
    GG(b,c,d,a,W[12],20,0x8d2a4c8aUL)
    HH(a,b,c,d,W[5],4,0xfffa3942UL)
    HH(d,a,b,c,W[8],11,0x8771f681UL)
    HH(c,d,a,b,W[11],16,0x6d9d6122UL)
    HH(b,c,d,a,W[14],23,0xfde5380cUL)
    HH(a,b,c,d,W[1],4,0xa4beea44UL)
    HH(d,a,b,c,W[4],11,0x4bdecfa9UL)
    HH(c,d,a,b,W[7],16,0xf6bb4b60UL)
    HH(b,c,d,a,W[10],23,0xbebfbc70UL)
    HH(a,b,c,d,W[13],4,0x289b7ec6UL)
    HH(d,a,b,c,W[0],11,0xeaa127faUL)
    HH(c,d,a,b,W[3],16,0xd4ef3085UL)
    HH(b,c,d,a,W[6],23,0x04881d05UL)
    HH(a,b,c,d,W[9],4,0xd9d4d039UL)
    HH(d,a,b,c,W[12],11,0xe6db99e5UL)
```

```
190     HH(c,d,a,b,W[15],16,0x1fa27cf8UL)
191     HH(b,c,d,a,W[2],23,0xc4ac5665UL)
192     II(a,b,c,d,W[0],6,0xf4292244UL)
193     II(d,a,b,c,W[7],10,0x432aff97UL)
194     II(c,d,a,b,W[14],15,0xab9423a7UL)
195     II(b,c,d,a,W[5],21,0xfc93a039UL)
196     II(a,b,c,d,W[12],6,0x655b59c3UL)
197     II(d,a,b,c,W[3],10,0x8f0ccc92UL)
198     II(c,d,a,b,W[10],15,0xffeff47dUL)
199     II(b,c,d,a,W[1],21,0x85845dd1UL)
200     II(a,b,c,d,W[8],6,0x6fa87e4fUL)
201     II(d,a,b,c,W[15],10,0xfe2ce6e0UL)
202     II(c,d,a,b,W[6],15,0xa3014314UL)
203     II(b,c,d,a,W[13],21,0x4e0811a1UL)
204     II(a,b,c,d,W[4],6,0xf7537e82UL)
205     II(d,a,b,c,W[11],10,0xbd3af235UL)
206     II(c,d,a,b,W[2],15,0x2ad7d2bbUL)
207     II(b,c,d,a,W[9],21,0xeb86d391UL)
208 #endif
209
210     md->md5.state[0] = md->md5.state[0] + a;
211     md->md5.state[1] = md->md5.state[1] + b;
212     md->md5.state[2] = md->md5.state[2] + c;
213     md->md5.state[3] = md->md5.state[3] + d;
214
215     return CRYPT_OK;
216 }
```

5.59.3.2 int md5_done (hash_state * *md*, unsigned char * *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (16 bytes)

Returns:

CRYPT_OK if successful

Definition at line 260 of file md5.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and md5_compress().

Referenced by md5_test().

```
261 {
262     int i;
263
264     LTC_ARGCHK(md  != NULL);
265     LTC_ARGCHK(out != NULL);
266
267     if (md->md5.curlen >= sizeof(md->md5.buf)) {
268        return CRYPT_INVALID_ARG;
269     }
270
271
272     /* increase the length of the message */
273     md->md5.length += md->md5.curlen * 8;
274
275     /* append the '1' bit */
276     md->md5.buf[md->md5.curlen++] = (unsigned char)0x80;
```

```

    /* if the length is currently above 56 bytes we append zeros
     * then compress.  Then we can fall back to padding zeros and length
     * encoding like normal.
     */
    if (md->md5.curlen > 56) {
        while (md->md5.curlen < 64) {
            md->md5.buf[md->md5.curlen++] = (unsigned char)0;
        }
        md5_compress(md, md->md5.buf);
        md->md5.curlen = 0;
    }

    /* pad upto 56 bytes of zeroes */
    while (md->md5.curlen < 56) {
        md->md5.buf[md->md5.curlen++] = (unsigned char)0;
    }

    /* store length */
    STORE64L(md->md5.length, md->md5.buf+56);
    md5_compress(md, md->md5.buf);

    /* copy output */
    for (i = 0; i < 4; i++) {
        STORE32L(md->md5.state[i], out+(4*i));
    }
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.59.3.3 int md5_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 233 of file md5.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by md5_test().

```
{
   LTC_ARGCHK(md != NULL);
   md->md5.state[0] = 0x67452301UL;
   md->md5.state[1] = 0xefcdab89UL;
   md->md5.state[2] = 0x98badcfeUL;
   md->md5.state[3] = 0x10325476UL;
   md->md5.curlen = 0;
   md->md5.length = 0;
   return CRYPT_OK;
}
```

5.59.3.4 int md5_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 313 of file md5.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, md5_done(), md5_init(), and XMEMCMP.

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
  static const struct {
      char *msg;
      unsigned char hash[16];
  } tests[] = {
    { "",
      { 0xd4, 0x1d, 0x8c, 0xd9, 0x8f, 0x00, 0xb2, 0x04,
        0xe9, 0x80, 0x09, 0x98, 0xec, 0xf8, 0x42, 0x7e } },
    { "a",
      {0x0c, 0xc1, 0x75, 0xb9, 0xc0, 0xf1, 0xb6, 0xa8,
       0x31, 0xc3, 0x99, 0xe2, 0x69, 0x77, 0x26, 0x61 } },
    { "abc",
      { 0x90, 0x01, 0x50, 0x98, 0x3c, 0xd2, 0x4f, 0xb0,
        0xd6, 0x96, 0x3f, 0x7d, 0x28, 0xe1, 0x7f, 0x72 } },
    { "message digest",
      { 0xf9, 0x6b, 0x69, 0x7d, 0x7c, 0xb7, 0x93, 0x8d,
        0x52, 0x5a, 0x2f, 0x31, 0xaa, 0xf1, 0x61, 0xd0 } },
    { "abcdefghijklmnopqrstuvwxyz",
      { 0xc3, 0xfc, 0xd3, 0xd7, 0x61, 0x92, 0xe4, 0x00,
        0x7d, 0xfb, 0x49, 0x6c, 0xca, 0x67, 0xe1, 0x3b } },
    { "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
      { 0xd1, 0x74, 0xab, 0x98, 0xd2, 0x77, 0xd9, 0xf5,
        0xa5, 0x61, 0x1c, 0x2c, 0x9f, 0x41, 0x9d, 0x9f } },
    { "12345678901234567890123456789012345678901234567890123456789012345678901234567890",
      { 0x57, 0xed, 0xf4, 0xa2, 0x2b, 0xe3, 0xc9, 0x55,
        0xac, 0x49, 0xda, 0x2e, 0x21, 0x07, 0xb6, 0x7a } },
    { NULL, { 0 } }
  };

  int i;
  unsigned char tmp[16];
  hash_state md;

  for (i = 0; tests[i].msg != NULL; i++) {
      md5_init(&md);
      md5_process(&md, (unsigned char *)tests[i].msg, (unsigned long)strlen(tests[i].msg));
      md5_done(&md, tmp);
      if (XMEMCMP(tmp, tests[i].hash, 16) != 0) {
         return CRYPT_FAIL_TESTVECTOR;
      }
  }
  return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.59.4 Variable Documentation

5.59.4.1 const struct ltc_hash_descriptor md5_desc

Initial value:

```
{
    "md5",
    3,
    16,
    64,


   { 1, 2, 840, 113549, 2, 5,  },
   6,

    &md5_init,
    &md5_process,
    &md5_done,
    &md5_test,
    NULL
}
```

Definition at line 21 of file md5.c.

Referenced by yarrow_start().

5.60 hashes/rmd128.c File Reference

`#include "tomcrypt.h"`

Include dependency graph for rmd128.c:

Defines

- #define F(x, y, z) ((x) ^ (y) ^ (z))
- #define G(x, y, z) (((x) & (y)) | (∼(x) & (z)))
- #define H(x, y, z) (((x) | ∼(y)) ^ (z))
- #define I(x, y, z) (((x) & (z)) | ((y) & ∼(z)))
- #define FF(a, b, c, d, x, s)
- #define GG(a, b, c, d, x, s)
- #define HH(a, b, c, d, x, s)
- #define II(a, b, c, d, x, s)
- #define FFF(a, b, c, d, x, s)
- #define GGG(a, b, c, d, x, s)
- #define HHH(a, b, c, d, x, s)
- #define III(a, b, c, d, x, s)

Functions

- static int rmd128_compress (hash_state ∗md, unsigned char ∗buf)
- int rmd128_init (hash_state ∗md)

  Initialize the hash state.

- int rmd128_done (hash_state ∗md, unsigned char ∗out)

  Terminate the hash to get the digest.

- int rmd128_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor rmd128_desc

5.60.1 Define Documentation

5.60.1.1 #define F(x, y, z) ((x) ^ (y) ^ (z))

Definition at line 45 of file rmd128.c.

5.60.1.2 #define FF(a, b, c, d, x, s)

Value:

```
(a) += F((b), (c), (d)) + (x);\
      (a) = ROLc((a), (s));
```

Definition at line 51 of file rmd128.c.

5.60.1.3 #define FFF(a, b, c, d, x, s)

Value:

```
(a) += F((b), (c), (d)) + (x);\
      (a) = ROLc((a), (s));
```

Definition at line 67 of file rmd128.c.

5.60.1.4 #define G(x, y, z) (((x) & (y)) | (∼(x) & (z)))

Definition at line 46 of file rmd128.c.

5.60.1.5 #define GG(a, b, c, d, x, s)

Value:

```
(a) += G((b), (c), (d)) + (x) + 0x5a827999UL;\
      (a) = ROLc((a), (s));
```

Definition at line 55 of file rmd128.c.

5.60.1.6 #define GGG(a, b, c, d, x, s)

Value:

```
(a) += G((b), (c), (d)) + (x) + 0x6d703ef3UL;\
      (a) = ROLc((a), (s));
```

Definition at line 71 of file rmd128.c.

5.60.1.7 #define H(x, y, z) (((x) | ∼(y)) ^ (z))

Definition at line 47 of file rmd128.c.

5.60.1.8 #define HH(a, b, c, d, x, s)

Value:

```
(a) += H((b), (c), (d)) + (x) + 0x6ed9eba1UL;\
      (a) = ROLc((a), (s));
```

Definition at line 59 of file rmd128.c.

5.60.1.9 #define HHH(a, b, c, d, x, s)

Value:

```
(a) += H((b), (c), (d)) + (x) + 0x5c4dd124UL;\
      (a) = ROLc((a), (s));
```

Definition at line 75 of file rmd128.c.

5.60.1.10 #define I(x, y, z) (((x) & (z)) | ((y) & ∼(z)))

Definition at line 48 of file rmd128.c.

5.60.1.11 #define II(a, b, c, d, x, s)

Value:

```
(a) += I((b), (c), (d)) + (x) + 0x8f1bbcdcUL;\
      (a) = ROLc((a), (s));
```

Definition at line 63 of file rmd128.c.

5.60.1.12 #define III(a, b, c, d, x, s)

Value:

```
(a) += I((b), (c), (d)) + (x) + 0x50a28be6UL;\
      (a) = ROLc((a), (s));
```

Definition at line 79 of file rmd128.c.

5.60.2 Function Documentation

5.60.2.1 static int rmd128_compress (hash_state ∗ *md*, unsigned char ∗ *buf*) [static]

Definition at line 86 of file rmd128.c.

Referenced by rmd128_done().

```
{
   ulong32 aa,bb,cc,dd,aaa,bbb,ccc,ddd,X[16];
   int i;

   /* load words X */
   for (i = 0; i < 16; i++){
      LOAD32L(X[i], buf + (4 * i));
   }

   /* load state */
   aa = aaa = md->rmd128.state[0];
   bb = bbb = md->rmd128.state[1];
   cc = ccc = md->rmd128.state[2];
   dd = ddd = md->rmd128.state[3];

```

```
   /* round 1 */
   FF(aa, bb, cc, dd, X[ 0], 11);
   FF(dd, aa, bb, cc, X[ 1], 14);
   FF(cc, dd, aa, bb, X[ 2], 15);
   FF(bb, cc, dd, aa, X[ 3], 12);
   FF(aa, bb, cc, dd, X[ 4],  5);
   FF(dd, aa, bb, cc, X[ 5],  8);
   FF(cc, dd, aa, bb, X[ 6],  7);
   FF(bb, cc, dd, aa, X[ 7],  9);
   FF(aa, bb, cc, dd, X[ 8], 11);
   FF(dd, aa, bb, cc, X[ 9], 13);
   FF(cc, dd, aa, bb, X[10], 14);
   FF(bb, cc, dd, aa, X[11], 15);
   FF(aa, bb, cc, dd, X[12],  6);
   FF(dd, aa, bb, cc, X[13],  7);
   FF(cc, dd, aa, bb, X[14],  9);
   FF(bb, cc, dd, aa, X[15],  8);

   /* round 2 */
   GG(aa, bb, cc, dd, X[ 7],  7);
   GG(dd, aa, bb, cc, X[ 4],  6);
   GG(cc, dd, aa, bb, X[13],  8);
   GG(bb, cc, dd, aa, X[ 1], 13);
   GG(aa, bb, cc, dd, X[10], 11);
   GG(dd, aa, bb, cc, X[ 6],  9);
   GG(cc, dd, aa, bb, X[15],  7);
   GG(bb, cc, dd, aa, X[ 3], 15);
   GG(aa, bb, cc, dd, X[12],  7);
   GG(dd, aa, bb, cc, X[ 0], 12);
   GG(cc, dd, aa, bb, X[ 9], 15);
   GG(bb, cc, dd, aa, X[ 5],  9);
   GG(aa, bb, cc, dd, X[ 2], 11);
   GG(dd, aa, bb, cc, X[14],  7);
   GG(cc, dd, aa, bb, X[11], 13);
   GG(bb, cc, dd, aa, X[ 8], 12);

   /* round 3 */
   HH(aa, bb, cc, dd, X[ 3], 11);
   HH(dd, aa, bb, cc, X[10], 13);
   HH(cc, dd, aa, bb, X[14],  6);
   HH(bb, cc, dd, aa, X[ 4],  7);
   HH(aa, bb, cc, dd, X[ 9], 14);
   HH(dd, aa, bb, cc, X[15],  9);
   HH(cc, dd, aa, bb, X[ 8], 13);
   HH(bb, cc, dd, aa, X[ 1], 15);
   HH(aa, bb, cc, dd, X[ 2], 14);
   HH(dd, aa, bb, cc, X[ 7],  8);
   HH(cc, dd, aa, bb, X[ 0], 13);
   HH(bb, cc, dd, aa, X[ 6],  6);
   HH(aa, bb, cc, dd, X[13],  5);
   HH(dd, aa, bb, cc, X[11], 12);
   HH(cc, dd, aa, bb, X[ 5],  7);
   HH(bb, cc, dd, aa, X[12],  5);

   /* round 4 */
   II(aa, bb, cc, dd, X[ 1], 11);
   II(dd, aa, bb, cc, X[ 9], 12);
   II(cc, dd, aa, bb, X[11], 14);
   II(bb, cc, dd, aa, X[10], 15);
   II(aa, bb, cc, dd, X[ 0], 14);
   II(dd, aa, bb, cc, X[ 8], 15);
   II(cc, dd, aa, bb, X[12],  9);
   II(bb, cc, dd, aa, X[ 4],  8);
   II(aa, bb, cc, dd, X[13],  9);
   II(dd, aa, bb, cc, X[ 3], 14);
   II(cc, dd, aa, bb, X[ 7],  5);
   II(bb, cc, dd, aa, X[15],  6);
```

```
    II(aa, bb, cc, dd, X[14],  8);
    II(dd, aa, bb, cc, X[ 5],  6);
    II(cc, dd, aa, bb, X[ 6],  5);
    II(bb, cc, dd, aa, X[ 2], 12);

    /* parallel round 1 */
    III(aaa, bbb, ccc, ddd, X[ 5],  8);
    III(ddd, aaa, bbb, ccc, X[14],  9);
    III(ccc, ddd, aaa, bbb, X[ 7],  9);
    III(bbb, ccc, ddd, aaa, X[ 0], 11);
    III(aaa, bbb, ccc, ddd, X[ 9], 13);
    III(ddd, aaa, bbb, ccc, X[ 2], 15);
    III(ccc, ddd, aaa, bbb, X[11], 15);
    III(bbb, ccc, ddd, aaa, X[ 4],  5);
    III(aaa, bbb, ccc, ddd, X[13],  7);
    III(ddd, aaa, bbb, ccc, X[ 6],  7);
    III(ccc, ddd, aaa, bbb, X[15],  8);
    III(bbb, ccc, ddd, aaa, X[ 8], 11);
    III(aaa, bbb, ccc, ddd, X[ 1], 14);
    III(ddd, aaa, bbb, ccc, X[10], 14);
    III(ccc, ddd, aaa, bbb, X[ 3], 12);
    III(bbb, ccc, ddd, aaa, X[12],  6);

    /* parallel round 2 */
    HHH(aaa, bbb, ccc, ddd, X[ 6],  9);
    HHH(ddd, aaa, bbb, ccc, X[11], 13);
    HHH(ccc, ddd, aaa, bbb, X[ 3], 15);
    HHH(bbb, ccc, ddd, aaa, X[ 7],  7);
    HHH(aaa, bbb, ccc, ddd, X[ 0], 12);
    HHH(ddd, aaa, bbb, ccc, X[13],  8);
    HHH(ccc, ddd, aaa, bbb, X[ 5],  9);
    HHH(bbb, ccc, ddd, aaa, X[10], 11);
    HHH(aaa, bbb, ccc, ddd, X[14],  7);
    HHH(ddd, aaa, bbb, ccc, X[15],  7);
    HHH(ccc, ddd, aaa, bbb, X[ 8], 12);
    HHH(bbb, ccc, ddd, aaa, X[12],  7);
    HHH(aaa, bbb, ccc, ddd, X[ 4],  6);
    HHH(ddd, aaa, bbb, ccc, X[ 9], 15);
    HHH(ccc, ddd, aaa, bbb, X[ 1], 13);
    HHH(bbb, ccc, ddd, aaa, X[ 2], 11);

    /* parallel round 3 */
    GGG(aaa, bbb, ccc, ddd, X[15],  9);
    GGG(ddd, aaa, bbb, ccc, X[ 5],  7);
    GGG(ccc, ddd, aaa, bbb, X[ 1], 15);
    GGG(bbb, ccc, ddd, aaa, X[ 3], 11);
    GGG(aaa, bbb, ccc, ddd, X[ 7],  8);
    GGG(ddd, aaa, bbb, ccc, X[14],  6);
    GGG(ccc, ddd, aaa, bbb, X[ 6],  6);
    GGG(bbb, ccc, ddd, aaa, X[ 9], 14);
    GGG(aaa, bbb, ccc, ddd, X[11], 12);
    GGG(ddd, aaa, bbb, ccc, X[ 8], 13);
    GGG(ccc, ddd, aaa, bbb, X[12],  5);
    GGG(bbb, ccc, ddd, aaa, X[ 2], 14);
    GGG(aaa, bbb, ccc, ddd, X[10], 13);
    GGG(ddd, aaa, bbb, ccc, X[ 0], 13);
    GGG(ccc, ddd, aaa, bbb, X[ 4],  7);
    GGG(bbb, ccc, ddd, aaa, X[13],  5);

    /* parallel round 4 */
    FFF(aaa, bbb, ccc, ddd, X[ 8], 15);
    FFF(ddd, aaa, bbb, ccc, X[ 6],  5);
    FFF(ccc, ddd, aaa, bbb, X[ 4],  8);
    FFF(bbb, ccc, ddd, aaa, X[ 1], 11);
    FFF(aaa, bbb, ccc, ddd, X[ 3], 14);
    FFF(ddd, aaa, bbb, ccc, X[11], 14);
    FFF(ccc, ddd, aaa, bbb, X[15],  6);
```

```
237    FFF(bbb, ccc, ddd, aaa, X[ 0], 14);
238    FFF(aaa, bbb, ccc, ddd, X[ 5],  6);
239    FFF(ddd, aaa, bbb, ccc, X[12],  9);
240    FFF(ccc, ddd, aaa, bbb, X[ 2], 12);
241    FFF(bbb, ccc, ddd, aaa, X[13],  9);
242    FFF(aaa, bbb, ccc, ddd, X[ 9], 12);
243    FFF(ddd, aaa, bbb, ccc, X[ 7],  5);
244    FFF(ccc, ddd, aaa, bbb, X[10], 15);
245    FFF(bbb, ccc, ddd, aaa, X[14],  8);
246 
247    /* combine results */
248    ddd += cc + md->rmd128.state[1];               /* final result for MDbuf[0] */
249    md->rmd128.state[1] = md->rmd128.state[2] + dd + aaa;
250    md->rmd128.state[2] = md->rmd128.state[3] + aa + bbb;
251    md->rmd128.state[3] = md->rmd128.state[0] + bb + ccc;
252    md->rmd128.state[0] = ddd;
253 
254    return CRYPT_OK;
255 }
```

5.60.2.2 int rmd128_done (hash_state ∗ *md*, unsigned char ∗ *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (16 bytes)

Returns:

CRYPT_OK if successful

Definition at line 299 of file rmd128.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and rmd128_compress().

Referenced by rmd128_test().

```
300 {
301     int i;
302 
303     LTC_ARGCHK(md  != NULL);
304     LTC_ARGCHK(out != NULL);
305 
306     if (md->rmd128.curlen >= sizeof(md->rmd128.buf)) {
307        return CRYPT_INVALID_ARG;
308     }
309 
310 
311     /* increase the length of the message */
312     md->rmd128.length += md->rmd128.curlen * 8;
313 
314     /* append the '1' bit */
315     md->rmd128.buf[md->rmd128.curlen++] = (unsigned char)0x80;
316 
317     /* if the length is currently above 56 bytes we append zeros
318      * then compress.  Then we can fall back to padding zeros and length
319      * encoding like normal.
320      */
321     if (md->rmd128.curlen > 56) {
322         while (md->rmd128.curlen < 64) {
323             md->rmd128.buf[md->rmd128.curlen++] = (unsigned char)0;
```

```
        }
        rmd128_compress(md, md->rmd128.buf);
        md->rmd128.curlen = 0;
    }

    /* pad upto 56 bytes of zeroes */
    while (md->rmd128.curlen < 56) {
        md->rmd128.buf[md->rmd128.curlen++] = (unsigned char)0;
    }

    /* store length */
    STORE64L(md->rmd128.length, md->rmd128.buf+56);
    rmd128_compress(md, md->rmd128.buf);

    /* copy output */
    for (i = 0; i < 4; i++) {
        STORE32L(md->rmd128.state[i], out+(4*i));
    }
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.60.2.3 int rmd128_init (hash_state ∗ *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 272 of file rmd128.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by rmd128_test().

```
{
   LTC_ARGCHK(md != NULL);
   md->rmd128.state[0] = 0x67452301UL;
   md->rmd128.state[1] = 0xefcdab89UL;
   md->rmd128.state[2] = 0x98badcfeUL;
   md->rmd128.state[3] = 0x10325476UL;
   md->rmd128.curlen   = 0;
   md->rmd128.length   = 0;
   return CRYPT_OK;
}
```

5.60.2.4 int rmd128_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 352 of file rmd128.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, rmd128_done(), rmd128_init(), and XMEMCMP.

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   static const struct {
        char *msg;
        unsigned char md[16];
   } tests[] = {
   { "",
     { 0xcd, 0xf2, 0x62, 0x13, 0xa1, 0x50, 0xdc, 0x3e,
       0xcb, 0x61, 0x0f, 0x18, 0xf6, 0xb3, 0x8b, 0x46 }
   },
   { "a",
     { 0x86, 0xbe, 0x7a, 0xfa, 0x33, 0x9d, 0x0f, 0xc7,
       0xcf, 0xc7, 0x85, 0xe7, 0x2f, 0x57, 0x8d, 0x33 }
   },
   { "abc",
     { 0xc1, 0x4a, 0x12, 0x19, 0x9c, 0x66, 0xe4, 0xba,
       0x84, 0x63, 0x6b, 0x0f, 0x69, 0x14, 0x4c, 0x77 }
   },
   { "message digest",
     { 0x9e, 0x32, 0x7b, 0x3d, 0x6e, 0x52, 0x30, 0x62,
       0xaf, 0xc1, 0x13, 0x2d, 0x7d, 0xf9, 0xd1, 0xb8 }
   },
   { "abcdefghijklmnopqrstuvwxyz",
     { 0xfd, 0x2a, 0xa6, 0x07, 0xf7, 0x1d, 0xc8, 0xf5,
       0x10, 0x71, 0x49, 0x22, 0xb3, 0x71, 0x83, 0x4e }
   },
   { "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
     { 0xd1, 0xe9, 0x59, 0xeb, 0x17, 0x9c, 0x91, 0x1f,
       0xae, 0xa4, 0x62, 0x4c, 0x60, 0xc5, 0xc7, 0x02 }
   }
   };
   int x;
   unsigned char buf[16];
   hash_state md;

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       rmd128_init(&md);
       rmd128_process(&md, (unsigned char *)tests[x].msg, strlen(tests[x].msg));
       rmd128_done(&md, buf);
       if (XMEMCMP(buf, tests[x].md, 16) != 0) {
       #if 0
          printf("Failed test %d\n", x);
       #endif
          return CRYPT_FAIL_TESTVECTOR;
       }
   }
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.60.3 Variable Documentation

5.60.3.1 const struct ltc_hash_descriptor rmd128_desc

Initial value:

```
{
    "rmd128",
    8,
    16,
    64,


   { 1, 0, 10118, 3, 0, 50 },
   6,

    &rmd128_init,
    &rmd128_process,
    &rmd128_done,
    &rmd128_test,
    NULL
}
```

Parameters:

rmd128.c RMD128 Hash function

Definition at line 26 of file rmd128.c.

Referenced by yarrow_start().

5.61 hashes/rmd160.c File Reference

5.61.1 Detailed Description

RMD160 hash function.

Definition in file rmd160.c.

```
#include "tomcrypt.h"
```

Include dependency graph for rmd160.c:

Defines

- #define F(x, y, z) ((x) ^ (y) ^ (z))
- #define G(x, y, z) (((x) & (y)) | (∼(x) & (z)))
- #define H(x, y, z) (((x) | ∼(y)) ^ (z))
- #define I(x, y, z) (((x) & (z)) | ((y) & ∼(z)))
- #define J(x, y, z) ((x) ^ ((y) | ∼(z)))
- #define FF(a, b, c, d, e, x, s)
- #define GG(a, b, c, d, e, x, s)
- #define HH(a, b, c, d, e, x, s)
- #define II(a, b, c, d, e, x, s)
- #define JJ(a, b, c, d, e, x, s)
- #define FFF(a, b, c, d, e, x, s)
- #define GGG(a, b, c, d, e, x, s)
- #define HHH(a, b, c, d, e, x, s)
- #define III(a, b, c, d, e, x, s)
- #define JJJ(a, b, c, d, e, x, s)

Functions

- static int rmd160_compress (hash_state ∗md, unsigned char ∗buf)
- int rmd160_init (hash_state ∗md)

    Initialize the hash state.

- int rmd160_done (hash_state ∗md, unsigned char ∗out)

    Terminate the hash to get the digest.

- int rmd160_test (void)

    Self-test the hash.

Variables

- const struct ltc_hash_descriptor rmd160_desc

5.61.2 Define Documentation

5.61.2.1 #define F(x, y, z) ((x) ^ (y) ^ (z))

Definition at line 45 of file rmd160.c.

5.61.2.2 #define FF(a, b, c, d, e, x, s)

Value:

```
(a) += F((b), (c), (d)) + (x);\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 52 of file rmd160.c.

5.61.2.3 #define FFF(a, b, c, d, e, x, s)

Value:

```
(a) += F((b), (c), (d)) + (x);\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 77 of file rmd160.c.

5.61.2.4 #define G(x, y, z) (((x) & (y)) | (∼(x) & (z)))

Definition at line 46 of file rmd160.c.

5.61.2.5 #define GG(a, b, c, d, e, x, s)

Value:

```
(a) += G((b), (c), (d)) + (x) + 0x5a827999UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 57 of file rmd160.c.

5.61.2.6 #define GGG(a, b, c, d, e, x, s)

Value:

```
(a) += G((b), (c), (d)) + (x) + 0x7a6d76e9UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 82 of file rmd160.c.

5.61.2.7 #define H(x, y, z) (((x) | ∼(y)) ^ (z))

Definition at line 47 of file rmd160.c.

5.61.2.8 #define HH(a, b, c, d, e, x, s)

Value:

```
(a) += H((b), (c), (d)) + (x) + 0x6ed9eba1UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 62 of file rmd160.c.

5.61.2.9 #define HHH(a, b, c, d, e, x, s)

Value:

```
(a) += H((b), (c), (d)) + (x) + 0x6d703ef3UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 87 of file rmd160.c.

5.61.2.10 #define I(x, y, z) (((x) & (z)) | ((y) & ∼(z)))

Definition at line 48 of file rmd160.c.

5.61.2.11 #define II(a, b, c, d, e, x, s)

Value:

```
(a) += I((b), (c), (d)) + (x) + 0x8f1bbcdcUL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 67 of file rmd160.c.

5.61.2.12 #define III(a, b, c, d, e, x, s)

Value:

```
(a) += I((b), (c), (d)) + (x) + 0x5c4dd124UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 92 of file rmd160.c.

5.61.2.13 #define J(x, y, z) ((x) ^ ((y) | ∼(z)))

Definition at line 49 of file rmd160.c.

5.61.2.14 #define JJ(a, b, c, d, e, x, s)

Value:

```
(a) += J((b), (c), (d)) + (x) + 0xa953fd4eUL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 72 of file rmd160.c.

5.61.2.15 #define JJJ(a, b, c, d, e, x, s)

Value:

```
(a) += J((b), (c), (d)) + (x) + 0x50a28be6UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 97 of file rmd160.c.

5.61.3 Function Documentation

5.61.3.1 static int rmd160_compress (hash_state * *md*, unsigned char * *buf*) [static]

Definition at line 106 of file rmd160.c.

Referenced by rmd160_done().

```
{
   ulong32 aa,bb,cc,dd,ee,aaa,bbb,ccc,ddd,eee,X[16];
   int i;

   /* load words X */
   for (i = 0; i < 16; i++){
      LOAD32L(X[i], buf + (4 * i));
   }

   /* load state */
   aa = aaa = md->rmd160.state[0];
   bb = bbb = md->rmd160.state[1];
   cc = ccc = md->rmd160.state[2];
   dd = ddd = md->rmd160.state[3];
   ee = eee = md->rmd160.state[4];

   /* round 1 */
   FF(aa, bb, cc, dd, ee, X[ 0], 11);
   FF(ee, aa, bb, cc, dd, X[ 1], 14);
   FF(dd, ee, aa, bb, cc, X[ 2], 15);
   FF(cc, dd, ee, aa, bb, X[ 3], 12);
   FF(bb, cc, dd, ee, aa, X[ 4],  5);
   FF(aa, bb, cc, dd, ee, X[ 5],  8);
   FF(ee, aa, bb, cc, dd, X[ 6],  7);
   FF(dd, ee, aa, bb, cc, X[ 7],  9);
   FF(cc, dd, ee, aa, bb, X[ 8], 11);
   FF(bb, cc, dd, ee, aa, X[ 9], 13);
   FF(aa, bb, cc, dd, ee, X[10], 14);
   FF(ee, aa, bb, cc, dd, X[11], 15);
   FF(dd, ee, aa, bb, cc, X[12],  6);
   FF(cc, dd, ee, aa, bb, X[13],  7);
```

```
    FF(bb, cc, dd, ee, aa, X[14],  9);
    FF(aa, bb, cc, dd, ee, X[15],  8);

    /* round 2 */
    GG(ee, aa, bb, cc, dd, X[ 7],  7);
    GG(dd, ee, aa, bb, cc, X[ 4],  6);
    GG(cc, dd, ee, aa, bb, X[13],  8);
    GG(bb, cc, dd, ee, aa, X[ 1], 13);
    GG(aa, bb, cc, dd, ee, X[10], 11);
    GG(ee, aa, bb, cc, dd, X[ 6],  9);
    GG(dd, ee, aa, bb, cc, X[15],  7);
    GG(cc, dd, ee, aa, bb, X[ 3], 15);
    GG(bb, cc, dd, ee, aa, X[12],  7);
    GG(aa, bb, cc, dd, ee, X[ 0], 12);
    GG(ee, aa, bb, cc, dd, X[ 9], 15);
    GG(dd, ee, aa, bb, cc, X[ 5],  9);
    GG(cc, dd, ee, aa, bb, X[ 2], 11);
    GG(bb, cc, dd, ee, aa, X[14],  7);
    GG(aa, bb, cc, dd, ee, X[11], 13);
    GG(ee, aa, bb, cc, dd, X[ 8], 12);

    /* round 3 */
    HH(dd, ee, aa, bb, cc, X[ 3], 11);
    HH(cc, dd, ee, aa, bb, X[10], 13);
    HH(bb, cc, dd, ee, aa, X[14],  6);
    HH(aa, bb, cc, dd, ee, X[ 4],  7);
    HH(ee, aa, bb, cc, dd, X[ 9], 14);
    HH(dd, ee, aa, bb, cc, X[15],  9);
    HH(cc, dd, ee, aa, bb, X[ 8], 13);
    HH(bb, cc, dd, ee, aa, X[ 1], 15);
    HH(aa, bb, cc, dd, ee, X[ 2], 14);
    HH(ee, aa, bb, cc, dd, X[ 7],  8);
    HH(dd, ee, aa, bb, cc, X[ 0], 13);
    HH(cc, dd, ee, aa, bb, X[ 6],  6);
    HH(bb, cc, dd, ee, aa, X[13],  5);
    HH(aa, bb, cc, dd, ee, X[11], 12);
    HH(ee, aa, bb, cc, dd, X[ 5],  7);
    HH(dd, ee, aa, bb, cc, X[12],  5);

    /* round 4 */
    II(cc, dd, ee, aa, bb, X[ 1], 11);
    II(bb, cc, dd, ee, aa, X[ 9], 12);
    II(aa, bb, cc, dd, ee, X[11], 14);
    II(ee, aa, bb, cc, dd, X[10], 15);
    II(dd, ee, aa, bb, cc, X[ 0], 14);
    II(cc, dd, ee, aa, bb, X[ 8], 15);
    II(bb, cc, dd, ee, aa, X[12],  9);
    II(aa, bb, cc, dd, ee, X[ 4],  8);
    II(ee, aa, bb, cc, dd, X[13],  9);
    II(dd, ee, aa, bb, cc, X[ 3], 14);
    II(cc, dd, ee, aa, bb, X[ 7],  5);
    II(bb, cc, dd, ee, aa, X[15],  6);
    II(aa, bb, cc, dd, ee, X[14],  8);
    II(ee, aa, bb, cc, dd, X[ 5],  6);
    II(dd, ee, aa, bb, cc, X[ 6],  5);
    II(cc, dd, ee, aa, bb, X[ 2], 12);

    /* round 5 */
    JJ(bb, cc, dd, ee, aa, X[ 4],  9);
    JJ(aa, bb, cc, dd, ee, X[ 0], 15);
    JJ(ee, aa, bb, cc, dd, X[ 5],  5);
    JJ(dd, ee, aa, bb, cc, X[ 9], 11);
    JJ(cc, dd, ee, aa, bb, X[ 7],  6);
    JJ(bb, cc, dd, ee, aa, X[12],  8);
    JJ(aa, bb, cc, dd, ee, X[ 2], 13);
    JJ(ee, aa, bb, cc, dd, X[10], 12);
    JJ(dd, ee, aa, bb, cc, X[14],  5);
```

```
   JJ(cc, dd, ee, aa, bb, X[ 1], 12);
   JJ(bb, cc, dd, ee, aa, X[ 3], 13);
   JJ(aa, bb, cc, dd, ee, X[ 8], 14);
   JJ(ee, aa, bb, cc, dd, X[11], 11);
   JJ(dd, ee, aa, bb, cc, X[ 6],  8);
   JJ(cc, dd, ee, aa, bb, X[15],  5);
   JJ(bb, cc, dd, ee, aa, X[13],  6);

   /* parallel round 1 */
   JJJ(aaa, bbb, ccc, ddd, eee, X[ 5],  8);
   JJJ(eee, aaa, bbb, ccc, ddd, X[14],  9);
   JJJ(ddd, eee, aaa, bbb, ccc, X[ 7],  9);
   JJJ(ccc, ddd, eee, aaa, bbb, X[ 0], 11);
   JJJ(bbb, ccc, ddd, eee, aaa, X[ 9], 13);
   JJJ(aaa, bbb, ccc, ddd, eee, X[ 2], 15);
   JJJ(eee, aaa, bbb, ccc, ddd, X[11], 15);
   JJJ(ddd, eee, aaa, bbb, ccc, X[ 4],  5);
   JJJ(ccc, ddd, eee, aaa, bbb, X[13],  7);
   JJJ(bbb, ccc, ddd, eee, aaa, X[ 6],  7);
   JJJ(aaa, bbb, ccc, ddd, eee, X[15],  8);
   JJJ(eee, aaa, bbb, ccc, ddd, X[ 8], 11);
   JJJ(ddd, eee, aaa, bbb, ccc, X[ 1], 14);
   JJJ(ccc, ddd, eee, aaa, bbb, X[10], 14);
   JJJ(bbb, ccc, ddd, eee, aaa, X[ 3], 12);
   JJJ(aaa, bbb, ccc, ddd, eee, X[12],  6);

   /* parallel round 2 */
   III(eee, aaa, bbb, ccc, ddd, X[ 6],  9);
   III(ddd, eee, aaa, bbb, ccc, X[11], 13);
   III(ccc, ddd, eee, aaa, bbb, X[ 3], 15);
   III(bbb, ccc, ddd, eee, aaa, X[ 7],  7);
   III(aaa, bbb, ccc, ddd, eee, X[ 0], 12);
   III(eee, aaa, bbb, ccc, ddd, X[13],  8);
   III(ddd, eee, aaa, bbb, ccc, X[ 5],  9);
   III(ccc, ddd, eee, aaa, bbb, X[10], 11);
   III(bbb, ccc, ddd, eee, aaa, X[14],  7);
   III(aaa, bbb, ccc, ddd, eee, X[15],  7);
   III(eee, aaa, bbb, ccc, ddd, X[ 8], 12);
   III(ddd, eee, aaa, bbb, ccc, X[12],  7);
   III(ccc, ddd, eee, aaa, bbb, X[ 4],  6);
   III(bbb, ccc, ddd, eee, aaa, X[ 9], 15);
   III(aaa, bbb, ccc, ddd, eee, X[ 1], 13);
   III(eee, aaa, bbb, ccc, ddd, X[ 2], 11);

   /* parallel round 3 */
   HHH(ddd, eee, aaa, bbb, ccc, X[15],  9);
   HHH(ccc, ddd, eee, aaa, bbb, X[ 5],  7);
   HHH(bbb, ccc, ddd, eee, aaa, X[ 1], 15);
   HHH(aaa, bbb, ccc, ddd, eee, X[ 3], 11);
   HHH(eee, aaa, bbb, ccc, ddd, X[ 7],  8);
   HHH(ddd, eee, aaa, bbb, ccc, X[14],  6);
   HHH(ccc, ddd, eee, aaa, bbb, X[ 6],  6);
   HHH(bbb, ccc, ddd, eee, aaa, X[ 9], 14);
   HHH(aaa, bbb, ccc, ddd, eee, X[11], 12);
   HHH(eee, aaa, bbb, ccc, ddd, X[ 8], 13);
   HHH(ddd, eee, aaa, bbb, ccc, X[12],  5);
   HHH(ccc, ddd, eee, aaa, bbb, X[ 2], 14);
   HHH(bbb, ccc, ddd, eee, aaa, X[10], 13);
   HHH(aaa, bbb, ccc, ddd, eee, X[ 0], 13);
   HHH(eee, aaa, bbb, ccc, ddd, X[ 4],  7);
   HHH(ddd, eee, aaa, bbb, ccc, X[13],  5);

   /* parallel round 4 */
   GGG(ccc, ddd, eee, aaa, bbb, X[ 8], 15);
   GGG(bbb, ccc, ddd, eee, aaa, X[ 6],  5);
   GGG(aaa, bbb, ccc, ddd, eee, X[ 4],  8);
   GGG(eee, aaa, bbb, ccc, ddd, X[ 1], 11);
```

```
   GGG(ddd, eee, aaa, bbb, ccc, X[ 3], 14);
   GGG(ccc, ddd, eee, aaa, bbb, X[11], 14);
   GGG(bbb, ccc, ddd, eee, aaa, X[15],  6);
   GGG(aaa, bbb, ccc, ddd, eee, X[ 0], 14);
   GGG(eee, aaa, bbb, ccc, ddd, X[ 5],  6);
   GGG(ddd, eee, aaa, bbb, ccc, X[12],  9);
   GGG(ccc, ddd, eee, aaa, bbb, X[ 2], 12);
   GGG(bbb, ccc, ddd, eee, aaa, X[13],  9);
   GGG(aaa, bbb, ccc, ddd, eee, X[ 9], 12);
   GGG(eee, aaa, bbb, ccc, ddd, X[ 7],  5);
   GGG(ddd, eee, aaa, bbb, ccc, X[10], 15);
   GGG(ccc, ddd, eee, aaa, bbb, X[14],  8);

   /* parallel round 5 */
   FFF(bbb, ccc, ddd, eee, aaa, X[12] ,  8);
   FFF(aaa, bbb, ccc, ddd, eee, X[15] ,  5);
   FFF(eee, aaa, bbb, ccc, ddd, X[10] , 12);
   FFF(ddd, eee, aaa, bbb, ccc, X[ 4] ,  9);
   FFF(ccc, ddd, eee, aaa, bbb, X[ 1] , 12);
   FFF(bbb, ccc, ddd, eee, aaa, X[ 5] ,  5);
   FFF(aaa, bbb, ccc, ddd, eee, X[ 8] , 14);
   FFF(eee, aaa, bbb, ccc, ddd, X[ 7] ,  6);
   FFF(ddd, eee, aaa, bbb, ccc, X[ 6] ,  8);
   FFF(ccc, ddd, eee, aaa, bbb, X[ 2] , 13);
   FFF(bbb, ccc, ddd, eee, aaa, X[13] ,  6);
   FFF(aaa, bbb, ccc, ddd, eee, X[14] ,  5);
   FFF(eee, aaa, bbb, ccc, ddd, X[ 0] , 15);
   FFF(ddd, eee, aaa, bbb, ccc, X[ 3] , 13);
   FFF(ccc, ddd, eee, aaa, bbb, X[ 9] , 11);
   FFF(bbb, ccc, ddd, eee, aaa, X[11] , 11);

   /* combine results */
   ddd += cc + md->rmd160.state[1];               /* final result for md->rmd160.state[0] */
   md->rmd160.state[1] = md->rmd160.state[2] + dd + eee;
   md->rmd160.state[2] = md->rmd160.state[3] + ee + aaa;
   md->rmd160.state[3] = md->rmd160.state[4] + aa + bbb;
   md->rmd160.state[4] = md->rmd160.state[0] + bb + ccc;
   md->rmd160.state[0] = ddd;

   return CRYPT_OK;
}
```

5.61.3.2 int rmd160_done (hash_state ∗ *md*, unsigned char ∗ *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (20 bytes)

Returns:

CRYPT_OK if successful

Definition at line 358 of file rmd160.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and rmd160_compress().

Referenced by rmd160_test().

```
{
    int i;
```

```

    LTC_ARGCHK(md  != NULL);
    LTC_ARGCHK(out != NULL);

    if (md->rmd160.curlen >= sizeof(md->rmd160.buf)) {
       return CRYPT_INVALID_ARG;
    }


    /* increase the length of the message */
    md->rmd160.length += md->rmd160.curlen * 8;

    /* append the '1' bit */
    md->rmd160.buf[md->rmd160.curlen++] = (unsigned char)0x80;

    /* if the length is currently above 56 bytes we append zeros
     * then compress.  Then we can fall back to padding zeros and length
     * encoding like normal.
     */
    if (md->rmd160.curlen > 56) {
        while (md->rmd160.curlen < 64) {
            md->rmd160.buf[md->rmd160.curlen++] = (unsigned char)0;
        }
        rmd160_compress(md, md->rmd160.buf);
        md->rmd160.curlen = 0;
    }

    /* pad upto 56 bytes of zeroes */
    while (md->rmd160.curlen < 56) {
        md->rmd160.buf[md->rmd160.curlen++] = (unsigned char)0;
    }

    /* store length */
    STORE64L(md->rmd160.length, md->rmd160.buf+56);
    rmd160_compress(md, md->rmd160.buf);

    /* copy output */
    for (i = 0; i < 5; i++) {
        STORE32L(md->rmd160.state[i], out+(4*i));
    }
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.61.3.3 int rmd160_init (hash_state ∗ *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 330 of file rmd160.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by rmd160_test().

```
{
   LTC_ARGCHK(md != NULL);
   md->rmd160.state[0] = 0x67452301UL;
   md->rmd160.state[1] = 0xefcdab89UL;
   md->rmd160.state[2] = 0x98badcfeUL;
   md->rmd160.state[3] = 0x10325476UL;
   md->rmd160.state[4] = 0xc3d2e1f0UL;
   md->rmd160.curlen   = 0;
   md->rmd160.length   = 0;
   return CRYPT_OK;
}
```

5.61.3.4 int rmd160_test (void)

Self-test the hash.

Returns:
CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 411 of file rmd160.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, rmd160_done(), rmd160_init(), and XMEMCMP.

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   static const struct {
        char *msg;
        unsigned char md[20];
   } tests[] = {
   { "",
     { 0x9c, 0x11, 0x85, 0xa5, 0xc5, 0xe9, 0xfc, 0x54, 0x61, 0x28,
       0x08, 0x97, 0x7e, 0xe8, 0xf5, 0x48, 0xb2, 0x25, 0x8d, 0x31 }
   },
   { "a",
     { 0x0b, 0xdc, 0x9d, 0x2d, 0x25, 0x6b, 0x3e, 0xe9, 0xda, 0xae,
       0x34, 0x7b, 0xe6, 0xf4, 0xdc, 0x83, 0x5a, 0x46, 0x7f, 0xfe }
   },
   { "abc",
     { 0x8e, 0xb2, 0x08, 0xf7, 0xe0, 0x5d, 0x98, 0x7a, 0x9b, 0x04,
       0x4a, 0x8e, 0x98, 0xc6, 0xb0, 0x87, 0xf1, 0x5a, 0x0b, 0xfc }
   },
   { "message digest",
     { 0x5d, 0x06, 0x89, 0xef, 0x49, 0xd2, 0xfa, 0xe5, 0x72, 0xb8,
       0x81, 0xb1, 0x23, 0xa8, 0x5f, 0xfa, 0x21, 0x59, 0x5f, 0x36 }
   },
   { "abcdefghijklmnopqrstuvwxyz",
     { 0xf7, 0x1c, 0x27, 0x10, 0x9c, 0x69, 0x2c, 0x1b, 0x56, 0xbb,
       0xdc, 0xeb, 0x5b, 0x9d, 0x28, 0x65, 0xb3, 0x70, 0x8d, 0xbc }
   },
   { "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq",
     { 0x12, 0xa0, 0x53, 0x38, 0x4a, 0x9c, 0x0c, 0x88, 0xe4, 0x05,
       0xa0, 0x6c, 0x27, 0xdc, 0xf4, 0x9a, 0xda, 0x62, 0xeb, 0x2b }
   }
   };
   int x;
   unsigned char buf[20];
   hash_state md;

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       rmd160_init(&md);
```

```
        rmd160_process(&md, (unsigned char *)tests[x].msg, strlen(tests[x].msg));
        rmd160_done(&md, buf);
        if (XMEMCMP(buf, tests[x].md, 20) != 0) {
#if 0
           printf("Failed test %d\n", x);
#endif
           return CRYPT_FAIL_TESTVECTOR;
        }
    }
    return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.61.4 Variable Documentation

5.61.4.1 const struct ltc_hash_descriptor rmd160_desc

Initial value:

```
{
    "rmd160",
    9,
    20,
    64,


   { 1, 3, 36, 3, 2, 1,  },
   6,

    &rmd160_init,
    &rmd160_process,
    &rmd160_done,
    &rmd160_test,
    NULL
}
```

Definition at line 26 of file rmd160.c.

Referenced by yarrow_start().

5.62 hashes/rmd256.c File Reference

`#include "tomcrypt.h"`

Include dependency graph for rmd256.c:

Defines

- #define F(x, y, z) ((x) ^ (y) ^ (z))
- #define G(x, y, z) (((x) & (y)) | (∼(x) & (z)))
- #define H(x, y, z) (((x) | ∼(y)) ^ (z))
- #define I(x, y, z) (((x) & (z)) | ((y) & ∼(z)))
- #define FF(a, b, c, d, x, s)
- #define GG(a, b, c, d, x, s)
- #define HH(a, b, c, d, x, s)
- #define II(a, b, c, d, x, s)
- #define FFF(a, b, c, d, x, s)
- #define GGG(a, b, c, d, x, s)
- #define HHH(a, b, c, d, x, s)
- #define III(a, b, c, d, x, s)

Functions

- static int rmd256_compress (hash_state ∗md, unsigned char ∗buf)
- int rmd256_init (hash_state ∗md)

  Initialize the hash state.
- int rmd256_done (hash_state ∗md, unsigned char ∗out)

  Terminate the hash to get the digest.
- int rmd256_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor rmd256_desc

5.62.1 Define Documentation

5.62.1.1 #define F(x, y, z) ((x) ^ (y) ^ (z))

Definition at line 39 of file rmd256.c.

5.62.1.2 #define FF(a, b, c, d, x, s)

Value:

```
(a) += F((b), (c), (d)) + (x);\
      (a) = ROLc((a), (s));
```

Definition at line 45 of file rmd256.c.

5.62.1.3 #define FFF(a, b, c, d, x, s)

Value:

```
(a) += F((b), (c), (d)) + (x);\
      (a) = ROLc((a), (s));
```

Definition at line 61 of file rmd256.c.

5.62.1.4 #define G(x, y, z) (((x) & (y)) | (~(x) & (z)))

Definition at line 40 of file rmd256.c.

5.62.1.5 #define GG(a, b, c, d, x, s)

Value:

```
(a) += G((b), (c), (d)) + (x) + 0x5a827999UL;\
      (a) = ROLc((a), (s));
```

Definition at line 49 of file rmd256.c.

5.62.1.6 #define GGG(a, b, c, d, x, s)

Value:

```
(a) += G((b), (c), (d)) + (x) + 0x6d703ef3UL;\
      (a) = ROLc((a), (s));
```

Definition at line 65 of file rmd256.c.

5.62.1.7 #define H(x, y, z) (((x) | ~(y)) ^ (z))

Definition at line 41 of file rmd256.c.

5.62.1.8 #define HH(a, b, c, d, x, s)

Value:

```
(a) += H((b), (c), (d)) + (x) + 0x6ed9eba1UL;\
      (a) = ROLc((a), (s));
```

Definition at line 53 of file rmd256.c.

5.62.1.9 #define HHH(a, b, c, d, x, s)

Value:

```
(a) += H((b), (c), (d)) + (x) + 0x5c4dd124UL;\
      (a) = ROLc((a), (s));
```

Definition at line 69 of file rmd256.c.

5.62.1.10 #define I(x, y, z) (((x) & (z)) | ((y) & ∼(z)))

Definition at line 42 of file rmd256.c.

5.62.1.11 #define II(a, b, c, d, x, s)

Value:

```
(a) += I((b), (c), (d)) + (x) + 0x8f1bbcdcUL;\
      (a) = ROLc((a), (s));
```

Definition at line 57 of file rmd256.c.

5.62.1.12 #define III(a, b, c, d, x, s)

Value:

```
(a) += I((b), (c), (d)) + (x) + 0x50a28be6UL;\
      (a) = ROLc((a), (s));
```

Definition at line 73 of file rmd256.c.

5.62.2 Function Documentation

5.62.2.1 static int rmd256_compress (hash_state ∗ *md*, unsigned char ∗ *buf*) [static]

Definition at line 80 of file rmd256.c.

Referenced by rmd256_done().

```
{
   ulong32 aa,bb,cc,dd,aaa,bbb,ccc,ddd,tmp,X[16];
   int i;

   /* load words X */
   for (i = 0; i < 16; i++){
      LOAD32L(X[i], buf + (4 * i));
   }

   /* load state */
   aa = md->rmd256.state[0];
   bb = md->rmd256.state[1];
   cc = md->rmd256.state[2];
   dd = md->rmd256.state[3];
   aaa = md->rmd256.state[4];
```

```
   bbb = md->rmd256.state[5];
   ccc = md->rmd256.state[6];
   ddd = md->rmd256.state[7];

    /* round 1 */
    FF(aa, bb, cc, dd, X[ 0], 11);
    FF(dd, aa, bb, cc, X[ 1], 14);
    FF(cc, dd, aa, bb, X[ 2], 15);
    FF(bb, cc, dd, aa, X[ 3], 12);
    FF(aa, bb, cc, dd, X[ 4],  5);
    FF(dd, aa, bb, cc, X[ 5],  8);
    FF(cc, dd, aa, bb, X[ 6],  7);
    FF(bb, cc, dd, aa, X[ 7],  9);
    FF(aa, bb, cc, dd, X[ 8], 11);
    FF(dd, aa, bb, cc, X[ 9], 13);
    FF(cc, dd, aa, bb, X[10], 14);
    FF(bb, cc, dd, aa, X[11], 15);
    FF(aa, bb, cc, dd, X[12],  6);
    FF(dd, aa, bb, cc, X[13],  7);
    FF(cc, dd, aa, bb, X[14],  9);
    FF(bb, cc, dd, aa, X[15],  8);

    /* parallel round 1 */
    III(aaa, bbb, ccc, ddd, X[ 5],  8);
    III(ddd, aaa, bbb, ccc, X[14],  9);
    III(ccc, ddd, aaa, bbb, X[ 7],  9);
    III(bbb, ccc, ddd, aaa, X[ 0], 11);
    III(aaa, bbb, ccc, ddd, X[ 9], 13);
    III(ddd, aaa, bbb, ccc, X[ 2], 15);
    III(ccc, ddd, aaa, bbb, X[11], 15);
    III(bbb, ccc, ddd, aaa, X[ 4],  5);
    III(aaa, bbb, ccc, ddd, X[13],  7);
    III(ddd, aaa, bbb, ccc, X[ 6],  7);
    III(ccc, ddd, aaa, bbb, X[15],  8);
    III(bbb, ccc, ddd, aaa, X[ 8], 11);
    III(aaa, bbb, ccc, ddd, X[ 1], 14);
    III(ddd, aaa, bbb, ccc, X[10], 14);
    III(ccc, ddd, aaa, bbb, X[ 3], 12);
    III(bbb, ccc, ddd, aaa, X[12],  6);

    tmp = aa; aa = aaa; aaa = tmp;

    /* round 2 */
    GG(aa, bb, cc, dd, X[ 7],  7);
    GG(dd, aa, bb, cc, X[ 4],  6);
    GG(cc, dd, aa, bb, X[13],  8);
    GG(bb, cc, dd, aa, X[ 1], 13);
    GG(aa, bb, cc, dd, X[10], 11);
    GG(dd, aa, bb, cc, X[ 6],  9);
    GG(cc, dd, aa, bb, X[15],  7);
    GG(bb, cc, dd, aa, X[ 3], 15);
    GG(aa, bb, cc, dd, X[12],  7);
    GG(dd, aa, bb, cc, X[ 0], 12);
    GG(cc, dd, aa, bb, X[ 9], 15);
    GG(bb, cc, dd, aa, X[ 5],  9);
    GG(aa, bb, cc, dd, X[ 2], 11);
    GG(dd, aa, bb, cc, X[14],  7);
    GG(cc, dd, aa, bb, X[11], 13);
    GG(bb, cc, dd, aa, X[ 8], 12);

    /* parallel round 2 */
    HHH(aaa, bbb, ccc, ddd, X[ 6],  9);
    HHH(ddd, aaa, bbb, ccc, X[11], 13);
    HHH(ccc, ddd, aaa, bbb, X[ 3], 15);
    HHH(bbb, ccc, ddd, aaa, X[ 7],  7);
    HHH(aaa, bbb, ccc, ddd, X[ 0], 12);
    HHH(ddd, aaa, bbb, ccc, X[13],  8);
```

```
   HHH(ccc, ddd, aaa, bbb, X[ 5],  9);
   HHH(bbb, ccc, ddd, aaa, X[10], 11);
   HHH(aaa, bbb, ccc, ddd, X[14],  7);
   HHH(ddd, aaa, bbb, ccc, X[15],  7);
   HHH(ccc, ddd, aaa, bbb, X[ 8], 12);
   HHH(bbb, ccc, ddd, aaa, X[12],  7);
   HHH(aaa, bbb, ccc, ddd, X[ 4],  6);
   HHH(ddd, aaa, bbb, ccc, X[ 9], 15);
   HHH(ccc, ddd, aaa, bbb, X[ 1], 13);
   HHH(bbb, ccc, ddd, aaa, X[ 2], 11);

   tmp = bb; bb = bbb; bbb = tmp;

   /* round 3 */
   HH(aa, bb, cc, dd, X[ 3], 11);
   HH(dd, aa, bb, cc, X[10], 13);
   HH(cc, dd, aa, bb, X[14],  6);
   HH(bb, cc, dd, aa, X[ 4],  7);
   HH(aa, bb, cc, dd, X[ 9], 14);
   HH(dd, aa, bb, cc, X[15],  9);
   HH(cc, dd, aa, bb, X[ 8], 13);
   HH(bb, cc, dd, aa, X[ 1], 15);
   HH(aa, bb, cc, dd, X[ 2], 14);
   HH(dd, aa, bb, cc, X[ 7],  8);
   HH(cc, dd, aa, bb, X[ 0], 13);
   HH(bb, cc, dd, aa, X[ 6],  6);
   HH(aa, bb, cc, dd, X[13],  5);
   HH(dd, aa, bb, cc, X[11], 12);
   HH(cc, dd, aa, bb, X[ 5],  7);
   HH(bb, cc, dd, aa, X[12],  5);

   /* parallel round 3 */
   GGG(aaa, bbb, ccc, ddd, X[15],  9);
   GGG(ddd, aaa, bbb, ccc, X[ 5],  7);
   GGG(ccc, ddd, aaa, bbb, X[ 1], 15);
   GGG(bbb, ccc, ddd, aaa, X[ 3], 11);
   GGG(aaa, bbb, ccc, ddd, X[ 7],  8);
   GGG(ddd, aaa, bbb, ccc, X[14],  6);
   GGG(ccc, ddd, aaa, bbb, X[ 6],  6);
   GGG(bbb, ccc, ddd, aaa, X[ 9], 14);
   GGG(aaa, bbb, ccc, ddd, X[11], 12);
   GGG(ddd, aaa, bbb, ccc, X[ 8], 13);
   GGG(ccc, ddd, aaa, bbb, X[12],  5);
   GGG(bbb, ccc, ddd, aaa, X[ 2], 14);
   GGG(aaa, bbb, ccc, ddd, X[10], 13);
   GGG(ddd, aaa, bbb, ccc, X[ 0], 13);
   GGG(ccc, ddd, aaa, bbb, X[ 4],  7);
   GGG(bbb, ccc, ddd, aaa, X[13],  5);

   tmp = cc; cc = ccc; ccc = tmp;

   /* round 4 */
   II(aa, bb, cc, dd, X[ 1], 11);
   II(dd, aa, bb, cc, X[ 9], 12);
   II(cc, dd, aa, bb, X[11], 14);
   II(bb, cc, dd, aa, X[10], 15);
   II(aa, bb, cc, dd, X[ 0], 14);
   II(dd, aa, bb, cc, X[ 8], 15);
   II(cc, dd, aa, bb, X[12],  9);
   II(bb, cc, dd, aa, X[ 4],  8);
   II(aa, bb, cc, dd, X[13],  9);
   II(dd, aa, bb, cc, X[ 3], 14);
   II(cc, dd, aa, bb, X[ 7],  5);
   II(bb, cc, dd, aa, X[15],  6);
   II(aa, bb, cc, dd, X[14],  8);
   II(dd, aa, bb, cc, X[ 5],  6);
   II(cc, dd, aa, bb, X[ 6],  5);
```

```
    II(bb, cc, dd, aa, X[ 2], 12);

    /* parallel round 4 */
    FFF(aaa, bbb, ccc, ddd, X[ 8], 15);
    FFF(ddd, aaa, bbb, ccc, X[ 6],  5);
    FFF(ccc, ddd, aaa, bbb, X[ 4],  8);
    FFF(bbb, ccc, ddd, aaa, X[ 1], 11);
    FFF(aaa, bbb, ccc, ddd, X[ 3], 14);
    FFF(ddd, aaa, bbb, ccc, X[11], 14);
    FFF(ccc, ddd, aaa, bbb, X[15],  6);
    FFF(bbb, ccc, ddd, aaa, X[ 0], 14);
    FFF(aaa, bbb, ccc, ddd, X[ 5],  6);
    FFF(ddd, aaa, bbb, ccc, X[12],  9);
    FFF(ccc, ddd, aaa, bbb, X[ 2], 12);
    FFF(bbb, ccc, ddd, aaa, X[13],  9);
    FFF(aaa, bbb, ccc, ddd, X[ 9], 12);
    FFF(ddd, aaa, bbb, ccc, X[ 7],  5);
    FFF(ccc, ddd, aaa, bbb, X[10], 15);
    FFF(bbb, ccc, ddd, aaa, X[14],  8);

    tmp = dd; dd = ddd; ddd = tmp;

    /* combine results */
    md->rmd256.state[0] += aa;
    md->rmd256.state[1] += bb;
    md->rmd256.state[2] += cc;
    md->rmd256.state[3] += dd;
    md->rmd256.state[4] += aaa;
    md->rmd256.state[5] += bbb;
    md->rmd256.state[6] += ccc;
    md->rmd256.state[7] += ddd;

    return CRYPT_OK;
}
```

5.62.2.2 int rmd256_done (hash_state * *md*, unsigned char * *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (16 bytes)

Returns:

CRYPT_OK if successful

Definition at line 312 of file rmd256.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and rmd256_compress().

Referenced by rmd256_test().

```
{
    int i;

    LTC_ARGCHK(md  != NULL);
    LTC_ARGCHK(out != NULL);

    if (md->rmd256.curlen >= sizeof(md->rmd256.buf)) {
       return CRYPT_INVALID_ARG;
    }
```

```


    /* increase the length of the message */
    md->rmd256.length += md->rmd256.curlen * 8;

    /* append the '1' bit */
    md->rmd256.buf[md->rmd256.curlen++] = (unsigned char)0x80;

    /* if the length is currently above 56 bytes we append zeros
     * then compress.  Then we can fall back to padding zeros and length
     * encoding like normal.
     */
    if (md->rmd256.curlen > 56) {
        while (md->rmd256.curlen < 64) {
            md->rmd256.buf[md->rmd256.curlen++] = (unsigned char)0;
        }
        rmd256_compress(md, md->rmd256.buf);
        md->rmd256.curlen = 0;
    }

    /* pad upto 56 bytes of zeroes */
    while (md->rmd256.curlen < 56) {
        md->rmd256.buf[md->rmd256.curlen++] = (unsigned char)0;
    }

    /* store length */
    STORE64L(md->rmd256.length, md->rmd256.buf+56);
    rmd256_compress(md, md->rmd256.buf);

    /* copy output */
    for (i = 0; i < 8; i++) {
        STORE32L(md->rmd256.state[i], out+(4*i));
    }
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.62.2.3 int rmd256_init (hash_state ∗ *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 281 of file rmd256.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by rmd256_test().

```
{
   LTC_ARGCHK(md != NULL);
   md->rmd256.state[0] = 0x67452301UL;
   md->rmd256.state[1] = 0xefcdab89UL;
   md->rmd256.state[2] = 0x98badcfeUL;
```

```
   md->rmd256.state[3] = 0x10325476UL;
   md->rmd256.state[4] = 0x76543210UL;
   md->rmd256.state[5] = 0xfedcba98UL;
   md->rmd256.state[6] = 0x89abcdefUL;
   md->rmd256.state[7] = 0x01234567UL;
   md->rmd256.curlen   = 0;
   md->rmd256.length   = 0;
   return CRYPT_OK;
}
```

5.62.2.4 int rmd256_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 365 of file rmd256.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, rmd256_done(), rmd256_init(), and XMEMCMP.

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   static const struct {
        char *msg;
        unsigned char md[32];
   } tests[] = {
   { "",
     { 0x02, 0xba, 0x4c, 0x4e, 0x5f, 0x8e, 0xcd, 0x18,
       0x77, 0xfc, 0x52, 0xd6, 0x4d, 0x30, 0xe3, 0x7a,
       0x2d, 0x97, 0x74, 0xfb, 0x1e, 0x5d, 0x02, 0x63,
       0x80, 0xae, 0x01, 0x68, 0xe3, 0xc5, 0x52, 0x2d }
   },
   { "a",
     { 0xf9, 0x33, 0x3e, 0x45, 0xd8, 0x57, 0xf5, 0xd9,
       0x0a, 0x91, 0xba, 0xb7, 0x0a, 0x1e, 0xba, 0x0c,
       0xfb, 0x1b, 0xe4, 0xb0, 0x78, 0x3c, 0x9a, 0xcf,
       0xcd, 0x88, 0x3a, 0x91, 0x34, 0x69, 0x29, 0x25 }
   },
   { "abc",
     { 0xaf, 0xbd, 0x6e, 0x22, 0x8b, 0x9d, 0x8c, 0xbb,
       0xce, 0xf5, 0xca, 0x2d, 0x03, 0xe6, 0xdb, 0xa1,
       0x0a, 0xc0, 0xbc, 0x7d, 0xcb, 0xe4, 0x68, 0x0e,
       0x1e, 0x42, 0xd2, 0xe9, 0x75, 0x45, 0x9b, 0x65 }
   },
   { "message digest",
     { 0x87, 0xe9, 0x71, 0x75, 0x9a, 0x1c, 0xe4, 0x7a,
       0x51, 0x4d, 0x5c, 0x91, 0x4c, 0x39, 0x2c, 0x90,
       0x18, 0xc7, 0xc4, 0x6b, 0xc1, 0x44, 0x65, 0x55,
       0x4a, 0xfc, 0xdf, 0x54, 0xa5, 0x07, 0x0c, 0x0e }
   },
   { "abcdefghijklmnopqrstuvwxyz",
     { 0x64, 0x9d, 0x30, 0x34, 0x75, 0x1e, 0xa2, 0x16,
       0x77, 0x6b, 0xf9, 0xa1, 0x8a, 0xcc, 0x81, 0xbc,
       0x78, 0x96, 0x11, 0x8a, 0x51, 0x97, 0x96, 0x87,
       0x82, 0xdd, 0x1f, 0xd9, 0x7d, 0x8d, 0x51, 0x33 }
   },
   { "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
     { 0x57, 0x40, 0xa4, 0x08, 0xac, 0x16, 0xb7, 0x20,
       0xb8, 0x44, 0x24, 0xae, 0x93, 0x1c, 0xbb, 0x1f,
```

```
            0xe3, 0x63, 0xd1, 0xd0, 0xbf, 0x40, 0x17, 0xf1,
            0xa8, 0x9f, 0x7e, 0xa6, 0xde, 0x77, 0xa0, 0xb8 }
    }
    };
    int x;
    unsigned char buf[32];
    hash_state md;

    for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
        rmd256_init(&md);
        rmd256_process(&md, (unsigned char *)tests[x].msg, strlen(tests[x].msg));
        rmd256_done(&md, buf);
        if (XMEMCMP(buf, tests[x].md, 32) != 0) {
        #if 0
           printf("Failed test %d\n", x);
        #endif
           return CRYPT_FAIL_TESTVECTOR;
        }
    }
    return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.62.3 Variable Documentation

5.62.3.1 const struct ltc_hash_descriptor rmd256_desc

Initial value:

```
{
    "rmd256",
    8,
    16,
    64,


   { 1, 3, 36, 3, 2, 3 },
   6,

    &rmd256_init,
    &rmd256_process,
    &rmd256_done,
    &rmd256_test,
    NULL
}
```

Parameters:

rmd256.c RMD256 Hash function

Definition at line 20 of file rmd256.c.

Referenced by yarrow_start().

5.63 hashes/rmd320.c File Reference

5.63.1 Detailed Description

RMD320 hash function.

Definition in file rmd320.c.

`#include "tomcrypt.h"`

Include dependency graph for rmd320.c:

Defines

- #define F(x, y, z) ((x) ^ (y) ^ (z))
- #define G(x, y, z) (((x) & (y)) | (~(x) & (z)))
- #define H(x, y, z) (((x) | ~(y)) ^ (z))
- #define I(x, y, z) (((x) & (z)) | ((y) & ~(z)))
- #define J(x, y, z) ((x) ^ ((y) | ~(z)))
- #define FF(a, b, c, d, e, x, s)
- #define GG(a, b, c, d, e, x, s)
- #define HH(a, b, c, d, e, x, s)
- #define II(a, b, c, d, e, x, s)
- #define JJ(a, b, c, d, e, x, s)
- #define FFF(a, b, c, d, e, x, s)
- #define GGG(a, b, c, d, e, x, s)
- #define HHH(a, b, c, d, e, x, s)
- #define III(a, b, c, d, e, x, s)
- #define JJJ(a, b, c, d, e, x, s)

Functions

- static int rmd320_compress (hash_state *md, unsigned char *buf)
- int rmd320_init (hash_state *md)

  Initialize the hash state.

- int rmd320_done (hash_state *md, unsigned char *out)

  Terminate the hash to get the digest.

- int rmd320_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor rmd320_desc

5.63.2 Define Documentation

5.63.2.1 #define F(x, y, z) ((x) ^ (y) ^ (z))

Definition at line 39 of file rmd320.c.

5.63.2.2 #define FF(a, b, c, d, e, x, s)

Value:

```
(a) += F((b), (c), (d)) + (x);\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 46 of file rmd320.c.

5.63.2.3 #define FFF(a, b, c, d, e, x, s)

Value:

```
(a) += F((b), (c), (d)) + (x);\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 71 of file rmd320.c.

5.63.2.4 #define G(x, y, z) (((x) & (y)) | (~(x) & (z)))

Definition at line 40 of file rmd320.c.

5.63.2.5 #define GG(a, b, c, d, e, x, s)

Value:

```
(a) += G((b), (c), (d)) + (x) + 0x5a827999UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 51 of file rmd320.c.

5.63.2.6 #define GGG(a, b, c, d, e, x, s)

Value:

```
(a) += G((b), (c), (d)) + (x) + 0x7a6d76e9UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 76 of file rmd320.c.

5.63.2.7 #define H(x, y, z) (((x) | ~(y)) ^ (z))

Definition at line 41 of file rmd320.c.

5.63.2.8 #define HH(a, b, c, d, e, x, s)

Value:

```
(a) += H((b), (c), (d)) + (x) + 0x6ed9eba1UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 56 of file rmd320.c.

5.63.2.9 #define HHH(a, b, c, d, e, x, s)

Value:

```
(a) += H((b), (c), (d)) + (x) + 0x6d703ef3UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 81 of file rmd320.c.

5.63.2.10 #define I(x, y, z) (((x) & (z)) | ((y) & ~(z)))

Definition at line 42 of file rmd320.c.

5.63.2.11 #define II(a, b, c, d, e, x, s)

Value:

```
(a) += I((b), (c), (d)) + (x) + 0x8f1bbcdcUL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 61 of file rmd320.c.

5.63.2.12 #define III(a, b, c, d, e, x, s)

Value:

```
(a) += I((b), (c), (d)) + (x) + 0x5c4dd124UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 86 of file rmd320.c.

5.63.2.13 #define J(x, y, z) ((x) ^ ((y) | ~(z)))

Definition at line 43 of file rmd320.c.

5.63.2.14 #define JJ(a, b, c, d, e, x, s)

Value:

```
(a) += J((b), (c), (d)) + (x) + 0xa953fd4eUL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 66 of file rmd320.c.

5.63.2.15 #define JJJ(a, b, c, d, e, x, s)

Value:

```
(a) += J((b), (c), (d)) + (x) + 0x50a28be6UL;\
      (a) = ROLc((a), (s)) + (e);\
      (c) = ROLc((c), 10);
```

Definition at line 91 of file rmd320.c.

5.63.3 Function Documentation

5.63.3.1 static int rmd320_compress (hash_state * *md*, unsigned char * *buf*) [static]

Definition at line 100 of file rmd320.c.

Referenced by rmd320_done().

```
{
   ulong32 aa,bb,cc,dd,ee,aaa,bbb,ccc,ddd,eee,tmp,X[16];
   int i;

   /* load words X */
   for (i = 0; i < 16; i++){
      LOAD32L(X[i], buf + (4 * i));
   }

   /* load state */
   aa = md->rmd320.state[0];
   bb = md->rmd320.state[1];
   cc = md->rmd320.state[2];
   dd = md->rmd320.state[3];
   ee = md->rmd320.state[4];
   aaa = md->rmd320.state[5];
   bbb = md->rmd320.state[6];
   ccc = md->rmd320.state[7];
   ddd = md->rmd320.state[8];
   eee = md->rmd320.state[9];

   /* round 1 */
   FF(aa, bb, cc, dd, ee, X[ 0], 11);
   FF(ee, aa, bb, cc, dd, X[ 1], 14);
   FF(dd, ee, aa, bb, cc, X[ 2], 15);
   FF(cc, dd, ee, aa, bb, X[ 3], 12);
   FF(bb, cc, dd, ee, aa, X[ 4],  5);
   FF(aa, bb, cc, dd, ee, X[ 5],  8);
   FF(ee, aa, bb, cc, dd, X[ 6],  7);
   FF(dd, ee, aa, bb, cc, X[ 7],  9);
   FF(cc, dd, ee, aa, bb, X[ 8], 11);
```

```
   FF(bb, cc, dd, ee, aa, X[ 9], 13);
   FF(aa, bb, cc, dd, ee, X[10], 14);
   FF(ee, aa, bb, cc, dd, X[11], 15);
   FF(dd, ee, aa, bb, cc, X[12],  6);
   FF(cc, dd, ee, aa, bb, X[13],  7);
   FF(bb, cc, dd, ee, aa, X[14],  9);
   FF(aa, bb, cc, dd, ee, X[15],  8);

   /* parallel round 1 */
   JJJ(aaa, bbb, ccc, ddd, eee, X[ 5],  8);
   JJJ(eee, aaa, bbb, ccc, ddd, X[14],  9);
   JJJ(ddd, eee, aaa, bbb, ccc, X[ 7],  9);
   JJJ(ccc, ddd, eee, aaa, bbb, X[ 0], 11);
   JJJ(bbb, ccc, ddd, eee, aaa, X[ 9], 13);
   JJJ(aaa, bbb, ccc, ddd, eee, X[ 2], 15);
   JJJ(eee, aaa, bbb, ccc, ddd, X[11], 15);
   JJJ(ddd, eee, aaa, bbb, ccc, X[ 4],  5);
   JJJ(ccc, ddd, eee, aaa, bbb, X[13],  7);
   JJJ(bbb, ccc, ddd, eee, aaa, X[ 6],  7);
   JJJ(aaa, bbb, ccc, ddd, eee, X[15],  8);
   JJJ(eee, aaa, bbb, ccc, ddd, X[ 8], 11);
   JJJ(ddd, eee, aaa, bbb, ccc, X[ 1], 14);
   JJJ(ccc, ddd, eee, aaa, bbb, X[10], 14);
   JJJ(bbb, ccc, ddd, eee, aaa, X[ 3], 12);
   JJJ(aaa, bbb, ccc, ddd, eee, X[12],  6);

   tmp = aa; aa = aaa; aaa = tmp;

   /* round 2 */
   GG(ee, aa, bb, cc, dd, X[ 7],  7);
   GG(dd, ee, aa, bb, cc, X[ 4],  6);
   GG(cc, dd, ee, aa, bb, X[13],  8);
   GG(bb, cc, dd, ee, aa, X[ 1], 13);
   GG(aa, bb, cc, dd, ee, X[10], 11);
   GG(ee, aa, bb, cc, dd, X[ 6],  9);
   GG(dd, ee, aa, bb, cc, X[15],  7);
   GG(cc, dd, ee, aa, bb, X[ 3], 15);
   GG(bb, cc, dd, ee, aa, X[12],  7);
   GG(aa, bb, cc, dd, ee, X[ 0], 12);
   GG(ee, aa, bb, cc, dd, X[ 9], 15);
   GG(dd, ee, aa, bb, cc, X[ 5],  9);
   GG(cc, dd, ee, aa, bb, X[ 2], 11);
   GG(bb, cc, dd, ee, aa, X[14],  7);
   GG(aa, bb, cc, dd, ee, X[11], 13);
   GG(ee, aa, bb, cc, dd, X[ 8], 12);

   /* parallel round 2 */
   III(eee, aaa, bbb, ccc, ddd, X[ 6],  9);
   III(ddd, eee, aaa, bbb, ccc, X[11], 13);
   III(ccc, ddd, eee, aaa, bbb, X[ 3], 15);
   III(bbb, ccc, ddd, eee, aaa, X[ 7],  7);
   III(aaa, bbb, ccc, ddd, eee, X[ 0], 12);
   III(eee, aaa, bbb, ccc, ddd, X[13],  8);
   III(ddd, eee, aaa, bbb, ccc, X[ 5],  9);
   III(ccc, ddd, eee, aaa, bbb, X[10], 11);
   III(bbb, ccc, ddd, eee, aaa, X[14],  7);
   III(aaa, bbb, ccc, ddd, eee, X[15],  7);
   III(eee, aaa, bbb, ccc, ddd, X[ 8], 12);
   III(ddd, eee, aaa, bbb, ccc, X[12],  7);
   III(ccc, ddd, eee, aaa, bbb, X[ 4],  6);
   III(bbb, ccc, ddd, eee, aaa, X[ 9], 15);
   III(aaa, bbb, ccc, ddd, eee, X[ 1], 13);
   III(eee, aaa, bbb, ccc, ddd, X[ 2], 11);

   tmp = bb; bb = bbb; bbb = tmp;

   /* round 3 */
```

```
HH(dd, ee, aa, bb, cc, X[ 3], 11);
HH(cc, dd, ee, aa, bb, X[10], 13);
HH(bb, cc, dd, ee, aa, X[14],  6);
HH(aa, bb, cc, dd, ee, X[ 4],  7);
HH(ee, aa, bb, cc, dd, X[ 9], 14);
HH(dd, ee, aa, bb, cc, X[15],  9);
HH(cc, dd, ee, aa, bb, X[ 8], 13);
HH(bb, cc, dd, ee, aa, X[ 1], 15);
HH(aa, bb, cc, dd, ee, X[ 2], 14);
HH(ee, aa, bb, cc, dd, X[ 7],  8);
HH(dd, ee, aa, bb, cc, X[ 0], 13);
HH(cc, dd, ee, aa, bb, X[ 6],  6);
HH(bb, cc, dd, ee, aa, X[13],  5);
HH(aa, bb, cc, dd, ee, X[11], 12);
HH(ee, aa, bb, cc, dd, X[ 5],  7);
HH(dd, ee, aa, bb, cc, X[12],  5);

/* parallel round 3 */
HHH(ddd, eee, aaa, bbb, ccc, X[15],  9);
HHH(ccc, ddd, eee, aaa, bbb, X[ 5],  7);
HHH(bbb, ccc, ddd, eee, aaa, X[ 1], 15);
HHH(aaa, bbb, ccc, ddd, eee, X[ 3], 11);
HHH(eee, aaa, bbb, ccc, ddd, X[ 7],  8);
HHH(ddd, eee, aaa, bbb, ccc, X[14],  6);
HHH(ccc, ddd, eee, aaa, bbb, X[ 6],  6);
HHH(bbb, ccc, ddd, eee, aaa, X[ 9], 14);
HHH(aaa, bbb, ccc, ddd, eee, X[11], 12);
HHH(eee, aaa, bbb, ccc, ddd, X[ 8], 13);
HHH(ddd, eee, aaa, bbb, ccc, X[12],  5);
HHH(ccc, ddd, eee, aaa, bbb, X[ 2], 14);
HHH(bbb, ccc, ddd, eee, aaa, X[10], 13);
HHH(aaa, bbb, ccc, ddd, eee, X[ 0], 13);
HHH(eee, aaa, bbb, ccc, ddd, X[ 4],  7);
HHH(ddd, eee, aaa, bbb, ccc, X[13],  5);

tmp = cc; cc = ccc; ccc = tmp;

/* round 4 */
II(cc, dd, ee, aa, bb, X[ 1], 11);
II(bb, cc, dd, ee, aa, X[ 9], 12);
II(aa, bb, cc, dd, ee, X[11], 14);
II(ee, aa, bb, cc, dd, X[10], 15);
II(dd, ee, aa, bb, cc, X[ 0], 14);
II(cc, dd, ee, aa, bb, X[ 8], 15);
II(bb, cc, dd, ee, aa, X[12],  9);
II(aa, bb, cc, dd, ee, X[ 4],  8);
II(ee, aa, bb, cc, dd, X[13],  9);
II(dd, ee, aa, bb, cc, X[ 3], 14);
II(cc, dd, ee, aa, bb, X[ 7],  5);
II(bb, cc, dd, ee, aa, X[15],  6);
II(aa, bb, cc, dd, ee, X[14],  8);
II(ee, aa, bb, cc, dd, X[ 5],  6);
II(dd, ee, aa, bb, cc, X[ 6],  5);
II(cc, dd, ee, aa, bb, X[ 2], 12);

/* parallel round 4 */
GGG(ccc, ddd, eee, aaa, bbb, X[ 8], 15);
GGG(bbb, ccc, ddd, eee, aaa, X[ 6],  5);
GGG(aaa, bbb, ccc, ddd, eee, X[ 4],  8);
GGG(eee, aaa, bbb, ccc, ddd, X[ 1], 11);
GGG(ddd, eee, aaa, bbb, ccc, X[ 3], 14);
GGG(ccc, ddd, eee, aaa, bbb, X[11], 14);
GGG(bbb, ccc, ddd, eee, aaa, X[15],  6);
GGG(aaa, bbb, ccc, ddd, eee, X[ 0], 14);
GGG(eee, aaa, bbb, ccc, ddd, X[ 5],  6);
GGG(ddd, eee, aaa, bbb, ccc, X[12],  9);
GGG(ccc, ddd, eee, aaa, bbb, X[ 2], 12);
```

```
   GGG(bbb, ccc, ddd, eee, aaa, X[13],  9);
   GGG(aaa, bbb, ccc, ddd, eee, X[ 9], 12);
   GGG(eee, aaa, bbb, ccc, ddd, X[ 7],  5);
   GGG(ddd, eee, aaa, bbb, ccc, X[10], 15);
   GGG(ccc, ddd, eee, aaa, bbb, X[14],  8);

   tmp = dd; dd = ddd; ddd = tmp;

   /* round 5 */
   JJ(bb, cc, dd, ee, aa, X[ 4],  9);
   JJ(aa, bb, cc, dd, ee, X[ 0], 15);
   JJ(ee, aa, bb, cc, dd, X[ 5],  5);
   JJ(dd, ee, aa, bb, cc, X[ 9], 11);
   JJ(cc, dd, ee, aa, bb, X[ 7],  6);
   JJ(bb, cc, dd, ee, aa, X[12],  8);
   JJ(aa, bb, cc, dd, ee, X[ 2], 13);
   JJ(ee, aa, bb, cc, dd, X[10], 12);
   JJ(dd, ee, aa, bb, cc, X[14],  5);
   JJ(cc, dd, ee, aa, bb, X[ 1], 12);
   JJ(bb, cc, dd, ee, aa, X[ 3], 13);
   JJ(aa, bb, cc, dd, ee, X[ 8], 14);
   JJ(ee, aa, bb, cc, dd, X[11], 11);
   JJ(dd, ee, aa, bb, cc, X[ 6],  8);
   JJ(cc, dd, ee, aa, bb, X[15],  5);
   JJ(bb, cc, dd, ee, aa, X[13],  6);

   /* parallel round 5 */
   FFF(bbb, ccc, ddd, eee, aaa, X[12] ,  8);
   FFF(aaa, bbb, ccc, ddd, eee, X[15] ,  5);
   FFF(eee, aaa, bbb, ccc, ddd, X[10] , 12);
   FFF(ddd, eee, aaa, bbb, ccc, X[ 4] ,  9);
   FFF(ccc, ddd, eee, aaa, bbb, X[ 1] , 12);
   FFF(bbb, ccc, ddd, eee, aaa, X[ 5] ,  5);
   FFF(aaa, bbb, ccc, ddd, eee, X[ 8] , 14);
   FFF(eee, aaa, bbb, ccc, ddd, X[ 7] ,  6);
   FFF(ddd, eee, aaa, bbb, ccc, X[ 6] ,  8);
   FFF(ccc, ddd, eee, aaa, bbb, X[ 2] , 13);
   FFF(bbb, ccc, ddd, eee, aaa, X[13] ,  6);
   FFF(aaa, bbb, ccc, ddd, eee, X[14] ,  5);
   FFF(eee, aaa, bbb, ccc, ddd, X[ 0] , 15);
   FFF(ddd, eee, aaa, bbb, ccc, X[ 3] , 13);
   FFF(ccc, ddd, eee, aaa, bbb, X[ 9] , 11);
   FFF(bbb, ccc, ddd, eee, aaa, X[11] , 11);

   tmp = ee; ee = eee; eee = tmp;

   /* combine results */
   md->rmd320.state[0] += aa;
   md->rmd320.state[1] += bb;
   md->rmd320.state[2] += cc;
   md->rmd320.state[3] += dd;
   md->rmd320.state[4] += ee;
   md->rmd320.state[5] += aaa;
   md->rmd320.state[6] += bbb;
   md->rmd320.state[7] += ccc;
   md->rmd320.state[8] += ddd;
   md->rmd320.state[9] += eee;

   return CRYPT_OK;
}
```

5.63.3.2 int rmd320_done (hash_state ∗ *md*, unsigned char ∗ *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (20 bytes)

Returns:

CRYPT_OK if successful

Definition at line 376 of file rmd320.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and rmd320_compress().

Referenced by rmd320_test().

```
{
    int i;

    LTC_ARGCHK(md  != NULL);
    LTC_ARGCHK(out != NULL);

    if (md->rmd320.curlen >= sizeof(md->rmd320.buf)) {
       return CRYPT_INVALID_ARG;
    }


    /* increase the length of the message */
    md->rmd320.length += md->rmd320.curlen * 8;

    /* append the '1' bit */
    md->rmd320.buf[md->rmd320.curlen++] = (unsigned char)0x80;

    /* if the length is currently above 56 bytes we append zeros
     * then compress.  Then we can fall back to padding zeros and length
     * encoding like normal.
     */
    if (md->rmd320.curlen > 56) {
        while (md->rmd320.curlen < 64) {
            md->rmd320.buf[md->rmd320.curlen++] = (unsigned char)0;
        }
        rmd320_compress(md, md->rmd320.buf);
        md->rmd320.curlen = 0;
    }

    /* pad upto 56 bytes of zeroes */
    while (md->rmd320.curlen < 56) {
        md->rmd320.buf[md->rmd320.curlen++] = (unsigned char)0;
    }

    /* store length */
    STORE64L(md->rmd320.length, md->rmd320.buf+56);
    rmd320_compress(md, md->rmd320.buf);

    /* copy output */
    for (i = 0; i < 10; i++) {
        STORE32L(md->rmd320.state[i], out+(4*i));
    }
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.63.3.3 int rmd320_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 343 of file rmd320.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by rmd320_test().

```
{
   LTC_ARGCHK(md != NULL);
   md->rmd320.state[0] = 0x67452301UL;
   md->rmd320.state[1] = 0xefcdab89UL;
   md->rmd320.state[2] = 0x98badcfeUL;
   md->rmd320.state[3] = 0x10325476UL;
   md->rmd320.state[4] = 0xc3d2e1f0UL;
   md->rmd320.state[5] = 0x76543210UL;
   md->rmd320.state[6] = 0xfedcba98UL;
   md->rmd320.state[7] = 0x89abcdefUL;
   md->rmd320.state[8] = 0x01234567UL;
   md->rmd320.state[9] = 0x3c2d1e0fUL;
   md->rmd320.curlen   = 0;
   md->rmd320.length   = 0;
   return CRYPT_OK;
}
```

5.63.3.4 int rmd320_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 429 of file rmd320.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, rmd320_done(), rmd320_init(), and XMEMCMP.

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   static const struct {
        char *msg;
        unsigned char md[40];
   } tests[] = {
   { "",
     { 0x22, 0xd6, 0x5d, 0x56, 0x61, 0x53, 0x6c, 0xdc, 0x75, 0xc1,
       0xfd, 0xf5, 0xc6, 0xde, 0x7b, 0x41, 0xb9, 0xf2, 0x73, 0x25,
       0xeb, 0xc6, 0x1e, 0x85, 0x57, 0x17, 0x7d, 0x70, 0x5a, 0x0e,
       0xc8, 0x80, 0x15, 0x1c, 0x3a, 0x32, 0xa0, 0x08, 0x99, 0xb8 }
   },
```

```
    { "a",
      { 0xce, 0x78, 0x85, 0x06, 0x38, 0xf9, 0x26, 0x58, 0xa5, 0xa5,
        0x85, 0x09, 0x75, 0x79, 0x92, 0x6d, 0xda, 0x66, 0x7a, 0x57,
        0x16, 0x56, 0x2c, 0xfc, 0xf6, 0xfb, 0xe7, 0x7f, 0x63, 0x54,
        0x2f, 0x99, 0xb0, 0x47, 0x05, 0xd6, 0x97, 0x0d, 0xff, 0x5d }
    },
    { "abc",
      { 0xde, 0x4c, 0x01, 0xb3, 0x05, 0x4f, 0x89, 0x30, 0xa7, 0x9d,
        0x09, 0xae, 0x73, 0x8e, 0x92, 0x30, 0x1e, 0x5a, 0x17, 0x08,
        0x5b, 0xef, 0xfd, 0xc1, 0xb8, 0xd1, 0x16, 0x71, 0x3e, 0x74,
        0xf8, 0x2f, 0xa9, 0x42, 0xd6, 0x4c, 0xdb, 0xc4, 0x68, 0x2d }
    },
    { "message digest",
      { 0x3a, 0x8e, 0x28, 0x50, 0x2e, 0xd4, 0x5d, 0x42, 0x2f, 0x68,
        0x84, 0x4f, 0x9d, 0xd3, 0x16, 0xe7, 0xb9, 0x85, 0x33, 0xfa,
        0x3f, 0x2a, 0x91, 0xd2, 0x9f, 0x84, 0xd4, 0x25, 0xc8, 0x8d,
        0x6b, 0x4e, 0xff, 0x72, 0x7d, 0xf6, 0x6a, 0x7c, 0x01, 0x97 }
    },
    { "abcdefghijklmnopqrstuvwxyz",
      { 0xca, 0xbd, 0xb1, 0x81, 0x0b, 0x92, 0x47, 0x0a, 0x20, 0x93,
        0xaa, 0x6b, 0xce, 0x05, 0x95, 0x2c, 0x28, 0x34, 0x8c, 0xf4,
        0x3f, 0xf6, 0x08, 0x41, 0x97, 0x51, 0x66, 0xbb, 0x40, 0xed,
        0x23, 0x40, 0x04, 0xb8, 0x82, 0x44, 0x63, 0xe6, 0xb0, 0x09 }
    },
    { "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq",
      { 0xd0, 0x34, 0xa7, 0x95, 0x0c, 0xf7, 0x22, 0x02, 0x1b, 0xa4,
        0xb8, 0x4d, 0xf7, 0x69, 0xa5, 0xde, 0x20, 0x60, 0xe2, 0x59,
        0xdf, 0x4c, 0x9b, 0xb4, 0xa4, 0x26, 0x8c, 0x0e, 0x93, 0x5b,
        0xbc, 0x74, 0x70, 0xa9, 0x69, 0xc9, 0xd0, 0x72, 0xa1, 0xac }
    }
    };
    int x;
    unsigned char buf[40];
    hash_state md;

    for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
        rmd320_init(&md);
        rmd320_process(&md, (unsigned char *)tests[x].msg, strlen(tests[x].msg));
        rmd320_done(&md, buf);
        if (XMEMCMP(buf, tests[x].md, 40) != 0) {
#if 0
           printf("Failed test %d\n", x);
#endif
           return CRYPT_FAIL_TESTVECTOR;
        }
    }
    return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.63.4 Variable Documentation

5.63.4.1 const struct ltc_hash_descriptor rmd320_desc

Initial value:

```
{
    "rmd320",
    9,
    20,
    64,
```

```
   { 0 },
   0,

    &rmd320_init,
    &rmd320_process,
    &rmd320_done,
    &rmd320_test,
    NULL
}
```

Definition at line 20 of file rmd320.c.

Referenced by yarrow_start().

5.64 hashes/sha1.c File Reference

5.64.1 Detailed Description

SHA1 code by Tom St Denis.

Definition in file sha1.c.

`#include "tomcrypt.h"`

Include dependency graph for sha1.c:

Defines

- #define F0(x, y, z) (z ^ (x & (y ^ z)))
- #define F1(x, y, z) (x ^ y ^ z)
- #define F2(x, y, z) ((x & y) | (z & (x | y)))
- #define F3(x, y, z) (x ^ y ^ z)
- #define FF0(a, b, c, d, e, i) e = (ROLc(a, 5) + F0(b,c,d) + e + W[i] + 0x5a827999UL); b = ROLc(b, 30);
- #define FF1(a, b, c, d, e, i) e = (ROLc(a, 5) + F1(b,c,d) + e + W[i] + 0x6ed9eba1UL); b = ROLc(b, 30);
- #define FF2(a, b, c, d, e, i) e = (ROLc(a, 5) + F2(b,c,d) + e + W[i] + 0x8f1bbcdcUL); b = ROLc(b, 30);
- #define FF3(a, b, c, d, e, i) e = (ROLc(a, 5) + F3(b,c,d) + e + W[i] + 0xca62c1d6UL); b = ROLc(b, 30);

Functions

- static int sha1_compress (hash_state ∗md, unsigned char ∗buf)
- int sha1_init (hash_state ∗md)

  Initialize the hash state.

- int sha1_done (hash_state ∗md, unsigned char ∗out)

  Terminate the hash to get the digest.

- int sha1_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor sha1_desc

5.64.2 Define Documentation

5.64.2.1 #define F0(x, y, z) (z ^ (x & (y ^ z)))

Definition at line 39 of file sha1.c.

5.64.2.2 #define F1(x, y, z) (x ^ y ^ z)

Definition at line 40 of file sha1.c.

5.64.2.3 #define F2(x, y, z) ((x & y) | (z & (x | y)))

Definition at line 41 of file sha1.c.

5.64.2.4 #define F3(x, y, z) (x ^ y ^ z)

Definition at line 42 of file sha1.c.

5.64.2.5 #define FF0(a, b, c, d, e, i) e = (ROLc(a, 5) + F0(b,c,d) + e + W[i] + 0x5a827999UL); b = ROLc(b, 30);

5.64.2.6 #define FF1(a, b, c, d, e, i) e = (ROLc(a, 5) + F1(b,c,d) + e + W[i] + 0x6ed9eba1UL); b = ROLc(b, 30);

5.64.2.7 #define FF2(a, b, c, d, e, i) e = (ROLc(a, 5) + F2(b,c,d) + e + W[i] + 0x8f1bbcdcUL); b = ROLc(b, 30);

5.64.2.8 #define FF3(a, b, c, d, e, i) e = (ROLc(a, 5) + F3(b,c,d) + e + W[i] + 0xca62c1d6UL); b = ROLc(b, 30);

5.64.3 Function Documentation

5.64.3.1 static int sha1_compress (hash_state * *md*, unsigned char * *buf*) [static]

Definition at line 47 of file sha1.c.

References c.

Referenced by sha1_done().

```
{
    ulong32 a,b,c,d,e,W[80],i;
#ifdef LTC_SMALL_CODE
    ulong32 t;
#endif

    /* copy the state into 512-bits into W[0..15] */
    for (i = 0; i < 16; i++) {
        LOAD32H(W[i], buf + (4*i));
    }

    /* copy state */
    a = md->sha1.state[0];
    b = md->sha1.state[1];
    c = md->sha1.state[2];
    d = md->sha1.state[3];
    e = md->sha1.state[4];

    /* expand it */
    for (i = 16; i < 80; i++) {
        W[i] = ROL(W[i-3] ^ W[i-8] ^ W[i-14] ^ W[i-16], 1);
    }

```

```
    /* compress */
    /* round one */
    #define FF0(a,b,c,d,e,i) e = (ROLc(a, 5) + F0(b,c,d) + e + W[i] + 0x5a827999UL); b = ROLc(b, 30);
    #define FF1(a,b,c,d,e,i) e = (ROLc(a, 5) + F1(b,c,d) + e + W[i] + 0x6ed9eba1UL); b = ROLc(b, 30);
    #define FF2(a,b,c,d,e,i) e = (ROLc(a, 5) + F2(b,c,d) + e + W[i] + 0x8f1bbcdcUL); b = ROLc(b, 30);
    #define FF3(a,b,c,d,e,i) e = (ROLc(a, 5) + F3(b,c,d) + e + W[i] + 0xca62c1d6UL); b = ROLc(b, 30);

#ifdef LTC_SMALL_CODE

    for (i = 0; i < 20; ) {
       FF0(a,b,c,d,e,i++); t = e; e = d; d = c; c = b; b = a; a = t;
    }

    for (; i < 40; ) {
       FF1(a,b,c,d,e,i++); t = e; e = d; d = c; c = b; b = a; a = t;
    }

    for (; i < 60; ) {
       FF2(a,b,c,d,e,i++); t = e; e = d; d = c; c = b; b = a; a = t;
    }

    for (; i < 80; ) {
       FF3(a,b,c,d,e,i++); t = e; e = d; d = c; c = b; b = a; a = t;
    }

#else

    for (i = 0; i < 20; ) {
        FF0(a,b,c,d,e,i++);
        FF0(e,a,b,c,d,i++);
        FF0(d,e,a,b,c,i++);
        FF0(c,d,e,a,b,i++);
        FF0(b,c,d,e,a,i++);
     }

     /* round two */
     for (; i < 40; )  {
        FF1(a,b,c,d,e,i++);
        FF1(e,a,b,c,d,i++);
        FF1(d,e,a,b,c,i++);
        FF1(c,d,e,a,b,i++);
        FF1(b,c,d,e,a,i++);
     }

     /* round three */
     for (; i < 60; )  {
        FF2(a,b,c,d,e,i++);
        FF2(e,a,b,c,d,i++);
        FF2(d,e,a,b,c,i++);
        FF2(c,d,e,a,b,i++);
        FF2(b,c,d,e,a,i++);
     }

     /* round four */
     for (; i < 80; )  {
        FF3(a,b,c,d,e,i++);
        FF3(e,a,b,c,d,i++);
        FF3(d,e,a,b,c,i++);
        FF3(c,d,e,a,b,i++);
        FF3(b,c,d,e,a,i++);
     }
#endif

     #undef FF0
     #undef FF1
     #undef FF2
     #undef FF3
```

```
139
140     /* store */
141     md->sha1.state[0] = md->sha1.state[0] + a;
142     md->sha1.state[1] = md->sha1.state[1] + b;
143     md->sha1.state[2] = md->sha1.state[2] + c;
144     md->sha1.state[3] = md->sha1.state[3] + d;
145     md->sha1.state[4] = md->sha1.state[4] + e;
146
147     return CRYPT_OK;
148 }
```

5.64.3.2 int sha1_done (hash_state ∗ *md*, unsigned char ∗ *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (20 bytes)

Returns:

CRYPT_OK if successful

Definition at line 193 of file sha1.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and sha1_compress().

Referenced by sha1_test().

```
194 {
195     int i;
196
197     LTC_ARGCHK(md  != NULL);
198     LTC_ARGCHK(out != NULL);
199
200     if (md->sha1.curlen >= sizeof(md->sha1.buf)) {
201        return CRYPT_INVALID_ARG;
202     }
203
204     /* increase the length of the message */
205     md->sha1.length += md->sha1.curlen * 8;
206
207     /* append the '1' bit */
208     md->sha1.buf[md->sha1.curlen++] = (unsigned char)0x80;
209
210     /* if the length is currently above 56 bytes we append zeros
211      * then compress.  Then we can fall back to padding zeros and length
212      * encoding like normal.
213      */
214     if (md->sha1.curlen > 56) {
215         while (md->sha1.curlen < 64) {
216             md->sha1.buf[md->sha1.curlen++] = (unsigned char)0;
217         }
218         sha1_compress(md, md->sha1.buf);
219         md->sha1.curlen = 0;
220     }
221
222     /* pad upto 56 bytes of zeroes */
223     while (md->sha1.curlen < 56) {
224         md->sha1.buf[md->sha1.curlen++] = (unsigned char)0;
225     }
226
```

```
    /* store length */
    STORE64H(md->sha1.length, md->sha1.buf+56);
    sha1_compress(md, md->sha1.buf);

    /* copy output */
    for (i = 0; i < 5; i++) {
        STORE32H(md->sha1.state[i], out+(4*i));
    }
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.64.3.3 int sha1_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 165 of file sha1.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by sha1_test().

```
{
   LTC_ARGCHK(md != NULL);
   md->sha1.state[0] = 0x67452301UL;
   md->sha1.state[1] = 0xefcdab89UL;
   md->sha1.state[2] = 0x98badcfeUL;
   md->sha1.state[3] = 0x10325476UL;
   md->sha1.state[4] = 0xc3d2e1f0UL;
   md->sha1.curlen = 0;
   md->sha1.length = 0;
   return CRYPT_OK;
}
```

5.64.3.4 int sha1_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 245 of file sha1.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, sha1_done(), sha1_init(), and XMEMCMP.

```
{
 #ifndef LTC_TEST
```

```
     return CRYPT_NOP;
  #else
   static const struct {
       char *msg;
       unsigned char hash[20];
   } tests[] = {
     { "abc",
       { 0xa9, 0x99, 0x3e, 0x36, 0x47, 0x06, 0x81, 0x6a,
         0xba, 0x3e, 0x25, 0x71, 0x78, 0x50, 0xc2, 0x6c,
         0x9c, 0xd0, 0xd8, 0x9d }
     },
     { "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq",
       { 0x84, 0x98, 0x3E, 0x44, 0x1C, 0x3B, 0xD2, 0x6E,
         0xBA, 0xAE, 0x4A, 0xA1, 0xF9, 0x51, 0x29, 0xE5,
         0xE5, 0x46, 0x70, 0xF1 }
     }
   };

   int i;
   unsigned char tmp[20];
   hash_state md;

   for (i = 0; i < (int)(sizeof(tests) / sizeof(tests[0]));  i++) {
       sha1_init(&md);
       sha1_process(&md, (unsigned char*)tests[i].msg, (unsigned long)strlen(tests[i].msg));
       sha1_done(&md, tmp);
       if (XMEMCMP(tmp, tests[i].hash, 20) != 0) {
          return CRYPT_FAIL_TESTVECTOR;
       }
   }
   return CRYPT_OK;
   #endif
 }
```

Here is the call graph for this function:

5.64.4 Variable Documentation

5.64.4.1 const struct ltc_hash_descriptor sha1_desc

Initial value:

```
{
    "sha1",
    2,
    20,
    64,


   { 1, 3, 14, 3, 2, 26,  },
   6,

    &sha1_init,
    &sha1_process,
    &sha1_done,
    &sha1_test,
    NULL
}
```

Definition at line 21 of file sha1.c.

Referenced by yarrow_start().

5.65 hashes/sha2/sha224.c File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int sha224_init (hash_state *md)

  Initialize the hash state.

- int sha224_done (hash_state *md, unsigned char *out)

  Terminate the hash to get the digest.

- int sha224_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor sha224_desc

5.65.1 Function Documentation

5.65.1.1 int sha224_done (hash_state * *md*, unsigned char * *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (28 bytes)

Returns:

CRYPT_OK if successful

Definition at line 63 of file sha224.c.

References LTC_ARGCHK, sha256_done(), XMEMCPY, and zeromem().

Referenced by sha224_test().

```
{
    unsigned char buf[32];
    int err;

    LTC_ARGCHK(md  != NULL);
    LTC_ARGCHK(out != NULL);

    err = sha256_done(md, buf);
    XMEMCPY(out, buf, 28);
#ifdef LTC_CLEAN_STACK
    zeromem(buf, sizeof(buf));
#endif
    return err;
}
```

Here is the call graph for this function:

5.65.1.2 int sha224_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 40 of file sha224.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by sha224_test().

```
41 {
42     LTC_ARGCHK(md != NULL);
43
44     md->sha256.curlen = 0;
45     md->sha256.length = 0;
46     md->sha256.state[0] = 0xc1059ed8UL;
47     md->sha256.state[1] = 0x367cd507UL;
48     md->sha256.state[2] = 0x3070dd17UL;
49     md->sha256.state[3] = 0xf70e5939UL;
50     md->sha256.state[4] = 0xffc00b31UL;
51     md->sha256.state[5] = 0x68581511UL;
52     md->sha256.state[6] = 0x64f98fa7UL;
53     md->sha256.state[7] = 0xbefa4fa4UL;
54     return CRYPT_OK;
55 }
```

5.65.1.3 int sha224_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 83 of file sha224.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, sha224_done(), sha224_init(), and XMEM-CMP.

```
84 {
85  #ifndef LTC_TEST
86     return CRYPT_NOP;
87  #else
88   static const struct {
89       char *msg;
90       unsigned char hash[28];
91   } tests[] = {
92     { "abc",
93       { 0x23, 0x09, 0x7d, 0x22, 0x34, 0x05, 0xd8,
94         0x22, 0x86, 0x42, 0xa4, 0x77, 0xbd, 0xa2,
95         0x55, 0xb3, 0x2a, 0xad, 0xbc, 0xe4, 0xbd,
96         0xa0, 0xb3, 0xf7, 0xe3, 0x6c, 0x9d, 0xa7 }
97     },
98     { "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq",
```

```
     { 0x75, 0x38, 0x8b, 0x16, 0x51, 0x27, 0x76,
        0xcc, 0x5d, 0xba, 0x5d, 0xa1, 0xfd, 0x89,
        0x01, 0x50, 0xb0, 0xc6, 0x45, 0x5c, 0xb4,
        0xf5, 0x8b, 0x19, 0x52, 0x52, 0x25, 0x25 }
    },
  };

  int i;
  unsigned char tmp[28];
  hash_state md;

  for (i = 0; i < (int)(sizeof(tests) / sizeof(tests[0])); i++) {
      sha224_init(&md);
      sha224_process(&md, (unsigned char*)tests[i].msg, (unsigned long)strlen(tests[i].msg));
      sha224_done(&md, tmp);
      if (XMEMCMP(tmp, tests[i].hash, 28) != 0) {
         return CRYPT_FAIL_TESTVECTOR;
      }
  }
  return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.65.2 Variable Documentation

5.65.2.1 const struct ltc_hash_descriptor sha224_desc

Initial value:

```
{
    "sha224",
    10,
    28,
    64,


   { 2, 16, 840, 1, 101, 3, 4, 2, 4,   },
   9,

    &sha224_init,
    &sha256_process,
    &sha224_done,
    &sha224_test,
    NULL
}
```

Parameters:

sha224.c SHA-224 new NIST standard based off of SHA-256 truncated to 224 bits (Tom St Denis)

Definition at line 16 of file sha224.c.

5.66 hashes/sha2/sha256.c File Reference

5.66.1 Detailed Description

SHA256 by Tom St Denis.

Definition in file sha256.c.

```
#include "tomcrypt.h"
#include "sha224.c"
```

Include dependency graph for sha256.c:

Defines

- #define Ch(x, y, z) (z ^ (x & (y ^ z)))
- #define Maj(x, y, z) (((x | y) & z) | (x & y))
- #define S(x, n) RORc((x),(n))
- #define R(x, n) (((x)&0xFFFFFFFFUL)>>(n))
- #define Sigma0(x) (S(x, 2) ^ S(x, 13) ^ S(x, 22))
- #define Sigma1(x) (S(x, 6) ^ S(x, 11) ^ S(x, 25))
- #define Gamma0(x) (S(x, 7) ^ S(x, 18) ^ R(x, 3))
- #define Gamma1(x) (S(x, 17) ^ S(x, 19) ^ R(x, 10))
- #define RND(a, b, c, d, e, f, g, h, i, ki)

Functions

- static int sha256_compress (hash_state ∗md, unsigned char ∗buf)
- int sha256_init (hash_state ∗md)

    Initialize the hash state.

- int sha256_done (hash_state ∗md, unsigned char ∗out)

    Terminate the hash to get the digest.

- int sha256_test (void)

    Self-test the hash.

Variables

- const struct ltc_hash_descriptor sha256_desc

5.66.2 Define Documentation

5.66.2.1 #define Ch(x, y, z) (z ^ (x & (y ^ z)))

Definition at line 58 of file sha256.c.

5.66.2.2 #define Gamma0(x) (S(x, 7) ^ S(x, 18) ^ R(x, 3))

Definition at line 64 of file sha256.c.

5.66.2.3 #define Gamma1(x) (S(x, 17) ^ S(x, 19) ^ R(x, 10))

Definition at line 65 of file sha256.c.

5.66.2.4 #define Maj(x, y, z) (((x | y) & z) | (x & y))

Definition at line 59 of file sha256.c.

5.66.2.5 #define R(x, n) (((x)&0xFFFFFFFFUL)>>(n))

Definition at line 61 of file sha256.c.

5.66.2.6 #define RND(a, b, c, d, e, f, g, h, i, ki)

Value:

```
t0 = h + Sigma1(e) + Ch(e, f, g) + ki + W[i];   \
     t1 = Sigma0(a) + Maj(a, b, c);                  \
     d += t0;                                        \
     h  = t0 + t1;
```

5.66.2.7 #define S(x, n) RORc((x),(n))

Definition at line 60 of file sha256.c.

Referenced by khazad_setup(), rc5_setup(), rc6_setup(), sha256_compress(), sha512_compress(), and twofish_setup().

5.66.2.8 #define Sigma0(x) (S(x, 2) ^ S(x, 13) ^ S(x, 22))

Definition at line 62 of file sha256.c.

5.66.2.9 #define Sigma1(x) (S(x, 6) ^ S(x, 11) ^ S(x, 25))

Definition at line 63 of file sha256.c.

5.66.3 Function Documentation

5.66.3.1 static int sha256_compress (hash_state ∗ *md*, unsigned char ∗ *buf*) `[static]`

Definition at line 71 of file sha256.c.

References S, and t1.

Referenced by sha256_done().

```
{
    ulong32 S[8], W[64], t0, t1;
#ifdef LTC_SMALL_CODE
    ulong32 t;
#endif
    int i;

    /* copy state into S */
    for (i = 0; i < 8; i++) {
        S[i] = md->sha256.state[i];
    }

    /* copy the state into 512-bits into W[0..15] */
    for (i = 0; i < 16; i++) {
        LOAD32H(W[i], buf + (4*i));
    }

    /* fill W[16..63] */
    for (i = 16; i < 64; i++) {
        W[i] = Gamma1(W[i - 2]) + W[i - 7] + Gamma0(W[i - 15]) + W[i - 16];
    }

    /* Compress */
#ifdef LTC_SMALL_CODE
#define RND(a,b,c,d,e,f,g,h,i)                         \
     t0 = h + Sigma1(e) + Ch(e, f, g) + K[i] + W[i];   \
     t1 = Sigma0(a) + Maj(a, b, c);                    \
     d += t0;                                          \
     h  = t0 + t1;

     for (i = 0; i < 64; ++i) {
         RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],i);
         t = S[7]; S[7] = S[6]; S[6] = S[5]; S[5] = S[4];
         S[4] = S[3]; S[3] = S[2]; S[2] = S[1]; S[1] = S[0]; S[0] = t;
     }
#else
#define RND(a,b,c,d,e,f,g,h,i,ki)                    \
     t0 = h + Sigma1(e) + Ch(e, f, g) + ki + W[i];   \
     t1 = Sigma0(a) + Maj(a, b, c);                  \
     d += t0;                                        \
     h  = t0 + t1;

    RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],0,0x428a2f98);
    RND(S[7],S[0],S[1],S[2],S[3],S[4],S[5],S[6],1,0x71374491);
    RND(S[6],S[7],S[0],S[1],S[2],S[3],S[4],S[5],2,0xb5c0fbcf);
    RND(S[5],S[6],S[7],S[0],S[1],S[2],S[3],S[4],3,0xe9b5dba5);
    RND(S[4],S[5],S[6],S[7],S[0],S[1],S[2],S[3],4,0x3956c25b);
    RND(S[3],S[4],S[5],S[6],S[7],S[0],S[1],S[2],5,0x59f111f1);
    RND(S[2],S[3],S[4],S[5],S[6],S[7],S[0],S[1],6,0x923f82a4);
    RND(S[1],S[2],S[3],S[4],S[5],S[6],S[7],S[0],7,0xab1c5ed5);
    RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],8,0xd807aa98);
    RND(S[7],S[0],S[1],S[2],S[3],S[4],S[5],S[6],9,0x12835b01);
    RND(S[6],S[7],S[0],S[1],S[2],S[3],S[4],S[5],10,0x243185be);
    RND(S[5],S[6],S[7],S[0],S[1],S[2],S[3],S[4],11,0x550c7dc3);
    RND(S[4],S[5],S[6],S[7],S[0],S[1],S[2],S[3],12,0x72be5d74);
    RND(S[3],S[4],S[5],S[6],S[7],S[0],S[1],S[2],13,0x80deb1fe);
    RND(S[2],S[3],S[4],S[5],S[6],S[7],S[0],S[1],14,0x9bdc06a7);
    RND(S[1],S[2],S[3],S[4],S[5],S[6],S[7],S[0],15,0xc19bf174);
    RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],16,0xe49b69c1);
    RND(S[7],S[0],S[1],S[2],S[3],S[4],S[5],S[6],17,0xefbe4786);
    RND(S[6],S[7],S[0],S[1],S[2],S[3],S[4],S[5],18,0x0fc19dc6);
    RND(S[5],S[6],S[7],S[0],S[1],S[2],S[3],S[4],19,0x240ca1cc);
    RND(S[4],S[5],S[6],S[7],S[0],S[1],S[2],S[3],20,0x2de92c6f);
    RND(S[3],S[4],S[5],S[6],S[7],S[0],S[1],S[2],21,0x4a7484aa);
    RND(S[2],S[3],S[4],S[5],S[6],S[7],S[0],S[1],22,0x5cb0a9dc);
    RND(S[1],S[2],S[3],S[4],S[5],S[6],S[7],S[0],23,0x76f988da);
    RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],24,0x983e5152);
```

```
    RND(S[7],S[0],S[1],S[2],S[3],S[4],S[5],S[6],25,0xa831c66d);
    RND(S[6],S[7],S[0],S[1],S[2],S[3],S[4],S[5],26,0xb00327c8);
    RND(S[5],S[6],S[7],S[0],S[1],S[2],S[3],S[4],27,0xbf597fc7);
    RND(S[4],S[5],S[6],S[7],S[0],S[1],S[2],S[3],28,0xc6e00bf3);
    RND(S[3],S[4],S[5],S[6],S[7],S[0],S[1],S[2],29,0xd5a79147);
    RND(S[2],S[3],S[4],S[5],S[6],S[7],S[0],S[1],30,0x06ca6351);
    RND(S[1],S[2],S[3],S[4],S[5],S[6],S[7],S[0],31,0x14292967);
    RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],32,0x27b70a85);
    RND(S[7],S[0],S[1],S[2],S[3],S[4],S[5],S[6],33,0x2e1b2138);
    RND(S[6],S[7],S[0],S[1],S[2],S[3],S[4],S[5],34,0x4d2c6dfc);
    RND(S[5],S[6],S[7],S[0],S[1],S[2],S[3],S[4],35,0x53380d13);
    RND(S[4],S[5],S[6],S[7],S[0],S[1],S[2],S[3],36,0x650a7354);
    RND(S[3],S[4],S[5],S[6],S[7],S[0],S[1],S[2],37,0x766a0abb);
    RND(S[2],S[3],S[4],S[5],S[6],S[7],S[0],S[1],38,0x81c2c92e);
    RND(S[1],S[2],S[3],S[4],S[5],S[6],S[7],S[0],39,0x92722c85);
    RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],40,0xa2bfe8a1);
    RND(S[7],S[0],S[1],S[2],S[3],S[4],S[5],S[6],41,0xa81a664b);
    RND(S[6],S[7],S[0],S[1],S[2],S[3],S[4],S[5],42,0xc24b8b70);
    RND(S[5],S[6],S[7],S[0],S[1],S[2],S[3],S[4],43,0xc76c51a3);
    RND(S[4],S[5],S[6],S[7],S[0],S[1],S[2],S[3],44,0xd192e819);
    RND(S[3],S[4],S[5],S[6],S[7],S[0],S[1],S[2],45,0xd6990624);
    RND(S[2],S[3],S[4],S[5],S[6],S[7],S[0],S[1],46,0xf40e3585);
    RND(S[1],S[2],S[3],S[4],S[5],S[6],S[7],S[0],47,0x106aa070);
    RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],48,0x19a4c116);
    RND(S[7],S[0],S[1],S[2],S[3],S[4],S[5],S[6],49,0x1e376c08);
    RND(S[6],S[7],S[0],S[1],S[2],S[3],S[4],S[5],50,0x2748774c);
    RND(S[5],S[6],S[7],S[0],S[1],S[2],S[3],S[4],51,0x34b0bcb5);
    RND(S[4],S[5],S[6],S[7],S[0],S[1],S[2],S[3],52,0x391c0cb3);
    RND(S[3],S[4],S[5],S[6],S[7],S[0],S[1],S[2],53,0x4ed8aa4a);
    RND(S[2],S[3],S[4],S[5],S[6],S[7],S[0],S[1],54,0x5b9cca4f);
    RND(S[1],S[2],S[3],S[4],S[5],S[6],S[7],S[0],55,0x682e6ff3);
    RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],56,0x748f82ee);
    RND(S[7],S[0],S[1],S[2],S[3],S[4],S[5],S[6],57,0x78a5636f);
    RND(S[6],S[7],S[0],S[1],S[2],S[3],S[4],S[5],58,0x84c87814);
    RND(S[5],S[6],S[7],S[0],S[1],S[2],S[3],S[4],59,0x8cc70208);
    RND(S[4],S[5],S[6],S[7],S[0],S[1],S[2],S[3],60,0x90befffa);
    RND(S[3],S[4],S[5],S[6],S[7],S[0],S[1],S[2],61,0xa4506ceb);
    RND(S[2],S[3],S[4],S[5],S[6],S[7],S[0],S[1],62,0xbef9a3f7);
    RND(S[1],S[2],S[3],S[4],S[5],S[6],S[7],S[0],63,0xc67178f2);

#undef RND

#endif

    /* feedback */
    for (i = 0; i < 8; i++) {
        md->sha256.state[i] = md->sha256.state[i] + S[i];
    }
    return CRYPT_OK;
}
```

5.66.3.2 int sha256_done (hash_state * *md*, unsigned char * *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (32 bytes)

Returns:

CRYPT_OK if successful

Definition at line 238 of file sha256.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and sha256_compress().

Referenced by fortuna_done(), fortuna_export(), fortuna_reseed(), fortuna_start(), sha224_done(), and sha256_test().

```
239 {
240     int i;
241 
242     LTC_ARGCHK(md  != NULL);
243     LTC_ARGCHK(out != NULL);
244 
245     if (md->sha256.curlen >= sizeof(md->sha256.buf)) {
246        return CRYPT_INVALID_ARG;
247     }
248 
249 
250     /* increase the length of the message */
251     md->sha256.length += md->sha256.curlen * 8;
252 
253     /* append the '1' bit */
254     md->sha256.buf[md->sha256.curlen++] = (unsigned char)0x80;
255 
256     /* if the length is currently above 56 bytes we append zeros
257      * then compress.  Then we can fall back to padding zeros and length
258      * encoding like normal.
259      */
260     if (md->sha256.curlen > 56) {
261         while (md->sha256.curlen < 64) {
262             md->sha256.buf[md->sha256.curlen++] = (unsigned char)0;
263         }
264         sha256_compress(md, md->sha256.buf);
265         md->sha256.curlen = 0;
266     }
267 
268     /* pad upto 56 bytes of zeroes */
269     while (md->sha256.curlen < 56) {
270         md->sha256.buf[md->sha256.curlen++] = (unsigned char)0;
271     }
272 
273     /* store length */
274     STORE64H(md->sha256.length, md->sha256.buf+56);
275     sha256_compress(md, md->sha256.buf);
276 
277     /* copy output */
278     for (i = 0; i < 8; i++) {
279         STORE32H(md->sha256.state[i], out+(4*i));
280     }
281 #ifdef LTC_CLEAN_STACK
282     zeromem(md, sizeof(hash_state));
283 #endif
284     return CRYPT_OK;
285 }
```

Here is the call graph for this function:

5.66.3.3 int sha256_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 206 of file sha256.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by fortuna_reseed(), fortuna_start(), and sha256_test().

```
{
    LTC_ARGCHK(md != NULL);

    md->sha256.curlen = 0;
    md->sha256.length = 0;
    md->sha256.state[0] = 0x6A09E667UL;
    md->sha256.state[1] = 0xBB67AE85UL;
    md->sha256.state[2] = 0x3C6EF372UL;
    md->sha256.state[3] = 0xA54FF53AUL;
    md->sha256.state[4] = 0x510E527FUL;
    md->sha256.state[5] = 0x9B05688CUL;
    md->sha256.state[6] = 0x1F83D9ABUL;
    md->sha256.state[7] = 0x5BE0CD19UL;
    return CRYPT_OK;
}
```

5.66.3.4 int sha256_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 291 of file sha256.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, sha256_done(), sha256_init(), and XMEMCMP.

Referenced by fortuna_test().

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
  static const struct {
      char *msg;
      unsigned char hash[32];
  } tests[] = {
    { "abc",
      { 0xba, 0x78, 0x16, 0xbf, 0x8f, 0x01, 0xcf, 0xea,
        0x41, 0x41, 0x40, 0xde, 0x5d, 0xae, 0x22, 0x23,
        0xb0, 0x03, 0x61, 0xa3, 0x96, 0x17, 0x7a, 0x9c,
        0xb4, 0x10, 0xff, 0x61, 0xf2, 0x00, 0x15, 0xad }
    },
    { "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq",
      { 0x24, 0x8d, 0x6a, 0x61, 0xd2, 0x06, 0x38, 0xb8,
        0xe5, 0xc0, 0x26, 0x93, 0x0c, 0x3e, 0x60, 0x39,
        0xa3, 0x3c, 0xe4, 0x59, 0x64, 0xff, 0x21, 0x67,
        0xf6, 0xec, 0xed, 0xd4, 0x19, 0xdb, 0x06, 0xc1 }
    },
  };

```

```
  int i;
  unsigned char tmp[32];
  hash_state md;

  for (i = 0; i < (int)(sizeof(tests) / sizeof(tests[0])); i++) {
      sha256_init(&md);
      sha256_process(&md, (unsigned char*)tests[i].msg, (unsigned long)strlen(tests[i].msg));
      sha256_done(&md, tmp);
      if (XMEMCMP(tmp, tests[i].hash, 32) != 0) {
         return CRYPT_FAIL_TESTVECTOR;
      }
  }
  return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.66.4 Variable Documentation

5.66.4.1 const struct ltc_hash_descriptor sha256_desc

Initial value:

```
{
    "sha256",
    0,
    32,
    64,


   { 2, 16, 840, 1, 101, 3, 4, 2, 1,  },
   9,

    &sha256_init,
    &sha256_process,
    &sha256_done,
    &sha256_test,
    NULL
}
```

Definition at line 20 of file sha256.c.

Referenced by yarrow_start().

5.67 hashes/sha2/sha384.c File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int sha384_init (hash_state ∗md)

    Initialize the hash state.

- int sha384_done (hash_state ∗md, unsigned char ∗out)

    Terminate the hash to get the digest.

- int sha384_test (void)

    Self-test the hash.

Variables

- const struct ltc_hash_descriptor sha384_desc

5.67.1 Function Documentation

5.67.1.1 int sha384_done (hash_state ∗ *md*, unsigned char ∗ *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (48 bytes)

Returns:

CRYPT_OK if successful

Definition at line 62 of file sha384.c.

References CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, sha512_done(), XMEMCPY, and zeromem().

Referenced by sha384_test().

```
{
   unsigned char buf[64];

   LTC_ARGCHK(md  != NULL);
   LTC_ARGCHK(out != NULL);

    if (md->sha512.curlen >= sizeof(md->sha512.buf)) {
       return CRYPT_INVALID_ARG;
    }

   sha512_done(md, buf);
   XMEMCPY(out, buf, 48);
#ifdef LTC_CLEAN_STACK
```

```
   zeromem(buf, sizeof(buf));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.67.1.2 int sha384_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 39 of file sha384.c.

References CONST64, CRYPT_OK, and LTC_ARGCHK.

Referenced by sha384_test().

```
{
    LTC_ARGCHK(md != NULL);

    md->sha512.curlen = 0;
    md->sha512.length = 0;
    md->sha512.state[0] = CONST64(0xcbbb9d5dc1059ed8);
    md->sha512.state[1] = CONST64(0x629a292a367cd507);
    md->sha512.state[2] = CONST64(0x9159015a3070dd17);
    md->sha512.state[3] = CONST64(0x152fecd8f70e5939);
    md->sha512.state[4] = CONST64(0x67332667ffc00b31);
    md->sha512.state[5] = CONST64(0x8eb44a8768581511);
    md->sha512.state[6] = CONST64(0xdb0c2e0d64f98fa7);
    md->sha512.state[7] = CONST64(0x47b5481dbefa4fa4);
    return CRYPT_OK;
}
```

5.67.1.3 int sha384_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 85 of file sha384.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, sha384_done(), sha384_init(), and XMEM-CMP.

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
  static const struct {
      char *msg;
```

```
      unsigned char hash[48];
  } tests[] = {
    { "abc",
      { 0xcb, 0x00, 0x75, 0x3f, 0x45, 0xa3, 0x5e, 0x8b,
        0xb5, 0xa0, 0x3d, 0x69, 0x9a, 0xc6, 0x50, 0x07,
        0x27, 0x2c, 0x32, 0xab, 0x0e, 0xde, 0xd1, 0x63,
        0x1a, 0x8b, 0x60, 0x5a, 0x43, 0xff, 0x5b, 0xed,
        0x80, 0x86, 0x07, 0x2b, 0xa1, 0xe7, 0xcc, 0x23,
        0x58, 0xba, 0xec, 0xa1, 0x34, 0xc8, 0x25, 0xa7 }
    },
    { "abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmnoijklmnopjklmnopqklmnopqrlmnopqr
      { 0x09, 0x33, 0x0c, 0x33, 0xf7, 0x11, 0x47, 0xe8,
        0x3d, 0x19, 0x2f, 0xc7, 0x82, 0xcd, 0x1b, 0x47,
        0x53, 0x11, 0x1b, 0x17, 0x3b, 0x3b, 0x05, 0xd2,
        0x2f, 0xa0, 0x80, 0x86, 0xe3, 0xb0, 0xf7, 0x12,
        0xfc, 0xc7, 0xc7, 0x1a, 0x55, 0x7e, 0x2d, 0xb9,
        0x66, 0xc3, 0xe9, 0xfa, 0x91, 0x74, 0x60, 0x39 }
    },
  };

  int i;
  unsigned char tmp[48];
  hash_state md;

  for (i = 0; i < (int)(sizeof(tests) / sizeof(tests[0])); i++) {
      sha384_init(&md);
      sha384_process(&md, (unsigned char*)tests[i].msg, (unsigned long)strlen(tests[i].msg));
      sha384_done(&md, tmp);
      if (XMEMCMP(tmp, tests[i].hash, 48) != 0) {
         return CRYPT_FAIL_TESTVECTOR;
      }
  }
  return CRYPT_OK;
 #endif
}
```

Here is the call graph for this function:

5.67.2 Variable Documentation

5.67.2.1 const struct ltc_hash_descriptor sha384_desc

Initial value:

```
{
    "sha384",
    4,
    48,
    128,


   { 2, 16, 840, 1, 101, 3, 4, 2, 2,  },
   9,

    &sha384_init,
    &sha512_process,
    &sha384_done,
    &sha384_test,
    NULL
}
```

Parameters:

sha384.c SHA384 hash included in sha512.c, Tom St Denis

Definition at line 16 of file sha384.c.

5.68 hashes/sha2/sha512.c File Reference

`#include "tomcrypt.h"`

`#include "sha384.c"`

Include dependency graph for sha512.c:

Defines

- #define Ch(x, y, z) (z ^ (x & (y ^ z)))
- #define Maj(x, y, z) (((x | y) & z) | (x & y))
- #define S(x, n) ROR64c(x, n)
- #define R(x, n) (((x)&CONST64(0xFFFFFFFFFFFFFFFF))>>((ulong64)n))
- #define Sigma0(x) (S(x, 28) ^ S(x, 34) ^ S(x, 39))
- #define Sigma1(x) (S(x, 14) ^ S(x, 18) ^ S(x, 41))
- #define Gamma0(x) (S(x, 1) ^ S(x, 8) ^ R(x, 7))
- #define Gamma1(x) (S(x, 19) ^ S(x, 61) ^ R(x, 6))
- #define RND(a, b, c, d, e, f, g, h, i)

Functions

- static int sha512_compress (hash_state *md, unsigned char *buf)
- int sha512_init (hash_state *md)

  Initialize the hash state.

- int sha512_done (hash_state *md, unsigned char *out)

  Terminate the hash to get the digest.

- int sha512_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor sha512_desc
- static const ulong64 K [80]

5.68.1 Define Documentation

5.68.1.1 #define Ch(x, y, z) (z ^ (x & (y ^ z)))

Definition at line 83 of file sha512.c.

5.68.1.2 #define Gamma0(x) (S(x, 1) ^ S(x, 8) ^ R(x, 7))

Definition at line 89 of file sha512.c.

5.68.1.3 #define Gamma1(x) (S(x, 19) ^ S(x, 61) ^ R(x, 6))

Definition at line 90 of file sha512.c.

5.68.1.4 #define Maj(x, y, z) (((x | y) & z) | (x & y))

Definition at line 84 of file sha512.c.

5.68.1.5 #define R(x, n) (((x)&CONST64(0xFFFFFFFFFFFFFFFF))>>((ulong64)n))

Definition at line 86 of file sha512.c.

5.68.1.6 #define RND(a, b, c, d, e, f, g, h, i)

Value:

```
t0 = h + Sigma1(e) + Ch(e, f, g) + K[i] + W[i];   \
     t1 = Sigma0(a) + Maj(a, b, c);                  \
     d += t0;                                        \
     h  = t0 + t1;
```

5.68.1.7 #define S(x, n) ROR64c(x, n)

Definition at line 85 of file sha512.c.

5.68.1.8 #define Sigma0(x) (S(x, 28) ^ S(x, 34) ^ S(x, 39))

Definition at line 87 of file sha512.c.

5.68.1.9 #define Sigma1(x) (S(x, 14) ^ S(x, 18) ^ S(x, 41))

Definition at line 88 of file sha512.c.

5.68.2 Function Documentation

5.68.2.1 static int sha512_compress (hash_state * *md*, unsigned char * *buf*) [static]

Definition at line 96 of file sha512.c.

References S, and t1.

Referenced by sha512_done().

```
{
    ulong64 S[8], W[80], t0, t1;
    int i;

    /* copy state into S */
    for (i = 0; i < 8; i++) {
        S[i] = md->sha512.state[i];
    }
```

```

    /* copy the state into 1024-bits into W[0..15] */
    for (i = 0; i < 16; i++) {
        LOAD64H(W[i], buf + (8*i));
    }

    /* fill W[16..79] */
    for (i = 16; i < 80; i++) {
        W[i] = Gamma1(W[i - 2]) + W[i - 7] + Gamma0(W[i - 15]) + W[i - 16];
    }

    /* Compress */
#ifdef LTC_SMALL_CODE
    for (i = 0; i < 80; i++) {
        t0 = S[7] + Sigma1(S[4]) + Ch(S[4], S[5], S[6]) + K[i] + W[i];
        t1 = Sigma0(S[0]) + Maj(S[0], S[1], S[2]);
        S[7] = S[6];
        S[6] = S[5];
        S[5] = S[4];
        S[4] = S[3] + t0;
        S[3] = S[2];
        S[2] = S[1];
        S[1] = S[0];
        S[0] = t0 + t1;
    }
#else
#define RND(a,b,c,d,e,f,g,h,i)                    \
     t0 = h + Sigma1(e) + Ch(e, f, g) + K[i] + W[i];   \
     t1 = Sigma0(a) + Maj(a, b, c);                  \
     d += t0;                                        \
     h  = t0 + t1;

     for (i = 0; i < 80; i += 8) {
         RND(S[0],S[1],S[2],S[3],S[4],S[5],S[6],S[7],i+0);
         RND(S[7],S[0],S[1],S[2],S[3],S[4],S[5],S[6],i+1);
         RND(S[6],S[7],S[0],S[1],S[2],S[3],S[4],S[5],i+2);
         RND(S[5],S[6],S[7],S[0],S[1],S[2],S[3],S[4],i+3);
         RND(S[4],S[5],S[6],S[7],S[0],S[1],S[2],S[3],i+4);
         RND(S[3],S[4],S[5],S[6],S[7],S[0],S[1],S[2],i+5);
         RND(S[2],S[3],S[4],S[5],S[6],S[7],S[0],S[1],i+6);
         RND(S[1],S[2],S[3],S[4],S[5],S[6],S[7],S[0],i+7);
     }
#endif


    /* feedback */
    for (i = 0; i < 8; i++) {
        md->sha512.state[i] = md->sha512.state[i] + S[i];
    }

    return CRYPT_OK;
}
```

5.68.2.2 int sha512_done (hash_state * *md*, unsigned char * *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (64 bytes)

Returns:

CRYPT_OK if successful

Definition at line 206 of file sha512.c.

References CONST64, CRYPT_INVALID_ARG, LTC_ARGCHK, and sha512_compress().

Referenced by sha384_done(), and sha512_test().

```
{
    int i;

    LTC_ARGCHK(md  != NULL);
    LTC_ARGCHK(out != NULL);

    if (md->sha512.curlen >= sizeof(md->sha512.buf)) {
       return CRYPT_INVALID_ARG;
    }

    /* increase the length of the message */
    md->sha512.length += md->sha512.curlen * CONST64(8);

    /* append the '1' bit */
    md->sha512.buf[md->sha512.curlen++] = (unsigned char)0x80;

    /* if the length is currently above 112 bytes we append zeros
     * then compress.  Then we can fall back to padding zeros and length
     * encoding like normal.
     */
    if (md->sha512.curlen > 112) {
        while (md->sha512.curlen < 128) {
            md->sha512.buf[md->sha512.curlen++] = (unsigned char)0;
        }
        sha512_compress(md, md->sha512.buf);
        md->sha512.curlen = 0;
    }

    /* pad upto 120 bytes of zeroes
     * note: that from 112 to 120 is the 64 MSB of the length.  We assume that you won't hash
     * > 2^64 bits of data... :-)
     */
    while (md->sha512.curlen < 120) {
        md->sha512.buf[md->sha512.curlen++] = (unsigned char)0;
    }

    /* store length */
    STORE64H(md->sha512.length, md->sha512.buf+120);
    sha512_compress(md, md->sha512.buf);

    /* copy output */
    for (i = 0; i < 8; i++) {
        STORE64H(md->sha512.state[i], out+(8*i));
    }
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.68.2.3 int sha512_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:
CRYPT_OK if successful

Definition at line 175 of file sha512.c.

References CONST64, CRYPT_OK, and LTC_ARGCHK.

Referenced by sha512_test().

```
176 {
177     LTC_ARGCHK(md != NULL);
178     md->sha512.curlen = 0;
179     md->sha512.length = 0;
180     md->sha512.state[0] = CONST64(0x6a09e667f3bcc908);
181     md->sha512.state[1] = CONST64(0xbb67ae8584caa73b);
182     md->sha512.state[2] = CONST64(0x3c6ef372fe94f82b);
183     md->sha512.state[3] = CONST64(0xa54ff53a5f1d36f1);
184     md->sha512.state[4] = CONST64(0x510e527fade682d1);
185     md->sha512.state[5] = CONST64(0x9b05688c2b3e6c1f);
186     md->sha512.state[6] = CONST64(0x1f83d9abfb41bd6b);
187     md->sha512.state[7] = CONST64(0x5be0cd19137e2179);
188     return CRYPT_OK;
189 }
```

5.68.2.4 int sha512_test (void)

Self-test the hash.

Returns:
CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 261 of file sha512.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, sha512_done(), sha512_init(), and XMEMCMP.

```
262 {
263  #ifndef LTC_TEST
264     return CRYPT_NOP;
265  #else
266   static const struct {
267       char *msg;
268       unsigned char hash[64];
269   } tests[] = {
270     { "abc",
271      { 0xdd, 0xaf, 0x35, 0xa1, 0x93, 0x61, 0x7a, 0xba,
272        0xcc, 0x41, 0x73, 0x49, 0xae, 0x20, 0x41, 0x31,
273        0x12, 0xe6, 0xfa, 0x4e, 0x89, 0xa9, 0x7e, 0xa2,
274        0x0a, 0x9e, 0xee, 0xe6, 0x4b, 0x55, 0xd3, 0x9a,
275        0x21, 0x92, 0x99, 0x2a, 0x27, 0x4f, 0xc1, 0xa8,
276        0x36, 0xba, 0x3c, 0x23, 0xa3, 0xfe, 0xeb, 0xbd,
277        0x45, 0x4d, 0x44, 0x23, 0x64, 0x3c, 0xe8, 0x0e,
278        0x2a, 0x9a, 0xc9, 0x4f, 0xa5, 0x4c, 0xa4, 0x9f }
279     },
280     { "abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmnoijklmnopjklmnopqklmnopqrlmnopqr
281      { 0x8e, 0x95, 0x9b, 0x75, 0xda, 0xe3, 0x13, 0xda,
282        0x8c, 0xf4, 0xf7, 0x28, 0x14, 0xfc, 0x14, 0x3f,
283        0x8f, 0x77, 0x79, 0xc6, 0xeb, 0x9f, 0x7f, 0xa1,
284        0x72, 0x99, 0xae, 0xad, 0xb6, 0x88, 0x90, 0x18,
285        0x50, 0x1d, 0x28, 0x9e, 0x49, 0x00, 0xf7, 0xe4,
286        0x33, 0x1b, 0x99, 0xde, 0xc4, 0xb5, 0x43, 0x3a,
```

```
          0xc7, 0xd3, 0x29, 0xee, 0xb6, 0xdd, 0x26, 0x54,
          0x5e, 0x96, 0xe5, 0x5b, 0x87, 0x4b, 0xe9, 0x09 }
      },
    };

    int i;
    unsigned char tmp[64];
    hash_state md;

    for (i = 0; i < (int)(sizeof(tests) / sizeof(tests[0])); i++) {
        sha512_init(&md);
        sha512_process(&md, (unsigned char *)tests[i].msg, (unsigned long)strlen(tests[i].msg));
        sha512_done(&md, tmp);
        if (XMEMCMP(tmp, tests[i].hash, 64) != 0) {
           return CRYPT_FAIL_TESTVECTOR;
        }
    }
    return CRYPT_OK;
    #endif
}
```

Here is the call graph for this function:

5.68.3 Variable Documentation

5.68.3.1 const ulong64 K[80] `[static]`

Definition at line 39 of file sha512.c.

Referenced by f9_test(), gcm_test(), pelican_test(), rc5_ecb_decrypt(), rc5_ecb_encrypt(), rc6_ecb_decrypt(), rc6_ecb_encrypt(), whirlpool_compress(), xcbc_test(), and xtea_setup().

5.68.3.2 const struct ltc_hash_descriptor sha512_desc

Initial value:

```
{
    "sha512",
    5,
    64,
    128,


   { 2, 16, 840, 1, 101, 3, 4, 2, 3,  },
   9,

    &sha512_init,
    &sha512_process,
    &sha512_done,
    &sha512_test,
    NULL
}
```

Parameters:

sha512.c SHA512 by Tom St Denis

Definition at line 20 of file sha512.c.

Referenced by yarrow_start().

5.69 hashes/tiger.c File Reference

5.69.1 Detailed Description

Tiger hash function, Tom St Denis.

Definition in file tiger.c.

```
#include "tomcrypt.h"
```

Include dependency graph for tiger.c:

Defines

- #define t1 (table)
- #define t2 (table+256)
- #define t3 (table+256∗2)
- #define t4 (table+256∗3)
- #define INLINE

Functions

- static INLINE void tiger_round (ulong64 ∗a, ulong64 ∗b, ulong64 ∗c, ulong64 x, int mul)
- static void pass (ulong64 ∗a, ulong64 ∗b, ulong64 ∗c, ulong64 ∗x, int mul)
- static void key_schedule (ulong64 ∗x)
- static int tiger_compress (hash_state ∗md, unsigned char ∗buf)
- int tiger_init (hash_state ∗md)

  Initialize the hash state.
- int tiger_done (hash_state ∗md, unsigned char ∗out)

  Terminate the hash to get the digest.
- int tiger_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor tiger_desc
- static const ulong64 table [4 ∗256]

5.69.2 Define Documentation

5.69.2.1 #define INLINE

Definition at line 561 of file tiger.c.

5.69.2.2 #define t1 (table)

Definition at line 39 of file tiger.c.

Referenced by ECB_ENC(), four_rounds(), is_point(), ltc_ecc_map(), ltc_ecc_projective_add_point(), ltc_ecc_projective_dbl_point(), sha256_compress(), sha512_compress(), tiger_round(), twofish_ecb_-decrypt(), and twofish_ecb_encrypt().

5.69.2.3 #define t2 (table+256)

Definition at line 40 of file tiger.c.

Referenced by ECB_ENC(), four_rounds(), is_point(), ltc_ecc_map(), ltc_ecc_projective_add_point(), ltc_ecc_projective_dbl_point(), tiger_round(), twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.69.2.4 #define t3 (table+256*2)

Definition at line 41 of file tiger.c.

Referenced by ECB_ENC(), four_rounds(), and tiger_round().

5.69.2.5 #define t4 (table+256*3)

Definition at line 42 of file tiger.c.

Referenced by tiger_round().

5.69.3 Function Documentation

5.69.3.1 static void key_schedule (ulong64 * *x*) [static]

Definition at line 592 of file tiger.c.

References CONST64.

```
{
    x[0] -= x[7] ^ CONST64(0xA5A5A5A5A5A5A5A5);
    x[1] ^= x[0];
    x[2] += x[1];
    x[3] -= x[2] ^ ((~x[1])<<19);
    x[4] ^= x[3];
    x[5] += x[4];
    x[6] -= x[5] ^ ((~x[4])>>23);
    x[7] ^= x[6];
    x[0] += x[7];
    x[1] -= x[0] ^ ((~x[7])<<19);
    x[2] ^= x[1];
    x[3] += x[2];
    x[4] -= x[3] ^ ((~x[2])>>23);
    x[5] ^= x[4];
    x[6] += x[5];
    x[7] -= x[6] ^ CONST64(0x0123456789ABCDEF);
}
```

5.69.3.2 static void pass (ulong64 ∗ *a*, ulong64 ∗ *b*, ulong64 ∗ *c*, ulong64 ∗ *x*, int *mul*) [static]

Definition at line 579 of file tiger.c.

References tiger_round().

```
{
   tiger_round(a,b,c,x[0],mul);
   tiger_round(b,c,a,x[1],mul);
   tiger_round(c,a,b,x[2],mul);
   tiger_round(a,b,c,x[3],mul);
   tiger_round(b,c,a,x[4],mul);
   tiger_round(c,a,b,x[5],mul);
   tiger_round(a,b,c,x[6],mul);
   tiger_round(b,c,a,x[7],mul);
}
```

Here is the call graph for this function:

5.69.3.3 static int tiger_compress (hash_state ∗ *md*, unsigned char ∗ *buf*) [static]

Definition at line 615 of file tiger.c.

References c.

Referenced by tiger_done().

```
{
    ulong64 a, b, c, x[8];
    unsigned long i;

    /* load words */
    for (i = 0; i < 8; i++) {
        LOAD64L(x[i],&buf[8*i]);
    }
    a = md->tiger.state[0];
    b = md->tiger.state[1];
    c = md->tiger.state[2];

    pass(&a,&b,&c,x,5);
    key_schedule(x);
    pass(&c,&a,&b,x,7);
    key_schedule(x);
    pass(&b,&c,&a,x,9);

    /* store state */
    md->tiger.state[0] = a ^ md->tiger.state[0];
    md->tiger.state[1] = b - md->tiger.state[1];
    md->tiger.state[2] = c + md->tiger.state[2];

    return CRYPT_OK;
}
```

5.69.3.4 int tiger_done (hash_state ∗ *md*, unsigned char ∗ *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (24 bytes)

Returns:
CRYPT_OK if successful

Definition at line 684 of file tiger.c.

References CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, tiger_compress(), and zeromem().

Referenced by tiger_test().

```
{
    LTC_ARGCHK(md  != NULL);
    LTC_ARGCHK(out != NULL);

    if (md->tiger.curlen >= sizeof(md->tiger.buf)) {
       return CRYPT_INVALID_ARG;
    }

    /* increase the length of the message */
    md->tiger.length += md->tiger.curlen * 8;

    /* append the '1' bit */
    md->tiger.buf[md->tiger.curlen++] = (unsigned char)0x01;

    /* if the length is currently above 56 bytes we append zeros
     * then compress.  Then we can fall back to padding zeros and length
     * encoding like normal. */
    if (md->tiger.curlen > 56) {
        while (md->tiger.curlen < 64) {
            md->tiger.buf[md->tiger.curlen++] = (unsigned char)0;
        }
        tiger_compress(md, md->tiger.buf);
        md->tiger.curlen = 0;
    }

    /* pad upto 56 bytes of zeroes */
    while (md->tiger.curlen < 56) {
        md->tiger.buf[md->tiger.curlen++] = (unsigned char)0;
    }

    /* store length */
    STORE64L(md->tiger.length, md->tiger.buf+56);
    tiger_compress(md, md->tiger.buf);

    /* copy output */
    STORE64L(md->tiger.state[0], &out[0]);
    STORE64L(md->tiger.state[1], &out[8]);
    STORE64L(md->tiger.state[2], &out[16]);
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif

    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.69.3.5 int tiger_init (hash_state * *md*)

Initialize the hash state.

Parameters:
md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 658 of file tiger.c.

References CONST64, CRYPT_OK, and LTC_ARGCHK.

Referenced by tiger_test().

```
{
    LTC_ARGCHK(md != NULL);
    md->tiger.state[0] = CONST64(0x0123456789ABCDEF);
    md->tiger.state[1] = CONST64(0xFEDCBA9876543210);
    md->tiger.state[2] = CONST64(0xF096A5B4C3B2E187);
    md->tiger.curlen = 0;
    md->tiger.length = 0;
    return CRYPT_OK;
}
```

5.69.3.6 static INLINE void tiger_round (ulong64 * *a*, ulong64 * *b*, ulong64 * *c*, ulong64 *x*, int *mul*) [static]

Definition at line 565 of file tiger.c.

References byte, t1, t2, t3, and t4.

Referenced by pass().

```
{
    ulong64 tmp;
    tmp = (*c ^= x);
           *a -= t1[byte(tmp, 0)] ^ t2[byte(tmp, 2)] ^ t3[byte(tmp, 4)] ^ t4[byte(tmp, 6)];
    tmp = (*b += t4[byte(tmp, 1)] ^ t3[byte(tmp, 3)] ^ t2[byte(tmp,5)] ^ t1[byte(tmp,7)]);
    switch (mul) {
        case 5:  *b = (tmp << 2) + tmp; break;
        case 7:  *b = (tmp << 3) - tmp; break;
        case 9:  *b = (tmp << 3) + tmp; break;
    }
}
```

5.69.3.7 int tiger_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 734 of file tiger.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, tiger_done(), tiger_init(), and XMEMCMP.

```
{
 #ifndef LTC_TEST
    return CRYPT_NOP;
 #else
  static const struct {
      char *msg;
      unsigned char hash[24];
```

```
    } tests[] = {
      { "",
       { 0x32, 0x93, 0xac, 0x63, 0x0c, 0x13, 0xf0, 0x24,
         0x5f, 0x92, 0xbb, 0xb1, 0x76, 0x6e, 0x16, 0x16,
         0x7a, 0x4e, 0x58, 0x49, 0x2d, 0xde, 0x73, 0xf3 }
      },
      { "abc",
       { 0x2a, 0xab, 0x14, 0x84, 0xe8, 0xc1, 0x58, 0xf2,
         0xbf, 0xb8, 0xc5, 0xff, 0x41, 0xb5, 0x7a, 0x52,
         0x51, 0x29, 0x13, 0x1c, 0x95, 0x7b, 0x5f, 0x93 }
      },
      { "Tiger",
       { 0xdd, 0x00, 0x23, 0x07, 0x99, 0xf5, 0x00, 0x9f,
         0xec, 0x6d, 0xeb, 0xc8, 0x38, 0xbb, 0x6a, 0x27,
         0xdf, 0x2b, 0x9d, 0x6f, 0x11, 0x0c, 0x79, 0x37 }
      },
      { "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+-",
       { 0xf7, 0x1c, 0x85, 0x83, 0x90, 0x2a, 0xfb, 0x87,
         0x9e, 0xdf, 0xe6, 0x10, 0xf8, 0x2c, 0x0d, 0x47,
         0x86, 0xa3, 0xa5, 0x34, 0x50, 0x44, 0x86, 0xb5 }
      },
      { "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+-ABCDEFGHIJKLMNOPQRSTUVWXYZabcde
       { 0xc5, 0x40, 0x34, 0xe5, 0xb4, 0x3e, 0xb8, 0x00,
         0x58, 0x48, 0xa7, 0xe0, 0xae, 0x6a, 0xac, 0x76,
         0xe4, 0xff, 0x59, 0x0a, 0xe7, 0x15, 0xfd, 0x25 }
      },
    };

    int i;
    unsigned char tmp[24];
    hash_state md;

    for (i = 0; i < (int)(sizeof(tests) / sizeof(tests[0])); i++) {
        tiger_init(&md);
        tiger_process(&md, (unsigned char *)tests[i].msg, (unsigned long)strlen(tests[i].msg));
        tiger_done(&md, tmp);
        if (XMEMCMP(tmp, tests[i].hash, 24) != 0) {
            return CRYPT_FAIL_TESTVECTOR;
        }
    }
    return CRYPT_OK;
    #endif
}
```

Here is the call graph for this function:

5.69.4 Variable Documentation

5.69.4.1 const ulong64 table[4 *256] [static]

Definition at line 44 of file tiger.c.

5.69.4.2 const struct ltc_hash_descriptor tiger_desc

Initial value:

```
{
    "tiger",
    1,
    24,
    64,
```

```
   { 1, 3, 6, 1, 4, 1, 11591, 12, 2,  },
   9,

    &tiger_init,
    &tiger_process,
    &tiger_done,
    &tiger_test,
    NULL
}
```

Definition at line 21 of file tiger.c.

Referenced by yarrow_start().

5.70 hashes/whirl/whirl.c File Reference

5.70.1 Detailed Description

WHIRLPOOL (using their new sbox) hash function by Tom St Denis.

Definition in file whirl.c.

`#include "tomcrypt.h"`

`#include "whirltab.c"`

Include dependency graph for whirl.c:

Defines

- #define GB(a, i, j) ((a[(i) & 7] >> (8 ∗ (j))) & 255)
- #define theta_pi_gamma(a, i)

Functions

- static int whirlpool_compress (hash_state ∗md, unsigned char ∗buf)
- int whirlpool_init (hash_state ∗md)

  Initialize the hash state.

- int whirlpool_done (hash_state ∗md, unsigned char ∗out)

  Terminate the hash to get the digest.

- int whirlpool_test (void)

  Self-test the hash.

Variables

- const struct ltc_hash_descriptor whirlpool_desc

5.70.2 Define Documentation

5.70.2.1 #define GB(a, i, j) ((a[(i) & 7] >> (8 ∗ (j))) & 255)

Definition at line 43 of file whirl.c.

5.70.2.2 #define theta_pi_gamma(a, i)

Value:

```
SB0(GB(a, i-0, 7)) ^                    \
     SB1(GB(a, i-1, 6)) ^                    \
     SB2(GB(a, i-2, 5)) ^                    \
     SB3(GB(a, i-3, 4)) ^                    \
     SB4(GB(a, i-4, 3)) ^                    \
     SB5(GB(a, i-5, 2)) ^                    \
```

```
    SB6(GB(a, i-6, 1)) ^                  \
    SB7(GB(a, i-7, 0))
```

Definition at line 46 of file whirl.c.

5.70.3 Function Documentation

5.70.3.1 static int whirlpool_compress (hash_state * *md*, unsigned char * *buf*) [static]

Definition at line 59 of file whirl.c.

References K.

Referenced by whirlpool_done().

```
{
   ulong64 K[2][8], T[3][8];
   int x, y;

   /* load the block/state */
   for (x = 0; x < 8; x++) {
      K[0][x] = md->whirlpool.state[x];

      LOAD64H(T[0][x], buf + (8 * x));
      T[2][x]  = T[0][x];
      T[0][x] ^= K[0][x];
   }

   /* do rounds 1..10 */
   for (x = 0; x < 10; x += 2) {
       /* odd round */
       /* apply main transform to K[0] into K[1] */
       for (y = 0; y < 8; y++) {
           K[1][y] = theta_pi_gamma(K[0], y);
       }
       /* xor the constant */
       K[1][0] ^= cont[x];

       /* apply main transform to T[0] into T[1] */
       for (y = 0; y < 8; y++) {
           T[1][y] = theta_pi_gamma(T[0], y) ^ K[1][y];
       }

       /* even round */
       /* apply main transform to K[1] into K[0] */
       for (y = 0; y < 8; y++) {
           K[0][y] = theta_pi_gamma(K[1], y);
       }
       /* xor the constant */
       K[0][0] ^= cont[x+1];

       /* apply main transform to T[1] into T[0] */
       for (y = 0; y < 8; y++) {
           T[0][y] = theta_pi_gamma(T[1], y) ^ K[0][y];
       }
   }

   /* store state */
   for (x = 0; x < 8; x++) {
      md->whirlpool.state[x] ^= T[0][x] ^ T[2][x];
   }

   return CRYPT_OK;
}
```

5.70.3.2 int whirlpool_done (hash_state ∗ *md*, unsigned char ∗ *out*)

Terminate the hash to get the digest.

Parameters:

md The hash state

out [out] The destination of the hash (64 bytes)

Returns:

CRYPT_OK if successful

Definition at line 150 of file whirl.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and whirlpool_compress().

Referenced by whirlpool_test().

```
151 {
152     int i;
153 
154     LTC_ARGCHK(md  != NULL);
155     LTC_ARGCHK(out != NULL);
156 
157     if (md->whirlpool.curlen >= sizeof(md->whirlpool.buf)) {
158        return CRYPT_INVALID_ARG;
159     }
160 
161     /* increase the length of the message */
162     md->whirlpool.length += md->whirlpool.curlen * 8;
163 
164     /* append the '1' bit */
165     md->whirlpool.buf[md->whirlpool.curlen++] = (unsigned char)0x80;
166 
167     /* if the length is currently above 32 bytes we append zeros
168      * then compress.  Then we can fall back to padding zeros and length
169      * encoding like normal.
170      */
171     if (md->whirlpool.curlen > 32) {
172         while (md->whirlpool.curlen < 64) {
173             md->whirlpool.buf[md->whirlpool.curlen++] = (unsigned char)0;
174         }
175         whirlpool_compress(md, md->whirlpool.buf);
176         md->whirlpool.curlen = 0;
177     }
178 
179     /* pad upto 56 bytes of zeroes (should be 32 but we only support 64-bit lengths)  */
180     while (md->whirlpool.curlen < 56) {
181         md->whirlpool.buf[md->whirlpool.curlen++] = (unsigned char)0;
182     }
183 
184     /* store length */
185     STORE64H(md->whirlpool.length, md->whirlpool.buf+56);
186     whirlpool_compress(md, md->whirlpool.buf);
187 
188     /* copy output */
189     for (i = 0; i < 8; i++) {
190         STORE64H(md->whirlpool.state[i], out+(8*i));
191     }
192 #ifdef LTC_CLEAN_STACK
193     zeromem(md, sizeof(*md));
194 #endif
195     return CRYPT_OK;
196 }
```

Here is the call graph for this function:

5.70.3.3 int whirlpool_init (hash_state * *md*)

Initialize the hash state.

Parameters:

md The hash state you wish to initialize

Returns:

CRYPT_OK if successful

Definition at line 128 of file whirl.c.

References CRYPT_OK, LTC_ARGCHK, and zeromem().

Referenced by whirlpool_test().

```
129 {
130    LTC_ARGCHK(md != NULL);
131    zeromem(&md->whirlpool, sizeof(md->whirlpool));
132    return CRYPT_OK;
133 }
```

Here is the call graph for this function:

5.70.3.4 int whirlpool_test (void)

Self-test the hash.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-tests have been disabled

Definition at line 202 of file whirl.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, len, whirlpool_done(), whirlpool_init(), and XMEMCMP.

```
203 {
204  #ifndef LTC_TEST
205     return CRYPT_NOP;
206  #else
207   static const struct {
208       int len;
209       unsigned char msg[128], hash[64];
210   } tests[] = {
211
212   /* NULL Message */
213 {
214   0,
215   { 0x00 },
216   { 0x19, 0xFA, 0x61, 0xD7, 0x55, 0x22, 0xA4, 0x66, 0x9B, 0x44, 0xE3, 0x9C, 0x1D, 0x2E, 0x17, 0x26,
217     0xC5, 0x30, 0x23, 0x21, 0x30, 0xD4, 0x07, 0xF8, 0x9A, 0xFE, 0xE0, 0x96, 0x49, 0x97, 0xF7, 0xA7,
218     0x3E, 0x83, 0xBE, 0x69, 0x8B, 0x28, 0x8F, 0xEB, 0xCF, 0x88, 0xE3, 0xE0, 0x3C, 0x4F, 0x07, 0x57,
219     0xEA, 0x89, 0x64, 0xE5, 0x9B, 0x63, 0xD9, 0x37, 0x08, 0xB1, 0x38, 0xCC, 0x42, 0xA6, 0x6E, 0xB3 }
220 },
221
222
223    /* 448-bits of 0 bits */
224 {
225
```

```
  56,
  { 0x00 },
  { 0x0B, 0x3F, 0x53, 0x78, 0xEB, 0xED, 0x2B, 0xF4, 0xD7, 0xBE, 0x3C, 0xFD, 0x81, 0x8C, 0x1B, 0x03,
    0xB6, 0xBB, 0x03, 0xD3, 0x46, 0x94, 0x8B, 0x04, 0xF4, 0xF4, 0x0C, 0x72, 0x6F, 0x07, 0x58, 0x70,
    0x2A, 0x0F, 0x1E, 0x22, 0x58, 0x80, 0xE3, 0x8D, 0xD5, 0xF6, 0xED, 0x6D, 0xE9, 0xB1, 0xE9, 0x61,
    0xE4, 0x9F, 0xC1, 0x31, 0x8D, 0x7C, 0xB7, 0x48, 0x22, 0xF3, 0xD0, 0xE2, 0xE9, 0xA7, 0xE7, 0xB0 }
},

   /* 520-bits of 0 bits */
{
  65,
  { 0x00 },
  { 0x85, 0xE1, 0x24, 0xC4, 0x41, 0x5B, 0xCF, 0x43, 0x19, 0x54, 0x3E, 0x3A, 0x63, 0xFF, 0x57, 0x1D,
    0x09, 0x35, 0x4C, 0xEE, 0xBE, 0xE1, 0xE3, 0x25, 0x30, 0x8C, 0x90, 0x69, 0xF4, 0x3E, 0x2A, 0xE4,
    0xD0, 0xE5, 0x1D, 0x4E, 0xB1, 0xE8, 0x64, 0x28, 0x70, 0x19, 0x4E, 0x95, 0x30, 0xD8, 0xD8, 0xAF,
    0x65, 0x89, 0xD1, 0xBF, 0x69, 0x49, 0xDD, 0xF9, 0x0A, 0x7F, 0x12, 0x08, 0x62, 0x37, 0x95, 0xB9 }
},

   /* 512-bits, leading set */
{
  64,
  { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
  { 0x10, 0x3E, 0x00, 0x55, 0xA9, 0xB0, 0x90, 0xE1, 0x1C, 0x8F, 0xDD, 0xEB, 0xBA, 0x06, 0xC0, 0x5A,
    0xCE, 0x8B, 0x64, 0xB8, 0x96, 0x12, 0x8F, 0x6E, 0xED, 0x30, 0x71, 0xFC, 0xF3, 0xDC, 0x16, 0x94,
    0x67, 0x78, 0xE0, 0x72, 0x23, 0x23, 0x3F, 0xD1, 0x80, 0xFC, 0x40, 0xCC, 0xDB, 0x84, 0x30, 0xA6,
    0x40, 0xE3, 0x76, 0x34, 0x27, 0x1E, 0x65, 0x5C, 0xA1, 0x67, 0x4E, 0xBF, 0xF5, 0x07, 0xF8, 0xCB }
},

   /* 512-bits, leading set of second byte */
{
  64,
  { 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
  { 0x35, 0x7B, 0x42, 0xEA, 0x79, 0xBC, 0x97, 0x86, 0x97, 0x5A, 0x3C, 0x44, 0x70, 0xAA, 0xB2, 0x3E,
    0x62, 0x29, 0x79, 0x7B, 0xAD, 0xBD, 0x54, 0x36, 0x5B, 0x54, 0x96, 0xE5, 0x5D, 0x9D, 0xD7, 0x9F,
    0xE9, 0x62, 0x4F, 0xB4, 0x22, 0x66, 0x93, 0x0A, 0x62, 0x8E, 0xD4, 0xDB, 0x08, 0xF9, 0xDD, 0x35,
    0xEF, 0x1B, 0xE1, 0x04, 0x53, 0xFC, 0x18, 0xF4, 0x2C, 0x7F, 0x5E, 0x1F, 0x9B, 0xAE, 0x55, 0xE0 }
},

   /* 512-bits, leading set of last byte */
{
  64,
  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80 },
  { 0x8B, 0x39, 0x04, 0xDD, 0x19, 0x81, 0x41, 0x26, 0xFD, 0x02, 0x74, 0xAB, 0x49, 0xC5, 0x97, 0xF6,
    0xD7, 0x75, 0x33, 0x52, 0xA2, 0xDD, 0x91, 0xFD, 0x8F, 0x9F, 0x54, 0x05, 0x4C, 0x54, 0xBF, 0x0F,
    0x06, 0xDB, 0x4F, 0xF7, 0x08, 0xA3, 0xA2, 0x8B, 0xC3, 0x7A, 0x92, 0x1E, 0xEE, 0x11, 0xED, 0x7B,
    0x6A, 0x53, 0x79, 0x32, 0xCC, 0x5E, 0x94, 0xEE, 0x1E, 0xA6, 0x57, 0x60, 0x7E, 0x36, 0xC9, 0xF7 }
},

};

  int i;
  unsigned char tmp[64];
  hash_state md;

  for (i = 0; i < (int)(sizeof(tests)/sizeof(tests[0])); i++) {
      whirlpool_init(&md);
      whirlpool_process(&md, (unsigned char *)tests[i].msg, tests[i].len);
      whirlpool_done(&md, tmp);
```

```
      if (XMEMCMP(tmp, tests[i].hash, 64) != 0) {
#if 0
         printf("\nFailed test %d\n", i);
         for (i = 0; i < 64; ) {
            printf("%02x ", tmp[i]);
            if (!(++i & 15)) printf("\n");
         }
#endif
         return CRYPT_FAIL_TESTVECTOR;
      }
   }
   return CRYPT_OK;
  #endif
}
```

Here is the call graph for this function:

5.70.4 Variable Documentation

5.70.4.1 const struct ltc_hash_descriptor whirlpool_desc

Initial value:

```
{
    "whirlpool",
    11,
    64,
    64,


   { 1, 0, 10118, 3, 0, 55 },
   6,

    &whirlpool_init,
    &whirlpool_process,
    &whirlpool_done,
    &whirlpool_test,
    NULL
}
```

Definition at line 21 of file whirl.c.

Referenced by yarrow_start().

5.71 hashes/whirl/whirltab.c File Reference

5.71.1 Detailed Description

WHIRLPOOL tables, Tom St Denis.

Definition in file whirltab.c.

This graph shows which files directly or indirectly include this file:

Defines

- #define SB0(x) sbox0[x]
- #define SB1(x) sbox1[x]
- #define SB2(x) sbox2[x]
- #define SB3(x) sbox3[x]
- #define SB4(x) sbox4[x]
- #define SB5(x) sbox5[x]
- #define SB6(x) sbox6[x]
- #define SB7(x) sbox7[x]

Variables

- static const ulong64 sbox0 [ ]
- static const ulong64 sbox1 [ ]
- static const ulong64 sbox2 [ ]
- static const ulong64 sbox3 [ ]
- static const ulong64 sbox4 [ ]
- static const ulong64 sbox5 [ ]
- static const ulong64 sbox6 [ ]
- static const ulong64 sbox7 [ ]
- static const ulong64 cont [ ]

5.71.2 Define Documentation

5.71.2.1 #define SB0(x) sbox0[x]

Definition at line 85 of file whirltab.c.

5.71.2.2 #define SB1(x) sbox1[x]

Definition at line 86 of file whirltab.c.

5.71.2.3 #define SB2(x) sbox2[x]

Definition at line 87 of file whirltab.c.

5.71.2.4 #define SB3(x) sbox3[x]

Definition at line 88 of file whirltab.c.

5.71.2.5 #define SB4(x) sbox4[x]

Definition at line 89 of file whirltab.c.

5.71.2.6 #define SB5(x) sbox5[x]

Definition at line 90 of file whirltab.c.

5.71.2.7 #define SB6(x) sbox6[x]

Definition at line 91 of file whirltab.c.

5.71.2.8 #define SB7(x) sbox7[x]

Definition at line 92 of file whirltab.c.

5.71.3 Variable Documentation

5.71.3.1 const ulong64 cont[ ] [static]

Initial value:

```
 {
CONST64(0x1823c6e887b8014f),
CONST64(0x36a6d2f5796f9152),
CONST64(0x60bc9b8ea30c7b35),
CONST64(0x1de0d7c22e4bfe57),
CONST64(0x157737e59ff04ada),
CONST64(0x58c9290ab1a06b85),
CONST64(0xbd5d10f4cb3e0567),
CONST64(0xe427418ba77d95d8),
CONST64(0xfbee7c66dd17479e),
CONST64(0xca2dbf07ad5a8333),
CONST64(0x6302aa71c81949d9),
}
```

Definition at line 566 of file whirltab.c.

5.71.3.2 const ulong64 sbox0[ ] [static]

Definition at line 5 of file whirltab.c.

5.71.3.3 const ulong64 sbox1[ ] [static]

Definition at line 95 of file whirltab.c.

5.71.3.4 const ulong64 sbox2[ ] [static]

Definition at line 162 of file whirltab.c.

5.71.3.5 const ulong64 sbox3[ ] [static]

Definition at line 229 of file whirltab.c.

5.71.3.6 const ulong64 sbox4[ ] [static]

Definition at line 296 of file whirltab.c.

5.71.3.7 const ulong64 sbox5[ ] [static]

Definition at line 363 of file whirltab.c.

5.71.3.8 const ulong64 sbox6[ ] [static]

Definition at line 430 of file whirltab.c.

5.71.3.9 const ulong64 sbox7[ ] [static]

Definition at line 497 of file whirltab.c.

5.72 headers/tomcrypt.h File Reference

```
#include <assert.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>
#include <limits.h>
#include <tomcrypt_custom.h>
#include <tomcrypt_cfg.h>
#include <tomcrypt_macros.h>
#include <tomcrypt_cipher.h>
#include <tomcrypt_hash.h>
#include <tomcrypt_mac.h>
#include <tomcrypt_prng.h>
#include <tomcrypt_pk.h>
#include <tomcrypt_math.h>
#include <tomcrypt_misc.h>
#include <tomcrypt_argchk.h>
#include <tomcrypt_pkcs.h>
```

Include dependency graph for tomcrypt.h:

This graph shows which files directly or indirectly include this file:

Defines

- #define CRYPT 0x0115
- #define SCRYPT "1.15"
- #define MAXBLOCKSIZE 128
- #define TAB_SIZE 32

Enumerations

- enum {
  CRYPT_OK = 0,
  CRYPT_ERROR,
  CRYPT_NOP,
  CRYPT_INVALID_KEYSIZE,
  CRYPT_INVALID_ROUNDS,
  CRYPT_FAIL_TESTVECTOR,

CRYPT_BUFFER_OVERFLOW,
CRYPT_INVALID_PACKET,
CRYPT_INVALID_PRNGSIZE,
CRYPT_ERROR_READPRNG,
CRYPT_INVALID_CIPHER,
CRYPT_INVALID_HASH,
CRYPT_INVALID_PRNG,
CRYPT_MEM,
CRYPT_PK_TYPE_MISMATCH,
CRYPT_PK_NOT_PRIVATE,
CRYPT_INVALID_ARG,
CRYPT_FILE_NOTFOUND,
CRYPT_PK_INVALID_TYPE,
CRYPT_PK_INVALID_SYSTEM,
CRYPT_PK_DUP,
CRYPT_PK_NOT_FOUND,
CRYPT_PK_INVALID_SIZE,
CRYPT_INVALID_PRIME_SIZE,
CRYPT_PK_INVALID_PADDING }

5.72.1 Define Documentation

5.72.1.1 #define CRYPT 0x0115

Definition at line 19 of file tomcrypt.h.

5.72.1.2 #define MAXBLOCKSIZE 128

Definition at line 23 of file tomcrypt.h.

Referenced by chc_compress(), chc_init(), dsa_decrypt_key(), dsa_encrypt_key(), eax_done(), eax_init(), eax_test(), ecc_decrypt_key(), ecc_encrypt_key(), f8_encrypt(), f8_start(), find_hash_any(), fortuna_reseed(), fortuna_start(), hmac_test(), ocb_decrypt(), ocb_done_decrypt(), ocb_encrypt(), ocb_test(), pkcs_5_alg1(), pkcs_5_alg2(), pmac_process(), pmac_test(), and s_ocb_done().

5.72.1.3 #define SCRYPT "1.15"

Definition at line 20 of file tomcrypt.h.

5.72.1.4 #define TAB_SIZE 32

Definition at line 26 of file tomcrypt.h.

Referenced by cipher_is_valid(), find_cipher(), find_cipher_any(), find_cipher_id(), find_hash(), find_hash_any(), find_hash_id(), find_hash_oid(), find_prng(), hash_is_valid(), prng_is_valid(), register_cipher(), register_hash(), register_prng(), unregister_cipher(), unregister_hash(), and unregister_prng().

5.72.2 Enumeration Type Documentation

5.72.2.1 anonymous enum

Enumerator:

CRYPT_OK

CRYPT_ERROR

CRYPT_NOP

CRYPT_INVALID_KEYSIZE

CRYPT_INVALID_ROUNDS

CRYPT_FAIL_TESTVECTOR

CRYPT_BUFFER_OVERFLOW

CRYPT_INVALID_PACKET

CRYPT_INVALID_PRNGSIZE

CRYPT_ERROR_READPRNG

CRYPT_INVALID_CIPHER

CRYPT_INVALID_HASH

CRYPT_INVALID_PRNG

CRYPT_MEM

CRYPT_PK_TYPE_MISMATCH

CRYPT_PK_NOT_PRIVATE

CRYPT_INVALID_ARG

CRYPT_FILE_NOTFOUND

CRYPT_PK_INVALID_TYPE

CRYPT_PK_INVALID_SYSTEM

CRYPT_PK_DUP

CRYPT_PK_NOT_FOUND

CRYPT_PK_INVALID_SIZE

CRYPT_INVALID_PRIME_SIZE

CRYPT_PK_INVALID_PADDING

Definition at line 29 of file tomcrypt.h.

```
29      {
30    CRYPT_OK=0,             /* Result OK */
31    CRYPT_ERROR,            /* Generic Error */
32    CRYPT_NOP,              /* Not a failure but no operation was performed */
33
34    CRYPT_INVALID_KEYSIZE,  /* Invalid key size given */
35    CRYPT_INVALID_ROUNDS,   /* Invalid number of rounds */
36    CRYPT_FAIL_TESTVECTOR,  /* Algorithm failed test vectors */
37
38    CRYPT_BUFFER_OVERFLOW,  /* Not enough space for output */
39    CRYPT_INVALID_PACKET,   /* Invalid input packet given */
40
41    CRYPT_INVALID_PRNGSIZE, /* Invalid number of bits for a PRNG */
42    CRYPT_ERROR_READPRNG,   /* Could not read enough from PRNG */
43
44    CRYPT_INVALID_CIPHER,   /* Invalid cipher specified */
```

```
   CRYPT_INVALID_HASH,     /* Invalid hash specified */
   CRYPT_INVALID_PRNG,     /* Invalid PRNG specified */

   CRYPT_MEM,              /* Out of memory */

   CRYPT_PK_TYPE_MISMATCH, /* Not equivalent types of PK keys */
   CRYPT_PK_NOT_PRIVATE,   /* Requires a private PK key */

   CRYPT_INVALID_ARG,      /* Generic invalid argument */
   CRYPT_FILE_NOTFOUND,    /* File Not Found */

   CRYPT_PK_INVALID_TYPE,  /* Invalid type of PK key */
   CRYPT_PK_INVALID_SYSTEM,/* Invalid PK system specified */
   CRYPT_PK_DUP,           /* Duplicate key already in key ring */
   CRYPT_PK_NOT_FOUND,     /* Key not found in keyring */
   CRYPT_PK_INVALID_SIZE,  /* Invalid size input for PK parameters */

   CRYPT_INVALID_PRIME_SIZE,/* Invalid size of prime requested */
   CRYPT_PK_INVALID_PADDING /* Invalid padding on input */
};
```

5.73 headers/tomcrypt_argchk.h File Reference

`#include <signal.h>`

Include dependency graph for tomcrypt_argchk.h:

This graph shows which files directly or indirectly include this file:

Defines

- #define LTC_ARGCHK(x) if (!(x)) { crypt_argchk(#x, __FILE__, __LINE__); }
- #define LTC_ARGCHKVD(x) LTC_ARGCHK(x)

Functions

- void crypt_argchk (char ∗v, char ∗s, int d)

5.73.1 Define Documentation

5.73.1.1 #define LTC_ARGCHK(x) if (!(x)) { crypt_argchk(#x, __FILE__, __LINE__); }

Definition at line 9 of file tomcrypt_argchk.h.

Referenced by anubis_ecb_decrypt(), anubis_ecb_encrypt(), anubis_keysize(), anubis_setup(), base64_decode(), base64_encode(), blowfish_ecb_decrypt(), blowfish_ecb_encrypt(), blowfish_keysize(), blowfish_setup(), cast5_ecb_decrypt(), cast5_ecb_encrypt(), cast5_keysize(), cast5_setup(), cbc_decrypt(), cbc_done(), cbc_encrypt(), cbc_getiv(), cbc_setiv(), cbc_start(), ccm_memory(), cfb_decrypt(), cfb_done(), cfb_encrypt(), cfb_getiv(), cfb_setiv(), cfb_start(), chc_init(), ctr_decrypt(), ctr_done(), ctr_encrypt(), ctr_getiv(), ctr_setiv(), ctr_start(), der_decode_bit_string(), der_decode_boolean(), der_decode_choice(), der_decode_ia5_string(), der_decode_integer(), der_decode_object_identifier(), der_decode_octet_string(), der_decode_printable_string(), der_decode_sequence_ex(), der_decode_sequence_flexi(), der_decode_sequence_multi(), der_decode_short_integer(), der_decode_utctime(), der_encode_bit_string(), der_encode_boolean(), der_encode_ia5_string(), der_encode_integer(), der_encode_object_identifier(), der_encode_octet_string(), der_encode_printable_string(), der_encode_sequence_ex(), der_encode_sequence_multi(), der_encode_short_integer(), der_encode_utctime(), der_length_bit_string(), der_length_boolean(), der_length_ia5_string(), der_length_integer(), der_length_object_identifier(), der_length_octet_string(), der_length_printable_string(), der_length_sequence(), der_length_short_integer(), der_length_utctime(), des3_ecb_decrypt(), des3_ecb_encrypt(), des3_keysize(), des3_setup(), des_ecb_decrypt(), des_ecb_encrypt(), des_keysize(), des_setup(), dsa_decrypt_key(), dsa_encrypt_key(), dsa_export(), dsa_import(), dsa_make_key(), dsa_shared_secret(), dsa_sign_hash(), dsa_sign_hash_raw(), dsa_verify_hash_raw(), dsa_verify_key(), eax_addheader(), eax_decrypt(), eax_decrypt_verify_memory(), eax_done(), eax_encrypt(), eax_encrypt_authenticate_memory(), eax_init(), ECB_DEC(), ecb_decrypt(), ecb_done(), ECB_ENC(), ecb_encrypt(), ECB_KS(), ecb_start(), ecc_ansi_x963_export(), ecc_ansi_x963_import(), ecc_decrypt_key(), ecc_encrypt_key(), ecc_export(), ecc_get_size(), ecc_import(), ecc_make_key(), ecc_shared_secret(), ecc_sign_hash(), ecc_verify_hash(), f8_decrypt(), f8_done(), f8_encrypt(), f8_getiv(), f8_setiv(), f8_start(), f9_done(), f9_file(), f9_init(), f9_memory_multi(), f9_process(), find_cipher(), find_cipher_any(), find_hash(), find_hash_any(), find_hash_oid(), find_prng(), fortuna_add_entropy(), fortuna_done(), fortuna_export(), fortuna_import(), fortuna_read(), fortuna_start(), gcm_add_aad(), gcm_add_iv(), gcm_done(), gcm_init(), gcm_process(), gcm_reset(), hash_file(), hash_filehandle(), hash_memory(), hash_memory_multi(), hmac_done(), hmac_file(), hmac_init(), hmac_memory(), hmac_memory_multi(), hmac_process(), kasumi_ecb_decrypt(), kasumi_ecb_encrypt(), kasumi_keysize(), kasumi_setup(), khazad_ecb_decrypt(), khazad_ecb_encrypt(), khazad_keysize(), khazad_setup(), kseed_keysize(), lrw_decrypt(), lrw_done(), lrw_encrypt(), lrw_getiv(), lrw_process(), lrw_setiv(), lrw_start(), ltc_ecc_map(), ltc_ecc_mulmod(), ltc_ecc_projective_add_point(), ltc_ecc_projective_dbl_point(), md2_done(), md2_init(), md2_process(), md4_done(), md4_init(), md5_done(), md5_init(), noekeon_ecb_decrypt(), noekeon_ecb_encrypt(), noekeon_keysize(), noekeon_setup(), ocb_decrypt(), ocb_decrypt_verify_memory(), ocb_done_decrypt(), ocb_done_encrypt(), ocb_encrypt(), ocb_encrypt_authenticate_memory(), ocb_init(), ofb_decrypt(), ofb_done(), ofb_encrypt(), ofb_getiv(), ofb_setiv(), ofb_start(), omac_done(), omac_file(), omac_init(), omac_memory(), omac_memory_multi(), omac_process(), pelican_done(), pelican_init(), pelican_process(), pkcs_1_mgf1(), pkcs_1_oaep_decode(), pkcs_1_oaep_encode(), pkcs_1_pss_decode(), pkcs_1_pss_encode(), pkcs_5_alg1(), pkcs_5_alg2(), pmac_done(), pmac_file(), pmac_init(), pmac_memory(), pmac_memory_multi(), pmac_process(), rand_prime(), rc2_ecb_decrypt(), rc2_ecb_encrypt(), rc2_keysize(), rc2_setup(), rc4_add_entropy(), rc4_done(), rc4_export(), rc4_import(), rc4_read(), rc4_ready(), rc4_start(), rc5_ecb_decrypt(), rc5_ecb_encrypt(), rc5_keysize(), rc5_setup(), rc6_ecb_decrypt(), rc6_ecb_encrypt(), rc6_keysize(), rc6_setup(), register_cipher(), register_hash(), register_prng(), rmd128_done(), rmd128_init(), rmd160_done(), rmd160_init(), rmd256_done(), rmd256_init(), rmd320_done(), rmd320_init(), rng_get_bytes(), rng_make_prng(), rsa_decrypt_key_ex(), rsa_encrypt_key_ex(), rsa_export(), rsa_exptmod(), rsa_import(), rsa_make_key(), rsa_sign_hash_ex(), rsa_verify_hash_ex(), s_ocb_done(), safer_128_keysize(), safer_64_keysize(), safer_ecb_decrypt(), safer_ecb_encrypt(), safer_k128_setup(), safer_k64_setup(), safer_sk128_setup(), safer_sk64_setup(), saferp_ecb_decrypt(), saferp_ecb_encrypt(), saferp_keysize(), saferp_setup(), SETUP(), sha1_done(), sha1_init(), sha224_done(), sha224_init(), sha256_done(), sha256_init(), sha384_done(), sha384_init(), sha512_done(), sha512_init(), skipjack_ecb_decrypt(), skipjack_ecb_encrypt(), skipjack_keysize(), skipjack_setup(), sober128_add_entropy(), sober128_done(), sober128_export(), sober128_import(), sober128_read(), sober128_start(), sprng_export(), sprng_read(), tiger_done(), tiger_init(), twofish_ecb_decrypt(), twofish_ecb_encrypt(), twofish_keysize(), twofish_setup(), unregister_cipher(), unregister_hash(), unregister_prng(), whirlpool_done(), whirlpool_init(), xcbc_done(), xcbc_file(), xcbc_init(), xcbc_memory_multi(), xcbc_process(), xtea_ecb_decrypt(), xtea_ecb_encrypt(), xtea_keysize(), xtea_setup(), yarrow_add_entropy(), yarrow_done(), yarrow_export(), yarrow_import(), yarrow_read(), yarrow_ready(), and yarrow_start().

5.73.1.2 #define LTC_ARGCHKVD(x) LTC_ARGCHK(x)

Definition at line 10 of file tomcrypt_argchk.h.

Referenced by dsa_free(), ecc_free(), ecc_sizes(), rsa_free(), and zeromem().

5.73.2 Function Documentation

5.73.2.1 void crypt_argchk (char ∗ *v*, char ∗ *s*, int *d*)

Definition at line 20 of file crypt_argchk.c.

```
{
 fprintf(stderr, "LTC_ARGCHK '%s' failure on line %d of file %s\n",
         v, d, s);
 (void)raise(SIGABRT);
}
```

5.74 headers/tomcrypt_cfg.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define ARGTYPE 0
- #define ENDIAN_NEUTRAL

Functions

- LTC_EXPORT void ∗LTC_CALL XMALLOC (size_t n)
- LTC_EXPORT void ∗LTC_CALL XREALLOC (void ∗p, size_t n)
- LTC_EXPORT void ∗LTC_CALL XCALLOC (size_t n, size_t s)
- LTC_EXPORT void LTC_CALL XFREE (void ∗p)
- LTC_EXPORT void LTC_CALL XQSORT (void ∗base, size_t nmemb, size_t size, int(∗compar)(const void ∗, const void ∗))
- LTC_EXPORT clock_t LTC_CALL XCLOCK (void)
- LTC_EXPORT void ∗LTC_CALL XMEMCPY (void ∗dest, const void ∗src, size_t n)
- LTC_EXPORT int LTC_CALL XMEMCMP (const void ∗s1, const void ∗s2, size_t n)
- LTC_EXPORT void ∗LTC_CALL XMEMSET (void ∗s, int c, size_t n)

5.74.1 Define Documentation

5.74.1.1 #define ARGTYPE 0

Definition at line 46 of file tomcrypt_cfg.h.

5.74.1.2 #define ENDIAN_NEUTRAL

Definition at line 126 of file tomcrypt_cfg.h.

5.74.2 Function Documentation

5.74.2.1 LTC_EXPORT void* LTC_CALL XCALLOC (size_t *n*, size_t *s*)

5.74.2.2 LTC_EXPORT clock_t LTC_CALL XCLOCK (void)

5.74.2.3 LTC_EXPORT void LTC_CALL XFREE (void * *p*)

5.74.2.4 LTC_EXPORT void* LTC_CALL XMALLOC (size_t *n*)

5.74.2.5 LTC_EXPORT int LTC_CALL XMEMCMP (const void * *s1*, const void * *s2*, size_t *n*)

5.74.2.6 LTC_EXPORT void* LTC_CALL XMEMCPY (void * *dest*, const void * *src*, size_t *n*)

5.74.2.7 LTC_EXPORT void* LTC_CALL XMEMSET (void * *s*, int *c*, size_t *n*)

5.74.2.8 LTC_EXPORT void LTC_CALL XQSORT (void * *base*, size_t *nmemb*, size_t *size*, int(*)(const void *, const void *) *compar*)

5.74.2.9 LTC_EXPORT void* LTC_CALL XREALLOC (void * *p*, size_t *n*)

5.75 headers/tomcrypt_cipher.h File Reference

This graph shows which files directly or indirectly include this file:

Data Structures

- union Symmetric_key
- struct ltc_cipher_descriptor

  cipher descriptor table, last entry has "name == NULL" to mark the end of table

Typedefs

- typedef Symmetric_key symmetric_key

Functions

- int find_cipher (const char ∗name)

  Find a registered cipher by name.
- int find_cipher_any (const char ∗name, int blocklen, int keylen)

  Find a cipher flexibly.
- int find_cipher_id (unsigned char ID)

  Find a cipher by ID number.
- int register_cipher (const struct ltc_cipher_descriptor ∗cipher)

  Register a cipher with the descriptor table.
- int unregister_cipher (const struct ltc_cipher_descriptor ∗cipher)

  Unregister a cipher from the descriptor table.
- int cipher_is_valid (int idx)

Variables

- ltc_cipher_descriptor cipher_descriptor [ ]

  cipher descriptor table, last entry has "name == NULL" to mark the end of table

5.75.1 Typedef Documentation

5.75.1.1 typedef union Symmetric_key symmetric_key

5.75.2 Function Documentation

5.75.2.1 int cipher_is_valid (int *idx*)

Definition at line 23 of file crypt_cipher_is_valid.c.

References cipher_descriptor, CRYPT_INVALID_CIPHER, CRYPT_OK, LTC_MUTEX_LOCK, LTC_-MUTEX_UNLOCK, ltc_cipher_descriptor::name, and TAB_SIZE.

Referenced by cbc_decrypt(), cbc_done(), cbc_encrypt(), cbc_start(), ccm_memory(), cfb_decrypt(), cfb_-done(), cfb_encrypt(), cfb_setiv(), cfb_start(), chc_init(), chc_register(), ctr_done(), ctr_encrypt(), ctr_-setiv(), ctr_start(), eax_init(), ecb_decrypt(), ecb_done(), ecb_encrypt(), ecb_start(), f8_done(), f8_-encrypt(), f8_setiv(), f8_start(), f9_done(), f9_init(), f9_memory(), f9_process(), gcm_add_aad(), gcm_-add_iv(), gcm_done(), gcm_init(), gcm_memory(), gcm_process(), lrw_decrypt(), lrw_done(), lrw_-encrypt(), lrw_setiv(), lrw_start(), ocb_decrypt(), ocb_encrypt(), ocb_init(), ofb_done(), ofb_encrypt(), ofb_setiv(), ofb_start(), omac_done(), omac_init(), omac_memory(), omac_process(), pmac_done(), pmac_init(), pmac_process(), s_ocb_done(), xcbc_done(), xcbc_init(), xcbc_memory(), xcbc_process(), yarrow_ready(), and yarrow_start().

```
{
   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   if (idx < 0 || idx >= TAB_SIZE || cipher_descriptor[idx].name == NULL) {
      LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
      return CRYPT_INVALID_CIPHER;
   }
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
   return CRYPT_OK;
}
```

5.75.2.2 int find_cipher (const char ∗ *name*)

Find a registered cipher by name.

Parameters:

name The name of the cipher to look for

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_cipher.c.

References cipher_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, and TAB_SIZE.

Referenced by ccm_test(), ctr_test(), eax_test(), f8_test_mode(), f9_test(), find_cipher_any(), gcm_test(), lrw_test(), ocb_test(), omac_test(), pmac_test(), and xcbc_test().

```
{
   int x;
   LTC_ARGCHK(name != NULL);
   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (cipher_descriptor[x].name != NULL && !strcmp(cipher_descriptor[x].name, name)) {
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return x;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
   return -1;
}
```

5.75.2.3 int find_cipher_any (const char ∗ *name*, int *blocklen*, int *keylen*)

Find a cipher flexibly.

First by name then if not present by block and key size

Parameters:

name The name of the cipher desired

blocklen The minimum length of the block cipher desired (octets)

keylen The minimum length of the key size desired (octets)

Returns:

>= 0 if found, -1 if not present

Definition at line 25 of file crypt_find_cipher_any.c.

References cipher_descriptor, find_cipher(), LTC_ARGCHK, LTC_MUTEX_LOCK, and TAB_SIZE.

```
{
   int x;

   LTC_ARGCHK(name != NULL);

   x = find_cipher(name);
   if (x != -1) return x;

   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (cipher_descriptor[x].name == NULL) {
          continue;
       }
       if (blocklen <= (int)cipher_descriptor[x].block_length && keylen <= (int)cipher_descriptor[x].ma
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return x;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
   return -1;
}
```

Here is the call graph for this function:

5.75.2.4 int find_cipher_id (unsigned char *ID*)

Find a cipher by ID number.

Parameters:

ID The ID (not same as index) of the cipher to find

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_cipher_id.c.

References cipher_descriptor, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_cipher_-descriptor::name, and TAB_SIZE.

```
{
   int x;
   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (cipher_descriptor[x].ID == ID) {
          x = (cipher_descriptor[x].name == NULL) ? -1 : x;
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return x;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
   return -1;
}
```

5.75.2.5 int register_cipher (const struct ltc_cipher_descriptor * *cipher*)

Register a cipher with the descriptor table.

Parameters:

cipher The cipher you wish to register

Returns:

value >= 0 if successfully added (or already present), -1 if unsuccessful

Definition at line 23 of file crypt_register_cipher.c.

References cipher_descriptor, ltc_cipher_descriptor::ID, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_-MUTEX_UNLOCK, ltc_prng_descriptor::name, and TAB_SIZE.

Referenced by crypt_fsa(), and yarrow_start().

```
{
   int x;

   LTC_ARGCHK(cipher != NULL);

   /* is it already registered? */
   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (cipher_descriptor[x].name != NULL && cipher_descriptor[x].ID == cipher->ID) {
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return x;
       }
   }

   /* find a blank spot */
   for (x = 0; x < TAB_SIZE; x++) {
       if (cipher_descriptor[x].name == NULL) {
          XMEMCPY(&cipher_descriptor[x], cipher, sizeof(struct ltc_cipher_descriptor));
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return x;
       }
   }

   /* no spot */
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
   return -1;
}
```

5.75.2.6 int unregister_cipher (const struct ltc_cipher_descriptor ∗ *cipher*)

Unregister a cipher from the descriptor table.

Parameters:
: ***cipher*** The cipher descriptor to remove

Returns:
: CRYPT_OK on success

Definition at line 23 of file crypt_unregister_cipher.c.

References cipher_descriptor, CRYPT_OK, ltc_cipher_descriptor::ID, LTC_ARGCHK, LTC_MUTEX_-LOCK, LTC_MUTEX_UNLOCK, ltc_prng_descriptor::name, TAB_SIZE, and XMEMCMP.

```
{
   int x;

   LTC_ARGCHK(cipher != NULL);

   /* is it already registered? */
   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (XMEMCMP(&cipher_descriptor[x], cipher, sizeof(struct ltc_cipher_descriptor)) == 0) {
          cipher_descriptor[x].name = NULL;
          cipher_descriptor[x].ID   = 255;
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return CRYPT_OK;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
   return CRYPT_ERROR;
}
```

5.75.3 Variable Documentation

5.75.3.1 struct ltc_cipher_descriptor cipher_descriptor[ ]

cipher descriptor table, last entry has "name == NULL" to mark the end of table

Referenced by cbc_decrypt(), cbc_done(), cbc_encrypt(), cbc_start(), ccm_memory(), ccm_test(), cfb_-decrypt(), cfb_done(), cfb_encrypt(), cfb_setiv(), cfb_start(), chc_compress(), chc_init(), chc_register(), cipher_is_valid(), ctr_done(), ctr_encrypt(), ctr_setiv(), ctr_start(), eax_init(), ecb_decrypt(), ecb_done(), ecb_encrypt(), ecb_start(), f8_done(), f8_encrypt(), f8_setiv(), f8_start(), f9_done(), f9_init(), f9_-memory(), f9_process(), find_cipher(), find_cipher_any(), find_cipher_id(), gcm_init(), gcm_memory(), lrw_decrypt(), lrw_done(), lrw_encrypt(), lrw_setiv(), lrw_start(), ocb_decrypt(), ocb_encrypt(), ocb_-init(), ofb_done(), ofb_encrypt(), ofb_setiv(), ofb_start(), omac_done(), omac_init(), omac_memory(), omac_process(), pmac_init(), pmac_process(), register_cipher(), s_ocb_done(), unregister_cipher(), xcbc_done(), xcbc_init(), xcbc_memory(), xcbc_process(), yarrow_ready(), and yarrow_test().

5.76 headers/tomcrypt_custom.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define XMALLOC malloc
- #define XREALLOC realloc
- #define XCALLOC calloc
- #define XFREE free
- #define XMEMSET memset
- #define XMEMCPY memcpy
- #define XMEMCMP memcmp
- #define XCLOCK clock
- #define XCLOCKS_PER_SEC CLOCKS_PER_SEC
- #define XQSORT qsort
- #define LTC_TEST
- #define BLOWFISH
- #define RC2
- #define RC5
- #define RC6
- #define SAFERP
- #define RIJNDAEL
- #define XTEA
- #define TWOFISH
- #define TWOFISH_TABLES
- #define DES
- #define CAST5
- #define NOEKEON
- #define SKIPJACK
- #define SAFER
- #define KHAZAD
- #define ANUBIS
- #define ANUBIS_TWEAK
- #define KSEED
- #define LTC_KASUMI
- #define LTC_CFB_MODE
- #define LTC_OFB_MODE
- #define LTC_ECB_MODE
- #define LTC_CBC_MODE
- #define LTC_CTR_MODE
- #define LTC_F8_MODE
- #define LTC_LRW_MODE
- #define LRW_TABLES
- #define CHC_HASH
- #define WHIRLPOOL
- #define SHA512
- #define SHA384
- #define SHA256

- #define SHA224
- #define TIGER
- #define SHA1
- #define MD5
- #define MD4
- #define MD2
- #define RIPEMD128
- #define RIPEMD160
- #define RIPEMD256
- #define RIPEMD320
- #define LTC_HMAC
- #define LTC_OMAC
- #define LTC_PMAC
- #define LTC_XCBC
- #define LTC_F9_MODE
- #define PELICAN
- #define EAX_MODE
- #define OCB_MODE
- #define CCM_MODE
- #define GCM_MODE
- #define GCM_TABLES
- #define BASE64
- #define YARROW
- #define YARROW_AES 0
- #define SPRNG
- #define RC4
- #define FORTUNA
- #define FORTUNA_WD 10
- #define FORTUNA_POOLS 32
- #define SOBER128
- #define DEVRANDOM
- #define TRY_URANDOM_FIRST
- #define MRSA
- #define MDSA
- #define MECC
- #define PKCS_1
- #define PKCS_5
- #define LTC_DER
- #define ECC112
- #define ECC128
- #define ECC160
- #define ECC192
- #define ECC224
- #define ECC256
- #define ECC384
- #define ECC521
- #define MPI
- #define PKCS_1
- #define LTC_MUTEX_GLOBAL(x)
- #define LTC_MUTEX_PROTO(x)

- #define LTC_MUTEX_TYPE(x)
- #define LTC_MUTEX_INIT(x)
- #define LTC_MUTEX_LOCK(x)
- #define LTC_MUTEX_UNLOCK(x)

5.76.1 Define Documentation

5.76.1.1 #define ANUBIS

Definition at line 125 of file tomcrypt_custom.h.

5.76.1.2 #define ANUBIS_TWEAK

Definition at line 126 of file tomcrypt_custom.h.

5.76.1.3 #define BASE64

Definition at line 215 of file tomcrypt_custom.h.

5.76.1.4 #define BLOWFISH

Definition at line 101 of file tomcrypt_custom.h.

5.76.1.5 #define CAST5

Definition at line 120 of file tomcrypt_custom.h.

5.76.1.6 #define CCM_MODE

Definition at line 199 of file tomcrypt_custom.h.

5.76.1.7 #define CHC_HASH

Definition at line 159 of file tomcrypt_custom.h.

5.76.1.8 #define DES

Definition at line 119 of file tomcrypt_custom.h.

5.76.1.9 #define DEVRANDOM

Definition at line 247 of file tomcrypt_custom.h.

5.76.1.10 #define EAX_MODE

Definition at line 193 of file tomcrypt_custom.h.

5.76.1.11 #define ECC112

Definition at line 307 of file tomcrypt_custom.h.

5.76.1.12 #define ECC128

Definition at line 308 of file tomcrypt_custom.h.

5.76.1.13 #define ECC160

Definition at line 309 of file tomcrypt_custom.h.

5.76.1.14 #define ECC192

Definition at line 310 of file tomcrypt_custom.h.

5.76.1.15 #define ECC224

Definition at line 311 of file tomcrypt_custom.h.

5.76.1.16 #define ECC256

Definition at line 312 of file tomcrypt_custom.h.

5.76.1.17 #define ECC384

Definition at line 313 of file tomcrypt_custom.h.

5.76.1.18 #define ECC521

Definition at line 314 of file tomcrypt_custom.h.

5.76.1.19 #define FORTUNA

Definition at line 237 of file tomcrypt_custom.h.

5.76.1.20 #define FORTUNA_POOLS 32

Definition at line 241 of file tomcrypt_custom.h.

Referenced by fortuna_add_entropy(), fortuna_done(), fortuna_export(), fortuna_import(), fortuna_-reseed(), and fortuna_start().

5.76.1.21 #define FORTUNA_WD 10

Definition at line 239 of file tomcrypt_custom.h.

Referenced by fortuna_read().

5.76.1.22 #define GCM_MODE

Definition at line 200 of file tomcrypt_custom.h.

5.76.1.23 #define GCM_TABLES

Definition at line 204 of file tomcrypt_custom.h.

5.76.1.24 #define KHAZAD

Definition at line 124 of file tomcrypt_custom.h.

5.76.1.25 #define KSEED

Definition at line 127 of file tomcrypt_custom.h.

5.76.1.26 #define LRW_TABLES

Definition at line 151 of file tomcrypt_custom.h.

5.76.1.27 #define LTC_CBC_MODE

Definition at line 139 of file tomcrypt_custom.h.

5.76.1.28 #define LTC_CFB_MODE

Definition at line 136 of file tomcrypt_custom.h.

5.76.1.29 #define LTC_CTR_MODE

Definition at line 140 of file tomcrypt_custom.h.

5.76.1.30 #define LTC_DER

Definition at line 298 of file tomcrypt_custom.h.

5.76.1.31 #define LTC_ECB_MODE

Definition at line 138 of file tomcrypt_custom.h.

5.76.1.32 #define LTC_F8_MODE

Definition at line 143 of file tomcrypt_custom.h.

5.76.1.33 #define LTC_F9_MODE

Definition at line 184 of file tomcrypt_custom.h.

5.76.1.34 #define LTC_HMAC

Definition at line 180 of file tomcrypt_custom.h.

5.76.1.35 #define LTC_KASUMI

Definition at line 128 of file tomcrypt_custom.h.

5.76.1.36 #define LTC_LRW_MODE

Definition at line 146 of file tomcrypt_custom.h.

5.76.1.37 #define LTC_MUTEX_GLOBAL(x)

Definition at line 350 of file tomcrypt_custom.h.

5.76.1.38 #define LTC_MUTEX_INIT(x)

Definition at line 353 of file tomcrypt_custom.h.

Referenced by yarrow_start().

5.76.1.39 #define LTC_MUTEX_LOCK(x)

Definition at line 354 of file tomcrypt_custom.h.

Referenced by cipher_is_valid(), find_cipher(), find_cipher_any(), find_cipher_id(), find_hash(), find_hash_any(), find_hash_id(), find_hash_oid(), find_prng(), fortuna_add_entropy(), fortuna_-done(), fortuna_export(), fortuna_read(), hash_is_valid(), prng_is_valid(), register_cipher(), register_-hash(), register_prng(), unregister_cipher(), unregister_hash(), unregister_prng(), yarrow_add_entropy(), yarrow_done(), yarrow_export(), yarrow_import(), yarrow_read(), and yarrow_ready().

5.76.1.40 #define LTC_MUTEX_PROTO(x)

Definition at line 351 of file tomcrypt_custom.h.

5.76.1.41 #define LTC_MUTEX_TYPE(x)

Definition at line 352 of file tomcrypt_custom.h.

5.76.1.42 #define LTC_MUTEX_UNLOCK(x)

Definition at line 355 of file tomcrypt_custom.h.

Referenced by cipher_is_valid(), find_cipher(), find_cipher_id(), find_hash(), find_hash_id(), find_-hash_oid(), find_prng(), fortuna_add_entropy(), fortuna_done(), fortuna_export(), fortuna_read(), hash_is_valid(), prng_is_valid(), register_cipher(), register_hash(), register_prng(), unregister_cipher(), unregister_hash(), unregister_prng(), yarrow_add_entropy(), yarrow_done(), yarrow_export(), yarrow_-import(), yarrow_read(), and yarrow_ready().

5.76.1.43 #define LTC_OFB_MODE

Definition at line 137 of file tomcrypt_custom.h.

5.76.1.44 #define LTC_OMAC

Definition at line 181 of file tomcrypt_custom.h.

5.76.1.45 #define LTC_PMAC

Definition at line 182 of file tomcrypt_custom.h.

5.76.1.46 #define LTC_TEST

Definition at line 80 of file tomcrypt_custom.h.

5.76.1.47 #define LTC_XCBC

Definition at line 183 of file tomcrypt_custom.h.

5.76.1.48 #define MD2

Definition at line 169 of file tomcrypt_custom.h.

5.76.1.49 #define MD4

Definition at line 168 of file tomcrypt_custom.h.

5.76.1.50 #define MD5

Definition at line 167 of file tomcrypt_custom.h.

5.76.1.51 #define MDSA

Definition at line 274 of file tomcrypt_custom.h.

5.76.1.52 #define MECC

Definition at line 277 of file tomcrypt_custom.h.

5.76.1.53 #define MPI

Definition at line 320 of file tomcrypt_custom.h.

5.76.1.54 #define MRSA

Definition at line 268 of file tomcrypt_custom.h.

5.76.1.55 #define NOEKEON

Definition at line 121 of file tomcrypt_custom.h.

5.76.1.56 #define OCB_MODE

Definition at line 198 of file tomcrypt_custom.h.

5.76.1.57 #define PELICAN

Definition at line 185 of file tomcrypt_custom.h.

5.76.1.58 #define PKCS_1

Definition at line 324 of file tomcrypt_custom.h.

5.76.1.59 #define PKCS_1

Definition at line 324 of file tomcrypt_custom.h.

5.76.1.60 #define PKCS_5

Definition at line 295 of file tomcrypt_custom.h.

5.76.1.61 #define RC2

Definition at line 102 of file tomcrypt_custom.h.

5.76.1.62 #define RC4

Definition at line 234 of file tomcrypt_custom.h.

5.76.1.63 #define RC5

Definition at line 103 of file tomcrypt_custom.h.

5.76.1.64 #define RC6

Definition at line 104 of file tomcrypt_custom.h.

5.76.1.65 #define RIJNDAEL

Definition at line 106 of file tomcrypt_custom.h.

5.76.1.66 #define RIPEMD128

Definition at line 170 of file tomcrypt_custom.h.

5.76.1.67 #define RIPEMD160

Definition at line 171 of file tomcrypt_custom.h.

5.76.1.68 #define RIPEMD256

Definition at line 172 of file tomcrypt_custom.h.

5.76.1.69 #define RIPEMD320

Definition at line 173 of file tomcrypt_custom.h.

5.76.1.70 #define SAFER

Definition at line 123 of file tomcrypt_custom.h.

5.76.1.71 #define SAFERP

Definition at line 105 of file tomcrypt_custom.h.

5.76.1.72 #define SHA1

Definition at line 166 of file tomcrypt_custom.h.

5.76.1.73 #define SHA224

Definition at line 164 of file tomcrypt_custom.h.

5.76.1.74 #define SHA256

Definition at line 163 of file tomcrypt_custom.h.

5.76.1.75 #define SHA384

Definition at line 162 of file tomcrypt_custom.h.

5.76.1.76 #define SHA512

Definition at line 161 of file tomcrypt_custom.h.

5.76.1.77 #define SKIPJACK

Definition at line 122 of file tomcrypt_custom.h.

5.76.1.78 #define SOBER128

Definition at line 244 of file tomcrypt_custom.h.

5.76.1.79 #define SPRNG

Definition at line 231 of file tomcrypt_custom.h.

5.76.1.80 #define TIGER

Definition at line 165 of file tomcrypt_custom.h.

5.76.1.81 #define TRY_URANDOM_FIRST

Definition at line 249 of file tomcrypt_custom.h.

5.76.1.82 #define TWOFISH

Definition at line 110 of file tomcrypt_custom.h.

5.76.1.83 #define TWOFISH_TABLES

Definition at line 112 of file tomcrypt_custom.h.

5.76.1.84 #define WHIRLPOOL

Definition at line 160 of file tomcrypt_custom.h.

5.76.1.85 #define XCALLOC calloc

Definition at line 12 of file tomcrypt_custom.h.

Referenced by der_decode_sequence_flexi(), der_encode_set(), f9_memory(), rand_prime(), rsa_import(), xcbc_init(), and xcbc_memory().

5.76.1.86 #define XCLOCK clock

Definition at line 29 of file tomcrypt_custom.h.

5.76.1.87 #define XCLOCKS_PER_SEC CLOCKS_PER_SEC

Definition at line 32 of file tomcrypt_custom.h.

5.76.1.88 #define XFREE free

Definition at line 15 of file tomcrypt_custom.h.

Referenced by ccm_memory(), chc_compress(), chc_init(), der_decode_sequence_flexi(), der_sequence_-free(), dsa_decrypt_key(), dsa_encrypt_key(), dsa_sign_hash_raw(), eax_decrypt_verify_memory(), eax_-done(), eax_encrypt_authenticate_memory(), eax_init(), ecc_decrypt_key(), ecc_encrypt_key(), f9_-memory(), gcm_memory(), hash_memory(), hmac_done(), hmac_init(), hmac_memory(), ltc_ecc_-del_point(), ltc_ecc_new_point(), ocb_decrypt_verify_memory(), ocb_done_decrypt(), ocb_encrypt_-authenticate_memory(), omac_memory(), pelican_memory(), pkcs_1_mgf1(), pkcs_1_oaep_decode(), pkcs_1_oaep_encode(), pkcs_1_pss_decode(), pkcs_1_pss_encode(), pkcs_5_alg1(), pkcs_5_alg2(), pmac_memory(), rand_prime(), rsa_decrypt_key_ex(), rsa_sign_hash_ex(), rsa_verify_hash_ex(), s_ocb_-done(), and xcbc_memory().

5.76.1.89 #define XMALLOC malloc

Definition at line 6 of file tomcrypt_custom.h.

Referenced by ccm_memory(), chc_compress(), chc_init(), dsa_decrypt_key(), dsa_encrypt_key(), dsa_make_key(), dsa_sign_hash_raw(), eax_decrypt_verify_memory(), eax_done(), eax_encrypt_-authenticate_memory(), eax_init(), ecc_decrypt_key(), ecc_encrypt_key(), f9_memory_multi(), fortuna_-export(), gcm_memory(), hash_memory(), hash_memory_multi(), hmac_done(), hmac_init(), hmac_-memory(), hmac_memory_multi(), ltc_ecc_new_point(), ocb_decrypt_verify_memory(), ocb_done_-decrypt(), ocb_encrypt_authenticate_memory(), omac_memory(), omac_memory_multi(), pelican_-memory(), pkcs_1_mgf1(), pkcs_1_oaep_decode(), pkcs_1_oaep_encode(), pkcs_1_pss_decode(), pkcs_-1_pss_encode(), pkcs_5_alg1(), pkcs_5_alg2(), pmac_memory(), pmac_memory_multi(), rsa_decrypt_-key_ex(), rsa_sign_hash_ex(), rsa_verify_hash_ex(), s_ocb_done(), and xcbc_memory_multi().

5.76.1.90 #define XMEMCMP memcmp

Definition at line 25 of file tomcrypt_custom.h.

Referenced by anubis_test(), ccm_test(), eax_decrypt_verify_memory(), f8_test_mode(), find_hash_-oid(), khazad_test(), kseed_test(), md2_test(), md4_test(), md5_test(), ocb_done_decrypt(), qsort_helper(), register_hash(), register_prng(), rmd128_test(), rmd160_test(), rmd256_test(), rmd320_test(), rsa_verify_-hash_ex(), safer_k64_test(), safer_sk128_test(), safer_sk64_test(), sha1_test(), sha224_test(), sha256_-test(), sha384_test(), sha512_test(), tiger_test(), unregister_cipher(), unregister_hash(), unregister_prng(), whirlpool_test(), and xtea_test().

5.76.1.91 #define XMEMCPY memcpy

Definition at line 22 of file tomcrypt_custom.h.

Referenced by cast5_setup(), cbc_getiv(), cbc_setiv(), cfb_getiv(), chc_compress(), crypt_fsa(), ctr_-getiv(), ctr_setiv(), ecc_ansi_x963_export(), f8_getiv(), fortuna_export(), fortuna_read(), gcm_add_aad(), gcm_gf_mult(), hmac_init(), lrw_getiv(), lrw_process(), lrw_setiv(), lrw_start(), md2_done(), md2_-process(), ofb_getiv(), pkcs_1_oaep_decode(), pkcs_1_oaep_encode(), pkcs_1_pss_decode(), pkcs_1_-pss_encode(), pkcs_5_alg2(), rc4_ready(), s_ocb_done(), sha224_done(), and sha384_done().

5.76.1.92 #define XMEMSET memset

Definition at line 19 of file tomcrypt_custom.h.

Referenced by pkcs_1_oaep_encode(), and pkcs_1_pss_encode().

5.76.1.93 #define XQSORT qsort

Definition at line 36 of file tomcrypt_custom.h.

5.76.1.94 #define XREALLOC realloc

Definition at line 9 of file tomcrypt_custom.h.

Referenced by der_decode_sequence_flexi().

5.76.1.95 #define XTEA

Definition at line 107 of file tomcrypt_custom.h.

5.76.1.96 #define YARROW

Definition at line 221 of file tomcrypt_custom.h.

5.76.1.97 #define YARROW_AES 0

Definition at line 224 of file tomcrypt_custom.h.

5.77 headers/tomcrypt_hash.h File Reference

This graph shows which files directly or indirectly include this file:

Data Structures

- union Hash_state
- struct ltc_hash_descriptor

  hash descriptor

Defines

- #define HASH_PROCESS(func_name, compress_name, state_var, block_size)

Typedefs

- typedef Hash_state hash_state

Functions

- int find_hash (const char ∗name)

  Find a registered hash by name.

- int find_hash_id (unsigned char ID)

  Find a hash by ID number.

- int find_hash_oid (const unsigned long ∗ID, unsigned long IDlen)
- int find_hash_any (const char ∗name, int digestlen)

  Find a hash flexibly.

- int register_hash (const struct ltc_hash_descriptor ∗hash)

  Register a hash with the descriptor table.

- int unregister_hash (const struct ltc_hash_descriptor ∗hash)

  Unregister a hash from the descriptor table.

- int hash_is_valid (int idx)
- int hash_memory (int hash, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Hash a block of memory and store the digest.

- int hash_memory_multi (int hash, unsigned char ∗out, unsigned long ∗outlen, const unsigned char ∗in, unsigned long inlen,...)

  Hash multiple (non-adjacent) blocks of memory at once.

- int hash_filehandle (int hash, FILE ∗in, unsigned char ∗out, unsigned long ∗outlen)

  Hash data from an open file handle.

- int hash_file (int hash, const char ∗fname, unsigned char ∗out, unsigned long ∗outlen)

Variables

- ltc_hash_descriptor hash_descriptor [ ]

  hash descriptor

5.77.1 Define Documentation

5.77.1.1 #define HASH_PROCESS(func_name, compress_name, state_var, block_size)

Definition at line 341 of file tomcrypt_hash.h.

5.77.2 Typedef Documentation

5.77.2.1 typedef union Hash_state hash_state

5.77.3 Function Documentation

5.77.3.1 int find_hash (const char ∗ *name*)

Find a registered hash by name.

Parameters:

name The name of the hash to look for

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_hash.c.

References hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, and TAB_SIZE.

Referenced by chc_register(), find_hash_any(), and hmac_test().

```
{
   int x;
   LTC_ARGCHK(name != NULL);
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (hash_descriptor[x].name != NULL && strcmp(hash_descriptor[x].name, name) == 0) {
          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
          return x;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return -1;
}
```

5.77.3.2 int find_hash_any (const char ∗ *name*, int *digestlen*)

Find a hash flexibly.

First by name then if not present by digest size

Parameters:

name The name of the hash desired

digestlen The minimum length of the digest size (octets)

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_hash_any.c.

References find_hash(), hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, MAXBLOCKSIZE, and TAB_SIZE.

```
{
   int x, y, z;
   LTC_ARGCHK(name != NULL);

   x = find_hash(name);
   if (x != -1) return x;

   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   y = MAXBLOCKSIZE+1;
   z = -1;
   for (x = 0; x < TAB_SIZE; x++) {
       if (hash_descriptor[x].name == NULL) {
          continue;
       }
       if ((int)hash_descriptor[x].hashsize >= digestlen && (int)hash_descriptor[x].hashsize < y) {
          z = x;
          y = hash_descriptor[x].hashsize;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return z;
}
```

Here is the call graph for this function:

5.77.3.3 int find_hash_id (unsigned char *ID*)

Find a hash by ID number.

Parameters:

ID The ID (not same as index) of the hash to find

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_hash_id.c.

References hash_descriptor, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_cipher_-descriptor::name, and TAB_SIZE.

```
{
   int x;
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
      if (hash_descriptor[x].ID == ID) {
          x = (hash_descriptor[x].name == NULL) ? -1 : x;
```

```
30          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
31          return x;
32       }
33    }
34    LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
35    return -1;
36 }
```

5.77.3.4 int find_hash_oid (const unsigned long ∗ *ID*, unsigned long *IDlen*)

Definition at line 18 of file crypt_find_hash_oid.c.

References hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_-cipher_descriptor::name, TAB_SIZE, and XMEMCMP.

Referenced by dsa_decrypt_key(), and ecc_decrypt_key().

```
19 {
20    int x;
21    LTC_ARGCHK(ID != NULL);
22    LTC_MUTEX_LOCK(&ltc_hash_mutex);
23    for (x = 0; x < TAB_SIZE; x++) {
24        if (hash_descriptor[x].name != NULL && hash_descriptor[x].OIDlen == IDlen && !XMEMCMP(hash_descr
25           LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
26           return x;
27        }
28    }
29    LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
30    return -1;
31 }
```

5.77.3.5 int hash_file (int *hash*, const char ∗ *fname*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Parameters:

hash The index of the hash desired

fname The name of the file you wish to hash

out [out] The destination of the digest

outlen [in/out] The max size and resulting size of the message digest

Returns:

CRYPT_OK if successful

Definition at line 25 of file hash_file.c.

References CRYPT_ERROR, CRYPT_FILE_NOTFOUND, CRYPT_NOP, CRYPT_OK, hash_-filehandle(), hash_is_valid(), in, and LTC_ARGCHK.

```
26 {
27 #ifdef LTC_NO_FILE
28     return CRYPT_NOP;
29 #else
30     FILE *in;
31     int err;
32     LTC_ARGCHK(fname  != NULL);
33     LTC_ARGCHK(out    != NULL);
34     LTC_ARGCHK(outlen != NULL);
35
```

```
    if ((err = hash_is_valid(hash)) != CRYPT_OK) {
        return err;
    }

    in = fopen(fname, "rb");
    if (in == NULL) {
       return CRYPT_FILE_NOTFOUND;
    }

    err = hash_filehandle(hash, in, out, outlen);
    if (fclose(in) != 0) {
       return CRYPT_ERROR;
    }

    return err;
#endif
}
```

Here is the call graph for this function:

5.77.3.6 int hash_filehandle (int *hash*, FILE ∗ *in*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Hash data from an open file handle.

Parameters:

hash The index of the hash you want to use

in The FILE∗ handle of the file you want to hash

out [out] The destination of the digest

outlen [in/out] The max size and resulting size of the digest

Returns:

CRYPT_OK if successful

Definition at line 26 of file hash_filehandle.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_NOP, CRYPT_OK, ltc_hash_descriptor::done, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, and zeromem().

Referenced by hash_file().

```
{
#ifdef LTC_NO_FILE
    return CRYPT_NOP;
#else
    hash_state md;
    unsigned char buf[512];
    size_t x;
    int err;

    LTC_ARGCHK(out    != NULL);
    LTC_ARGCHK(outlen != NULL);
    LTC_ARGCHK(in     != NULL);

    if ((err = hash_is_valid(hash)) != CRYPT_OK) {
        return err;
    }

    if (*outlen < hash_descriptor[hash].hashsize) {
       *outlen = hash_descriptor[hash].hashsize;
       return CRYPT_BUFFER_OVERFLOW;
```

```
    }
    if ((err = hash_descriptor[hash].init(&md)) != CRYPT_OK) {
       return err;
    }

    *outlen = hash_descriptor[hash].hashsize;
    do {
        x = fread(buf, 1, sizeof(buf), in);
        if ((err = hash_descriptor[hash].process(&md, buf, x)) != CRYPT_OK) {
           return err;
        }
    } while (x == sizeof(buf));
    err = hash_descriptor[hash].done(&md, out);

#ifdef LTC_CLEAN_STACK
    zeromem(buf, sizeof(buf));
#endif
    return err;
#endif
}
```

Here is the call graph for this function:

5.77.3.7 int hash_is_valid (int *idx*)

Definition at line 23 of file crypt_hash_is_valid.c.

References CRYPT_INVALID_HASH, CRYPT_OK, hash_descriptor, LTC_MUTEX_LOCK, LTC_-MUTEX_UNLOCK, ltc_hash_descriptor::name, and TAB_SIZE.

Referenced by chc_register(), dsa_decrypt_key(), dsa_encrypt_key(), ecc_decrypt_key(), ecc_encrypt_-key(), hash_file(), hash_filehandle(), hash_memory(), hash_memory_multi(), hmac_done(), hmac_file(), hmac_init(), hmac_memory(), hmac_process(), pkcs_1_mgf1(), pkcs_1_oaep_decode(), pkcs_1_oaep_-encode(), pkcs_1_pss_decode(), pkcs_1_pss_encode(), pkcs_5_alg1(), pkcs_5_alg2(), rsa_decrypt_key_-ex(), rsa_encrypt_key_ex(), rsa_sign_hash_ex(), rsa_verify_hash_ex(), yarrow_add_entropy(), yarrow_-ready(), and yarrow_start().

```
{
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   if (idx < 0 || idx >= TAB_SIZE || hash_descriptor[idx].name == NULL) {
      LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
      return CRYPT_INVALID_HASH;
   }
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return CRYPT_OK;
}
```

5.77.3.8 int hash_memory (int *hash*, const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Hash a block of memory and store the digest.

Parameters:

hash The index of the hash you wish to use

in The data you wish to hash

inlen The length of the data to hash (octets)

out [out] Where to store the digest

outlen [in/out] Max size and resulting size of the digest

Returns:

CRYPT_OK if successful

Definition at line 27 of file hash_memory.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_MEM, CRYPT_OK, ltc_hash_descriptor::done, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, XFREE, XMALLOC, and zeromem().

Referenced by dsa_decrypt_key(), dsa_encrypt_key(), ecc_decrypt_key(), ecc_encrypt_key(), hmac_init(), pkcs_1_oaep_encode(), and pkcs_5_alg1().

```
{
    hash_state *md;
    int err;

    LTC_ARGCHK(in     != NULL);
    LTC_ARGCHK(out    != NULL);
    LTC_ARGCHK(outlen != NULL);

    if ((err = hash_is_valid(hash)) != CRYPT_OK) {
        return err;
    }

    if (*outlen < hash_descriptor[hash].hashsize) {
       *outlen = hash_descriptor[hash].hashsize;
       return CRYPT_BUFFER_OVERFLOW;
    }

    md = XMALLOC(sizeof(hash_state));
    if (md == NULL) {
       return CRYPT_MEM;
    }

    if ((err = hash_descriptor[hash].init(md)) != CRYPT_OK) {
       goto LBL_ERR;
    }
    if ((err = hash_descriptor[hash].process(md, in, inlen)) != CRYPT_OK) {
       goto LBL_ERR;
    }
    err = hash_descriptor[hash].done(md, out);
    *outlen = hash_descriptor[hash].hashsize;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
#endif
    XFREE(md);

    return err;
}
```

Here is the call graph for this function:

5.77.3.9 int hash_memory_multi (int *hash*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, const unsigned char ∗ *in*, unsigned long *inlen*, ...)

Hash multiple (non-adjacent) blocks of memory at once.

Parameters:

hash The index of the hash you wish to use

out [out] Where to store the digest

outlen [in/out] Max size and resulting size of the digest

in The data you wish to hash

inlen The length of the data to hash (octets)

... tuples of (data,len) pairs to hash, terminated with a (NULL,x) (x=don't care)

Returns:

CRYPT_OK if successful

Definition at line 28 of file hash_memory_multi.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_MEM, CRYPT_OK, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, and XMALLOC.

```
{
    hash_state          *md;
    int                  err;
    va_list              args;
    const unsigned char *curptr;
    unsigned long        curlen;

    LTC_ARGCHK(in     != NULL);
    LTC_ARGCHK(out    != NULL);
    LTC_ARGCHK(outlen != NULL);

    if ((err = hash_is_valid(hash)) != CRYPT_OK) {
        return err;
    }

    if (*outlen < hash_descriptor[hash].hashsize) {
       *outlen = hash_descriptor[hash].hashsize;
       return CRYPT_BUFFER_OVERFLOW;
    }

    md = XMALLOC(sizeof(hash_state));
    if (md == NULL) {
       return CRYPT_MEM;
    }

    if ((err = hash_descriptor[hash].init(md)) != CRYPT_OK) {
       goto LBL_ERR;
    }

    va_start(args, inlen);
    curptr = in;
    curlen = inlen;
    for (;;) {
       /* process buf */
       if ((err = hash_descriptor[hash].process(md, curptr, curlen)) != CRYPT_OK) {
          goto LBL_ERR;
       }
       /* step to next */
       curptr = va_arg(args, const unsigned char*);
       if (curptr == NULL) {
          break;
       }
       curlen = va_arg(args, unsigned long);
    }
    err = hash_descriptor[hash].done(md, out);
    *outlen = hash_descriptor[hash].hashsize;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
    zeromem(md, sizeof(hash_state));
```

```
#endif
    XFREE(md);
    va_end(args);
    return err;
}
```

Here is the call graph for this function:

5.77.3.10 int register_hash (const struct ltc_hash_descriptor * *hash*)

Register a hash with the descriptor table.

Parameters:

hash The hash you wish to register

Returns:

value >= 0 if successfully added (or already present), -1 if unsuccessful

Definition at line 23 of file crypt_register_hash.c.

References hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, TAB_-SIZE, and XMEMCMP.

Referenced by crypt_fsa(), and yarrow_start().

```
{
   int x;

   LTC_ARGCHK(hash != NULL);

   /* is it already registered? */
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (XMEMCMP(&hash_descriptor[x], hash, sizeof(struct ltc_hash_descriptor)) == 0) {
          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
          return x;
       }
   }

   /* find a blank spot */
   for (x = 0; x < TAB_SIZE; x++) {
       if (hash_descriptor[x].name == NULL) {
          XMEMCPY(&hash_descriptor[x], hash, sizeof(struct ltc_hash_descriptor));
          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
          return x;
       }
   }

   /* no spot */
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return -1;
}
```

5.77.3.11 int unregister_hash (const struct ltc_hash_descriptor * *hash*)

Unregister a hash from the descriptor table.

Parameters:

hash The hash descriptor to remove

Returns:

CRYPT_OK on success

Definition at line 23 of file crypt_unregister_hash.c.

References CRYPT_OK, hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_-UNLOCK, ltc_prng_descriptor::name, TAB_SIZE, and XMEMCMP.

```
{
   int x;

   LTC_ARGCHK(hash != NULL);

   /* is it already registered? */
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (XMEMCMP(&hash_descriptor[x], hash, sizeof(struct ltc_hash_descriptor)) == 0) {
          hash_descriptor[x].name = NULL;
          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
          return CRYPT_OK;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return CRYPT_ERROR;
}
```

5.77.4 Variable Documentation

5.77.4.1 struct ltc_hash_descriptor hash_descriptor[ ]

hash descriptor

Referenced by chc_register(), dsa_encrypt_key(), ecc_encrypt_key(), find_hash(), find_hash_any(), find_-hash_id(), find_hash_oid(), hash_filehandle(), hash_is_valid(), hash_memory(), hash_memory_multi(), hmac_done(), hmac_init(), hmac_memory(), hmac_process(), pkcs_1_mgf1(), pkcs_1_oaep_decode(), pkcs_1_oaep_encode(), pkcs_1_pss_decode(), pkcs_1_pss_encode(), pkcs_5_alg1(), register_hash(), rsa_-sign_hash_ex(), rsa_verify_hash_ex(), unregister_hash(), yarrow_add_entropy(), yarrow_ready(), and yarrow_test().

5.78 headers/tomcrypt_mac.h File Reference

This graph shows which files directly or indirectly include this file:

5.79 headers/tomcrypt_macros.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- #define CONST64(n) n ## ULL
- #define BSWAP(x)
- #define ROL(x, y) ((((unsigned long)(x)<<(unsigned long)((y)&31)) | (((unsigned long)(x)&0xFFFFFFFFUL)>>(unsigned long)(32-((y)&31)))) & 0xFFFFFFFFUL)
- #define ROR(x, y) (((((unsigned long)(x)&0xFFFFFFFFUL)>>(unsigned long)((y)&31)) | ((unsigned long)(x)<<(unsigned long)(32-((y)&31)))) & 0xFFFFFFFFUL)
- #define ROLc(x, y) ((((unsigned long)(x)<<(unsigned long)((y)&31)) | (((unsigned long)(x)&0xFFFFFFFFUL)>>(unsigned long)(32-((y)&31)))) & 0xFFFFFFFFUL)
- #define RORc(x, y) (((((unsigned long)(x)&0xFFFFFFFFUL)>>(unsigned long)((y)&31)) | ((unsigned long)(x)<<(unsigned long)(32-((y)&31)))) & 0xFFFFFFFFUL)
- #define ROL64(x, y)
- #define ROR64(x, y)
- #define ROL64c(x, y)
- #define ROR64c(x, y)
- #define MAX(x, y) (((x)>(y))?(x):(y))
- #define MIN(x, y) (((x)<(y))?(x):(y))
- #define byte(x, n) (((x) >> (8 * (n))) & 255)

Typedefs

- typedef unsigned long long ulong64
- typedef unsigned long ulong32

5.79.1 Define Documentation

5.79.1.1 #define BSWAP(x)

Value:

```
( ((x>>24)&0x000000FFUL) | ((x<<24)&0xFF000000UL)  | \
                    ((x>>8)&0x0000FF00UL)  | ((x<<8)&0x00FF0000UL) )
```

Definition at line 230 of file tomcrypt_macros.h.

Referenced by rc5_setup(), and rc6_setup().

5.79.1.2 #define byte(x, n) (((x) >> (8 * (n))) & 255)

Definition at line 419 of file tomcrypt_macros.h.

Referenced by desfunc(), ECB_DEC(), ECB_ENC(), FI(), FII(), FIII(), four_rounds(), setup_mix(), and tiger_round().

5.79.1.3 #define CONST64(n) n ## ULL

Definition at line 6 of file tomcrypt_macros.h.

Referenced by gcm_add_aad(), gcm_done(), gcm_process(), key_schedule(), sha384_init(), sha512_-done(), sha512_init(), and tiger_init().

5.79.1.4 #define MAX(x, y) (((x)>(y))?(x):(y))

Definition at line 408 of file tomcrypt_macros.h.

5.79.1.5 #define MIN(x, y) (((x)<(y))?(x):(y))

Definition at line 412 of file tomcrypt_macros.h.

Referenced by dsa_decrypt_key(), ecc_decrypt_key(), md2_process(), and qsort_helper().

5.79.1.6 #define ROL(x, y) ((((unsigned long)(x)<<(unsigned long)((y)&31)) | (((unsigned long)(x)&0xFFFFFFFFUL)>>(unsigned long)(32-((y)&31)))) & 0xFFFFFFFFUL)

Definition at line 335 of file tomcrypt_macros.h.

Referenced by FI(), FII(), FIII(), and rc5_ecb_encrypt().

5.79.1.7 #define ROL64(x, y)

Value:

```
( (((x)<<((ulong64)(y)&63)) | \
      (((x)&CONST64(0xFFFFFFFFFFFFFFFF))>>((ulong64)64-((y)&63)))) & CONST64(0xFFFFFFFFFFFFFFFF))
```

Definition at line 389 of file tomcrypt_macros.h.

5.79.1.8 #define ROL64c(x, y)

Value:

```
( (((x)<<((ulong64)(y)&63)) | \
      (((x)&CONST64(0xFFFFFFFFFFFFFFFF))>>((ulong64)64-((y)&63)))) & CONST64(0xFFFFFFFFFFFFFFFF))
```

Definition at line 397 of file tomcrypt_macros.h.

5.79.1.9 #define ROLc(x, y) ((((unsigned long)(x)<<(unsigned long)((y)&31)) | (((unsigned long)(x)&0xFFFFFFFFUL)>>(unsigned long)(32-((y)&31)))) & 0xFFFFFFFFUL)

Definition at line 337 of file tomcrypt_macros.h.

Referenced by desfunc(), twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.79.1.10 #define ROR(x, y) (((((unsigned long)(x)&0xFFFFFFFFUL)>>(unsigned long)((y)&31)) | ((unsigned long)(x)<<(unsigned long)(32-((y)&31)))) & 0xFFFFFFFFUL)

Definition at line 336 of file tomcrypt_macros.h.

Referenced by rc5_ecb_decrypt().

5.79.1.11 #define ROR64(x, y)

Value:

```
( ((((x)&CONST64(0xFFFFFFFFFFFFFFFF))>>((ulong64)(y)&CONST64(63))) | \
      ((x)<<((ulong64)(64-((y)&CONST64(63)))))) & CONST64(0xFFFFFFFFFFFFFFFF))
```

Definition at line 393 of file tomcrypt_macros.h.

5.79.1.12 #define ROR64c(x, y)

Value:

```
( ((((x)&CONST64(0xFFFFFFFFFFFFFFFF))>>((ulong64)(y)&CONST64(63))) | \
      ((x)<<((ulong64)(64-((y)&CONST64(63)))))) & CONST64(0xFFFFFFFFFFFFFFFF))
```

Definition at line 401 of file tomcrypt_macros.h.

5.79.1.13 #define RORc(x, y) (((((unsigned long)(x)&0xFFFFFFFFUL)>>(unsigned long)((y)&31)) | ((unsigned long)(x)<<(unsigned long)(32-((y)&31)))) & 0xFFFFFFFFUL)

Definition at line 338 of file tomcrypt_macros.h.

Referenced by desfunc(), twofish_ecb_decrypt(), and twofish_ecb_encrypt().

5.79.2 Typedef Documentation

5.79.2.1 typedef unsigned long ulong32

Definition at line 16 of file tomcrypt_macros.h.

5.79.2.2 typedef unsigned long long ulong64

Definition at line 7 of file tomcrypt_macros.h.

5.80 headers/tomcrypt_math.h File Reference

This graph shows which files directly or indirectly include this file:

Data Structures

- struct ltc_math_descriptor

  math descriptor

Defines

- #define LTC_MP_LT -1

  math functions

- #define LTC_MP_EQ 0
- #define LTC_MP_GT 1
- #define LTC_MP_NO 0
- #define LTC_MP_YES 1

Typedefs

- typedef void ecc_point
- typedef void rsa_key

Functions

- int ltc_init_multi (void ∗∗a,...)
- void ltc_deinit_multi (void ∗a,...)

Variables

- ltc_math_descriptor ltc_mp

5.80.1 Define Documentation

5.80.1.1 #define LTC_MP_EQ 0

Definition at line 4 of file tomcrypt_math.h.

Referenced by dsa_sign_hash_raw(), dsa_verify_hash_raw(), dsa_verify_key(), is_point(), and ltc_ecc_-projective_add_point().

5.80.1.2 #define LTC_MP_GT 1

Definition at line 5 of file tomcrypt_math.h.

Referenced by der_encode_integer(), dsa_sign_hash_raw(), and dsa_verify_key().

5.80.1.3 #define LTC_MP_LT -1

math functions

Definition at line 3 of file tomcrypt_math.h.

Referenced by der_encode_integer(), der_length_integer(), dsa_verify_hash_raw(), dsa_verify_key(), ecc_verify_hash(), is_point(), ltc_ecc_projective_add_point(), ltc_ecc_projective_dbl_point(), and rsa_-exptmod().

5.80.1.4 #define LTC_MP_NO 0

Definition at line 7 of file tomcrypt_math.h.

5.80.1.5 #define LTC_MP_YES 1

Definition at line 8 of file tomcrypt_math.h.

Referenced by der_encode_integer(), der_length_integer(), dsa_sign_hash_raw(), dsa_verify_hash_raw(), and dsa_verify_key().

5.80.2 Typedef Documentation

5.80.2.1 typedef void ecc_point

Definition at line 11 of file tomcrypt_math.h.

5.80.2.2 typedef void rsa_key

Definition at line 15 of file tomcrypt_math.h.

5.80.3 Function Documentation

5.80.3.1 void ltc_deinit_multi (void $*$ *a*, ...)

Definition at line 44 of file multi.c.

```
{
   void      *cur = a;
   va_list   args;

   va_start(args, a);
   while (cur != NULL) {
       mp_clear(cur);
       cur = va_arg(args, void *);
   }
   va_end(args);
}
```

5.80.3.2 int ltc_init_multi (void $**$ *a*, ...)

Definition at line 16 of file multi.c.

References CRYPT_MEM, and CRYPT_OK.

```
{
    void     **cur = a;
    int        np  = 0;
    va_list    args;

    va_start(args, a);
    while (cur != NULL) {
        if (mp_init(cur) != CRYPT_OK) {
           /* failed */
           va_list clean_list;

           va_start(clean_list, a);
           cur = a;
           while (np--) {
               mp_clear(*cur);
               cur = va_arg(clean_list, void**);
           }
           va_end(clean_list);
           return CRYPT_MEM;
        }
        ++np;
        cur = va_arg(args, void**);
    }
    va_end(args);
    return CRYPT_OK;
}
```

5.80.4 Variable Documentation

5.80.4.1 ltc_math_descriptor ltc_mp

Definition at line 13 of file crypt_ltc_mp_descriptor.c.

Referenced by crypt_fsa(), dsa_import(), dsa_make_key(), ecc_import(), ecc_make_key(), ecc_shared_-secret(), ecc_verify_hash(), rsa_decrypt_key_ex(), rsa_encrypt_key_ex(), rsa_import(), rsa_make_key(), rsa_sign_hash_ex(), and rsa_verify_hash_ex().

5.81 headers/tomcrypt_misc.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void zeromem (void ∗dst, size_t len)

  Zero a block of memory.

- void burn_stack (unsigned long len)

  Burn some stack memory.

- const char ∗ error_to_string (int err)

  Convert an LTC error code to ASCII.

- int crypt_fsa (void ∗mp,...)

Variables

- const char ∗ crypt_build_settings

5.81.1 Function Documentation

5.81.1.1 void burn_stack (unsigned long *len*)

Burn some stack memory.

Parameters:

len amount of stack to burn in bytes

Definition at line 22 of file burn_stack.c.

References burn_stack(), and zeromem().

Referenced by burn_stack().

```
{
   unsigned char buf[32];
   zeromem(buf, sizeof(buf));
   if (len > (unsigned long)sizeof(buf))
      burn_stack(len - sizeof(buf));
}
```

Here is the call graph for this function:

5.81.1.2 int crypt_fsa (void ∗ *mp*, ...)

Definition at line 20 of file crypt_fsa.c.

References CRYPT_OK, ltc_mp, register_cipher(), register_hash(), register_prng(), and XMEMCPY.

```
{
   int       err;
   va_list  args;
   void     *p;

   va_start(args, mp);
   if (mp != NULL) {
      XMEMCPY(&ltc_mp, mp, sizeof(ltc_mp));
   }

   while ((p = va_arg(args, void*)) != NULL) {
      if ((err = register_cipher(p)) != CRYPT_OK) {
         va_end(args);
         return err;
      }
   }

   while ((p = va_arg(args, void*)) != NULL) {
      if ((err = register_hash(p)) != CRYPT_OK) {
         va_end(args);
         return err;
      }
   }

   while ((p = va_arg(args, void*)) != NULL) {
      if ((err = register_prng(p)) != CRYPT_OK) {
         va_end(args);
         return err;
      }
   }

   va_end(args);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.81.1.3 const char∗ error_to_string (int *err*)

Convert an LTC error code to ASCII.

Parameters:

err The error code

Returns:

A pointer to the ASCII NUL terminated string for the error or "Invalid error code." if the err code was not valid.

Definition at line 62 of file error_to_string.c.

References err_2_str.

Referenced by hmac_test().

```
{
   if (err < 0 || err >= (int)(sizeof(err_2_str)/sizeof(err_2_str[0]))) {
      return "Invalid error code.";
   } else {
      return err_2_str[err];
   }
}
```

5.81.1.4 void zeromem (void ∗ *out*, size_t *outlen*)

Zero a block of memory.

Parameters:

out The destination of the area to zero

outlen The length of the area to zero (octets)

Definition at line 23 of file zeromem.c.

References LTC_ARGCHKVD.

Referenced by burn_stack(), cast5_setup(), chc_init(), dsa_shared_secret(), dsa_sign_hash_raw(), eax_decrypt_verify_memory(), eax_encrypt_authenticate_memory(), eax_init(), ECB_TEST(), ecc_ansi_x963_export(), ecc_shared_secret(), f8_encrypt(), f8_start(), f9_file(), fortuna_read(), gcm_add_aad(), gcm_gf_mult(), gcm_init(), gcm_reset(), hash_filehandle(), hash_memory(), hmac_file(), hmac_init(), hmac_memory(), lrw_start(), md2_done(), md2_init(), noekeon_test(), ocb_decrypt_verify_memory(), ocb_done_decrypt(), ocb_encrypt_authenticate_memory(), omac_done(), omac_file(), omac_init(), omac_memory(), pelican_init(), pkcs_1_i2osp(), pkcs_1_pss_encode(), pkcs_5_alg2(), pmac_file(), pmac_memory(), rc2_test(), rc4_read(), rng_make_prng(), rsa_exptmod(), rsa_verify_hash_ex(), sha224_done(), sha384_done(), sober128_read(), tiger_done(), whirlpool_init(), xcbc_file(), yarrow_read(), and yarrow_start().

```
{
   unsigned char *mem = out;
   LTC_ARGCHKVD(out != NULL);
   while (outlen-- > 0) {
      *mem++ = 0;
   }
}
```

5.81.2 Variable Documentation

5.81.2.1 const char∗ crypt_build_settings

Definition at line 18 of file crypt.c.

5.82 headers/tomcrypt_pk.h File Reference

This graph shows which files directly or indirectly include this file:

Enumerations

- enum {
  PK_PUBLIC = 0,
  PK_PRIVATE = 1 }

Functions

- int rand_prime (void ∗N, long len, prng_state ∗prng, int wprng)

5.82.1 Enumeration Type Documentation

5.82.1.1 anonymous enum

Enumerator:

PK_PUBLIC

PK_PRIVATE

Definition at line 3 of file tomcrypt_pk.h.

```
3      {
4    PK_PUBLIC=0,
5    PK_PRIVATE=1
6 };
```

5.82.2 Function Documentation

5.82.2.1 int rand_prime (void ∗ *N*, long *len*, prng_state ∗ *prng*, int *wprng*)

Definition at line 20 of file rand_prime.c.

References CRYPT_ERROR_READPRNG, CRYPT_INVALID_PRIME_SIZE, CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, prng_descriptor, prng_is_valid(), USE_BBS, XCALLOC, and XFREE.

Referenced by dsa_make_key(), and rsa_make_key().

```
21 {
22    int            err, res, type;
23    unsigned char *buf;
24
25    LTC_ARGCHK(N != NULL);
26
27    /* get type */
28    if (len < 0) {
29       type = USE_BBS;
30       len = -len;
31    } else {
32       type = 0;
33    }
```

```

   /* allow sizes between 2 and 512 bytes for a prime size */
   if (len < 2 || len > 512) {
      return CRYPT_INVALID_PRIME_SIZE;
   }

   /* valid PRNG? Better be! */
   if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
      return err;
   }

   /* allocate buffer to work with */
   buf = XCALLOC(1, len);
   if (buf == NULL) {
       return CRYPT_MEM;
   }

   do {
      /* generate value */
      if (prng_descriptor[wprng].read(buf, len, prng) != (unsigned long)len) {
         XFREE(buf);
         return CRYPT_ERROR_READPRNG;
      }

      /* munge bits */
      buf[0]     |= 0x80 | 0x40;
      buf[len-1] |= 0x01 | ((type & USE_BBS) ? 0x02 : 0x00);

      /* load value */
      if ((err = mp_read_unsigned_bin(N, buf, len)) != CRYPT_OK) {
         XFREE(buf);
         return err;
      }

      /* test */
      if ((err = mp_prime_is_prime(N, 8, &res)) != CRYPT_OK) {
         XFREE(buf);
         return err;
      }
   } while (res == LTC_MP_NO);

#ifdef LTC_CLEAN_STACK
   zeromem(buf, len);
#endif

   XFREE(buf);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.83 headers/tomcrypt_pkcs.h File Reference

This graph shows which files directly or indirectly include this file:

5.84 headers/tomcrypt_prng.h File Reference

This graph shows which files directly or indirectly include this file:

Data Structures

- union Prng_state
- struct ltc_prng_descriptor

  PRNG descriptor.

Typedefs

- typedef Prng_state prng_state

Functions

- int find_prng (const char ∗name)

  Find a registered PRNG by name.

- int register_prng (const struct ltc_prng_descriptor ∗prng)

  Register a PRNG with the descriptor table.

- int unregister_prng (const struct ltc_prng_descriptor ∗prng)

  Unregister a PRNG from the descriptor table.

- int prng_is_valid (int idx)
- unsigned long rng_get_bytes (unsigned char ∗out, unsigned long outlen, void(∗callback)(void))

  Read the system RNG.

- int rng_make_prng (int bits, int wprng, prng_state ∗prng, void(∗callback)(void))

  Create a PRNG from a RNG.

Variables

- ltc_prng_descriptor prng_descriptor [ ]

  PRNG descriptor.

5.84.1 Typedef Documentation

5.84.1.1 typedef union Prng_state prng_state

5.84.2 Function Documentation

5.84.2.1 int find_prng (const char ∗ *name*)

Find a registered PRNG by name.

Parameters:
name The name of the PRNG to look for

Returns:
>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_prng.c.

References LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, prng_descriptor, and TAB_SIZE.

```
{
   int x;
   LTC_ARGCHK(name != NULL);
   LTC_MUTEX_LOCK(&ltc_prng_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if ((prng_descriptor[x].name != NULL) && strcmp(prng_descriptor[x].name, name) == 0) {
          LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
          return x;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
   return -1;
}
```

5.84.2.2 int prng_is_valid (int *idx*)

Definition at line 23 of file crypt_prng_is_valid.c.

References CRYPT_INVALID_PRNG, CRYPT_OK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_prng_descriptor::name, prng_descriptor, and TAB_SIZE.

Referenced by dsa_encrypt_key(), dsa_make_key(), dsa_sign_hash_raw(), ecc_encrypt_key(), ecc_make_key(), ecc_sign_hash(), pkcs_1_oaep_encode(), pkcs_1_pss_encode(), pkcs_1_v1_5_encode(), rand_prime(), rng_make_prng(), rsa_encrypt_key_ex(), rsa_make_key(), and rsa_sign_hash_ex().

```
{
   LTC_MUTEX_LOCK(&ltc_prng_mutex);
   if (idx < 0 || idx >= TAB_SIZE || prng_descriptor[idx].name == NULL) {
      LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
      return CRYPT_INVALID_PRNG;
   }
   LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
   return CRYPT_OK;
}
```

5.84.2.3 int register_prng (const struct ltc_prng_descriptor ∗ *prng*)

Register a PRNG with the descriptor table.

Parameters:
prng The PRNG you wish to register

Returns:
value >= 0 if successfully added (or already present), -1 if unsuccessful

Definition at line 23 of file crypt_register_prng.c.

References LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, prng_descriptor, TAB_-SIZE, and XMEMCMP.

Referenced by crypt_fsa().

```
{
   int x;

   LTC_ARGCHK(prng != NULL);

   /* is it already registered? */
   LTC_MUTEX_LOCK(&ltc_prng_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (XMEMCMP(&prng_descriptor[x], prng, sizeof(struct ltc_prng_descriptor)) == 0) {
          LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
          return x;
       }
   }

   /* find a blank spot */
   for (x = 0; x < TAB_SIZE; x++) {
       if (prng_descriptor[x].name == NULL) {
          XMEMCPY(&prng_descriptor[x], prng, sizeof(struct ltc_prng_descriptor));
          LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
          return x;
       }
   }

   /* no spot */
   LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
   return -1;
}
```

5.84.2.4 unsigned long rng_get_bytes (unsigned char ∗ *out*, unsigned long *outlen*, void(∗)(void) *callback*)

Read the system RNG.

Parameters:

out Destination

outlen Length desired (octets)

callback Pointer to void function to act as "callback" when RNG is slow. This can be NULL

Returns:

Number of octets read

Definition at line 123 of file rng_get_bytes.c.

References LTC_ARGCHK, and rng_nix().

Referenced by rng_make_prng(), and sprng_read().

```
{
   unsigned long x;

   LTC_ARGCHK(out != NULL);

#if defined(DEVRANDOM)
```

```
   x = rng_nix(out, outlen, callback);   if (x != 0) { return x; }
#endif
#ifdef WIN32
   x = rng_win32(out, outlen, callback); if (x != 0) { return x; }
#endif
#ifdef ANSI_RNG
   x = rng_ansic(out, outlen, callback); if (x != 0) { return x; }
#endif
   return 0;
}
```

Here is the call graph for this function:

5.84.2.5 int rng_make_prng (int *bits*, int *wprng*, prng_state ∗ *prng*, void(∗)(void) *callback*)

Create a PRNG from a RNG.

Parameters:

bits Number of bits of entropy desired (64 ... 1024)

wprng Index of which PRNG to setup

prng [out] PRNG state to initialize

callback A pointer to a void function for when the RNG is slow, this can be NULL

Returns:

CRYPT_OK if successful

Definition at line 26 of file rng_make_prng.c.

References CRYPT_ERROR_READPRNG, CRYPT_INVALID_PRNGSIZE, CRYPT_OK, LTC_-ARGCHK, prng_descriptor, prng_is_valid(), rng_get_bytes(), and zeromem().

```
{
   unsigned char buf[256];
   int err;

   LTC_ARGCHK(prng != NULL);

   /* check parameter */
   if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
      return err;
   }

   if (bits < 64 || bits > 1024) {
      return CRYPT_INVALID_PRNGSIZE;
   }

   if ((err = prng_descriptor[wprng].start(prng)) != CRYPT_OK) {
      return err;
   }

   bits = ((bits/8)+((bits&7)!=0?1:0)) * 2;
   if (rng_get_bytes(buf, (unsigned long)bits, callback) != (unsigned long)bits) {
      return CRYPT_ERROR_READPRNG;
   }

   if ((err = prng_descriptor[wprng].add_entropy(buf, (unsigned long)bits, prng)) != CRYPT_OK) {
      return err;
   }

   if ((err = prng_descriptor[wprng].ready(prng)) != CRYPT_OK) {
```

```
        return err;
    }

    #ifdef LTC_CLEAN_STACK
        zeromem(buf, sizeof(buf));
    #endif
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.84.2.6 int unregister_prng (const struct ltc_prng_descriptor * *prng*)

Unregister a PRNG from the descriptor table.

Parameters:

prng The PRNG descriptor to remove

Returns:

CRYPT_OK on success

Definition at line 23 of file crypt_unregister_prng.c.

References CRYPT_OK, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_prng_-descriptor::name, prng_descriptor, TAB_SIZE, and XMEMCMP.

```
{
    int x;

    LTC_ARGCHK(prng != NULL);

    /* is it already registered? */
    LTC_MUTEX_LOCK(&ltc_prng_mutex);
    for (x = 0; x < TAB_SIZE; x++) {
        if (XMEMCMP(&prng_descriptor[x], prng, sizeof(struct ltc_prng_descriptor)) != 0) {
            prng_descriptor[x].name = NULL;
            LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
            return CRYPT_OK;
        }
    }
    LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
    return CRYPT_ERROR;
}
```

5.84.3 Variable Documentation

5.84.3.1 struct ltc_prng_descriptor prng_descriptor[ ]

PRNG descriptor.

Referenced by dsa_encrypt_key(), dsa_make_key(), dsa_sign_hash_raw(), find_prng(), pkcs_1_oaep_-encode(), pkcs_1_pss_encode(), pkcs_1_v1_5_encode(), prng_is_valid(), rand_prime(), register_prng(), rng_make_prng(), and unregister_prng().

5.85 mac/f9/f9_done.c File Reference

5.85.1 Detailed Description

f9 Support, terminate the state

Definition in file f9_done.c.

`#include "tomcrypt.h"`

Include dependency graph for f9_done.c:

Functions

- int f9_done (f9_state ∗f9, unsigned char ∗out, unsigned long ∗outlen)

  Terminate the f9-MAC state.

5.85.2 Function Documentation

5.85.2.1 int f9_done (f9_state ∗ *f9*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Terminate the f9-MAC state.

Parameters:

f9 f9 state to terminate

out [out] Destination for the MAC tag

outlen [in/out] Destination size and final tag size Return CRYPT_OK on success

Definition at line 26 of file f9_done.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, and LTC_ARGCHK.

Referenced by f9_file(), and f9_memory().

```
{
   int err, x;
   LTC_ARGCHK(f9 != NULL);
   LTC_ARGCHK(out  != NULL);

   /* check structure */
   if ((err = cipher_is_valid(f9->cipher)) != CRYPT_OK) {
      return err;
   }

   if ((f9->blocksize > cipher_descriptor[f9->cipher].block_length) || (f9->blocksize < 0) ||
       (f9->buflen > f9->blocksize) || (f9->buflen < 0)) {
      return CRYPT_INVALID_ARG;
   }

   if (f9->buflen != 0) {
      /* encrypt */
      cipher_descriptor[f9->cipher].ecb_encrypt(f9->IV, f9->IV, &f9->key);
      f9->buflen = 0;
      for (x = 0; x < f9->blocksize; x++) {
         f9->ACC[x] ^= f9->IV[x];
```

```
      }
   }

   /* schedule modified key */
   if ((err = cipher_descriptor[f9->cipher].setup(f9->akey, f9->keylen, 0, &f9->key)) != CRYPT_OK) {
      return err;
   }

   /* encrypt the ACC */
   cipher_descriptor[f9->cipher].ecb_encrypt(f9->ACC, f9->ACC, &f9->key);
   cipher_descriptor[f9->cipher].done(&f9->key);

   /* extract tag */
   for (x = 0; x < f9->blocksize && (unsigned long)x < *outlen; x++) {
      out[x] = f9->ACC[x];
   }
   *outlen = x;

#ifdef LTC_CLEAN_STACK
   zeromem(f9, sizeof(*f9));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.86 mac/f9/f9_file.c File Reference

5.86.1 Detailed Description

f9 support, process a file, Tom St Denis

Definition in file f9_file.c.

`#include "tomcrypt.h"`

Include dependency graph for f9_file.c:

Functions

- int f9_file (int cipher, const unsigned char *key, unsigned long keylen, const char *filename, unsigned char *out, unsigned long *outlen)

  f9 a file

5.86.2 Function Documentation

5.86.2.1 int f9_file (int *cipher*, const unsigned char * *key*, unsigned long *keylen*, const char * *filename*, unsigned char * *out*, unsigned long * *outlen*)

f9 a file

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the secret key (octets)

filename The name of the file you wish to f9

out [out] Where the authentication tag is to be stored

outlen [in/out] The max size and resulting size of the authentication tag

Returns:

CRYPT_OK if successful, CRYPT_NOP if file support has been disabled

Definition at line 30 of file f9_file.c.

References CRYPT_FILE_NOTFOUND, CRYPT_NOP, CRYPT_OK, f9_done(), f9_init(), f9_process(), in, LTC_ARGCHK, and zeromem().

```
{
#ifdef LTC_NO_FILE
   return CRYPT_NOP;
#else
   int err, x;
   f9_state f9;
   FILE *in;
   unsigned char buf[512];

   LTC_ARGCHK(key      != NULL);
   LTC_ARGCHK(filename != NULL);
   LTC_ARGCHK(out      != NULL);
```

```
   LTC_ARGCHK(outlen   != NULL);

   in = fopen(filename, "rb");
   if (in == NULL) {
      return CRYPT_FILE_NOTFOUND;
   }

   if ((err = f9_init(&f9, cipher, key, keylen)) != CRYPT_OK) {
      fclose(in);
      return err;
   }

   do {
      x = fread(buf, 1, sizeof(buf), in);
      if ((err = f9_process(&f9, buf, x)) != CRYPT_OK) {
         fclose(in);
         return err;
      }
   } while (x == sizeof(buf));
   fclose(in);

   if ((err = f9_done(&f9,  out, outlen)) != CRYPT_OK) {
      return err;
   }

#ifdef LTC_CLEAN_STACK
   zeromem(buf, sizeof(buf));
#endif

   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.87 mac/f9/f9_init.c File Reference

5.87.1 Detailed Description

F9 Support, start an F9 state.

Definition in file f9_init.c.

`#include "tomcrypt.h"`

Include dependency graph for f9_init.c:

Functions

- int f9_init (f9_state ∗f9, int cipher, const unsigned char ∗key, unsigned long keylen)
  Initialize F9-MAC state.

5.87.2 Function Documentation

5.87.2.1 int f9_init (f9_state ∗ *f9*, int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*)

Initialize F9-MAC state.

Parameters:

f9 [out] f9 state to initialize

cipher Index of cipher to use

key [in] Secret key

keylen Length of secret key in octets Return CRYPT_OK on success

Definition at line 27 of file f9_init.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_prng_-descriptor::done, and LTC_ARGCHK.

Referenced by f9_file(), f9_memory(), and f9_memory_multi().

```
{
   int              x, err;

   LTC_ARGCHK(f9   != NULL);
   LTC_ARGCHK(key  != NULL);

   /* schedule the key */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

#ifdef LTC_FAST
   if (cipher_descriptor[cipher].block_length % sizeof(LTC_FAST_TYPE)) {
       return CRYPT_INVALID_ARG;
   }
#endif

   if ((err = cipher_descriptor[cipher].setup(key, keylen, 0, &f9->key)) != CRYPT_OK) {
      goto done;
   }
```

```

   /* make the second key */
   for (x = 0; (unsigned)x < keylen; x++) {
      f9->akey[x] = key[x] ^ 0xAA;
   }

   /* setup struct */
   zeromem(f9->IV,  cipher_descriptor[cipher].block_length);
   zeromem(f9->ACC, cipher_descriptor[cipher].block_length);
   f9->blocksize = cipher_descriptor[cipher].block_length;
   f9->cipher    = cipher;
   f9->buflen    = 0;
   f9->keylen    = keylen;
done:
   return err;
}
```

Here is the call graph for this function:

5.88 mac/f9/f9_memory.c File Reference

`#include "tomcrypt.h"`

Include dependency graph for f9_memory.c:

Functions

- int f9_memory (int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  f9-MAC a block of memory

5.88.1 Function Documentation

5.88.1.1 int f9_memory (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

f9-MAC a block of memory

Parameters:

cipher Index of cipher to use

key [in] Secret key

keylen Length of key in octets

in [in] Message to MAC

inlen Length of input in octets

out [out] Destination for the MAC tag

outlen [in/out] Output size and final tag size Return CRYPT_OK on success.

Definition at line 30 of file f9_memory.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_MEM, CRYPT_OK, ltc_prng_descriptor::done, f9_done(), f9_init(), ltc_cipher_descriptor::f9_memory, f9_process(), XCALLOC, and XFREE.

Referenced by f9_test().

```
34 {
35    f9_state *f9;
36    int         err;
37
38    /* is the cipher valid? */
39    if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
40       return err;
41    }
42
43    /* Use accelerator if found */
44    if (cipher_descriptor[cipher].f9_memory != NULL) {
45       return cipher_descriptor[cipher].f9_memory(key, keylen, in, inlen, out, outlen);
46    }
47
48    f9 = XCALLOC(1, sizeof(*f9));
49    if (f9 == NULL) {
50       return CRYPT_MEM;
51    }
52
```

```
   if ((err = f9_init(f9, cipher, key, keylen)) != CRYPT_OK) {
     goto done;
   }

   if ((err = f9_process(f9, in, inlen)) != CRYPT_OK) {
     goto done;
   }

   err = f9_done(f9, out, outlen);
done:
   XFREE(f9);
   return err;
}
```

Here is the call graph for this function:

5.89 mac/f9/f9_memory_multi.c File Reference

5.89.1 Detailed Description

f9 support, process multiple blocks of memory, Tom St Denis

Definition in file f9_memory_multi.c.

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for f9_memory_multi.c:

Functions

- int f9_memory_multi (int cipher, const unsigned char ∗key, unsigned long keylen, unsigned char ∗out, unsigned long ∗outlen, const unsigned char ∗in, unsigned long inlen,...)

    f9 multiple blocks of memory

5.89.2 Function Documentation

5.89.2.1 int f9_memory_multi (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, const unsigned char ∗ *in*, unsigned long *inlen*, ...)

f9 multiple blocks of memory

Parameters:

cipher The index of the desired cipher

key The secret key

keylen The length of the secret key (octets)

out [out] The destination of the authentication tag

outlen [in/out] The max size and resulting size of the authentication tag (octets)

in The data to send through f9

inlen The length of the data to send through f9 (octets)

... tuples of (data,len) pairs to f9, terminated with a (NULL,x) (x=don't care)

Returns:

CRYPT_OK if successful

Definition at line 33 of file f9_memory_multi.c.

References CRYPT_MEM, CRYPT_OK, f9_init(), f9_process(), LTC_ARGCHK, and XMALLOC.

```
{
   int                    err;
   f9_state           *f9;
   va_list                args;
   const unsigned char *curptr;
   unsigned long          curlen;
```

```

   LTC_ARGCHK(key    != NULL);
   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* allocate ram for f9 state */
   f9 = XMALLOC(sizeof(f9_state));
   if (f9 == NULL) {
      return CRYPT_MEM;
   }

   /* f9 process the message */
   if ((err = f9_init(f9, cipher, key, keylen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   va_start(args, inlen);
   curptr = in;
   curlen = inlen;
   for (;;) {
      /* process buf */
      if ((err = f9_process(f9, curptr, curlen)) != CRYPT_OK) {
         goto LBL_ERR;
      }
      /* step to next */
      curptr = va_arg(args, const unsigned char*);
      if (curptr == NULL) {
         break;
      }
      curlen = va_arg(args, unsigned long);
   }
   if ((err = f9_done(f9, out, outlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(f9, sizeof(f9_state));
#endif
   XFREE(f9);
   va_end(args);
   return err;
}
```

Here is the call graph for this function:

5.90 mac/f9/f9_process.c File Reference

5.90.1 Detailed Description

f9 Support, terminate the state

Definition in file f9_process.c.

`#include "tomcrypt.h"`

Include dependency graph for f9_process.c:

Functions

- int f9_process (f9_state ∗f9, const unsigned char ∗in, unsigned long inlen)

  Process data through f9-MAC.

5.90.2 Function Documentation

5.90.2.1 int f9_process (f9_state ∗ *f9*, const unsigned char ∗ *in*, unsigned long *inlen*)

Process data through f9-MAC.

Parameters:

f9 The f9-MAC state

in Input data to process

inlen Length of input in octets Return CRYPT_OK on success

Definition at line 26 of file f9_process.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, and LTC_ARGCHK.

Referenced by f9_file(), f9_memory(), and f9_memory_multi().

```
{
   int err, x;

   LTC_ARGCHK(f9 != NULL);
   LTC_ARGCHK(in   != NULL);

   /* check structure */
   if ((err = cipher_is_valid(f9->cipher)) != CRYPT_OK) {
      return err;
   }

   if ((f9->blocksize > cipher_descriptor[f9->cipher].block_length) || (f9->blocksize < 0) ||
       (f9->buflen > f9->blocksize) || (f9->buflen < 0)) {
      return CRYPT_INVALID_ARG;
   }

#ifdef LTC_FAST
   if (f9->buflen == 0) {
       while (inlen >= (unsigned long)f9->blocksize) {
           for (x = 0; x < f9->blocksize; x += sizeof(LTC_FAST_TYPE)) {
              *((LTC_FAST_TYPE*)&(f9->IV[x])) ^= *((LTC_FAST_TYPE*)&(in[x]));
```

```
           }
           cipher_descriptor[f9->cipher].ecb_encrypt(f9->IV, f9->IV, &f9->key);
           for (x = 0; x < f9->blocksize; x += sizeof(LTC_FAST_TYPE)) {
              *((LTC_FAST_TYPE*)&(f9->ACC[x])) ^= *((LTC_FAST_TYPE*)&(f9->IV[x]));
           }
           in    += f9->blocksize;
           inlen -= f9->blocksize;
       }
  }
#endif

   while (inlen) {
     if (f9->buflen == f9->blocksize) {
         cipher_descriptor[f9->cipher].ecb_encrypt(f9->IV, f9->IV, &f9->key);
         for (x = 0; x < f9->blocksize; x++) {
            f9->ACC[x] ^= f9->IV[x];
         }
         f9->buflen = 0;
     }
     f9->IV[f9->buflen++] ^= *in++;
     --inlen;
  }
  return CRYPT_OK;
}
```

Here is the call graph for this function:

5.91 mac/f9/f9_test.c File Reference

5.91.1 Detailed Description

f9 Support, terminate the state

Definition in file f9_test.c.

`#include "tomcrypt.h"`

Include dependency graph for f9_test.c:

Functions

- int f9_test (void)

  Test f9-MAC mode Return CRYPT_OK on succes.

5.91.2 Function Documentation

5.91.2.1 int f9_test (void)

Test f9-MAC mode Return CRYPT_OK on succes.

Definition at line 23 of file f9_test.c.

References CRYPT_NOP, CRYPT_OK, f9_memory(), find_cipher(), and K.

```
24 {
25 #ifdef LTC_NO_TEST
26    return CRYPT_NOP;
27 #else
28    static const struct {
29        int msglen;
30        unsigned char K[16], M[128], T[4];
31    } tests[] = {
32 {
33    20,
34    { 0x2B, 0xD6, 0x45, 0x9F, 0x82, 0xC5, 0xB3, 0x00, 0x95, 0x2C, 0x49, 0x10, 0x48, 0x81, 0xFF, 0x48 },
35    { 0x38, 0xA6, 0xF0, 0x56, 0xB8, 0xAE, 0xFD, 0xA9, 0x33, 0x32, 0x34, 0x62, 0x63, 0x39, 0x38, 0x61, 0x
36    { 0x46, 0xE0, 0x0D, 0x4B }
37 },
38
39 {
40    105,
41    { 0x83, 0xFD, 0x23, 0xA2, 0x44, 0xA7, 0x4C, 0xF3, 0x58, 0xDA, 0x30, 0x19, 0xF1, 0x72, 0x26, 0x35 },
42    { 0x36, 0xAF, 0x61, 0x44, 0x4F, 0x30, 0x2A, 0xD2,
43      0x35, 0xC6, 0x87, 0x16, 0x63, 0x3C, 0x66, 0xFB, 0x75, 0x0C, 0x26, 0x68, 0x65, 0xD5, 0x3C, 0x11, 0x
44      0x47, 0x90, 0x28, 0x37, 0xF5, 0xAE, 0x96, 0xD5, 0xA0, 0x5B, 0xC8, 0xD6, 0x1C, 0xA8, 0xDB, 0xEF, 0x
45      0x93, 0x04, 0xC3, 0x82, 0xBE, 0x53, 0xA5, 0xAF, 0x05, 0x55, 0x61, 0x76, 0xF6, 0xEA, 0xA2, 0xEF, 0x
46      0x40|0x80 },
47    { 0x95, 0xAE, 0x41, 0xBA },
48 }
49 };
50   unsigned char T[16];
51   unsigned long taglen;
52   int err, x, idx;
53
54   /* find kasumi */
55   if ((idx = find_cipher("kasumi")) == -1) {
56      return CRYPT_NOP;
```

```
   }

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
      taglen = 4;
      if ((err = f9_memory(idx, tests[x].K, 16, tests[x].M, tests[x].msglen, T, &taglen)) != CRYPT_OK) {
         return err;
      }
      if (taglen != 4 || XMEMCMP(T, tests[x].T, 4)) {
         return CRYPT_FAIL_TESTVECTOR;
      }
   }

   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.92 mac/hmac/hmac_done.c File Reference

5.92.1 Detailed Description

HMAC support, terminate stream, Tom St Denis/Dobes Vandermeer.

Definition in file hmac_done.c.

`#include "tomcrypt.h"`

Include dependency graph for hmac_done.c:

Defines

- #define HMAC_BLOCKSIZE hash_descriptor[hash].blocksize

Functions

- int hmac_done (hmac_state ∗hmac, unsigned char ∗out, unsigned long ∗outlen)

  Terminate an HMAC session.

5.92.2 Define Documentation

5.92.2.1 #define HMAC_BLOCKSIZE hash_descriptor[hash].blocksize

Definition at line 20 of file hmac_done.c.

Referenced by hmac_done(), and hmac_init().

5.92.3 Function Documentation

5.92.3.1 int hmac_done (hmac_state ∗ *hmac*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Terminate an HMAC session.

Parameters:

hmac The HMAC state

out [out] The destination of the HMAC authentication tag

outlen [in/out] The max size and resulting size of the HMAC authentication tag

Returns:

CRYPT_OK if successful

Definition at line 29 of file hmac_done.c.

References CRYPT_MEM, CRYPT_OK, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, HMAC_BLOCKSIZE, LTC_ARGCHK, XFREE, and XMALLOC.

Referenced by hmac_file(), hmac_memory(), and pkcs_5_alg2().

```
{
    unsigned char *buf, *isha;
    unsigned long hashsize, i;
    int hash, err;

    LTC_ARGCHK(hmac  != NULL);
    LTC_ARGCHK(out   != NULL);

    /* test hash */
    hash = hmac->hash;
    if((err = hash_is_valid(hash)) != CRYPT_OK) {
        return err;
    }

    /* get the hash message digest size */
    hashsize = hash_descriptor[hash].hashsize;

    /* allocate buffers */
    buf  = XMALLOC(HMAC_BLOCKSIZE);
    isha = XMALLOC(hashsize);
    if (buf == NULL || isha == NULL) {
       if (buf != NULL) {
          XFREE(buf);
       }
       if (isha != NULL) {
          XFREE(isha);
       }
       return CRYPT_MEM;
    }

    /* Get the hash of the first HMAC vector plus the data */
    if ((err = hash_descriptor[hash].done(&hmac->md, isha)) != CRYPT_OK) {
       goto LBL_ERR;
    }

    /* Create the second HMAC vector vector for step (3) */
    for(i=0; i < HMAC_BLOCKSIZE; i++) {
        buf[i] = hmac->key[i] ^ 0x5C;
    }

    /* Now calculate the "outer" hash for step (5), (6), and (7) */
    if ((err = hash_descriptor[hash].init(&hmac->md)) != CRYPT_OK) {
       goto LBL_ERR;
    }
    if ((err = hash_descriptor[hash].process(&hmac->md, buf, HMAC_BLOCKSIZE)) != CRYPT_OK) {
       goto LBL_ERR;
    }
    if ((err = hash_descriptor[hash].process(&hmac->md, isha, hashsize)) != CRYPT_OK) {
       goto LBL_ERR;
    }
    if ((err = hash_descriptor[hash].done(&hmac->md, buf)) != CRYPT_OK) {
       goto LBL_ERR;
    }

    /* copy to output  */
    for (i = 0; i < hashsize && i < *outlen; i++) {
        out[i] = buf[i];
    }
    *outlen = i;

    err = CRYPT_OK;
LBL_ERR:
    XFREE(hmac->key);
#ifdef LTC_CLEAN_STACK
    zeromem(isha, hashsize);
    zeromem(buf,  hashsize);
    zeromem(hmac, sizeof(*hmac));
```

```
#endif

   XFREE(isha);
   XFREE(buf);

   return err;
}
```

Here is the call graph for this function:

5.93 mac/hmac/hmac_file.c File Reference

5.93.1 Detailed Description

HMAC support, process a file, Tom St Denis/Dobes Vandermeer.

Definition in file hmac_file.c.

`#include "tomcrypt.h"`

Include dependency graph for hmac_file.c:

Functions

- int hmac_file (int hash, const char ∗fname, const unsigned char ∗key, unsigned long keylen, unsigned char ∗out, unsigned long ∗outlen)

  HMAC a file.

5.93.2 Function Documentation

5.93.2.1 int hmac_file (int *hash*, const char ∗ *fname*, const unsigned char ∗ *key*, unsigned long *keylen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

HMAC a file.

Parameters:

hash The index of the hash you wish to use

fname The name of the file you wish to HMAC

key The secret key

keylen The length of the secret key

out [out] The HMAC authentication tag

outlen [in/out] The max size and resulting size of the authentication tag

Returns:

CRYPT_OK if successful, CRYPT_NOP if file support has been disabled

Definition at line 30 of file hmac_file.c.

References CRYPT_ERROR, CRYPT_FILE_NOTFOUND, CRYPT_NOP, CRYPT_OK, hash_is_valid(), hmac_done(), hmac_init(), hmac_process(), in, LTC_ARGCHK, and zeromem().

```
{
#ifdef LTC_NO_FILE
    return CRYPT_NOP;
#else
   hmac_state hmac;
   FILE *in;
   unsigned char buf[512];
   size_t x;
   int err;

   LTC_ARGCHK(fname  != NULL);
   LTC_ARGCHK(key    != NULL);
```

```
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   if((err = hash_is_valid(hash)) != CRYPT_OK) {
       return err;
   }

   if ((err = hmac_init(&hmac, hash, key, keylen)) != CRYPT_OK) {
       return err;
   }

   in = fopen(fname, "rb");
   if (in == NULL) {
      return CRYPT_FILE_NOTFOUND;
   }

   /* process the file contents */
   do {
      x = fread(buf, 1, sizeof(buf), in);
      if ((err = hmac_process(&hmac, buf, (unsigned long)x)) != CRYPT_OK) {
         /* we don't trap this error since we're already returning an error! */
         fclose(in);
         return err;
      }
   } while (x == sizeof(buf));

   if (fclose(in) != 0) {
      return CRYPT_ERROR;
   }

   /* get final hmac */
   if ((err = hmac_done(&hmac, out, outlen)) != CRYPT_OK) {
      return err;
   }

#ifdef LTC_CLEAN_STACK
   /* clear memory */
   zeromem(buf, sizeof(buf));
#endif
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.94 mac/hmac/hmac_init.c File Reference

5.94.1 Detailed Description

HMAC support, initialize state, Tom St Denis/Dobes Vandermeer.

Definition in file hmac_init.c.

`#include "tomcrypt.h"`

Include dependency graph for hmac_init.c:

Defines

- #define HMAC_BLOCKSIZE hash_descriptor[hash].blocksize

Functions

- int hmac_init (hmac_state *hmac, int hash, const unsigned char *key, unsigned long keylen)
  Initialize an HMAC context.

5.94.2 Define Documentation

5.94.2.1 #define HMAC_BLOCKSIZE hash_descriptor[hash].blocksize

Definition at line 20 of file hmac_init.c.

5.94.3 Function Documentation

5.94.3.1 int hmac_init (hmac_state * *hmac*, int *hash*, const unsigned char * *key*, unsigned long *keylen*)

Initialize an HMAC context.

Parameters:

hmac The HMAC state

hash The index of the hash you want to use

key The secret key

keylen The length of the secret key (octets)

Returns:

CRYPT_OK if successful

Definition at line 30 of file hmac_init.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_MEM, CRYPT_OK, hash_descriptor, hash_is_valid(), hash_memory(), ltc_hash_descriptor::hashsize, HMAC_BLOCKSIZE, LTC_ARGCHK, XFREE, XMALLOC, XMEMCPY, and zeromem().

Referenced by hmac_file(), hmac_memory(), hmac_memory_multi(), and pkcs_5_alg2().

```
{
    unsigned char *buf;
    unsigned long hashsize;
    unsigned long i, z;
    int err;

    LTC_ARGCHK(hmac != NULL);
    LTC_ARGCHK(key  != NULL);

    /* valid hash? */
    if ((err = hash_is_valid(hash)) != CRYPT_OK) {
        return err;
    }
    hmac->hash = hash;
    hashsize   = hash_descriptor[hash].hashsize;

    /* valid key length? */
    if (keylen == 0) {
        return CRYPT_INVALID_KEYSIZE;
    }

    /* allocate ram for buf */
    buf = XMALLOC(HMAC_BLOCKSIZE);
    if (buf == NULL) {
       return CRYPT_MEM;
    }

    /* allocate memory for key */
    hmac->key = XMALLOC(HMAC_BLOCKSIZE);
    if (hmac->key == NULL) {
       XFREE(buf);
       return CRYPT_MEM;
    }

    /* (1) make sure we have a large enough key */
    if(keylen > HMAC_BLOCKSIZE) {
        z = HMAC_BLOCKSIZE;
        if ((err = hash_memory(hash, key, keylen, hmac->key, &z)) != CRYPT_OK) {
           goto LBL_ERR;
        }
        if(hashsize < HMAC_BLOCKSIZE) {
            zeromem((hmac->key) + hashsize, (size_t)(HMAC_BLOCKSIZE - hashsize));
        }
        keylen = hashsize;
    } else {
        XMEMCPY(hmac->key, key, (size_t)keylen);
        if(keylen < HMAC_BLOCKSIZE) {
            zeromem((hmac->key) + keylen, (size_t)(HMAC_BLOCKSIZE - keylen));
        }
    }

    /* Create the initial vector for step (3) */
    for(i=0; i < HMAC_BLOCKSIZE;   i++) {
       buf[i] = hmac->key[i] ^ 0x36;
    }

    /* Pre-pend that to the hash data */
    if ((err = hash_descriptor[hash].init(&hmac->md)) != CRYPT_OK) {
       goto LBL_ERR;
    }

    if ((err = hash_descriptor[hash].process(&hmac->md, buf, HMAC_BLOCKSIZE)) != CRYPT_OK) {
       goto LBL_ERR;
    }
    goto done;
LBL_ERR:
    /* free the key since we failed */
```

```
    XFREE(hmac->key);
done:
#ifdef LTC_CLEAN_STACK
    zeromem(buf, HMAC_BLOCKSIZE);
#endif

    XFREE(buf);
    return err;
}
```

Here is the call graph for this function:

5.95 mac/hmac/hmac_memory.c File Reference

5.95.1 Detailed Description

HMAC support, process a block of memory, Tom St Denis/Dobes Vandermeer.

Definition in file hmac_memory.c.

`#include "tomcrypt.h"`

Include dependency graph for hmac_memory.c:

Functions

- int hmac_memory (int hash, const unsigned char *key, unsigned long keylen, const unsigned char *in, unsigned long inlen, unsigned char *out, unsigned long *outlen)

  HMAC a block of memory to produce the authentication tag.

5.95.2 Function Documentation

5.95.2.1 int hmac_memory (int *hash*, const unsigned char * *key*, unsigned long *keylen*, const unsigned char * *in*, unsigned long *inlen*, unsigned char * *out*, unsigned long * *outlen*)

HMAC a block of memory to produce the authentication tag.

Parameters:

hash The index of the hash to use

key The secret key

keylen The length of the secret key (octets)

in The data to HMAC

inlen The length of the data to HMAC (octets)

out [out] Destination of the authentication tag

outlen [in/out] Max size and resulting size of authentication tag

Returns:

CRYPT_OK if successful

Definition at line 31 of file hmac_memory.c.

References CRYPT_MEM, CRYPT_OK, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hmac_-block, hmac_done(), hmac_init(), hmac_process(), LTC_ARGCHK, XFREE, XMALLOC, and zeromem().

Referenced by hmac_test(), and pkcs_5_alg2().

```
{
    hmac_state *hmac;
    int         err;

    LTC_ARGCHK(key    != NULL);
    LTC_ARGCHK(in     != NULL);
    LTC_ARGCHK(out    != NULL);
```

```
    LTC_ARGCHK(outlen != NULL);

    /* make sure hash descriptor is valid */
    if ((err = hash_is_valid(hash)) != CRYPT_OK) {
       return err;
    }

    /* is there a descriptor? */
    if (hash_descriptor[hash].hmac_block != NULL) {
        return hash_descriptor[hash].hmac_block(key, keylen, in, inlen, out, outlen);
    }

    /* nope, so call the hmac functions */
    /* allocate ram for hmac state */
    hmac = XMALLOC(sizeof(hmac_state));
    if (hmac == NULL) {
       return CRYPT_MEM;
    }

    if ((err = hmac_init(hmac, hash, key, keylen)) != CRYPT_OK) {
       goto LBL_ERR;
    }

    if ((err = hmac_process(hmac, in, inlen)) != CRYPT_OK) {
       goto LBL_ERR;
    }

    if ((err = hmac_done(hmac, out, outlen)) != CRYPT_OK) {
       goto LBL_ERR;
    }

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(hmac, sizeof(hmac_state));
#endif

   XFREE(hmac);
   return err;
}
```

Here is the call graph for this function:

5.96 mac/hmac/hmac_memory_multi.c File Reference

5.96.1 Detailed Description

HMAC support, process multiple blocks of memory, Tom St Denis/Dobes Vandermeer.

Definition in file hmac_memory_multi.c.

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for hmac_memory_multi.c:

Functions

- int hmac_memory_multi (int hash, const unsigned char ∗key, unsigned long keylen, unsigned char ∗out, unsigned long ∗outlen, const unsigned char ∗in, unsigned long inlen,...)

  HMAC multiple blocks of memory to produce the authentication tag.

5.96.2 Function Documentation

5.96.2.1 int hmac_memory_multi (int *hash*, const unsigned char ∗ *key*, unsigned long *keylen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, const unsigned char ∗ *in*, unsigned long *inlen*, ...)

HMAC multiple blocks of memory to produce the authentication tag.

Parameters:

hash The index of the hash to use

key The secret key

keylen The length of the secret key (octets)

out [out] Destination of the authentication tag

outlen [in/out] Max size and resulting size of authentication tag

in The data to HMAC

inlen The length of the data to HMAC (octets)

... tuples of (data,len) pairs to HMAC, terminated with a (NULL,x) (x=don't care)

Returns:

CRYPT_OK if successful

Definition at line 33 of file hmac_memory_multi.c.

References CRYPT_MEM, CRYPT_OK, hmac_init(), hmac_process(), LTC_ARGCHK, and XMALLOC.

```
{
    hmac_state          *hmac;
    int                  err;
    va_list              args;
    const unsigned char *curptr;
    unsigned long        curlen;
```

```

    LTC_ARGCHK(key    != NULL);
    LTC_ARGCHK(in     != NULL);
    LTC_ARGCHK(out    != NULL);
    LTC_ARGCHK(outlen != NULL);

    /* allocate ram for hmac state */
    hmac = XMALLOC(sizeof(hmac_state));
    if (hmac == NULL) {
       return CRYPT_MEM;
    }

    if ((err = hmac_init(hmac, hash, key, keylen)) != CRYPT_OK) {
       goto LBL_ERR;
    }

    va_start(args, inlen);
    curptr = in;
    curlen = inlen;
    for (;;) {
       /* process buf */
       if ((err = hmac_process(hmac, curptr, curlen)) != CRYPT_OK) {
          goto LBL_ERR;
       }
       /* step to next */
       curptr = va_arg(args, const unsigned char*);
       if (curptr == NULL) {
          break;
       }
       curlen = va_arg(args, unsigned long);
    }
    if ((err = hmac_done(hmac, out, outlen)) != CRYPT_OK) {
       goto LBL_ERR;
    }
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(hmac, sizeof(hmac_state));
#endif
   XFREE(hmac);
   va_end(args);
   return err;
}
```

Here is the call graph for this function:

5.97 mac/hmac/hmac_process.c File Reference

5.97.1 Detailed Description

HMAC support, process data, Tom St Denis/Dobes Vandermeer.

Definition in file hmac_process.c.

`#include "tomcrypt.h"`

Include dependency graph for hmac_process.c:

Functions

- int hmac_process (hmac_state ∗hmac, const unsigned char ∗in, unsigned long inlen)

  Process data through HMAC.

5.97.2 Function Documentation

5.97.2.1 int hmac_process (hmac_state ∗ *hmac*, const unsigned char ∗ *in*, unsigned long *inlen*)

Process data through HMAC.

Parameters:

hmac The hmac state

in The data to send through HMAC

inlen The length of the data to HMAC (octets)

Returns:

CRYPT_OK if successful

Definition at line 27 of file hmac_process.c.

References CRYPT_OK, hash_descriptor, hash_is_valid(), LTC_ARGCHK, and ltc_hash_descriptor::process.

Referenced by hmac_file(), hmac_memory(), hmac_memory_multi(), and pkcs_5_alg2().

```
28 {
29     int err;
30     LTC_ARGCHK(hmac != NULL);
31     LTC_ARGCHK(in != NULL);
32     if ((err = hash_is_valid(hmac->hash)) != CRYPT_OK) {
33         return err;
34     }
35     return hash_descriptor[hmac->hash].process(&hmac->md, in, inlen);
36 }
```

Here is the call graph for this function:

5.98 mac/hmac/hmac_test.c File Reference

5.98.1 Detailed Description

HMAC support, self-test, Tom St Denis/Dobes Vandermeer.

Definition in file hmac_test.c.

`#include "tomcrypt.h"`

Include dependency graph for hmac_test.c:

Defines

- #define HMAC_BLOCKSIZE hash_descriptor[hash].blocksize

Functions

- int hmac_test (void)

  HMAC self-test.

5.98.2 Define Documentation

5.98.2.1 #define HMAC_BLOCKSIZE hash_descriptor[hash].blocksize

Definition at line 20 of file hmac_test.c.

5.98.3 Function Documentation

5.98.3.1 int hmac_test (void)

HMAC self-test.

Returns:

CRYPT_OK if successful, CRYPT_NOP if tests have been disabled.

Definition at line 37 of file hmac_test.c.

References CRYPT_NOP, CRYPT_OK, error_to_string(), find_hash(), hmac_memory(), and MAXBLOCKSIZE.

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   unsigned char digest[MAXBLOCKSIZE];
   int i;

   static const struct hmac_test_case {
       int num;
       char *algo;
       unsigned char key[128];
       unsigned long keylen;
```

```
        unsigned char data[128];
        unsigned long datalen;
        unsigned char digest[MAXBLOCKSIZE];
    } cases[] = {
        /*
        3. Test Cases for HMAC-SHA-1

        test_case =     1
        key =           0x0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c
        key_len =       20
        data =          "Hi Ther     20
        digest =        0x4c1a03424b55e07fe7f27be1d58bb9324a9a5a04
        digest-96 =     0x4c1a03424b55e07fe7f27be1
        */
        { 5, "sha1",
            {0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c,
             0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c,
             0x0c, 0x0c, 0x0c, 0x0c}, 20,
            "Test With Truncation", 20,
            {0x4c, 0x1a, 0x03, 0x42, 0x4b, 0x55, 0xe0, 0x7f, 0xe7, 0xf2,
             0x7b, 0xe1, 0xd5, 0x8b, 0xb9, 0x32, 0x4a, 0x9a, 0x5a, 0x04} },

        /*
        test_case =     6
        key =           0xaa repeated 80 times
        key_len =       80
        data =          "Test Using Larger Than Block-Size Key - Hash Key First"
        data_len =      54
        digest =        0xaa4ae5e15272d00e95705637ce8a3b55ed402112
        */
        { 6, "sha1",
            {0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa}, 80,
            "Test Using Larger Than Block-Size Key - Hash Key First", 54,
            {0xaa, 0x4a, 0xe5, 0xe1, 0x52, 0x72, 0xd0, 0x0e,
             0x95, 0x70, 0x56, 0x37, 0xce, 0x8a, 0x3b, 0x55,
             0xed, 0x40, 0x21, 0x12} },

        /*
        test_case =     7
        key =           0xaa repeated 80 times
        key_len =       80
        data =          "Test Using Larger Than Block-Size Key and Larger
                        Than One Block-Size Data"
        data_len =      73
        digest =        0xe8e99d0f45237d786d6bbaa7965c7808bbff1a91
        */
        { 7, "sha1",
            {0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa}, 80,
            "Test Using Larger Than Block-Size Key and Larger Than One Block-Size Data", 73,
```

```
        {0xe8, 0xe9, 0x9d, 0x0f, 0x45, 0x23, 0x7d, 0x78, 0x6d,
         0x6b, 0xba, 0xa7, 0x96, 0x5c, 0x78, 0x08, 0xbb, 0xff, 0x1a, 0x91} },

    /*
    2. Test Cases for HMAC-MD5

    test_case =     1
    key =           0x0b 0b 0b 0b
                      0b 0b 0b 0b
                      0b 0b 0b 0b
                      0b 0b 0b 0b
    key_len =       16
    data =          "Hi There"
    data_len =      8
    digest =        0x92 94 72 7a
                      36 38 bb 1c
                      13 f4 8e f8
                      15 8b fc 9d
    */
    { 1, "md5",
        {0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
         0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b}, 16,
        "Hi There", 8,
        {0x92, 0x94, 0x72, 0x7a, 0x36, 0x38, 0xbb, 0x1c,
         0x13, 0xf4, 0x8e, 0xf8, 0x15, 0x8b, 0xfc, 0x9d}  },
    /*
    test_case =     2
    key =           "Jefe"
    key_len =       4
    data =          "what do ya want for nothing?"
    data_len =      28
    digest =        0x750c783e6ab0b503eaa86e310a5db738
    */
    { 2, "md5",
        "Jefe", 4,
        "what do ya want for nothing?", 28,
        {0x75, 0x0c, 0x78, 0x3e, 0x6a, 0xb0, 0xb5, 0x03,
         0xea, 0xa8, 0x6e, 0x31, 0x0a, 0x5d, 0xb7, 0x38} },

    /*
    test_case =     3
    key =           0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
    key_len         16
    data =          0xdd repeated 50 times
    data_len =      50
    digest =        0x56be34521d144c88dbb8c733f0e8b3f6
    */
    { 3, "md5",
        {0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
         0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa}, 16,
        {0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd,
         0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd,
         0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd,
         0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd,
         0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd, 0xdd}, 50,
        {0x56, 0xbe, 0x34, 0x52, 0x1d, 0x14, 0x4c, 0x88,
         0xdb, 0xb8, 0xc7, 0x33, 0xf0, 0xe8, 0xb3, 0xf6} },
    /*

    test_case =     4
    key = 0x0102030405060708090a0b0c0d0e0f10111213141516171819
    key_len         25
    data =          0xcd repeated 50 times
    data_len =      50
    digest =        0x697eaf0aca3a3aea3a75164746ffaa79
    */
    { 4, "md5",
```

```
            {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a,
             0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10, 0x11, 0x12, 0x13, 0x14,
             0x15, 0x16, 0x17, 0x18, 0x19}, 25,
            {0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd,
             0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd,
             0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd,
             0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd,
             0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd, 0xcd}, 50,
            {0x69, 0x7e, 0xaf, 0x0a, 0xca, 0x3a, 0x3a, 0xea,
             0x3a, 0x75, 0x16, 0x47, 0x46, 0xff, 0xaa, 0x79} },


        /*

        test_case =     5
        key =           0x0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c0c
        key_len =       16
        data =          "Test With Truncation"
        data_len =      20
        digest =        0x56461ef2342edc00f9bab995690efd4c
        digest-96       0x56461ef2342edc00f9bab995
        */
        { 5, "md5",
            {0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c,
             0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c}, 16,
            "Test With Truncation", 20,
            {0x56, 0x46, 0x1e, 0xf2, 0x34, 0x2e, 0xdc, 0x00,
             0xf9, 0xba, 0xb9, 0x95, 0x69, 0x0e, 0xfd, 0x4c} },

        /*

        test_case =     6
        key =           0xaa repeated 80 times
        key_len =       80
        data =          "Test Using Larger Than Block-Size Key - Hash
Key First"
        data_len =      54
        digest =        0x6b1ab7fe4bd7bf8f0b62e6ce61b9d0cd
        */
        { 6, "md5",
            {0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,

             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
             0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa}, 80,
            "Test Using Larger Than Block-Size Key - Hash Key First", 54,
            {0x6b, 0x1a, 0xb7, 0xfe, 0x4b, 0xd7, 0xbf, 0x8f,
             0x0b, 0x62, 0xe6, 0xce, 0x61, 0xb9, 0xd0, 0xcd} },

        /*

        test_case =     7
        key =           0xaa repeated 80 times
        key_len =       80
        data =          "Test Using Larger Than Block-Size Key and Larger
                        Than One Block-Size Data"
        data_len =      73
        digest =        0x6f630fad67cda0ee1fb1f562db3aa53e
        */
        { 7, "md5",
            {0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
```

```
            0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
            0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
            0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
            0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
            0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
            0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
            0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
            0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
            0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa}, 80,
           "Test Using Larger Than Block-Size Key and Larger Than One Block-Size Data", 73,
           {0x6f, 0x63, 0x0f, 0xad, 0x67, 0xcd, 0xa0, 0xee,
            0x1f, 0xb1, 0xf5, 0x62, 0xdb, 0x3a, 0xa5, 0x3e} }
    };

    unsigned long outlen;
    int err;
    int tested=0,failed=0;
    for(i=0; i < (int)(sizeof(cases) / sizeof(cases[0])); i++) {
        int hash = find_hash(cases[i].algo);
        if (hash == -1) continue;
        ++tested;
        outlen = sizeof(digest);
        if((err = hmac_memory(hash, cases[i].key, cases[i].keylen, cases[i].data, cases[i].datalen, di
#if 0
            printf("HMAC-%s test #%d, %s\n", cases[i].algo, cases[i].num, error_to_string(err));
#endif
            return err;
        }

        if(XMEMCMP(digest, cases[i].digest, (size_t)hash_descriptor[hash].hashsize) != 0)  {
            failed++;
#if 0
            unsigned int j;
            printf("\nHMAC-%s test #%d:\n", cases[i].algo, cases[i].num);
            printf(  "Result:  0x");
            for(j=0; j < hash_descriptor[hash].hashsize; j++) {
                printf("%2x ", digest[j]);
            }
            printf("\nCorrect: 0x");
            for(j=0; j < hash_descriptor[hash].hashsize; j++) {
               printf("%2x ", cases[i].digest[j]);
            }
            printf("\n");
            return CRYPT_ERROR;
#endif
        } else {
            /* printf("HMAC-%s test #%d: Passed\n", cases[i].algo, cases[i].num); */
        }
    }

    if (failed != 0) {
        return CRYPT_FAIL_TESTVECTOR;
    } else if (tested == 0) {
        return CRYPT_NOP;
    } else {
        return CRYPT_OK;
    }
 #endif
}
```

Here is the call graph for this function:

5.99 mac/omac/omac_done.c File Reference

5.99.1 Detailed Description

OMAC1 support, terminate a stream, Tom St Denis.

Definition in file omac_done.c.

`#include "tomcrypt.h"`

Include dependency graph for omac_done.c:

Functions

- int omac_done (omac_state ∗omac, unsigned char ∗out, unsigned long ∗outlen)

  Terminate an OMAC stream.

5.99.2 Function Documentation

5.99.2.1 int omac_done (omac_state ∗ *omac*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Terminate an OMAC stream.

Parameters:

omac The OMAC state

out [out] Destination for the authentication tag

outlen [in/out] The max size and resulting size of the authentication tag

Returns:

CRYPT_OK if successful

Definition at line 27 of file omac_done.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::done, ltc_cipher_descriptor::ecb_encrypt, LTC_ARGCHK, and zeromem().

Referenced by eax_done(), eax_init(), omac_file(), and omac_memory().

```
28 {
29    int       err, mode;
30    unsigned  x;
31 
32    LTC_ARGCHK(omac   != NULL);
33    LTC_ARGCHK(out    != NULL);
34    LTC_ARGCHK(outlen != NULL);
35    if ((err = cipher_is_valid(omac->cipher_idx)) != CRYPT_OK) {
36       return err;
37    }
38 
39    if ((omac->buflen > (int)sizeof(omac->block)) || (omac->buflen < 0)  ||
40        (omac->blklen > (int)sizeof(omac->block)) || (omac->buflen > omac->blklen)) {
41       return CRYPT_INVALID_ARG;
42    }
43 
44    /* figure out mode */
```

```
   if (omac->buflen != omac->blklen) {
      /* add the 0x80 byte */
      omac->block[omac->buflen++] = 0x80;

      /* pad with 0x00 */
      while (omac->buflen < omac->blklen) {
         omac->block[omac->buflen++] = 0x00;
      }
      mode = 1;
   } else {
      mode = 0;
   }

   /* now xor prev + Lu[mode] */
   for (x = 0; x < (unsigned)omac->blklen; x++) {
       omac->block[x] ^= omac->prev[x] ^ omac->Lu[mode][x];
   }

   /* encrypt it */
   if ((err = cipher_descriptor[omac->cipher_idx].ecb_encrypt(omac->block, omac->block, &omac->key)) !=
      return err;
   }
   cipher_descriptor[omac->cipher_idx].done(&omac->key);

   /* output it */
   for (x = 0; x < (unsigned)omac->blklen && x < *outlen; x++) {
       out[x] = omac->block[x];
   }
   *outlen = x;

#ifdef LTC_CLEAN_STACK
   zeromem(omac, sizeof(*omac));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.100 mac/omac/omac_file.c File Reference

5.100.1 Detailed Description

OMAC1 support, process a file, Tom St Denis.

Definition in file omac_file.c.

`#include "tomcrypt.h"`

Include dependency graph for omac_file.c:

Functions

- int omac_file (int cipher, const unsigned char ∗key, unsigned long keylen, const char ∗filename, unsigned char ∗out, unsigned long ∗outlen)

  OMAC a file.

5.100.2 Function Documentation

5.100.2.1 int omac_file (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const char ∗ *filename*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

OMAC a file.

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the secret key (octets)

filename The name of the file you wish to OMAC

out [out] Where the authentication tag is to be stored

outlen [in/out] The max size and resulting size of the authentication tag

Returns:

CRYPT_OK if successful, CRYPT_NOP if file support has been disabled

Definition at line 30 of file omac_file.c.

References CRYPT_FILE_NOTFOUND, CRYPT_NOP, CRYPT_OK, in, LTC_ARGCHK, omac_done(), omac_init(), omac_process(), and zeromem().

```
34 {
35 #ifdef LTC_NO_FILE
36    return CRYPT_NOP;
37 #else
38    int err, x;
39    omac_state omac;
40    FILE *in;
41    unsigned char buf[512];
42 
43    LTC_ARGCHK(key      != NULL);
44    LTC_ARGCHK(filename != NULL);
45    LTC_ARGCHK(out      != NULL);
```

```
   LTC_ARGCHK(outlen   != NULL);

   in = fopen(filename, "rb");
   if (in == NULL) {
      return CRYPT_FILE_NOTFOUND;
   }

   if ((err = omac_init(&omac, cipher, key, keylen)) != CRYPT_OK) {
      fclose(in);
      return err;
   }

   do {
      x = fread(buf, 1, sizeof(buf), in);
      if ((err = omac_process(&omac, buf, x)) != CRYPT_OK) {
         fclose(in);
         return err;
      }
   } while (x == sizeof(buf));
   fclose(in);

   if ((err = omac_done(&omac, out, outlen)) != CRYPT_OK) {
      return err;
   }

#ifdef LTC_CLEAN_STACK
   zeromem(buf, sizeof(buf));
#endif

   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.101 mac/omac/omac_init.c File Reference

5.101.1 Detailed Description

OMAC1 support, initialize state, by Tom St Denis.

Definition in file omac_init.c.

`#include "tomcrypt.h"`

Include dependency graph for omac_init.c:

Functions

- int omac_init (omac_state ∗omac, int cipher, const unsigned char ∗key, unsigned long keylen)

    Initialize an OMAC state.

5.101.2 Function Documentation

5.101.2.1 int omac_init (omac_state ∗ *omac*, int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*)

Initialize an OMAC state.

Parameters:

omac The OMAC state to initialize

cipher The index of the desired cipher

key The secret key

keylen The length of the secret key (octets)

Returns:

CRYPT_OK if successful

Definition at line 29 of file omac_init.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ecb_encrypt(), len, LTC_ARGCHK, mask, and zeromem().

Referenced by eax_init(), omac_file(), omac_memory(), and omac_memory_multi().

```
30 {
31    int err, x, y, mask, msb, len;
32
33    LTC_ARGCHK(omac  != NULL);
34    LTC_ARGCHK(key   != NULL);
35
36    /* schedule the key */
37    if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
38       return err;
39    }
40
41 #ifdef LTC_FAST
42    if (cipher_descriptor[cipher].block_length % sizeof(LTC_FAST_TYPE)) {
43        return CRYPT_INVALID_ARG;
44    }
```

```
#endif

   /* now setup the system */
   switch (cipher_descriptor[cipher].block_length) {
       case 8:  mask = 0x1B;
                len  = 8;
                break;
       case 16: mask = 0x87;
                len  = 16;
                break;
       default: return CRYPT_INVALID_ARG;
   }

   if ((err = cipher_descriptor[cipher].setup(key, keylen, 0, &omac->key)) != CRYPT_OK) {
      return err;
   }

   /* ok now we need Lu and Lu^2 [calc one from the other] */

   /* first calc L which is Ek(0) */
   zeromem(omac->Lu[0], cipher_descriptor[cipher].block_length);
   if ((err = cipher_descriptor[cipher].ecb_encrypt(omac->Lu[0], omac->Lu[0], &omac->key)) != CRYPT_OK)
      return err;
   }

   /* now do the mults, whoopy! */
   for (x = 0; x < 2; x++) {
       /* if msb(L * u^(x+1)) = 0 then just shift, otherwise shift and xor constant mask */
       msb = omac->Lu[x][0] >> 7;

       /* shift left */
       for (y = 0; y < (len - 1); y++) {
           omac->Lu[x][y] = ((omac->Lu[x][y] << 1) | (omac->Lu[x][y+1] >> 7)) & 255;
       }
       omac->Lu[x][len - 1] = ((omac->Lu[x][len - 1] << 1) ^ (msb ? mask : 0)) & 255;

       /* copy up as require */
       if (x == 0) {
          XMEMCPY(omac->Lu[1], omac->Lu[0], sizeof(omac->Lu[0]));
       }
   }

   /* setup state */
   omac->cipher_idx = cipher;
   omac->buflen     = 0;
   omac->blklen     = len;
   zeromem(omac->prev,  sizeof(omac->prev));
   zeromem(omac->block, sizeof(omac->block));

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.102 mac/omac/omac_memory.c File Reference

5.102.1 Detailed Description

OMAC1 support, process a block of memory, Tom St Denis.

Definition in file omac_memory.c.

`#include "tomcrypt.h"`

Include dependency graph for omac_memory.c:

Functions

- int omac_memory (int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

    OMAC a block of memory.

5.102.2 Function Documentation

5.102.2.1 int omac_memory (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

OMAC a block of memory.

Parameters:

cipher The index of the desired cipher

key The secret key

keylen The length of the secret key (octets)

in The data to send through OMAC

inlen The length of the data to send through OMAC (octets)

out [out] The destination of the authentication tag

outlen [in/out] The max size and resulting size of the authentication tag (octets)

Returns:

CRYPT_OK if successful

Definition at line 31 of file omac_memory.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, omac_-done(), omac_init(), ltc_cipher_descriptor::omac_memory, omac_process(), XFREE, XMALLOC, and ze-romem().

Referenced by omac_test().

```
{
   int err;
   omac_state *omac;

   LTC_ARGCHK(key    != NULL);
   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
```

```
   LTC_ARGCHK(outlen != NULL);

   /* is the cipher valid? */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

   /* Use accelerator if found */
   if (cipher_descriptor[cipher].omac_memory != NULL) {
      return cipher_descriptor[cipher].omac_memory(key, keylen, in, inlen, out, outlen);
   }

   /* allocate ram for omac state */
   omac = XMALLOC(sizeof(omac_state));
   if (omac == NULL) {
      return CRYPT_MEM;
   }

   /* omac process the message */
   if ((err = omac_init(omac, cipher, key, keylen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   if ((err = omac_process(omac, in, inlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   if ((err = omac_done(omac, out, outlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(omac, sizeof(omac_state));
#endif

   XFREE(omac);
   return err;
}
```

Here is the call graph for this function:

5.103 mac/omac/omac_memory_multi.c File Reference

5.103.1 Detailed Description

OMAC1 support, process multiple blocks of memory, Tom St Denis.

Definition in file omac_memory_multi.c.

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for omac_memory_multi.c:

Functions

- int omac_memory_multi (int cipher, const unsigned char ∗key, unsigned long keylen, unsigned char ∗out, unsigned long ∗outlen, const unsigned char ∗in, unsigned long inlen,...)

    OMAC multiple blocks of memory.

5.103.2 Function Documentation

5.103.2.1 int omac_memory_multi (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, const unsigned char ∗ *in*, unsigned long *inlen*, ...)

OMAC multiple blocks of memory.

Parameters:

cipher The index of the desired cipher

key The secret key

keylen The length of the secret key (octets)

out [out] The destination of the authentication tag

outlen [in/out] The max size and resulting size of the authentication tag (octets)

in The data to send through OMAC

inlen The length of the data to send through OMAC (octets)

... tuples of (data,len) pairs to OMAC, terminated with a (NULL,x) (x=don't care)

Returns:

CRYPT_OK if successful

Definition at line 33 of file omac_memory_multi.c.

References CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, omac_init(), omac_process(), and XMALLOC.

```
{
   int                  err;
   omac_state          *omac;
   va_list              args;
   const unsigned char *curptr;
   unsigned long        curlen;
```

```

   LTC_ARGCHK(key    != NULL);
   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* allocate ram for omac state */
   omac = XMALLOC(sizeof(omac_state));
   if (omac == NULL) {
      return CRYPT_MEM;
   }

   /* omac process the message */
   if ((err = omac_init(omac, cipher, key, keylen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   va_start(args, inlen);
   curptr = in;
   curlen = inlen;
   for (;;) {
      /* process buf */
      if ((err = omac_process(omac, curptr, curlen)) != CRYPT_OK) {
         goto LBL_ERR;
      }
      /* step to next */
      curptr = va_arg(args, const unsigned char*);
      if (curptr == NULL) {
         break;
      }
      curlen = va_arg(args, unsigned long);
   }
   if ((err = omac_done(omac, out, outlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(omac, sizeof(omac_state));
#endif
   XFREE(omac);
   va_end(args);
   return err;
}
```

Here is the call graph for this function:

5.104 mac/omac/omac_process.c File Reference

5.104.1 Detailed Description

OMAC1 support, process data, Tom St Denis.

Definition in file omac_process.c.

`#include "tomcrypt.h"`

Include dependency graph for omac_process.c:

Functions

- int omac_process (omac_state ∗omac, const unsigned char ∗in, unsigned long inlen)

  Process data through OMAC.

5.104.2 Function Documentation

5.104.2.1 int omac_process (omac_state ∗ *omac*, const unsigned char ∗ *in*, unsigned long *inlen*)

Process data through OMAC.

Parameters:

omac The OMAC state

in The input data to send through OMAC

inlen The length of the input (octets)

Returns:

CRYPT_OK if successful

Definition at line 28 of file omac_process.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, and LTC_ARGCHK.

Referenced by eax_addheader(), eax_decrypt(), eax_encrypt(), eax_init(), omac_file(), omac_memory(), and omac_memory_multi().

```
{
   unsigned long n, x;
   int           err;

   LTC_ARGCHK(omac  != NULL);
   LTC_ARGCHK(in    != NULL);
   if ((err = cipher_is_valid(omac->cipher_idx)) != CRYPT_OK) {
      return err;
   }

   if ((omac->buflen > (int)sizeof(omac->block)) || (omac->buflen < 0)  ||
       (omac->blklen > (int)sizeof(omac->block)) || (omac->buflen > omac->blklen)) {
      return CRYPT_INVALID_ARG;
   }

#ifdef LTC_FAST
```

```
   if (omac->buflen == 0 && inlen > 16) {
      int y;
      for (x = 0; x < (inlen - 16); x += 16) {
          for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
              *((LTC_FAST_TYPE*)(&omac->prev[y])) ^= *((LTC_FAST_TYPE*)(&in[y]));
          }
          in += 16;
          if ((err = cipher_descriptor[omac->cipher_idx].ecb_encrypt(omac->prev, omac->prev, &omac->key
             return err;
          }
      }
      inlen -= x;
    }
#endif

   while (inlen != 0) {
       /* ok if the block is full we xor in prev, encrypt and replace prev */
       if (omac->buflen == omac->blklen) {
          for (x = 0; x < (unsigned long)omac->blklen; x++) {
              omac->block[x] ^= omac->prev[x];
          }
          if ((err = cipher_descriptor[omac->cipher_idx].ecb_encrypt(omac->block, omac->prev, &omac->ke
             return err;
          }
          omac->buflen = 0;
       }

       /* add bytes */
       n = MIN(inlen, (unsigned long)(omac->blklen - omac->buflen));
       XMEMCPY(omac->block + omac->buflen, in, n);
       omac->buflen  += n;
       inlen         -= n;
       in            += n;
   }

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.105 mac/omac/omac_test.c File Reference

5.105.1 Detailed Description

OMAC1 support, self-test, by Tom St Denis.

Definition in file omac_test.c.

`#include "tomcrypt.h"`

Include dependency graph for omac_test.c:

Functions

- int omac_test (void)

  Test the OMAC setup.

5.105.2 Function Documentation

5.105.2.1 int omac_test (void)

Test the OMAC setup.

Returns:

CRYPT_OK if successful, CRYPT_NOP if tests have been disabled

Definition at line 24 of file omac_test.c.

References CRYPT_NOP, CRYPT_OK, find_cipher(), len, and omac_memory().

```
25 {
26 #if !defined(LTC_TEST)
27     return CRYPT_NOP;
28 #else
29     static const struct {
30         int keylen, msglen;
31         unsigned char key[16], msg[64], tag[16];
32     } tests[] = {
33     { 16, 0,
34       { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
35         0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c },
36       { 0x00 },
37       { 0xbb, 0x1d, 0x69, 0x29, 0xe9, 0x59, 0x37, 0x28,
38         0x7f, 0xa3, 0x7d, 0x12, 0x9b, 0x75, 0x67, 0x46 }
39     },
40     { 16, 16,
41       { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
42         0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c },
43       { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
44         0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a },
45       { 0x07, 0x0a, 0x16, 0xb4, 0x6b, 0x4d, 0x41, 0x44,
46         0xf7, 0x9b, 0xdd, 0x9d, 0xd0, 0x4a, 0x28, 0x7c }
47     },
48     { 16, 40,
49       { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
50         0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c },
51       { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
52         0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
```

```
        0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c,
        0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
        0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11 },
      { 0xdf, 0xa6, 0x67, 0x47, 0xde, 0x9a, 0xe6, 0x30,
        0x30, 0xca, 0x32, 0x61, 0x14, 0x97, 0xc8, 0x27 }
    },
    { 16, 64,
      { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
        0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c },
      { 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
        0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
        0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c,
        0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
        0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11,
        0xe5, 0xfb, 0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
        0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17,
        0xad, 0x2b, 0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10 },
      { 0x51, 0xf0, 0xbe, 0xbf, 0x7e, 0x3b, 0x9d, 0x92,
        0xfc, 0x49, 0x74, 0x17, 0x79, 0x36, 0x3c, 0xfe }
    }

    };
    unsigned char out[16];
    int x, err, idx;
    unsigned long len;


    /* AES can be under rijndael or aes... try to find it */
    if ((idx = find_cipher("aes")) == -1) {
       if ((idx = find_cipher("rijndael")) == -1) {
          return CRYPT_NOP;
       }
    }

    for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       len = sizeof(out);
       if ((err = omac_memory(idx, tests[x].key, tests[x].keylen, tests[x].msg, tests[x].msglen, out, &
          return err;
       }

       if (XMEMCMP(out, tests[x].tag, 16) != 0) {
#if 0
          int y;
          printf("\n\nTag: ");
          for (y = 0; y < 16; y++) printf("%02x", out[y]); printf("\n\n");
#endif
          return CRYPT_FAIL_TESTVECTOR;
       }
    }
    return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.106 mac/pelican/pelican.c File Reference

5.106.1 Detailed Description

Pelican MAC, initialize state, by Tom St Denis.

Definition in file pelican.c.

```
#include "tomcrypt.h"
#include "../../ciphers/aes/aes_tab.c"
```

Include dependency graph for pelican.c:

Defines

- #define ENCRYPT_ONLY
- #define PELI_TAB

Functions

- int pelican_init (pelican_state ∗pelmac, const unsigned char ∗key, unsigned long keylen)

  Initialize a Pelican state.

- static void four_rounds (pelican_state ∗pelmac)
- int pelican_process (pelican_state ∗pelmac, const unsigned char ∗in, unsigned long inlen)

  Process a block of text through Pelican.

- int pelican_done (pelican_state ∗pelmac, unsigned char ∗out)

  Terminate Pelican MAC.

5.106.2 Define Documentation

5.106.2.1 #define ENCRYPT_ONLY

Definition at line 20 of file pelican.c.

5.106.2.2 #define PELI_TAB

Definition at line 21 of file pelican.c.

5.106.3 Function Documentation

5.106.3.1 static void four_rounds (pelican_state ∗ *pelmac*) [static]

Definition at line 55 of file pelican.c.

References byte, t1, t2, t3, Te0, Te1, Te2, and Te3.

Referenced by pelican_done(), and pelican_process().

```
56 {
57     ulong32 s0, s1, s2, s3, t0, t1, t2, t3;
58     int r;
59 
60     LOAD32H(s0, pelmac->state      );
61     LOAD32H(s1, pelmac->state  +  4);
62     LOAD32H(s2, pelmac->state  +  8);
63     LOAD32H(s3, pelmac->state  + 12);
64     for (r = 0; r < 4; r++) {
65         t0 =
66             Te0(byte(s0, 3)) ^
67             Te1(byte(s1, 2)) ^
68             Te2(byte(s2, 1)) ^
69             Te3(byte(s3, 0));
70         t1 =
71             Te0(byte(s1, 3)) ^
72             Te1(byte(s2, 2)) ^
73             Te2(byte(s3, 1)) ^
74             Te3(byte(s0, 0));
75         t2 =
76             Te0(byte(s2, 3)) ^
77             Te1(byte(s3, 2)) ^
78             Te2(byte(s0, 1)) ^
79             Te3(byte(s1, 0));
80         t3 =
81             Te0(byte(s3, 3)) ^
82             Te1(byte(s0, 2)) ^
83             Te2(byte(s1, 1)) ^
84             Te3(byte(s2, 0));
85         s0 = t0; s1 = t1; s2 = t2; s3 = t3;
86     }
87     STORE32H(s0, pelmac->state      );
88     STORE32H(s1, pelmac->state  +  4);
89     STORE32H(s2, pelmac->state  +  8);
90     STORE32H(s3, pelmac->state  + 12);
91 }
```

5.106.3.2 int pelican_done (pelican_state ∗ *pelmac*, unsigned char ∗ *out*)

Terminate Pelican MAC.

Parameters:

pelmac The Pelican MAC state

out [out] The TAG

Returns:

CRYPT_OK on sucess

Definition at line 141 of file pelican.c.

References CRYPT_INVALID_ARG, CRYPT_OK, four_rounds(), and LTC_ARGCHK.

Referenced by pelican_memory().

```
142 {
143    LTC_ARGCHK(pelmac  != NULL);
144    LTC_ARGCHK(out     != NULL);
145 
146    /* check range */
147    if (pelmac->buflen < 0 || pelmac->buflen > 16) {
148       return CRYPT_INVALID_ARG;
```

```
149     }
150
151     if  (pelmac->buflen == 16) {
152         four_rounds(pelmac);
153         pelmac->buflen = 0;
154     }
155     pelmac->state[pelmac->buflen++] ^= 0x80;
156     aes_ecb_encrypt(pelmac->state, out, &pelmac->K);
157     aes_done(&pelmac->K);
158     return CRYPT_OK;
159 }
```

Here is the call graph for this function:

5.106.3.3 int pelican_init (pelican_state ∗ *pelmac*, const unsigned char ∗ *key*, unsigned long *keylen*)

Initialize a Pelican state.

Parameters:

pelmac The Pelican state to initialize

key The secret key

keylen The length of the secret key (octets)

Returns:

CRYPT_OK if successful

Definition at line 31 of file pelican.c.

References CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, and zeromem().

Referenced by pelican_memory(), and pelican_test().

```
32 {
33     int err;
34
35     LTC_ARGCHK(pelmac != NULL);
36     LTC_ARGCHK(key    != NULL);
37
38 #ifdef LTC_FAST
39     if (16 % sizeof(LTC_FAST_TYPE)) {
40         return CRYPT_INVALID_ARG;
41     }
42 #endif
43
44     if ((err = aes_setup(key, keylen, 0, &pelmac->K)) != CRYPT_OK) {
45        return err;
46     }
47
48     zeromem(pelmac->state, 16);
49     aes_ecb_encrypt(pelmac->state, pelmac->state, &pelmac->K);
50     pelmac->buflen = 0;
51
52     return CRYPT_OK;
53 }
```

Here is the call graph for this function:

5.106.3.4 int pelican_process (pelican_state * *pelmac*, const unsigned char * *in*, unsigned long *inlen*)

Process a block of text through Pelican.

Parameters:

pelmac The Pelican MAC state

in The input

inlen The length input (octets)

Returns:

CRYPT_OK on success

Definition at line 100 of file pelican.c.

References CRYPT_INVALID_ARG, CRYPT_OK, four_rounds(), and LTC_ARGCHK.

Referenced by pelican_memory().

```
{

   LTC_ARGCHK(pelmac != NULL);
   LTC_ARGCHK(in     != NULL);

   /* check range */
   if (pelmac->buflen < 0 || pelmac->buflen > 15) {
      return CRYPT_INVALID_ARG;
   }

#ifdef LTC_FAST
   if (pelmac->buflen == 0) {
      while (inlen & ~15) {
         int x;
         for (x = 0; x < 16; x += sizeof(LTC_FAST_TYPE)) {
            *((LTC_FAST_TYPE*)((unsigned char *)pelmac->state + x)) ^= *((LTC_FAST_TYPE*)((unsigned ch
         }
         four_rounds(pelmac);
         in    += 16;
         inlen -= 16;
      }
   }
#endif

   while (inlen--) {
       pelmac->state[pelmac->buflen++] ^= *in++;
       if (pelmac->buflen == 16) {
          four_rounds(pelmac);
          pelmac->buflen = 0;
       }
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.107 mac/pelican/pelican_memory.c File Reference

5.107.1 Detailed Description

Pelican MAC, MAC a block of memory, by Tom St Denis.

Definition in file pelican_memory.c.

`#include "tomcrypt.h"`

Include dependency graph for pelican_memory.c:

Functions

- int pelican_memory (const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out)

    Pelican block of memory.

5.107.2 Function Documentation

5.107.2.1 int pelican_memory (const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*)

Pelican block of memory.

Parameters:

key The key for the MAC

keylen The length of the key (octets)

in The input to MAC

inlen The length of the input (octets)

out [out] The output TAG

Returns:

CRYPT_OK on success

Definition at line 29 of file pelican_memory.c.

References CRYPT_MEM, CRYPT_OK, pelican_done(), pelican_init(), pelican_process(), XFREE, and XMALLOC.

```
{
   pelican_state *pel;
   int err;

   pel = XMALLOC(sizeof(*pel));
   if (pel == NULL) {
      return CRYPT_MEM;
   }

   if ((err = pelican_init(pel, key, keylen)) != CRYPT_OK) {
      XFREE(pel);
      return err;
   }
```

```
   if ((err = pelican_process(pel, in ,inlen)) != CRYPT_OK) {
      XFREE(pel);
      return err;
   }
   err = pelican_done(pel, out);
   XFREE(pel);
   return err;
}
```

Here is the call graph for this function:

5.108 mac/pelican/pelican_test.c File Reference

5.108.1 Detailed Description

Pelican MAC, test, by Tom St Denis.

Definition in file pelican_test.c.

`#include "tomcrypt.h"`

Include dependency graph for pelican_test.c:

Functions

- int pelican_test (void)

5.108.2 Function Documentation

5.108.2.1 int pelican_test (void)

Definition at line 20 of file pelican_test.c.

References CRYPT_NOP, CRYPT_OK, K, and pelican_init().

```
21 {
22 #ifndef LTC_TEST
23    return CRYPT_NOP;
24 #else
25    static const struct {
26         unsigned char K[32], MSG[64], T[16];
27     int keylen, ptlen;
28    } tests[] = {
29 /* K=16, M=0 */
30 {
31    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
32      0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F },
33    { 0 },
34    { 0xeb, 0x58, 0x37, 0x15, 0xf8, 0x34, 0xde, 0xe5,
35      0xa4, 0xd1, 0x6e, 0xe4, 0xb9, 0xd7, 0x76, 0x0e, },
36    16, 0
37 },
38
39 /* K=16, M=3 */
40 {
41    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
42      0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F },
43    { 0x00, 0x01, 0x02 },
44    { 0x1c, 0x97, 0x40, 0x60, 0x6c, 0x58, 0x17, 0x2d,
45      0x03, 0x94, 0x19, 0x70, 0x81, 0xc4, 0x38, 0x54, },
46    16, 3
47 },
48
49 /* K=16, M=16 */
50 {
51    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
52      0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F },
53    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
54      0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F },
55    { 0x03, 0xcc, 0x46, 0xb8, 0xac, 0xa7, 0x9c, 0x36,
56      0x1e, 0x8c, 0x6e, 0xa6, 0x7b, 0x89, 0x32, 0x49, },
57    16, 16
```

```
},

/* K=16, M=32 */
{
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F },
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
     0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
     0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F },
   { 0x89, 0xcc, 0x36, 0x58, 0x1b, 0xdd, 0x4d, 0xb5,
     0x78, 0xbb, 0xac, 0xf0, 0xff, 0x8b, 0x08, 0x15, },
   16, 32
},

/* K=16, M=35 */
{
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F },
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
     0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
     0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
     0x20, 0x21, 0x23 },
   { 0x4a, 0x7d, 0x45, 0x4d, 0xcd, 0xb5, 0xda, 0x8d,
     0x48, 0x78, 0x16, 0x48, 0x5d, 0x45, 0x95, 0x99, },
   16, 35
},
};
   int x, err;
   unsigned char out[16];
   pelican_state pel;

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       if ((err = pelican_init(&pel, tests[x].K, tests[x].keylen)) != CRYPT_OK) {
          return err;
       }
       if ((err = pelican_process(&pel, tests[x].MSG, tests[x].ptlen)) != CRYPT_OK) {
          return err;
       }
       if ((err = pelican_done(&pel, out)) != CRYPT_OK) {
          return err;
       }

       if (XMEMCMP(out, tests[x].T, 16)) {
#if 0
           int y;
           printf("\nFailed test %d\n", x);
           printf("{ "); for (y = 0; y < 16; ) { printf("0x%02x, ", out[y]); if (!(++y & 7)) printf("\
#endif
           return CRYPT_FAIL_TESTVECTOR;
       }
   }
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.109 mac/pmac/pmac_done.c File Reference

5.109.1 Detailed Description

PMAC implementation, terminate a session, by Tom St Denis.

Definition in file pmac_done.c.

`#include "tomcrypt.h"`

Include dependency graph for pmac_done.c:

Functions

- int pmac_done (pmac_state ∗state, unsigned char ∗out, unsigned long ∗outlen)

5.109.2 Function Documentation

5.109.2.1 int pmac_done (pmac_state ∗ *state*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Definition at line 20 of file pmac_done.c.

References cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, and LTC_ARGCHK.

Referenced by pmac_file(), and pmac_memory().

```
{
   int err, x;

   LTC_ARGCHK(state != NULL);
   LTC_ARGCHK(out   != NULL);
   if ((err = cipher_is_valid(state->cipher_idx)) != CRYPT_OK) {
      return err;
   }

   if ((state->buflen > (int)sizeof(state->block)) || (state->buflen < 0) ||
       (state->block_len > (int)sizeof(state->block)) || (state->buflen > state->block_len)) {
      return CRYPT_INVALID_ARG;
   }


   /* handle padding.  If multiple xor in L/x */

   if (state->buflen == state->block_len) {
      /* xor Lr against the checksum */
      for (x = 0; x < state->block_len; x++) {
          state->checksum[x] ^= state->block[x] ^ state->Lr[x];
      }
   } else {
      /* otherwise xor message bytes then the 0x80 byte */
      for (x = 0; x < state->buflen; x++) {
          state->checksum[x] ^= state->block[x];
      }
      state->checksum[x] ^= 0x80;
   }

   /* encrypt it */
   if ((err = cipher_descriptor[state->cipher_idx].ecb_encrypt(state->checksum, state->checksum, &state
      return err;
   }
   cipher_descriptor[state->cipher_idx].done(&state->key);
```

```

   /* store it */
   for (x = 0; x < state->block_len && x < (int)*outlen; x++) {
       out[x] = state->checksum[x];
   }
   *outlen = x;

#ifdef LTC_CLEAN_STACK
   zeromem(state, sizeof(*state));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.110 mac/pmac/pmac_file.c File Reference

5.110.1 Detailed Description

PMAC implementation, process a file, by Tom St Denis.

Definition in file pmac_file.c.

`#include "tomcrypt.h"`

Include dependency graph for pmac_file.c:

Functions

- int pmac_file (int cipher, const unsigned char ∗key, unsigned long keylen, const char ∗filename, unsigned char ∗out, unsigned long ∗outlen)

  PMAC a file.

5.110.2 Function Documentation

5.110.2.1 int pmac_file (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const char ∗ *filename*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

PMAC a file.

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the secret key (octets)

filename The name of the file to send through PMAC

out [out] Destination for the authentication tag

outlen [in/out] Max size and resulting size of the authentication tag

Returns:

CRYPT_OK if successful, CRYPT_NOP if file support has been disabled

Definition at line 30 of file pmac_file.c.

References CRYPT_FILE_NOTFOUND, CRYPT_NOP, CRYPT_OK, in, LTC_ARGCHK, pmac_done(), pmac_init(), pmac_process(), and zeromem().

```
{
#ifdef LTC_NO_FILE
   return CRYPT_NOP;
#else
   int err, x;
   pmac_state pmac;
   FILE *in;
   unsigned char buf[512];


   LTC_ARGCHK(key      != NULL);
   LTC_ARGCHK(filename != NULL);
```

```
   LTC_ARGCHK(out      != NULL);
   LTC_ARGCHK(outlen   != NULL);

   in = fopen(filename, "rb");
   if (in == NULL) {
      return CRYPT_FILE_NOTFOUND;
   }

   if ((err = pmac_init(&pmac, cipher, key, keylen)) != CRYPT_OK) {
      fclose(in);
      return err;
   }

   do {
      x = fread(buf, 1, sizeof(buf), in);
      if ((err = pmac_process(&pmac, buf, x)) != CRYPT_OK) {
         fclose(in);
         return err;
      }
   } while (x == sizeof(buf));
   fclose(in);

   if ((err = pmac_done(&pmac, out, outlen)) != CRYPT_OK) {
      return err;
   }

#ifdef LTC_CLEAN_STACK
   zeromem(buf, sizeof(buf));
#endif

   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.111 mac/pmac/pmac_init.c File Reference

5.111.1 Detailed Description

PMAC implementation, initialize state, by Tom St Denis.

Definition in file pmac_init.c.

`#include "tomcrypt.h"`

Include dependency graph for pmac_init.c:

Functions

- int pmac_init (pmac_state ∗pmac, int cipher, const unsigned char ∗key, unsigned long keylen)

  Initialize a PMAC state.

Variables

- struct {
  int len
  unsigned char poly_div [MAXBLOCKSIZE]
  unsigned char poly_mul [MAXBLOCKSIZE]
  int code
  int value
  } polys [ ]

5.111.2 Function Documentation

5.111.2.1 int pmac_init (pmac_state ∗ *pmac*, int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*)

Initialize a PMAC state.

Parameters:

pmac The PMAC state to initialize

cipher The index of the desired cipher

key The secret key

keylen The length of the secret key (octets)

Returns:

CRYPT_OK if successful

Definition at line 46 of file pmac_init.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_OK, len, LTC_ARGCHK, poly, and polys.

Referenced by pmac_file(), pmac_memory(), and pmac_memory_multi().

```
{
   int poly, x, y, m, err;
   unsigned char *L;

   LTC_ARGCHK(pmac  != NULL);
   LTC_ARGCHK(key   != NULL);

   /* valid cipher? */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

   /* determine which polys to use */
   pmac->block_len = cipher_descriptor[cipher].block_length;
   for (poly = 0; poly < (int)(sizeof(polys)/sizeof(polys[0])); poly++) {
       if (polys[poly].len == pmac->block_len) {
          break;
       }
   }
   if (polys[poly].len != pmac->block_len) {
      return CRYPT_INVALID_ARG;
   }

#ifdef LTC_FAST
   if (pmac->block_len % sizeof(LTC_FAST_TYPE)) {
      return CRYPT_INVALID_ARG;
   }
#endif


   /* schedule the key */
   if ((err = cipher_descriptor[cipher].setup(key, keylen, 0, &pmac->key)) != CRYPT_OK) {
      return err;
   }

   /* allocate L */
   L = XMALLOC(pmac->block_len);
   if (L == NULL) {
      return CRYPT_MEM;
   }

   /* find L = E[0] */
   zeromem(L, pmac->block_len);
   if ((err = cipher_descriptor[cipher].ecb_encrypt(L, L, &pmac->key)) != CRYPT_OK) {
      goto error;
   }

   /* find Ls[i] = L << i for i == 0..31 */
   XMEMCPY(pmac->Ls[0], L, pmac->block_len);
   for (x = 1; x < 32; x++) {
       m = pmac->Ls[x-1][0] >> 7;
       for (y = 0; y < pmac->block_len-1; y++) {
           pmac->Ls[x][y] = ((pmac->Ls[x-1][y] << 1) | (pmac->Ls[x-1][y+1] >> 7)) & 255;
       }
       pmac->Ls[x][pmac->block_len-1] = (pmac->Ls[x-1][pmac->block_len-1] << 1) & 255;

       if (m == 1) {
          for (y = 0; y < pmac->block_len; y++) {
              pmac->Ls[x][y] ^= polys[poly].poly_mul[y];
          }
       }
    }

    /* find Lr = L / x */
    m = L[pmac->block_len-1] & 1;

    /* shift right */
```

```
    for (x = pmac->block_len - 1; x > 0; x--) {
        pmac->Lr[x] = ((L[x] >> 1) | (L[x-1] << 7)) & 255;
    }
    pmac->Lr[0] = L[0] >> 1;

    if (m == 1) {
       for (x = 0; x < pmac->block_len; x++) {
           pmac->Lr[x] ^= polys[poly].poly_div[x];
       }
    }

    /* zero buffer, counters, etc... */
    pmac->block_index = 1;
    pmac->cipher_idx  = cipher;
    pmac->buflen      = 0;
    zeromem(pmac->block,    sizeof(pmac->block));
    zeromem(pmac->Li,       sizeof(pmac->Li));
    zeromem(pmac->checksum, sizeof(pmac->checksum));
    err = CRYPT_OK;
error:
#ifdef LTC_CLEAN_STACK
    zeromem(L, pmac->block_len);
#endif

    XFREE(L);

    return err;
}
```

Here is the call graph for this function:

5.111.3 Variable Documentation

5.111.3.1 int len

Definition at line 21 of file pmac_init.c.

5.111.3.2 unsigned char poly_div[MAXBLOCKSIZE]

Definition at line 22 of file pmac_init.c.

5.111.3.3 unsigned char poly_mul[MAXBLOCKSIZE]

Definition at line 22 of file pmac_init.c.

5.111.3.4 const { ... } polys[ ] `[static]`

5.112 mac/pmac/pmac_memory.c File Reference

5.112.1 Detailed Description

PMAC implementation, process a block of memory, by Tom St Denis.

Definition in file pmac_memory.c.

`#include "tomcrypt.h"`

Include dependency graph for pmac_memory.c:

Functions

- int pmac_memory (int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  PMAC a block of memory.

5.112.2 Function Documentation

5.112.2.1 int pmac_memory (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

PMAC a block of memory.

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the secret key (octets)

in The data you wish to send through PMAC

inlen The length of data you wish to send through PMAC (octets)

out [out] Destination for the authentication tag

outlen [in/out] The max size and resulting size of the authentication tag

Returns:

CRYPT_OK if successful

Definition at line 31 of file pmac_memory.c.

References CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, pmac_done(), pmac_init(), pmac_process(), XFREE, XMALLOC, and zeromem().

Referenced by pmac_test().

```
{
   int err;
   pmac_state *pmac;

   LTC_ARGCHK(key    != NULL);
   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);
```

```

   /* allocate ram for pmac state */
   pmac = XMALLOC(sizeof(pmac_state));
   if (pmac == NULL) {
      return CRYPT_MEM;
   }

   if ((err = pmac_init(pmac, cipher, key, keylen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   if ((err = pmac_process(pmac, in, inlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   if ((err = pmac_done(pmac, out, outlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(pmac, sizeof(pmac_state));
#endif

   XFREE(pmac);
   return err;
}
```

Here is the call graph for this function:

5.113 mac/pmac/pmac_memory_multi.c File Reference

5.113.1 Detailed Description

PMAC implementation, process multiple blocks of memory, by Tom St Denis.

Definition in file pmac_memory_multi.c.

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for pmac_memory_multi.c:

Functions

- int pmac_memory_multi (int cipher, const unsigned char ∗key, unsigned long keylen, unsigned char ∗out, unsigned long ∗outlen, const unsigned char ∗in, unsigned long inlen,...)

  PMAC multiple blocks of memory.

5.113.2 Function Documentation

5.113.2.1 int pmac_memory_multi (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, const unsigned char ∗ *in*, unsigned long *inlen*, ...)

PMAC multiple blocks of memory.

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the secret key (octets)

out [out] Destination for the authentication tag

outlen [in/out] The max size and resulting size of the authentication tag

in The data you wish to send through PMAC

inlen The length of data you wish to send through PMAC (octets)

... tuples of (data,len) pairs to PMAC, terminated with a (NULL,x) (x=don't care)

Returns:

CRYPT_OK if successful

Definition at line 33 of file pmac_memory_multi.c.

References CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, pmac_init(), pmac_process(), and XMALLOC.

```
{
   int                  err;
   pmac_state          *pmac;
   va_list              args;
   const unsigned char *curptr;
   unsigned long        curlen;
```

```c

   LTC_ARGCHK(key    != NULL);
   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* allocate ram for pmac state */
   pmac = XMALLOC(sizeof(pmac_state));
   if (pmac == NULL) {
      return CRYPT_MEM;
   }

   if ((err = pmac_init(pmac, cipher, key, keylen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   va_start(args, inlen);
   curptr = in;
   curlen = inlen;
   for (;;) {
      /* process buf */
      if ((err = pmac_process(pmac, curptr, curlen)) != CRYPT_OK) {
         goto LBL_ERR;
      }
      /* step to next */
      curptr = va_arg(args, const unsigned char*);
      if (curptr == NULL) {
         break;
      }
      curlen = va_arg(args, unsigned long);
   }
   if ((err = pmac_done(pmac, out, outlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(pmac, sizeof(pmac_state));
#endif
   XFREE(pmac);
   va_end(args);
   return err;
}
```

Here is the call graph for this function:

5.114 mac/pmac/pmac_ntz.c File Reference

5.114.1 Detailed Description

PMAC implementation, internal function, by Tom St Denis.

Definition in file pmac_ntz.c.

`#include "tomcrypt.h"`

Include dependency graph for pmac_ntz.c:

Functions

- int pmac_ntz (unsigned long x)

    Internal PMAC function.

5.114.2 Function Documentation

5.114.2.1 int pmac_ntz (unsigned long *x*)

Internal PMAC function.

Definition at line 23 of file pmac_ntz.c.

References c.

Referenced by pmac_shift_xor().

```
24 {
25    int c;
26    x &= 0xFFFFFFFFUL;
27    c = 0;
28    while ((x & 1) == 0) {
29       ++c;
30       x >>= 1;
31    }
32    return c;
33 }
```

5.115 mac/pmac/pmac_process.c File Reference

5.115.1 Detailed Description

PMAC implementation, process data, by Tom St Denis.

Definition in file pmac_process.c.

`#include "tomcrypt.h"`

Include dependency graph for pmac_process.c:

Functions

- int pmac_process (pmac_state ∗pmac, const unsigned char ∗in, unsigned long inlen)

  Process data in a PMAC stream.

5.115.2 Function Documentation

5.115.2.1 int pmac_process (pmac_state ∗ *pmac*, const unsigned char ∗ *in*, unsigned long *inlen*)

Process data in a PMAC stream.

Parameters:

pmac The PMAC state

in The data to send through PMAC

inlen The length of the data to send through PMAC

Returns:

CRYPT_OK if successful

Definition at line 28 of file pmac_process.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, LTC_ARGCHK, MAXBLOCKSIZE, and pmac_shift_xor().

Referenced by pmac_file(), pmac_memory(), and pmac_memory_multi().

```
29 {
30    int err, n;
31    unsigned long x;
32    unsigned char Z[MAXBLOCKSIZE];
33
34    LTC_ARGCHK(pmac != NULL);
35    LTC_ARGCHK(in   != NULL);
36    if ((err = cipher_is_valid(pmac->cipher_idx)) != CRYPT_OK) {
37       return err;
38    }
39
40    if ((pmac->buflen > (int)sizeof(pmac->block)) || (pmac->buflen < 0) ||
41        (pmac->block_len > (int)sizeof(pmac->block)) || (pmac->buflen > pmac->block_len)) {
42       return CRYPT_INVALID_ARG;
43    }
44
45 #ifdef LTC_FAST
```

```
   if (pmac->buflen == 0 && inlen > 16) {
      unsigned long y;
      for (x = 0; x < (inlen - 16); x += 16) {
          pmac_shift_xor(pmac);
          for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
              *((LTC_FAST_TYPE*)(&Z[y])) = *((LTC_FAST_TYPE*)(&in[y])) ^ *((LTC_FAST_TYPE*)(&pmac->Li[y
          }
          if ((err = cipher_descriptor[pmac->cipher_idx].ecb_encrypt(Z, Z, &pmac->key)) != CRYPT_OK) {
             return err;
          }
          for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
              *((LTC_FAST_TYPE*)(&pmac->checksum[y])) ^= *((LTC_FAST_TYPE*)(&Z[y]));
          }
          in += 16;
      }
      inlen -= x;
   }
#endif

   while (inlen != 0) {
       /* ok if the block is full we xor in prev, encrypt and replace prev */
       if (pmac->buflen == pmac->block_len) {
          pmac_shift_xor(pmac);
          for (x = 0; x < (unsigned long)pmac->block_len; x++) {
               Z[x] = pmac->Li[x] ^ pmac->block[x];
          }
          if ((err = cipher_descriptor[pmac->cipher_idx].ecb_encrypt(Z, Z, &pmac->key)) != CRYPT_OK) {
             return err;
           }
          for (x = 0; x < (unsigned long)pmac->block_len; x++) {
              pmac->checksum[x] ^= Z[x];
          }
          pmac->buflen = 0;
       }

       /* add bytes */
       n = MIN(inlen, (unsigned long)(pmac->block_len - pmac->buflen));
       XMEMCPY(pmac->block + pmac->buflen, in, n);
       pmac->buflen  += n;
       inlen         -= n;
       in            += n;
   }

#ifdef LTC_CLEAN_STACK
   zeromem(Z, sizeof(Z));
#endif

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.116 mac/pmac/pmac_shift_xor.c File Reference

5.116.1 Detailed Description

PMAC implementation, internal function, by Tom St Denis.

Definition in file pmac_shift_xor.c.

`#include "tomcrypt.h"`

Include dependency graph for pmac_shift_xor.c:

Functions

- void pmac_shift_xor (pmac_state ∗pmac)

  Internal function.

5.116.2 Function Documentation

5.116.2.1 void pmac_shift_xor (pmac_state ∗ *pmac*)

Internal function.

Performs the state update (adding correct multiple)

Parameters:

pmac The PMAC state.

Definition at line 24 of file pmac_shift_xor.c.

References pmac_ntz().

Referenced by pmac_process().

```
{
   int x, y;
   y = pmac_ntz(pmac->block_index++);
#ifdef LTC_FAST
   for (x = 0; x < pmac->block_len; x += sizeof(LTC_FAST_TYPE)) {
       *((LTC_FAST_TYPE*)((unsigned char *)pmac->Li + x)) ^=
       *((LTC_FAST_TYPE*)((unsigned char *)pmac->Ls[y] + x));
   }
#else
   for (x = 0; x < pmac->block_len; x++) {
       pmac->Li[x] ^= pmac->Ls[y][x];
   }
#endif
}
```

Here is the call graph for this function:

5.117 mac/pmac/pmac_test.c File Reference

5.117.1 Detailed Description

PMAC implementation, self-test, by Tom St Denis.

Definition in file pmac_test.c.

`#include "tomcrypt.h"`

Include dependency graph for pmac_test.c:

Functions

- int pmac_test (void)

  Test the OMAC implementation.

5.117.2 Function Documentation

5.117.2.1 int pmac_test (void)

Test the OMAC implementation.

Returns:

CRYPT_OK if successful, CRYPT_NOP if testing has been disabled

Definition at line 25 of file pmac_test.c.

References CRYPT_NOP, CRYPT_OK, find_cipher(), len, MAXBLOCKSIZE, and pmac_memory().

```
26 {
27 #if !defined(LTC_TEST)
28     return CRYPT_NOP;
29 #else
30     static const struct {
31         int msglen;
32         unsigned char key[16], msg[34], tag[16];
33     } tests[] = {
34 
35    /* PMAC-AES-128-0B */
36 {
37    0,
38    /* key */
39    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
40      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
41    /* msg */
42    { 0x00 },
43    /* tag */
44    { 0x43, 0x99, 0x57, 0x2c, 0xd6, 0xea, 0x53, 0x41,
45      0xb8, 0xd3, 0x58, 0x76, 0xa7, 0x09, 0x8a, 0xf7 }
46 },
47 
48    /* PMAC-AES-128-3B */
49 {
50    3,
51    /* key */
52    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
53      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
```

```
    /* msg */
    { 0x00, 0x01, 0x02 },
    /* tag */
    { 0x25, 0x6b, 0xa5, 0x19, 0x3c, 0x1b, 0x99, 0x1b,
      0x4d, 0xf0, 0xc5, 0x1f, 0x38, 0x8a, 0x9e, 0x27 }
},

    /* PMAC-AES-128-16B */
{
    16,
    /* key */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
    /* msg */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
    /* tag */
    { 0xeb, 0xbd, 0x82, 0x2f, 0xa4, 0x58, 0xda, 0xf6,
      0xdf, 0xda, 0xd7, 0xc2, 0x7d, 0xa7, 0x63, 0x38 }
},

    /* PMAC-AES-128-20B */
{
    20,
    /* key */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
    /* msg */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
      0x10, 0x11, 0x12, 0x13 },
    /* tag */
    { 0x04, 0x12, 0xca, 0x15, 0x0b, 0xbf, 0x79, 0x05,
      0x8d, 0x8c, 0x75, 0xa5, 0x8c, 0x99, 0x3f, 0x55 }
},

    /* PMAC-AES-128-32B */
{
    32,
    /* key */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
    /* msg */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
      0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f },
    /* tag */
    { 0xe9, 0x7a, 0xc0, 0x4e, 0x9e, 0x5e, 0x33, 0x99,
      0xce, 0x53, 0x55, 0xcd, 0x74, 0x07, 0xbc, 0x75 }
},

    /* PMAC-AES-128-34B */
{
    34,
    /* key */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },
    /* msg */
    { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
      0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
      0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
      0x20, 0x21 },
    /* tag */
    { 0x5c, 0xba, 0x7d, 0x5e, 0xb2, 0x4f, 0x7c, 0x86,
      0xcc, 0xc5, 0x46, 0x04, 0xe5, 0x3d, 0x55, 0x12 }
```

```
}

};
   int err, x, idx;
   unsigned long len;
   unsigned char outtag[MAXBLOCKSIZE];

    /* AES can be under rijndael or aes... try to find it */
    if ((idx = find_cipher("aes")) == -1) {
       if ((idx = find_cipher("rijndael")) == -1) {
          return CRYPT_NOP;
       }
    }

    for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
        len = sizeof(outtag);
        if ((err = pmac_memory(idx, tests[x].key, 16, tests[x].msg, tests[x].msglen, outtag, &len)) !=
           return err;
        }

        if (XMEMCMP(outtag, tests[x].tag, len)) {
#if 0
           unsigned long y;
           printf("\nTAG:\n");
           for (y = 0; y < len; ) {
               printf("0x%02x", outtag[y]);
               if (y < len-1) printf(", ");
               if (!(++y % 8)) printf("\n");
           }
#endif
           return CRYPT_FAIL_TESTVECTOR;
        }
     }
     return CRYPT_OK;
#endif /* LTC_TEST */
}
```

Here is the call graph for this function:

5.118 mac/xcbc/xcbc_done.c File Reference

5.118.1 Detailed Description

XCBC Support, terminate the state.

Definition in file xcbc_done.c.

`#include "tomcrypt.h"`

Include dependency graph for xcbc_done.c:

Functions

- int xcbc_done (xcbc_state ∗xcbc, unsigned char ∗out, unsigned long ∗outlen)

  Terminate the XCBC-MAC state.

5.118.2 Function Documentation

5.118.2.1 int xcbc_done (xcbc_state ∗ *xcbc*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Terminate the XCBC-MAC state.

Parameters:

xcbc XCBC state to terminate

out [out] Destination for the MAC tag

outlen [in/out] Destination size and final tag size Return CRYPT_OK on success

Definition at line 26 of file xcbc_done.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_OK, and LTC_ARGCHK.

Referenced by xcbc_file(), and xcbc_memory().

```
{
   int err, x;
   LTC_ARGCHK(xcbc != NULL);
   LTC_ARGCHK(out  != NULL);

   /* check structure */
   if ((err = cipher_is_valid(xcbc->cipher)) != CRYPT_OK) {
      return err;
   }

   if ((xcbc->blocksize > cipher_descriptor[xcbc->cipher].block_length) || (xcbc->blocksize < 0) ||
       (xcbc->buflen > xcbc->blocksize) || (xcbc->buflen < 0)) {
      return CRYPT_INVALID_ARG;
   }

   /* which key do we use? */
   if (xcbc->buflen == xcbc->blocksize) {
      /* k2 */
      for (x = 0; x < xcbc->blocksize; x++) {
         xcbc->IV[x] ^= xcbc->K[1][x];
      }
```

```
   } else {
      xcbc->IV[xcbc->buflen] ^= 0x80;
      /* k3 */
      for (x = 0; x < xcbc->blocksize; x++) {
         xcbc->IV[x] ^= xcbc->K[2][x];
      }
   }

   /* encrypt */
   cipher_descriptor[xcbc->cipher].ecb_encrypt(xcbc->IV, xcbc->IV, &xcbc->key);
   cipher_descriptor[xcbc->cipher].done(&xcbc->key);

   /* extract tag */
   for (x = 0; x < xcbc->blocksize && (unsigned long)x < *outlen; x++) {
      out[x] = xcbc->IV[x];
   }
   *outlen = x;

#ifdef LTC_CLEAN_STACK
   zeromem(xcbc, sizeof(*xcbc));
#endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.119 mac/xcbc/xcbc_file.c File Reference

5.119.1 Detailed Description

XCBC support, process a file, Tom St Denis.

Definition in file xcbc_file.c.

`#include "tomcrypt.h"`

Include dependency graph for xcbc_file.c:

Functions

- int xcbc_file (int cipher, const unsigned char ∗key, unsigned long keylen, const char ∗filename, unsigned char ∗out, unsigned long ∗outlen)

  XCBC a file.

5.119.2 Function Documentation

5.119.2.1 int xcbc_file (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const char ∗ *filename*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

XCBC a file.

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the secret key (octets)

filename The name of the file you wish to XCBC

out [out] Where the authentication tag is to be stored

outlen [in/out] The max size and resulting size of the authentication tag

Returns:

CRYPT_OK if successful, CRYPT_NOP if file support has been disabled

Definition at line 30 of file xcbc_file.c.

References CRYPT_FILE_NOTFOUND, CRYPT_NOP, CRYPT_OK, in, LTC_ARGCHK, xcbc_done(), xcbc_init(), xcbc_process(), and zeromem().

```
{
#ifdef LTC_NO_FILE
   return CRYPT_NOP;
#else
   int err, x;
   xcbc_state xcbc;
   FILE *in;
   unsigned char buf[512];

   LTC_ARGCHK(key      != NULL);
   LTC_ARGCHK(filename != NULL);
   LTC_ARGCHK(out      != NULL);
```

```
   LTC_ARGCHK(outlen   != NULL);

   in = fopen(filename, "rb");
   if (in == NULL) {
      return CRYPT_FILE_NOTFOUND;
   }

   if ((err = xcbc_init(&xcbc, cipher, key, keylen)) != CRYPT_OK) {
      fclose(in);
      return err;
   }

   do {
      x = fread(buf, 1, sizeof(buf), in);
      if ((err = xcbc_process(&xcbc, buf, x)) != CRYPT_OK) {
         fclose(in);
         return err;
      }
   } while (x == sizeof(buf));
   fclose(in);

   if ((err = xcbc_done(&xcbc, out, outlen)) != CRYPT_OK) {
      return err;
   }

#ifdef LTC_CLEAN_STACK
   zeromem(buf, sizeof(buf));
#endif

   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.120 mac/xcbc/xcbc_init.c File Reference

5.120.1 Detailed Description

XCBC Support, start an XCBC state.

Definition in file xcbc_init.c.

`#include "tomcrypt.h"`

Include dependency graph for xcbc_init.c:

Functions

- int xcbc_init (xcbc_state ∗xcbc, int cipher, const unsigned char ∗key, unsigned long keylen)

  Initialize XCBC-MAC state.

5.120.2 Function Documentation

5.120.2.1 int xcbc_init (xcbc_state ∗ *xcbc*, int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*)

Initialize XCBC-MAC state.

Parameters:

xcbc [out] XCBC state to initialize

cipher Index of cipher to use

key [in] Secret key

keylen Length of secret key in octets Return CRYPT_OK on success

Definition at line 27 of file xcbc_init.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, and XCALLOC.

Referenced by xcbc_file(), xcbc_memory(), and xcbc_memory_multi().

```
{
   int            x, y, err;
   symmetric_key *skey;

   LTC_ARGCHK(xcbc != NULL);
   LTC_ARGCHK(key  != NULL);

   /* schedule the key */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

#ifdef LTC_FAST
   if (cipher_descriptor[cipher].block_length % sizeof(LTC_FAST_TYPE)) {
       return CRYPT_INVALID_ARG;
   }
#endif

```

```
   /* schedule the user key */
   skey = XCALLOC(1, sizeof(*skey));
   if (skey == NULL) {
      return CRYPT_MEM;
   }

   if ((err = cipher_descriptor[cipher].setup(key, keylen, 0, skey)) != CRYPT_OK) {
      goto done;
   }

   /* make the three keys */
   for (y = 0; y < 3; y++) {
     for (x = 0; x < cipher_descriptor[cipher].block_length; x++) {
        xcbc->K[y][x] = y + 1;
     }
     cipher_descriptor[cipher].ecb_encrypt(xcbc->K[y], xcbc->K[y], skey);
   }

   /* setup K1 */
   err = cipher_descriptor[cipher].setup(xcbc->K[0], cipher_descriptor[cipher].block_length, 0, &xcbc->

   /* setup struct */
   zeromem(xcbc->IV, cipher_descriptor[cipher].block_length);
   xcbc->blocksize = cipher_descriptor[cipher].block_length;
   xcbc->cipher    = cipher;
   xcbc->buflen    = 0;
done:
   cipher_descriptor[cipher].done(skey);
#ifdef LTC_CLEAN_STACK
   zeromem(skey, sizeof(*skey));
#endif
   XFREE(skey);
   return err;
}
```

Here is the call graph for this function:

5.121 mac/xcbc/xcbc_memory.c File Reference

`#include "tomcrypt.h"`

Include dependency graph for xcbc_memory.c:

Functions

- int xcbc_memory (int cipher, const unsigned char ∗key, unsigned long keylen, const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  XCBC-MAC a block of memory.

5.121.1 Function Documentation

5.121.1.1 int xcbc_memory (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

XCBC-MAC a block of memory.

Parameters:

cipher Index of cipher to use

key [in] Secret key

keylen Length of key in octets

in [in] Message to MAC

inlen Length of input in octets

out [out] Destination for the MAC tag

outlen [in/out] Output size and final tag size Return CRYPT_OK on success.

Definition at line 30 of file xcbc_memory.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_MEM, CRYPT_OK, XCALLOC, xcbc_done(), xcbc_init(), ltc_cipher_descriptor::xcbc_memory, xcbc_process(), and XFREE.

Referenced by xcbc_test().

```
{
   xcbc_state *xcbc;
   int         err;

   /* is the cipher valid? */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

   /* Use accelerator if found */
   if (cipher_descriptor[cipher].xcbc_memory != NULL) {
      return cipher_descriptor[cipher].xcbc_memory(key, keylen, in, inlen, out, outlen);
   }

   xcbc = XCALLOC(1, sizeof(*xcbc));
   if (xcbc == NULL) {
      return CRYPT_MEM;
   }

```

```
   if ((err = xcbc_init(xcbc, cipher, key, keylen)) != CRYPT_OK) {
     goto done;
   }

   if ((err = xcbc_process(xcbc, in, inlen)) != CRYPT_OK) {
     goto done;
   }

   err = xcbc_done(xcbc, out, outlen);
done:
   XFREE(xcbc);
   return err;
}
```

Here is the call graph for this function:

5.122 mac/xcbc/xcbc_memory_multi.c File Reference

5.122.1 Detailed Description

XCBC support, process multiple blocks of memory, Tom St Denis.

Definition in file xcbc_memory_multi.c.

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for xcbc_memory_multi.c:

Functions

- int xcbc_memory_multi (int cipher, const unsigned char ∗key, unsigned long keylen, unsigned char ∗out, unsigned long ∗outlen, const unsigned char ∗in, unsigned long inlen,...)

  XCBC multiple blocks of memory.

5.122.2 Function Documentation

5.122.2.1 int xcbc_memory_multi (int *cipher*, const unsigned char ∗ *key*, unsigned long *keylen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, const unsigned char ∗ *in*, unsigned long *inlen*, ...)

XCBC multiple blocks of memory.

Parameters:

cipher The index of the desired cipher

key The secret key

keylen The length of the secret key (octets)

out [out] The destination of the authentication tag

outlen [in/out] The max size and resulting size of the authentication tag (octets)

in The data to send through XCBC

inlen The length of the data to send through XCBC (octets)

... tuples of (data,len) pairs to XCBC, terminated with a (NULL,x) (x=don't care)

Returns:

CRYPT_OK if successful

Definition at line 33 of file xcbc_memory_multi.c.

References CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, xcbc_init(), xcbc_process(), and XMALLOC.

```
{
   int                  err;
   xcbc_state          *xcbc;
   va_list              args;
   const unsigned char *curptr;
   unsigned long        curlen;
```

```

   LTC_ARGCHK(key    != NULL);
   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* allocate ram for xcbc state */
   xcbc = XMALLOC(sizeof(xcbc_state));
   if (xcbc == NULL) {
      return CRYPT_MEM;
   }

   /* xcbc process the message */
   if ((err = xcbc_init(xcbc, cipher, key, keylen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   va_start(args, inlen);
   curptr = in;
   curlen = inlen;
   for (;;) {
      /* process buf */
      if ((err = xcbc_process(xcbc, curptr, curlen)) != CRYPT_OK) {
         goto LBL_ERR;
      }
      /* step to next */
      curptr = va_arg(args, const unsigned char*);
      if (curptr == NULL) {
         break;
      }
      curlen = va_arg(args, unsigned long);
   }
   if ((err = xcbc_done(xcbc, out, outlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(xcbc, sizeof(xcbc_state));
#endif
   XFREE(xcbc);
   va_end(args);
   return err;
}
```

Here is the call graph for this function:

5.123 mac/xcbc/xcbc_process.c File Reference

5.123.1 Detailed Description

XCBC Support, terminate the state.

Definition in file xcbc_process.c.

`#include "tomcrypt.h"`

Include dependency graph for xcbc_process.c:

Functions

- int xcbc_process (xcbc_state ∗xcbc, const unsigned char ∗in, unsigned long inlen)

  Process data through XCBC-MAC.

5.123.2 Function Documentation

5.123.2.1 int xcbc_process (xcbc_state ∗ *xcbc*, const unsigned char ∗ *in*, unsigned long *inlen*)

Process data through XCBC-MAC.

Parameters:

xcbc The XCBC-MAC state

in Input data to process

inlen Length of input in octets Return CRYPT_OK on success

Definition at line 26 of file xcbc_process.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, and LTC_ARGCHK.

Referenced by xcbc_file(), xcbc_memory(), and xcbc_memory_multi().

```
{
   int err;
#ifdef LTC_FAST
   int x;
#endif

   LTC_ARGCHK(xcbc != NULL);
   LTC_ARGCHK(in   != NULL);

   /* check structure */
   if ((err = cipher_is_valid(xcbc->cipher)) != CRYPT_OK) {
      return err;
   }

   if ((xcbc->blocksize > cipher_descriptor[xcbc->cipher].block_length) || (xcbc->blocksize < 0) ||
       (xcbc->buflen > xcbc->blocksize) || (xcbc->buflen < 0)) {
      return CRYPT_INVALID_ARG;
   }

#ifdef LTC_FAST
   if (xcbc->buflen == 0) {
```

```
      while (inlen > (unsigned long)xcbc->blocksize) {
          for (x = 0; x < xcbc->blocksize; x += sizeof(LTC_FAST_TYPE)) {
             *((LTC_FAST_TYPE*)&(xcbc->IV[x])) ^= *((LTC_FAST_TYPE*)&(in[x]));
          }
          cipher_descriptor[xcbc->cipher].ecb_encrypt(xcbc->IV, xcbc->IV, &xcbc->key);
          in    += xcbc->blocksize;
          inlen -= xcbc->blocksize;
      }
  }
#endif

   while (inlen) {
     if (xcbc->buflen == xcbc->blocksize) {
         cipher_descriptor[xcbc->cipher].ecb_encrypt(xcbc->IV, xcbc->IV, &xcbc->key);
         xcbc->buflen = 0;
     }
     xcbc->IV[xcbc->buflen++] ^= *in++;
     --inlen;
  }
  return CRYPT_OK;
}
```

Here is the call graph for this function:

5.124 mac/xcbc/xcbc_test.c File Reference

5.124.1 Detailed Description

XCBC Support, terminate the state.

Definition in file xcbc_test.c.

`#include "tomcrypt.h"`

Include dependency graph for xcbc_test.c:

Functions

- int xcbc_test (void)

  Test XCBC-MAC mode Return CRYPT_OK on succes.

5.124.2 Function Documentation

5.124.2.1 int xcbc_test (void)

Test XCBC-MAC mode Return CRYPT_OK on succes.

Definition at line 23 of file xcbc_test.c.

References CRYPT_NOP, CRYPT_OK, find_cipher(), K, and xcbc_memory().

```
{
#ifdef LTC_NO_TEST
   return CRYPT_NOP;
#else
   static const struct {
       int msglen;
       unsigned char K[16], M[34], T[16];
   } tests[] = {
{
   0,
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },

   { 0 },

   { 0x75, 0xf0, 0x25, 0x1d, 0x52, 0x8a, 0xc0, 0x1c,
     0x45, 0x73, 0xdf, 0xd5, 0x84, 0xd7, 0x9f, 0x29 }
},

{
   3,
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },

   { 0x00, 0x01, 0x02 },

   { 0x5b, 0x37, 0x65, 0x80, 0xae, 0x2f, 0x19, 0xaf,
     0xe7, 0x21, 0x9c, 0xee, 0xf1, 0x72, 0x75, 0x6f }
},

{
   16,
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
```

```
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },

   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },

   { 0xd2, 0xa2, 0x46, 0xfa, 0x34, 0x9b, 0x68, 0xa7,
     0x99, 0x98, 0xa4, 0x39, 0x4f, 0xf7, 0xa2, 0x63 }
},

{
   32,
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },

   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
     0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
     0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f },

   { 0xf5, 0x4f, 0x0e, 0xc8, 0xd2, 0xb9, 0xf3, 0xd3,
     0x68, 0x07, 0x73, 0x4b, 0xd5, 0x28, 0x3f, 0xd4 }
},

{
   34,
   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f },

   { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
     0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
     0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
     0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
     0x20, 0x21 },

   { 0xbe, 0xcb, 0xb3, 0xbc, 0xcd, 0xb5, 0x18, 0xa3,
     0x06, 0x77, 0xd5, 0x48, 0x1f, 0xb6, 0xb4, 0xd8 },
},



};
  unsigned char T[16];
  unsigned long taglen;
   int err, x, idx;

   /* AES can be under rijndael or aes... try to find it */
   if ((idx = find_cipher("aes")) == -1) {
      if ((idx = find_cipher("rijndael")) == -1) {
         return CRYPT_NOP;
      }
   }

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
      taglen = 16;
      if ((err = xcbc_memory(idx, tests[x].K, 16, tests[x].M, tests[x].msglen, T, &taglen)) != CRYPT_OK
         return err;
      }
      if (taglen != 16 || XMEMCMP(T, tests[x].T, 16)) {
         return CRYPT_FAIL_TESTVECTOR;
      }
   }

   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.125 math/fp/ltc_ecc_fp_mulmod.c File Reference

5.125.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ltc_ecc_fp_mulmod.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ltc_ecc_fp_mulmod.c:

5.126 math/gmp_desc.c File Reference

```
#include "tomcrypt.h"
```

Include dependency graph for gmp_desc.c:

Defines

- #define DESC_DEF_ONLY

5.126.1 Define Documentation

5.126.1.1 #define DESC_DEF_ONLY

Definition at line 12 of file gmp_desc.c.

5.127 math/ltm_desc.c File Reference

```
#include "tomcrypt.h"
```

Include dependency graph for ltm_desc.c:

Defines

- #define DESC_DEF_ONLY

5.127.1 Define Documentation

5.127.1.1 #define DESC_DEF_ONLY

Definition at line 12 of file ltm_desc.c.

5.128 math/multi.c File Reference

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for multi.c:

Functions

- int ltc_init_multi (void **a,...)
- void ltc_deinit_multi (void *a,...)

5.128.1 Function Documentation

5.128.1.1 void ltc_deinit_multi (void * *a*, ...)

Definition at line 44 of file multi.c.

```
{
   void     *cur = a;
   va_list   args;

   va_start(args, a);
   while (cur != NULL) {
       mp_clear(cur);
       cur = va_arg(args, void *);
   }
   va_end(args);
}
```

5.128.1.2 int ltc_init_multi (void ** *a*, ...)

Definition at line 16 of file multi.c.

References CRYPT_MEM, and CRYPT_OK.

```
{
   void    **cur = a;
   int       np  = 0;
   va_list   args;

   va_start(args, a);
   while (cur != NULL) {
       if (mp_init(cur) != CRYPT_OK) {
          /* failed */
          va_list clean_list;

          va_start(clean_list, a);
          cur = a;
          while (np--) {
              mp_clear(*cur);
              cur = va_arg(clean_list, void**);
          }
          va_end(clean_list);
          return CRYPT_MEM;
       }
       ++np;
```

```
        cur = va_arg(args, void**);
    }
    va_end(args);
    return CRYPT_OK;
}
```

5.129 math/rand_prime.c File Reference

5.129.1 Detailed Description

Generate a random prime, Tom St Denis.

Definition in file rand_prime.c.

`#include "tomcrypt.h"`

Include dependency graph for rand_prime.c:

Defines

- #define USE_BBS 1

Functions

- int rand_prime (void ∗N, long len, prng_state ∗prng, int wprng)

5.129.2 Define Documentation

5.129.2.1 #define USE_BBS 1

Definition at line 18 of file rand_prime.c.

Referenced by rand_prime().

5.129.3 Function Documentation

5.129.3.1 int rand_prime (void ∗ *N*, long *len*, prng_state ∗ *prng*, int *wprng*)

Definition at line 20 of file rand_prime.c.

References CRYPT_ERROR_READPRNG, CRYPT_INVALID_PRIME_SIZE, CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, prng_descriptor, prng_is_valid(), USE_BBS, XCALLOC, and XFREE.

Referenced by dsa_make_key(), and rsa_make_key().

```
{
   int            err, res, type;
   unsigned char *buf;

   LTC_ARGCHK(N != NULL);

   /* get type */
   if (len < 0) {
      type = USE_BBS;
      len = -len;
   } else {
      type = 0;
   }

   /* allow sizes between 2 and 512 bytes for a prime size */
   if (len < 2 || len > 512) {
      return CRYPT_INVALID_PRIME_SIZE;
```

```
   }

   /* valid PRNG? Better be! */
   if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
      return err;
   }

   /* allocate buffer to work with */
   buf = XCALLOC(1, len);
   if (buf == NULL) {
       return CRYPT_MEM;
   }

   do {
      /* generate value */
      if (prng_descriptor[wprng].read(buf, len, prng) != (unsigned long)len) {
         XFREE(buf);
         return CRYPT_ERROR_READPRNG;
      }

      /* munge bits */
      buf[0]     |= 0x80 | 0x40;
      buf[len-1] |= 0x01 | ((type & USE_BBS) ? 0x02 : 0x00);

      /* load value */
      if ((err = mp_read_unsigned_bin(N, buf, len)) != CRYPT_OK) {
         XFREE(buf);
         return err;
      }

      /* test */
      if ((err = mp_prime_is_prime(N, 8, &res)) != CRYPT_OK) {
         XFREE(buf);
         return err;
      }
   } while (res == LTC_MP_NO);

#ifdef LTC_CLEAN_STACK
   zeromem(buf, len);
#endif

   XFREE(buf);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.130 math/tfm_desc.c File Reference

`#include "tomcrypt.h"`

Include dependency graph for tfm_desc.c:

Defines

- #define DESC_DEF_ONLY

5.130.1 Define Documentation

5.130.1.1 #define DESC_DEF_ONLY

Definition at line 12 of file tfm_desc.c.

5.131 misc/base64/base64_decode.c File Reference

5.131.1 Detailed Description

Compliant base64 code donated by Wayne Scott (wscott@bitmover.com).

Definition in file base64_decode.c.

`#include "tomcrypt.h"`

Include dependency graph for base64_decode.c:

Functions

- int base64_decode (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  base64 decode a block of memory

Variables

- static const unsigned char map [256]

5.131.2 Function Documentation

5.131.2.1 int base64_decode (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

base64 decode a block of memory

Parameters:

in The base64 data to decode

inlen The length of the base64 data

out [out] The destination of the binary decoded data

outlen [in/out] The max size and resulting size of the decoded data

Returns:

CRYPT_OK if successful

Definition at line 53 of file base64_decode.c.

References c, CRYPT_INVALID_PACKET, LTC_ARGCHK, and map.

```
{
   unsigned long t, x, y, z;
   unsigned char c;
   int           g;

   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   g = 3;
```

```
   for (x = y = z = t = 0; x < inlen; x++) {
       c = map[in[x]&0xFF];
       if (c == 255) continue;
       /* the final = symbols are read and used to trim the remaining bytes */
       if (c == 254) {
          c = 0;
          /* prevent g < 0 which would potentially allow an overflow later */
          if (--g < 0) {
             return CRYPT_INVALID_PACKET;
          }
       } else if (g != 3) {
          /* we only allow = to be at the end */
          return CRYPT_INVALID_PACKET;
       }

       t = (t<<6)|c;

       if (++y == 4) {
          if (z + g > *outlen) {
             return CRYPT_BUFFER_OVERFLOW;
          }
          out[z++] = (unsigned char)((t>>16)&255);
          if (g > 1) out[z++] = (unsigned char)((t>>8)&255);
          if (g > 2) out[z++] = (unsigned char)(t&255);
          y = t = 0;
       }
   }
   if (y != 0) {
       return CRYPT_INVALID_PACKET;
   }
   *outlen = z;
   return CRYPT_OK;
}
```

5.131.3 Variable Documentation

5.131.3.1 const unsigned char map[256] [static]

Initial value:

```
 {
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255,  62, 255, 255, 255,  63,
 52,  53,  54,  55,  56,  57,  58,  59,  60,  61, 255, 255,
255, 254, 255, 255, 255,   0,   1,   2,   3,   4,   5,   6,
  7,   8,   9,  10,  11,  12,  13,  14,  15,  16,  17,  18,
 19,  20,  21,  22,  23,  24,  25, 255, 255, 255, 255, 255,
255,  26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,
 37,  38,  39,  40,  41,  42,  43,  44,  45,  46,  47,  48,
 49,  50,  51, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255 }
```

Definition at line 21 of file base64_decode.c.

Referenced by base64_decode().

5.132 misc/base64/base64_encode.c File Reference

5.132.1 Detailed Description

Compliant base64 encoder donated by Wayne Scott (`wscott@bitmover.com`).

Definition in file base64_encode.c.

`#include "tomcrypt.h"`

Include dependency graph for base64_encode.c:

Functions

- int base64_encode (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  base64 Encode a buffer (NUL terminated)

Variables

- static const char ∗ codes

5.132.2 Function Documentation

5.132.2.1 int base64_encode (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

base64 Encode a buffer (NUL terminated)

Parameters:

in The input buffer to encode

inlen The length of the input buffer

out [out] The destination of the base64 encoded data

outlen [in/out] The max size and resulting size

Returns:

CRYPT_OK if successful

Definition at line 32 of file base64_encode.c.

References codes, CRYPT_BUFFER_OVERFLOW, and LTC_ARGCHK.

```
{
   unsigned long i, len2, leven;
   unsigned char *p;

   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* valid output size ? */
   len2 = 4 * ((inlen + 2) / 3);
```

```
    if (*outlen < len2 + 1) {
       *outlen = len2 + 1;
       return CRYPT_BUFFER_OVERFLOW;
    }
    p = out;
    leven = 3*(inlen / 3);
    for (i = 0; i < leven; i += 3) {
        *p++ = codes[(in[0] >> 2) & 0x3F];
        *p++ = codes[(((in[0] & 3) << 4) + (in[1] >> 4)) & 0x3F];
        *p++ = codes[(((in[1] & 0xf) << 2) + (in[2] >> 6)) & 0x3F];
        *p++ = codes[in[2] & 0x3F];
        in += 3;
    }
    /* Pad it if necessary...  */
    if (i < inlen) {
        unsigned a = in[0];
        unsigned b = (i+1 < inlen) ? in[1] : 0;

        *p++ = codes[(a >> 2) & 0x3F];
        *p++ = codes[(((a & 3) << 4) + (b >> 4)) & 0x3F];
        *p++ = (i+1 < inlen) ? codes[(((b & 0xf) << 2)) & 0x3F] : '=';
        *p++ = '=';
    }

    /* append a NULL byte */
    *p = '\0';

    /* return ok */
    *outlen = p - out;
    return CRYPT_OK;
 }
```

5.132.3 Variable Documentation

5.132.3.1 const char* codes [static]

Initial value:

```
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
```

Definition at line 21 of file base64_encode.c.

Referenced by base64_encode().

5.133 misc/burn_stack.c File Reference

5.133.1 Detailed Description

Burn stack, Tom St Denis.

Definition in file burn_stack.c.

```
#include "tomcrypt.h"
```

Include dependency graph for burn_stack.c:

Functions

- void burn_stack (unsigned long len)

  Burn some stack memory.

5.133.2 Function Documentation

5.133.2.1 void burn_stack (unsigned long *len*)

Burn some stack memory.

Parameters:

len amount of stack to burn in bytes

Definition at line 22 of file burn_stack.c.

References burn_stack(), and zeromem().

Referenced by burn_stack().

```
{
   unsigned char buf[32];
   zeromem(buf, sizeof(buf));
   if (len > (unsigned long)sizeof(buf))
      burn_stack(len - sizeof(buf));
}
```

Here is the call graph for this function:

5.134 misc/crypt/crypt.c File Reference

5.134.1 Detailed Description

Build strings, Tom St Denis.

Definition in file crypt.c.

```
#include "tomcrypt.h"
```

Include dependency graph for crypt.c:

Variables

- const char * crypt_build_settings

5.134.2 Variable Documentation

5.134.2.1 const char* crypt_build_settings

Definition at line 18 of file crypt.c.

5.135 misc/crypt/crypt_argchk.c File Reference

5.135.1 Detailed Description

Perform argument checking, Tom St Denis.

Definition in file crypt_argchk.c.

`#include "tomcrypt.h"`

`#include <signal.h>`

Include dependency graph for crypt_argchk.c:

Functions

- void crypt_argchk (char *v, char *s, int d)

5.135.2 Function Documentation

5.135.2.1 void crypt_argchk (char * *v*, char * *s*, int *d*)

Definition at line 20 of file crypt_argchk.c.

```
{
 fprintf(stderr, "LTC_ARGCHK '%s' failure on line %d of file %s\n",
         v, d, s);
 (void)raise(SIGABRT);
}
```

5.136 misc/crypt/crypt_cipher_descriptor.c File Reference

5.136.1 Detailed Description

Stores the cipher descriptor table, Tom St Denis.

Definition in file crypt_cipher_descriptor.c.

```
#include "tomcrypt.h"
```

Include dependency graph for crypt_cipher_descriptor.c:

Variables

- ltc_cipher_descriptor cipher_descriptor [TAB_SIZE]

5.136.2 Variable Documentation

5.136.2.1 struct ltc_cipher_descriptor cipher_descriptor[TAB_SIZE]

Initial value:

```
 {
{ NULL, 0, 0, 0, 0, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
 }
```

Definition at line 18 of file crypt_cipher_descriptor.c.

5.137 misc/crypt/crypt_cipher_is_valid.c File Reference

5.137.1 Detailed Description

Determine if cipher is valid, Tom St Denis.

Definition in file crypt_cipher_is_valid.c.

```
#include "tomcrypt.h"
```

Include dependency graph for crypt_cipher_is_valid.c:

Functions

- int cipher_is_valid (int idx)

5.137.2 Function Documentation

5.137.2.1 int cipher_is_valid (int *idx*)

Definition at line 23 of file crypt_cipher_is_valid.c.

References cipher_descriptor, CRYPT_INVALID_CIPHER, CRYPT_OK, LTC_MUTEX_LOCK, LTC_-MUTEX_UNLOCK, ltc_cipher_descriptor::name, and TAB_SIZE.

Referenced by cbc_decrypt(), cbc_done(), cbc_encrypt(), cbc_start(), ccm_memory(), cfb_decrypt(), cfb_-done(), cfb_encrypt(), cfb_setiv(), cfb_start(), chc_init(), chc_register(), ctr_done(), ctr_encrypt(), ctr_-setiv(), ctr_start(), eax_init(), ecb_decrypt(), ecb_done(), ecb_encrypt(), ecb_start(), f8_done(), f8_-encrypt(), f8_setiv(), f8_start(), f9_done(), f9_init(), f9_memory(), f9_process(), gcm_add_aad(), gcm_-add_iv(), gcm_done(), gcm_init(), gcm_memory(), gcm_process(), lrw_decrypt(), lrw_done(), lrw_-encrypt(), lrw_setiv(), lrw_start(), ocb_decrypt(), ocb_encrypt(), ocb_init(), ofb_done(), ofb_encrypt(), ofb_setiv(), ofb_start(), omac_done(), omac_init(), omac_memory(), omac_process(), pmac_done(), pmac_init(), pmac_process(), s_ocb_done(), xcbc_done(), xcbc_init(), xcbc_memory(), xcbc_process(), yarrow_ready(), and yarrow_start().

```
{
   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   if (idx < 0 || idx >= TAB_SIZE || cipher_descriptor[idx].name == NULL) {
      LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
      return CRYPT_INVALID_CIPHER;
   }
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
   return CRYPT_OK;
}
```

5.138 misc/crypt/crypt_find_cipher.c File Reference

5.138.1 Detailed Description

Find a cipher in the descriptor tables, Tom St Denis.

Definition in file crypt_find_cipher.c.

```
#include "tomcrypt.h"
```

Include dependency graph for crypt_find_cipher.c:

Functions

- int find_cipher (const char *name)

  Find a registered cipher by name.

5.138.2 Function Documentation

5.138.2.1 int find_cipher (const char * *name*)

Find a registered cipher by name.

Parameters:

name The name of the cipher to look for

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_cipher.c.

References cipher_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, and TAB_SIZE.

Referenced by ccm_test(), ctr_test(), eax_test(), f8_test_mode(), f9_test(), find_cipher_any(), gcm_test(), lrw_test(), ocb_test(), omac_test(), pmac_test(), and xcbc_test().

```
{
   int x;
   LTC_ARGCHK(name != NULL);
   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (cipher_descriptor[x].name != NULL && !strcmp(cipher_descriptor[x].name, name)) {
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return x;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
   return -1;
}
```

5.139 misc/crypt/crypt_find_cipher_any.c File Reference

5.139.1 Detailed Description

Find a cipher in the descriptor tables, Tom St Denis.

Definition in file crypt_find_cipher_any.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_find_cipher_any.c:

Functions

- int find_cipher_any (const char ∗name, int blocklen, int keylen)

    Find a cipher flexibly.

5.139.2 Function Documentation

5.139.2.1 int find_cipher_any (const char ∗ *name*, int *blocklen*, int *keylen*)

Find a cipher flexibly.

First by name then if not present by block and key size

Parameters:

name The name of the cipher desired

blocklen The minimum length of the block cipher desired (octets)

keylen The minimum length of the key size desired (octets)

Returns:

>= 0 if found, -1 if not present

Definition at line 25 of file crypt_find_cipher_any.c.

References cipher_descriptor, find_cipher(), LTC_ARGCHK, LTC_MUTEX_LOCK, and TAB_SIZE.

```
{
   int x;

   LTC_ARGCHK(name != NULL);

   x = find_cipher(name);
   if (x != -1) return x;

   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (cipher_descriptor[x].name == NULL) {
          continue;
       }
       if (blocklen <= (int)cipher_descriptor[x].block_length && keylen <= (int)cipher_descriptor[x].ma
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return x;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
```

```
   return -1;
}
```

Here is the call graph for this function:

5.140 misc/crypt/crypt_find_cipher_id.c File Reference

5.140.1 Detailed Description

Find cipher by ID, Tom St Denis.

Definition in file crypt_find_cipher_id.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_find_cipher_id.c:

Functions

- int find_cipher_id (unsigned char ID)

  Find a cipher by ID number.

5.140.2 Function Documentation

5.140.2.1 int find_cipher_id (unsigned char *ID*)

Find a cipher by ID number.

Parameters:

ID The ID (not same as index) of the cipher to find

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_cipher_id.c.

References cipher_descriptor, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_cipher_descriptor::name, and TAB_SIZE.

```
24 {
25    int x;
26    LTC_MUTEX_LOCK(&ltc_cipher_mutex);
27    for (x = 0; x < TAB_SIZE; x++) {
28        if (cipher_descriptor[x].ID == ID) {
29           x = (cipher_descriptor[x].name == NULL) ? -1 : x;
30           LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
31           return x;
32        }
33    }
34    LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
35    return -1;
36 }
```

5.141 misc/crypt/crypt_find_hash.c File Reference

5.141.1 Detailed Description

Find a hash, Tom St Denis.

Definition in file crypt_find_hash.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_find_hash.c:

Functions

- int find_hash (const char *name)

  Find a registered hash by name.

5.141.2 Function Documentation

5.141.2.1 int find_hash (const char * *name*)

Find a registered hash by name.

Parameters:

name The name of the hash to look for

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_hash.c.

References hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, and TAB_SIZE.

Referenced by chc_register(), find_hash_any(), and hmac_test().

```
{
   int x;
   LTC_ARGCHK(name != NULL);
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (hash_descriptor[x].name != NULL && strcmp(hash_descriptor[x].name, name) == 0) {
          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
          return x;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return -1;
}
```

5.142 misc/crypt/crypt_find_hash_any.c File Reference

5.142.1 Detailed Description

Find a hash, Tom St Denis.

Definition in file crypt_find_hash_any.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_find_hash_any.c:

Functions

- int find_hash_any (const char ∗name, int digestlen)

  Find a hash flexibly.

5.142.2 Function Documentation

5.142.2.1 int find_hash_any (const char ∗ *name*, int *digestlen*)

Find a hash flexibly.

First by name then if not present by digest size

Parameters:

name The name of the hash desired

digestlen The minimum length of the digest size (octets)

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_hash_any.c.

References find_hash(), hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, MAXBLOCKSIZE, and TAB_SIZE.

```
{
   int x, y, z;
   LTC_ARGCHK(name != NULL);

   x = find_hash(name);
   if (x != -1) return x;

   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   y = MAXBLOCKSIZE+1;
   z = -1;
   for (x = 0; x < TAB_SIZE; x++) {
       if (hash_descriptor[x].name == NULL) {
          continue;
       }
       if ((int)hash_descriptor[x].hashsize >= digestlen && (int)hash_descriptor[x].hashsize < y) {
          z = x;
          y = hash_descriptor[x].hashsize;
       }
   }
```

```
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return z;
}
```

Here is the call graph for this function:

5.143 misc/crypt/crypt_find_hash_id.c File Reference

5.143.1 Detailed Description

Find hash by ID, Tom St Denis.

Definition in file crypt_find_hash_id.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_find_hash_id.c:

Functions

- int find_hash_id (unsigned char ID)

  Find a hash by ID number.

5.143.2 Function Documentation

5.143.2.1 int find_hash_id (unsigned char *ID*)

Find a hash by ID number.

Parameters:

ID The ID (not same as index) of the hash to find

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_hash_id.c.

References hash_descriptor, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_cipher_descriptor::name, and TAB_SIZE.

```
24 {
25    int x;
26    LTC_MUTEX_LOCK(&ltc_hash_mutex);
27    for (x = 0; x < TAB_SIZE; x++) {
28       if (hash_descriptor[x].ID == ID) {
29           x = (hash_descriptor[x].name == NULL) ? -1 : x;
30           LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
31           return x;
32       }
33    }
34    LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
35    return -1;
36 }
```

5.144 misc/crypt/crypt_find_hash_oid.c File Reference

5.144.1 Detailed Description

Find a hash, Tom St Denis.

Definition in file crypt_find_hash_oid.c.

```
#include "tomcrypt.h"
```

Include dependency graph for crypt_find_hash_oid.c:

Functions

- int find_hash_oid (const unsigned long ∗ID, unsigned long IDlen)

5.144.2 Function Documentation

5.144.2.1 int find_hash_oid (const unsigned long ∗ *ID*, unsigned long *IDlen*)

Definition at line 18 of file crypt_find_hash_oid.c.

References hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_-cipher_descriptor::name, TAB_SIZE, and XMEMCMP.

Referenced by dsa_decrypt_key(), and ecc_decrypt_key().

```
{
   int x;
   LTC_ARGCHK(ID != NULL);
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (hash_descriptor[x].name != NULL && hash_descriptor[x].OIDlen == IDlen && !XMEMCMP(hash_descr
          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
          return x;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return -1;
}
```

5.145 misc/crypt/crypt_find_prng.c File Reference

5.145.1 Detailed Description

Find a PRNG, Tom St Denis.

Definition in file crypt_find_prng.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_find_prng.c:

Functions

- int find_prng (const char ∗name)

  Find a registered PRNG by name.

5.145.2 Function Documentation

5.145.2.1 int find_prng (const char ∗ *name*)

Find a registered PRNG by name.

Parameters:

name The name of the PRNG to look for

Returns:

>= 0 if found, -1 if not present

Definition at line 23 of file crypt_find_prng.c.

References LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, prng_descriptor, and TAB_SIZE.

```
24 {
25    int x;
26    LTC_ARGCHK(name != NULL);
27    LTC_MUTEX_LOCK(&ltc_prng_mutex);
28    for (x = 0; x < TAB_SIZE; x++) {
29        if ((prng_descriptor[x].name != NULL) && strcmp(prng_descriptor[x].name, name) == 0) {
30           LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
31           return x;
32        }
33    }
34    LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
35    return -1;
36 }
```

5.146 misc/crypt/crypt_fsa.c File Reference

5.146.1 Detailed Description

LibTomCrypt FULL SPEED AHEAD!, Tom St Denis.

Definition in file crypt_fsa.c.

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for crypt_fsa.c:

Functions

- int crypt_fsa (void ∗mp,...)

5.146.2 Function Documentation

5.146.2.1 int crypt_fsa (void ∗ *mp*, ...)

Definition at line 20 of file crypt_fsa.c.

References CRYPT_OK, ltc_mp, register_cipher(), register_hash(), register_prng(), and XMEMCPY.

```
21 {
22    int      err;
23    va_list  args;
24    void     *p;
25
26    va_start(args, mp);
27    if (mp != NULL) {
28       XMEMCPY(&ltc_mp, mp, sizeof(ltc_mp));
29    }
30
31    while ((p = va_arg(args, void*)) != NULL) {
32       if ((err = register_cipher(p)) != CRYPT_OK) {
33          va_end(args);
34          return err;
35       }
36    }
37
38    while ((p = va_arg(args, void*)) != NULL) {
39       if ((err = register_hash(p)) != CRYPT_OK) {
40          va_end(args);
41          return err;
42       }
43    }
44
45    while ((p = va_arg(args, void*)) != NULL) {
46       if ((err = register_prng(p)) != CRYPT_OK) {
47          va_end(args);
48          return err;
49       }
50    }
51
52    va_end(args);
53    return CRYPT_OK;
54 }
```

Here is the call graph for this function:

5.147 misc/crypt/crypt_hash_descriptor.c File Reference

5.147.1 Detailed Description

Stores the hash descriptor table, Tom St Denis.

Definition in file crypt_hash_descriptor.c.

```
#include "tomcrypt.h"
```

Include dependency graph for crypt_hash_descriptor.c:

Variables

- ltc_hash_descriptor hash_descriptor [TAB_SIZE]

5.147.2 Variable Documentation

5.147.2.1 struct ltc_hash_descriptor hash_descriptor[TAB_SIZE]

Initial value:

```
 {
{ NULL, 0, 0, 0, { 0 }, 0, NULL, NULL, NULL, NULL, NULL }
}
```

Definition at line 18 of file crypt_hash_descriptor.c.

5.148 misc/crypt/crypt_hash_is_valid.c File Reference

5.148.1 Detailed Description

Determine if hash is valid, Tom St Denis.

Definition in file crypt_hash_is_valid.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_hash_is_valid.c:

Functions

- int hash_is_valid (int idx)

5.148.2 Function Documentation

5.148.2.1 int hash_is_valid (int *idx*)

Definition at line 23 of file crypt_hash_is_valid.c.

References CRYPT_INVALID_HASH, CRYPT_OK, hash_descriptor, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_hash_descriptor::name, and TAB_SIZE.

Referenced by chc_register(), dsa_decrypt_key(), dsa_encrypt_key(), ecc_decrypt_key(), ecc_encrypt_key(), hash_file(), hash_filehandle(), hash_memory(), hash_memory_multi(), hmac_done(), hmac_file(), hmac_init(), hmac_memory(), hmac_process(), pkcs_1_mgf1(), pkcs_1_oaep_decode(), pkcs_1_oaep_encode(), pkcs_1_pss_decode(), pkcs_1_pss_encode(), pkcs_5_alg1(), pkcs_5_alg2(), rsa_decrypt_key_ex(), rsa_encrypt_key_ex(), rsa_sign_hash_ex(), rsa_verify_hash_ex(), yarrow_add_entropy(), yarrow_ready(), and yarrow_start().

```
{
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   if (idx < 0 || idx >= TAB_SIZE || hash_descriptor[idx].name == NULL) {
      LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
      return CRYPT_INVALID_HASH;
   }
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return CRYPT_OK;
}
```

5.149 misc/crypt/crypt_ltc_mp_descriptor.c File Reference

```
#include "tomcrypt.h"
```

Include dependency graph for crypt_ltc_mp_descriptor.c:

Variables

- ltc_math_descriptor ltc_mp

5.149.1 Variable Documentation

5.149.1.1 ltc_math_descriptor ltc_mp

Definition at line 13 of file crypt_ltc_mp_descriptor.c.

Referenced by crypt_fsa(), dsa_import(), dsa_make_key(), ecc_import(), ecc_make_key(), ecc_shared_-secret(), ecc_verify_hash(), rsa_decrypt_key_ex(), rsa_encrypt_key_ex(), rsa_import(), rsa_make_key(), rsa_sign_hash_ex(), and rsa_verify_hash_ex().

5.150 misc/crypt/crypt_prng_descriptor.c File Reference

5.150.1 Detailed Description

Stores the PRNG descriptors, Tom St Denis.

Definition in file crypt_prng_descriptor.c.

```
#include "tomcrypt.h"
```

Include dependency graph for crypt_prng_descriptor.c:

Variables

- ltc_prng_descriptor prng_descriptor [TAB_SIZE]

5.150.2 Variable Documentation

5.150.2.1 struct ltc_prng_descriptor prng_descriptor[TAB_SIZE]

Initial value:

```
 {
{ NULL, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL }
}
```

Definition at line 17 of file crypt_prng_descriptor.c.

5.151 misc/crypt/crypt_prng_is_valid.c File Reference

5.151.1 Detailed Description

Determine if PRNG is valid, Tom St Denis.

Definition in file crypt_prng_is_valid.c.

```
#include "tomcrypt.h"
```

Include dependency graph for crypt_prng_is_valid.c:

Functions

- int prng_is_valid (int idx)

5.151.2 Function Documentation

5.151.2.1 int prng_is_valid (int *idx*)

Definition at line 23 of file crypt_prng_is_valid.c.

References CRYPT_INVALID_PRNG, CRYPT_OK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_prng_descriptor::name, prng_descriptor, and TAB_SIZE.

Referenced by dsa_encrypt_key(), dsa_make_key(), dsa_sign_hash_raw(), ecc_encrypt_key(), ecc_make_-key(), ecc_sign_hash(), pkcs_1_oaep_encode(), pkcs_1_pss_encode(), pkcs_1_v1_5_encode(), rand_-prime(), rng_make_prng(), rsa_encrypt_key_ex(), rsa_make_key(), and rsa_sign_hash_ex().

```
{
   LTC_MUTEX_LOCK(&ltc_prng_mutex);
   if (idx < 0 || idx >= TAB_SIZE || prng_descriptor[idx].name == NULL) {
      LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
      return CRYPT_INVALID_PRNG;
   }
   LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
   return CRYPT_OK;
}
```

5.152 misc/crypt/crypt_register_cipher.c File Reference

5.152.1 Detailed Description

Register a cipher, Tom St Denis.

Definition in file crypt_register_cipher.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_register_cipher.c:

Functions

- int register_cipher (const struct ltc_cipher_descriptor ∗cipher)
  Register a cipher with the descriptor table.

5.152.2 Function Documentation

5.152.2.1 int register_cipher (const struct ltc_cipher_descriptor ∗ *cipher*)

Register a cipher with the descriptor table.

Parameters:
cipher The cipher you wish to register

Returns:
value >= 0 if successfully added (or already present), -1 if unsuccessful

Definition at line 23 of file crypt_register_cipher.c.

References cipher_descriptor, ltc_cipher_descriptor::ID, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_-MUTEX_UNLOCK, ltc_prng_descriptor::name, and TAB_SIZE.

Referenced by crypt_fsa(), and yarrow_start().

```
{
   int x;

   LTC_ARGCHK(cipher != NULL);

   /* is it already registered? */
   LTC_MUTEX_LOCK(&ltc_cipher_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (cipher_descriptor[x].name != NULL && cipher_descriptor[x].ID == cipher->ID) {
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return x;
       }
   }

   /* find a blank spot */
   for (x = 0; x < TAB_SIZE; x++) {
       if (cipher_descriptor[x].name == NULL) {
          XMEMCPY(&cipher_descriptor[x], cipher, sizeof(struct ltc_cipher_descriptor));
          LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
          return x;
       }
```

```
   }

   /* no spot */
   LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
   return -1;
}
```

5.153 misc/crypt/crypt_register_hash.c File Reference

5.153.1 Detailed Description

Register a HASH, Tom St Denis.

Definition in file crypt_register_hash.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_register_hash.c:

Functions

- int register_hash (const struct ltc_hash_descriptor ∗hash)

  Register a hash with the descriptor table.

5.153.2 Function Documentation

5.153.2.1 int register_hash (const struct ltc_hash_descriptor ∗ *hash*)

Register a hash with the descriptor table.

Parameters:

hash The hash you wish to register

Returns:

value >= 0 if successfully added (or already present), -1 if unsuccessful

Definition at line 23 of file crypt_register_hash.c.

References hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, TAB_-SIZE, and XMEMCMP.

Referenced by crypt_fsa(), and yarrow_start().

```
{
   int x;

   LTC_ARGCHK(hash != NULL);

   /* is it already registered? */
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (XMEMCMP(&hash_descriptor[x], hash, sizeof(struct ltc_hash_descriptor)) == 0) {
          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
          return x;
       }
   }

   /* find a blank spot */
   for (x = 0; x < TAB_SIZE; x++) {
       if (hash_descriptor[x].name == NULL) {
          XMEMCPY(&hash_descriptor[x], hash, sizeof(struct ltc_hash_descriptor));
          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
          return x;
       }
```

```
   }

   /* no spot */
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return -1;
}
```

5.154 misc/crypt/crypt_register_prng.c File Reference

5.154.1 Detailed Description

Register a PRNG, Tom St Denis.

Definition in file crypt_register_prng.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_register_prng.c:

Functions

- int register_prng (const struct ltc_prng_descriptor ∗prng)

  Register a PRNG with the descriptor table.

5.154.2 Function Documentation

5.154.2.1 int register_prng (const struct ltc_prng_descriptor ∗ *prng*)

Register a PRNG with the descriptor table.

Parameters:

prng The PRNG you wish to register

Returns:

value >= 0 if successfully added (or already present), -1 if unsuccessful

Definition at line 23 of file crypt_register_prng.c.

References LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, prng_descriptor, TAB_-SIZE, and XMEMCMP.

Referenced by crypt_fsa().

```
24 {
25    int x;
26
27    LTC_ARGCHK(prng != NULL);
28
29    /* is it already registered? */
30    LTC_MUTEX_LOCK(&ltc_prng_mutex);
31    for (x = 0; x < TAB_SIZE; x++) {
32        if (XMEMCMP(&prng_descriptor[x], prng, sizeof(struct ltc_prng_descriptor)) == 0) {
33           LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
34           return x;
35        }
36    }
37
38    /* find a blank spot */
39    for (x = 0; x < TAB_SIZE; x++) {
40        if (prng_descriptor[x].name == NULL) {
41           XMEMCPY(&prng_descriptor[x], prng, sizeof(struct ltc_prng_descriptor));
42           LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
43           return x;
44        }
```

```
   }

   /* no spot */
   LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
   return -1;
}
```

5.155 misc/crypt/crypt_unregister_cipher.c File Reference

5.155.1 Detailed Description

Unregister a cipher, Tom St Denis.

Definition in file crypt_unregister_cipher.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_unregister_cipher.c:

Functions

- int unregister_cipher (const struct ltc_cipher_descriptor ∗cipher)

  Unregister a cipher from the descriptor table.

5.155.2 Function Documentation

5.155.2.1 int unregister_cipher (const struct ltc_cipher_descriptor ∗ *cipher*)

Unregister a cipher from the descriptor table.

Parameters:

cipher The cipher descriptor to remove

Returns:

CRYPT_OK on success

Definition at line 23 of file crypt_unregister_cipher.c.

References cipher_descriptor, CRYPT_OK, ltc_cipher_descriptor::ID, LTC_ARGCHK, LTC_MUTEX_-LOCK, LTC_MUTEX_UNLOCK, ltc_prng_descriptor::name, TAB_SIZE, and XMEMCMP.

```
24 {
25    int x;
26 
27    LTC_ARGCHK(cipher != NULL);
28 
29    /* is it already registered? */
30    LTC_MUTEX_LOCK(&ltc_cipher_mutex);
31    for (x = 0; x < TAB_SIZE; x++) {
32        if (XMEMCMP(&cipher_descriptor[x], cipher, sizeof(struct ltc_cipher_descriptor)) == 0) {
33           cipher_descriptor[x].name = NULL;
34           cipher_descriptor[x].ID   = 255;
35           LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
36           return CRYPT_OK;
37        }
38    }
39    LTC_MUTEX_UNLOCK(&ltc_cipher_mutex);
40    return CRYPT_ERROR;
41 }
```

5.156 misc/crypt/crypt_unregister_hash.c File Reference

5.156.1 Detailed Description

Unregister a hash, Tom St Denis.

Definition in file crypt_unregister_hash.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_unregister_hash.c:

Functions

- int unregister_hash (const struct ltc_hash_descriptor ∗hash)

  Unregister a hash from the descriptor table.

5.156.2 Function Documentation

5.156.2.1 int unregister_hash (const struct ltc_hash_descriptor ∗ *hash*)

Unregister a hash from the descriptor table.

Parameters:

hash The hash descriptor to remove

Returns:

CRYPT_OK on success

Definition at line 23 of file crypt_unregister_hash.c.

References CRYPT_OK, hash_descriptor, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_-UNLOCK, ltc_prng_descriptor::name, TAB_SIZE, and XMEMCMP.

```
{
   int x;

   LTC_ARGCHK(hash != NULL);

   /* is it already registered? */
   LTC_MUTEX_LOCK(&ltc_hash_mutex);
   for (x = 0; x < TAB_SIZE; x++) {
       if (XMEMCMP(&hash_descriptor[x], hash, sizeof(struct ltc_hash_descriptor)) == 0) {
          hash_descriptor[x].name = NULL;
          LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
          return CRYPT_OK;
       }
   }
   LTC_MUTEX_UNLOCK(&ltc_hash_mutex);
   return CRYPT_ERROR;
}
```

5.157 misc/crypt/crypt_unregister_prng.c File Reference

5.157.1 Detailed Description

Unregister a PRNG, Tom St Denis.

Definition in file crypt_unregister_prng.c.

`#include "tomcrypt.h"`

Include dependency graph for crypt_unregister_prng.c:

Functions

- int unregister_prng (const struct ltc_prng_descriptor ∗prng)

  Unregister a PRNG from the descriptor table.

5.157.2 Function Documentation

5.157.2.1 int unregister_prng (const struct ltc_prng_descriptor ∗ *prng*)

Unregister a PRNG from the descriptor table.

Parameters:

prng The PRNG descriptor to remove

Returns:

CRYPT_OK on success

Definition at line 23 of file crypt_unregister_prng.c.

References CRYPT_OK, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, ltc_prng_-descriptor::name, prng_descriptor, TAB_SIZE, and XMEMCMP.

```
24 {
25    int x;
26
27    LTC_ARGCHK(prng != NULL);
28
29    /* is it already registered? */
30    LTC_MUTEX_LOCK(&ltc_prng_mutex);
31    for (x = 0; x < TAB_SIZE; x++) {
32        if (XMEMCMP(&prng_descriptor[x], prng, sizeof(struct ltc_prng_descriptor)) != 0) {
33           prng_descriptor[x].name = NULL;
34           LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
35           return CRYPT_OK;
36        }
37    }
38    LTC_MUTEX_UNLOCK(&ltc_prng_mutex);
39    return CRYPT_ERROR;
40 }
```

5.158 misc/error_to_string.c File Reference

5.158.1 Detailed Description

Convert error codes to ASCII strings, Tom St Denis.

Definition in file error_to_string.c.

```
#include "tomcrypt.h"
```

Include dependency graph for error_to_string.c:

Functions

- const char * error_to_string (int err)

  Convert an LTC error code to ASCII.

Variables

- static const char * err_2_str [ ]

5.158.2 Function Documentation

5.158.2.1 const char* error_to_string (int *err*)

Convert an LTC error code to ASCII.

Parameters:

err The error code

Returns:

A pointer to the ASCII NUL terminated string for the error or "Invalid error code." if the err code was not valid.

Definition at line 62 of file error_to_string.c.

References err_2_str.

Referenced by hmac_test().

```
{
   if (err < 0 || err >= (int)(sizeof(err_2_str)/sizeof(err_2_str[0]))) {
      return "Invalid error code.";
   } else {
      return err_2_str[err];
   }
}
```

5.158.3 Variable Documentation

5.158.3.1 const char* err_2_str[ ] `[static]`

Definition at line 19 of file error_to_string.c.

Referenced by error_to_string().

5.159 misc/pkcs5/pkcs_5_1.c File Reference

5.159.1 Detailed Description

PKCS #5, Algorithm #1, Tom St Denis.

Definition in file pkcs_5_1.c.

`#include <tomcrypt.h>`

Include dependency graph for pkcs_5_1.c:

Functions

- int pkcs_5_alg1 (const unsigned char ∗password, unsigned long password_len, const unsigned char ∗salt, int iteration_count, int hash_idx, unsigned char ∗out, unsigned long ∗outlen)

  Execute PKCS #5 v1.

5.159.2 Function Documentation

5.159.2.1 int pkcs_5_alg1 (const unsigned char ∗ *password*, unsigned long *password_len*, const unsigned char ∗ *salt*, int *iteration_count*, int *hash_idx*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Execute PKCS #5 v1.

Parameters:

password The password (or key)

password_len The length of the password (octet)

salt The salt (or nonce) which is 8 octets long

iteration_count The PKCS #5 v1 iteration count

hash_idx The index of the hash desired

out [out] The destination for this algorithm

outlen [in/out] The max size and resulting size of the algorithm output

Returns:

CRYPT_OK if successful

Definition at line 29 of file pkcs_5_1.c.

References CRYPT_MEM, CRYPT_OK, ltc_prng_descriptor::done, hash_descriptor, hash_is_valid(), hash_memory(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, MAXBLOCKSIZE, XFREE, and XMALLOC.

```
{
   int err;
   unsigned long x;
   hash_state    *md;
   unsigned char *buf;

   LTC_ARGCHK(password != NULL);
   LTC_ARGCHK(salt     != NULL);
```

```
   LTC_ARGCHK(out      != NULL);
   LTC_ARGCHK(outlen   != NULL);

   /* test hash IDX */
   if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
      return err;
   }

   /* allocate memory */
   md  = XMALLOC(sizeof(hash_state));
   buf = XMALLOC(MAXBLOCKSIZE);
   if (md == NULL || buf == NULL) {
      if (md != NULL) {
         XFREE(md);
      }
      if (buf != NULL) {
         XFREE(buf);
      }
      return CRYPT_MEM;
   }

   /* hash initial password + salt */
   if ((err = hash_descriptor[hash_idx].init(md)) != CRYPT_OK) {
       goto LBL_ERR;
   }
   if ((err = hash_descriptor[hash_idx].process(md, password, password_len)) != CRYPT_OK) {
       goto LBL_ERR;
   }
   if ((err = hash_descriptor[hash_idx].process(md, salt, 8)) != CRYPT_OK) {
       goto LBL_ERR;
   }
   if ((err = hash_descriptor[hash_idx].done(md, buf)) != CRYPT_OK) {
       goto LBL_ERR;
   }

   while (--iteration_count) {
      /* code goes here. */
      x = MAXBLOCKSIZE;
      if ((err = hash_memory(hash_idx, buf, hash_descriptor[hash_idx].hashsize, buf, &x)) != CRYPT_OK)
         goto LBL_ERR;
      }
   }

   /* copy upto outlen bytes */
   for (x = 0; x < hash_descriptor[hash_idx].hashsize && x < *outlen; x++) {
       out[x] = buf[x];
   }
   *outlen = x;
   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(buf, MAXBLOCKSIZE);
   zeromem(md, sizeof(hash_state));
#endif

   XFREE(buf);
   XFREE(md);

   return err;
}
```

Here is the call graph for this function:

5.160 misc/pkcs5/pkcs_5_2.c File Reference

5.160.1 Detailed Description

PKCS #5, Algorithm #2, Tom St Denis.

Definition in file pkcs_5_2.c.

`#include <tomcrypt.h>`

Include dependency graph for pkcs_5_2.c:

Functions

- int pkcs_5_alg2 (const unsigned char ∗password, unsigned long password_len, const unsigned char ∗salt, unsigned long salt_len, int iteration_count, int hash_idx, unsigned char ∗out, unsigned long ∗outlen)

    Execute PKCS #5 v2.

5.160.2 Function Documentation

5.160.2.1 int pkcs_5_alg2 (const unsigned char ∗ *password*, unsigned long *password_len*, const unsigned char ∗ *salt*, unsigned long *salt_len*, int *iteration_count*, int *hash_idx*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Execute PKCS #5 v2.

Parameters:

password The input password (or key)

password_len The length of the password (octets)

salt The salt (or nonce)

salt_len The length of the salt (octets)

iteration_count # of iterations desired for PKCS #5 v2 [read specs for more]

hash_idx The index of the hash desired

out [out] The destination for this algorithm

outlen [in/out] The max size and resulting size of the algorithm output

Returns:

CRYPT_OK if successful

Definition at line 31 of file pkcs_5_2.c.

References CRYPT_MEM, CRYPT_OK, hash_is_valid(), hmac_done(), hmac_init(), hmac_memory(), hmac_process(), LTC_ARGCHK, MAXBLOCKSIZE, XFREE, XMALLOC, XMEMCPY, and zeromem().

```
35 {
36    int err, itts;
37    ulong32  blkno;
38    unsigned long stored, left, x, y;
39    unsigned char *buf[2];
```

```
   hmac_state    *hmac;

   LTC_ARGCHK(password != NULL);
   LTC_ARGCHK(salt     != NULL);
   LTC_ARGCHK(out      != NULL);
   LTC_ARGCHK(outlen   != NULL);

   /* test hash IDX */
   if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
      return err;
   }

   buf[0] = XMALLOC(MAXBLOCKSIZE * 2);
   hmac   = XMALLOC(sizeof(hmac_state));
   if (hmac == NULL || buf[0] == NULL) {
      if (hmac != NULL) {
         XFREE(hmac);
      }
      if (buf[0] != NULL) {
         XFREE(buf[0]);
      }
      return CRYPT_MEM;
   }
   /* buf[1] points to the second block of MAXBLOCKSIZE bytes */
   buf[1] = buf[0] + MAXBLOCKSIZE;

   left   = *outlen;
   blkno  = 1;
   stored = 0;
   while (left != 0) {
       /* process block number blkno */
       zeromem(buf[0], MAXBLOCKSIZE*2);

       /* store current block number and increment for next pass */
       STORE32H(blkno, buf[1]);
       ++blkno;

       /* get PRF(P, S||int(blkno)) */
       if ((err = hmac_init(hmac, hash_idx, password, password_len)) != CRYPT_OK) {
          goto LBL_ERR;
       }
       if ((err = hmac_process(hmac, salt, salt_len)) != CRYPT_OK) {
          goto LBL_ERR;
       }
       if ((err = hmac_process(hmac, buf[1], 4)) != CRYPT_OK) {
          goto LBL_ERR;
       }
       x = MAXBLOCKSIZE;
       if ((err = hmac_done(hmac, buf[0], &x)) != CRYPT_OK) {
          goto LBL_ERR;
       }

       /* now compute repeated and XOR it in buf[1] */
       XMEMCPY(buf[1], buf[0], x);
       for (itts = 1; itts < iteration_count; ++itts) {
           if ((err = hmac_memory(hash_idx, password, password_len, buf[0], x, buf[0], &x)) != CRYPT_OK
              goto LBL_ERR;
           }
           for (y = 0; y < x; y++) {
               buf[1][y] ^= buf[0][y];
            }
        }

        /* now emit upto x bytes of buf[1] to output */
        for (y = 0; y < x && left != 0; ++y) {
            out[stored++] = buf[1][y];
            --left;
```

```
        }
    }
    *outlen = stored;

    err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
    zeromem(buf[0], MAXBLOCKSIZE*2);
    zeromem(hmac, sizeof(hmac_state));
#endif

    XFREE(hmac);
    XFREE(buf[0]);

    return err;
}
```

Here is the call graph for this function:

5.161 misc/zeromem.c File Reference

5.161.1 Detailed Description

Zero a block of memory, Tom St Denis.

Definition in file zeromem.c.

`#include "tomcrypt.h"`

Include dependency graph for zeromem.c:

Functions

- void zeromem (void ∗out, size_t outlen)

  Zero a block of memory.

5.161.2 Function Documentation

5.161.2.1 void zeromem (void ∗ *out*, size_t *outlen*)

Zero a block of memory.

Parameters:

out The destination of the area to zero

outlen The length of the area to zero (octets)

Definition at line 23 of file zeromem.c.

References LTC_ARGCHKVD.

Referenced by burn_stack(), cast5_setup(), chc_init(), dsa_shared_secret(), dsa_sign_hash_raw(), eax_decrypt_verify_memory(), eax_encrypt_authenticate_memory(), eax_init(), ECB_TEST(), ecc_ansi_x963_export(), ecc_shared_secret(), f8_encrypt(), f8_start(), f9_file(), fortuna_read(), gcm_add_aad(), gcm_gf_mult(), gcm_init(), gcm_reset(), hash_filehandle(), hash_memory(), hmac_file(), hmac_init(), hmac_memory(), lrw_start(), md2_done(), md2_init(), noekeon_test(), ocb_decrypt_verify_memory(), ocb_done_decrypt(), ocb_encrypt_authenticate_memory(), omac_done(), omac_file(), omac_init(), omac_memory(), pelican_init(), pkcs_1_i2osp(), pkcs_1_pss_encode(), pkcs_5_alg2(), pmac_file(), pmac_memory(), rc2_test(), rc4_read(), rng_make_prng(), rsa_exptmod(), rsa_verify_hash_ex(), sha224_done(), sha384_done(), sober128_read(), tiger_done(), whirlpool_init(), xcbc_file(), yarrow_read(), and yarrow_start().

```
{
   unsigned char *mem = out;
   LTC_ARGCHKVD(out != NULL);
   while (outlen-- > 0) {
      *mem++ = 0;
   }
}
```

5.162 modes/cbc/cbc_decrypt.c File Reference

5.162.1 Detailed Description

CBC implementation, encrypt block, Tom St Denis.

Definition in file cbc_decrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for cbc_decrypt.c:

Functions

- int cbc_decrypt (const unsigned char ∗ct, unsigned char ∗pt, unsigned long len, symmetric_CBC ∗cbc)

  CBC decrypt.

5.162.2 Function Documentation

5.162.2.1 int cbc_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, unsigned long *len*, symmetric_CBC ∗ *cbc*)

CBC decrypt.

Parameters:

ct Ciphertext

pt [out] Plaintext

len The number of bytes to process (must be multiple of block length)

cbc CBC state

Returns:

CRYPT_OK if successful

Definition at line 29 of file cbc_decrypt.c.

References ltc_cipher_descriptor::accel_cbc_decrypt, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_decrypt, and LTC_ARGCHK.

```
{
   int x, err;
   unsigned char tmp[16];
#ifdef LTC_FAST
   LTC_FAST_TYPE tmpy;
#else
   unsigned char tmpy;
#endif

   LTC_ARGCHK(pt  != NULL);
   LTC_ARGCHK(ct  != NULL);
   LTC_ARGCHK(cbc != NULL);

   if ((err = cipher_is_valid(cbc->cipher)) != CRYPT_OK) {
       return err;
```

```
   }

   /* is blocklen valid? */
   if (cbc->blocklen < 1 || cbc->blocklen > (int)sizeof(cbc->IV)) {
      return CRYPT_INVALID_ARG;
   }

   if (len % cbc->blocklen) {
      return CRYPT_INVALID_ARG;
   }
#ifdef LTC_FAST
   if (cbc->blocklen % sizeof(LTC_FAST_TYPE)) {
      return CRYPT_INVALID_ARG;
   }
#endif

   if (cipher_descriptor[cbc->cipher].accel_cbc_decrypt != NULL) {
      return cipher_descriptor[cbc->cipher].accel_cbc_decrypt(ct, pt, len / cbc->blocklen, cbc->IV, &cb
   } else {
      while (len) {
         /* decrypt */
         if ((err = cipher_descriptor[cbc->cipher].ecb_decrypt(ct, tmp, &cbc->key)) != CRYPT_OK) {
            return err;
         }

         /* xor IV against plaintext */
         #if defined(LTC_FAST)
         for (x = 0; x < cbc->blocklen; x += sizeof(LTC_FAST_TYPE)) {
             tmpy = *((LTC_FAST_TYPE*)((unsigned char *)cbc->IV + x)) ^ *((LTC_FAST_TYPE*)((unsigned ch
         *((LTC_FAST_TYPE*)((unsigned char *)cbc->IV + x)) = *((LTC_FAST_TYPE*)((unsigned char *)ct + x
         *((LTC_FAST_TYPE*)((unsigned char *)pt + x)) = tmpy;
         }
     #else
            for (x = 0; x < cbc->blocklen; x++) {
               tmpy       = tmp[x] ^ cbc->IV[x];
               cbc->IV[x] = ct[x];
               pt[x]      = tmpy;
            }
     #endif

         ct  += cbc->blocklen;
         pt  += cbc->blocklen;
         len -= cbc->blocklen;
      }
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.163 modes/cbc/cbc_done.c File Reference

5.163.1 Detailed Description

CBC implementation, finish chain, Tom St Denis.

Definition in file cbc_done.c.

`#include "tomcrypt.h"`

Include dependency graph for cbc_done.c:

Functions

- int cbc_done (symmetric_CBC ∗cbc)

  Terminate the chain.

5.163.2 Function Documentation

5.163.2.1 int cbc_done (symmetric_CBC ∗ *cbc*)

Terminate the chain.

Parameters:

cbc The CBC chain to terminate

Returns:

CRYPT_OK on success

Definition at line 24 of file cbc_done.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_OK, ltc_cipher_descriptor::done, and LTC_-ARGCHK.

```
{
   int err;
   LTC_ARGCHK(cbc != NULL);

   if ((err = cipher_is_valid(cbc->cipher)) != CRYPT_OK) {
      return err;
   }
   cipher_descriptor[cbc->cipher].done(&cbc->key);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.164 modes/cbc/cbc_encrypt.c File Reference

5.164.1 Detailed Description

CBC implementation, encrypt block, Tom St Denis.

Definition in file cbc_encrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for cbc_encrypt.c:

Functions

- int cbc_encrypt (const unsigned char ∗pt, unsigned char ∗ct, unsigned long len, symmetric_CBC ∗cbc)

  CBC encrypt.

5.164.2 Function Documentation

5.164.2.1 int cbc_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, unsigned long *len*, symmetric_CBC ∗ *cbc*)

CBC encrypt.

Parameters:

pt Plaintext

ct [out] Ciphertext

len The number of bytes to process (must be multiple of block length)

cbc CBC state

Returns:

CRYPT_OK if successful

Definition at line 29 of file cbc_encrypt.c.

References ltc_cipher_descriptor::accel_cbc_encrypt, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, and LTC_ARGCHK.

```
{
   int x, err;

   LTC_ARGCHK(pt != NULL);
   LTC_ARGCHK(ct != NULL);
   LTC_ARGCHK(cbc != NULL);

   if ((err = cipher_is_valid(cbc->cipher)) != CRYPT_OK) {
       return err;
   }

   /* is blocklen valid? */
   if (cbc->blocklen < 1 || cbc->blocklen > (int)sizeof(cbc->IV)) {
      return CRYPT_INVALID_ARG;
   }
```

```

   if (len % cbc->blocklen) {
      return CRYPT_INVALID_ARG;
   }
#ifdef LTC_FAST
   if (cbc->blocklen % sizeof(LTC_FAST_TYPE)) {
      return CRYPT_INVALID_ARG;
   }
#endif

   if (cipher_descriptor[cbc->cipher].accel_cbc_encrypt != NULL) {
      return cipher_descriptor[cbc->cipher].accel_cbc_encrypt(pt, ct, len / cbc->blocklen, cbc->IV, &cb
   } else {
      while (len) {
         /* xor IV against plaintext */
         #if defined(LTC_FAST)
         for (x = 0; x < cbc->blocklen; x += sizeof(LTC_FAST_TYPE)) {
             *((LTC_FAST_TYPE*)((unsigned char *)cbc->IV + x)) ^= *((LTC_FAST_TYPE*)((unsigned char *)p
         }
     #else
            for (x = 0; x < cbc->blocklen; x++) {
               cbc->IV[x] ^= pt[x];
            }
     #endif

         /* encrypt */
         if ((err = cipher_descriptor[cbc->cipher].ecb_encrypt(cbc->IV, ct, &cbc->key)) != CRYPT_OK) {
            return err;
         }

        /* store IV [ciphertext] for a future block */
         #if defined(LTC_FAST)
         for (x = 0; x < cbc->blocklen; x += sizeof(LTC_FAST_TYPE)) {
             *((LTC_FAST_TYPE*)((unsigned char *)cbc->IV + x)) = *((LTC_FAST_TYPE*)((unsigned char *)ct
         }
     #else
             for (x = 0; x < cbc->blocklen; x++) {
                cbc->IV[x] = ct[x];
             }
     #endif

        ct  += cbc->blocklen;
        pt  += cbc->blocklen;
        len -= cbc->blocklen;
     }
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.165 modes/cbc/cbc_getiv.c File Reference

5.165.1 Detailed Description

CBC implementation, get IV, Tom St Denis.

Definition in file cbc_getiv.c.

```
#include "tomcrypt.h"
```

Include dependency graph for cbc_getiv.c:

Functions

- int cbc_getiv (unsigned char ∗IV, unsigned long ∗len, symmetric_CBC ∗cbc)

  Get the current initial vector.

5.165.2 Function Documentation

5.165.2.1 int cbc_getiv (unsigned char ∗ *IV*, unsigned long ∗ *len*, symmetric_CBC ∗ *cbc*)

Get the current initial vector.

Parameters:

IV [out] The destination of the initial vector

len [in/out] The max size and resulting size of the initial vector

cbc The CBC state

Returns:

CRYPT_OK if successful

Definition at line 27 of file cbc_getiv.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, LTC_ARGCHK, and XMEMCPY.

```
{
   LTC_ARGCHK(IV  != NULL);
   LTC_ARGCHK(len != NULL);
   LTC_ARGCHK(cbc != NULL);
   if ((unsigned long)cbc->blocklen > *len) {
      *len = cbc->blocklen;
      return CRYPT_BUFFER_OVERFLOW;
   }
   XMEMCPY(IV, cbc->IV, cbc->blocklen);
   *len = cbc->blocklen;

   return CRYPT_OK;
}
```

5.166 modes/cbc/cbc_setiv.c File Reference

5.166.1 Detailed Description

CBC implementation, set IV, Tom St Denis.

Definition in file cbc_setiv.c.

`#include "tomcrypt.h"`

Include dependency graph for cbc_setiv.c:

Functions

- int cbc_setiv (const unsigned char ∗IV, unsigned long len, symmetric_CBC ∗cbc)

  Set an initial vector.

5.166.2 Function Documentation

5.166.2.1 int cbc_setiv (const unsigned char ∗ *IV*, unsigned long *len*, symmetric_CBC ∗ *cbc*)

Set an initial vector.

Parameters:

IV The initial vector

len The length of the vector (in octets)

cbc The CBC state

Returns:

CRYPT_OK if successful

Definition at line 28 of file cbc_setiv.c.

References CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, and XMEMCPY.

```
29 {
30    LTC_ARGCHK(IV  != NULL);
31    LTC_ARGCHK(cbc != NULL);
32    if (len != (unsigned long)cbc->blocklen) {
33       return CRYPT_INVALID_ARG;
34    }
35    XMEMCPY(cbc->IV, IV, len);
36    return CRYPT_OK;
37 }
```

5.167 modes/cbc/cbc_start.c File Reference

5.167.1 Detailed Description

CBC implementation, start chain, Tom St Denis.

Definition in file cbc_start.c.

`#include "tomcrypt.h"`

Include dependency graph for cbc_start.c:

Functions

- int cbc_start (int cipher, const unsigned char ∗IV, const unsigned char ∗key, int keylen, int num_rounds, symmetric_CBC ∗cbc)

  Initialize a CBC context.

5.167.2 Function Documentation

5.167.2.1 int cbc_start (int *cipher*, const unsigned char ∗ *IV*, const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_CBC ∗ *cbc*)

Initialize a CBC context.

Parameters:

cipher The index of the cipher desired

IV The initial vector

key The secret key

keylen The length of the secret key (octets)

num_rounds Number of rounds in the cipher desired (0 for default)

cbc The CBC state to initialize

Returns:

CRYPT_OK if successful

Definition at line 30 of file cbc_start.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_OK, and LTC_ARGCHK.

```
{
   int x, err;

   LTC_ARGCHK(IV != NULL);
   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(cbc != NULL);

   /* bad param? */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

```

```
   /* setup cipher */
   if ((err = cipher_descriptor[cipher].setup(key, keylen, num_rounds, &cbc->key)) != CRYPT_OK) {
      return err;
   }

   /* copy IV */
   cbc->blocklen = cipher_descriptor[cipher].block_length;
   cbc->cipher   = cipher;
   for (x = 0; x < cbc->blocklen; x++) {
       cbc->IV[x] = IV[x];
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.168 modes/cfb/cfb_decrypt.c File Reference

5.168.1 Detailed Description

CFB implementation, decrypt data, Tom St Denis.

Definition in file cfb_decrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for cfb_decrypt.c:

Functions

- int cfb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, unsigned long len, symmetric_CFB ∗cfb)

  CFB decrypt.

5.168.2 Function Documentation

5.168.2.1 int cfb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, unsigned long *len*, symmetric_CFB ∗ *cfb*)

CFB decrypt.

Parameters:

ct Ciphertext

pt [out] Plaintext

len Length of ciphertext (octets)

cfb CFB state

Returns:

CRYPT_OK if successful

Definition at line 28 of file cfb_decrypt.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_-descriptor::ecb_encrypt, and LTC_ARGCHK.

```
29 {
30    int err;
31
32    LTC_ARGCHK(pt != NULL);
33    LTC_ARGCHK(ct != NULL);
34    LTC_ARGCHK(cfb != NULL);
35
36    if ((err = cipher_is_valid(cfb->cipher)) != CRYPT_OK) {
37        return err;
38    }
39
40    /* is blocklen/padlen valid? */
41    if (cfb->blocklen < 0 || cfb->blocklen > (int)sizeof(cfb->IV) ||
42        cfb->padlen   < 0 || cfb->padlen   > (int)sizeof(cfb->pad)) {
43       return CRYPT_INVALID_ARG;
```

```
   }

   while (len-- > 0) {
       if (cfb->padlen == cfb->blocklen) {
          if ((err = cipher_descriptor[cfb->cipher].ecb_encrypt(cfb->pad, cfb->IV, &cfb->key)) != CRYPT
             return err;
          }
          cfb->padlen = 0;
       }
       cfb->pad[cfb->padlen] = *ct;
       *pt = *ct ^ cfb->IV[cfb->padlen];
       ++pt;
       ++ct;
       ++cfb->padlen;
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.169 modes/cfb/cfb_done.c File Reference

5.169.1 Detailed Description

CFB implementation, finish chain, Tom St Denis.

Definition in file cfb_done.c.

`#include "tomcrypt.h"`

Include dependency graph for cfb_done.c:

Functions

- int cfb_done (symmetric_CFB ∗cfb)

  Terminate the chain.

5.169.2 Function Documentation

5.169.2.1 int cfb_done (symmetric_CFB ∗ *cfb*)

Terminate the chain.

Parameters:

cfb The CFB chain to terminate

Returns:

CRYPT_OK on success

Definition at line 24 of file cfb_done.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_OK, ltc_cipher_descriptor::done, and LTC_-ARGCHK.

```
{
   int err;
   LTC_ARGCHK(cfb != NULL);

   if ((err = cipher_is_valid(cfb->cipher)) != CRYPT_OK) {
      return err;
   }
   cipher_descriptor[cfb->cipher].done(&cfb->key);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.170 modes/cfb/cfb_encrypt.c File Reference

5.170.1 Detailed Description

CFB implementation, encrypt data, Tom St Denis.

Definition in file cfb_encrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for cfb_encrypt.c:

Functions

- int cfb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, unsigned long len, symmetric_CFB ∗cfb)

  CFB encrypt.

5.170.2 Function Documentation

5.170.2.1 int cfb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, unsigned long *len*, symmetric_CFB ∗ *cfb*)

CFB encrypt.

Parameters:

pt Plaintext

ct [out] Ciphertext

len Length of plaintext (octets)

cfb CFB state

Returns:

CRYPT_OK if successful

Definition at line 28 of file cfb_encrypt.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, and LTC_ARGCHK.

```
29 {
30    int err;
31
32    LTC_ARGCHK(pt != NULL);
33    LTC_ARGCHK(ct != NULL);
34    LTC_ARGCHK(cfb != NULL);
35
36    if ((err = cipher_is_valid(cfb->cipher)) != CRYPT_OK) {
37        return err;
38    }
39
40    /* is blocklen/padlen valid? */
41    if (cfb->blocklen < 0 || cfb->blocklen > (int)sizeof(cfb->IV) ||
42        cfb->padlen   < 0 || cfb->padlen   > (int)sizeof(cfb->pad)) {
43       return CRYPT_INVALID_ARG;
```

```
   }

   while (len-- > 0) {
       if (cfb->padlen == cfb->blocklen) {
          if ((err = cipher_descriptor[cfb->cipher].ecb_encrypt(cfb->pad, cfb->IV, &cfb->key)) != CRYPT
             return err;
          }
          cfb->padlen = 0;
       }
       cfb->pad[cfb->padlen] = (*ct = *pt ^ cfb->IV[cfb->padlen]);
       ++pt;
       ++ct;
       ++cfb->padlen;
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.171 modes/cfb/cfb_getiv.c File Reference

5.171.1 Detailed Description

CFB implementation, get IV, Tom St Denis.

Definition in file cfb_getiv.c.

`#include "tomcrypt.h"`

Include dependency graph for cfb_getiv.c:

Functions

- int cfb_getiv (unsigned char ∗IV, unsigned long ∗len, symmetric_CFB ∗cfb)

  Get the current initial vector.

5.171.2 Function Documentation

5.171.2.1 int cfb_getiv (unsigned char ∗ *IV*, unsigned long ∗ *len*, symmetric_CFB ∗ *cfb*)

Get the current initial vector.

Parameters:

IV [out] The destination of the initial vector

len [in/out] The max size and resulting size of the initial vector

cfb The CFB state

Returns:

CRYPT_OK if successful

Definition at line 27 of file cfb_getiv.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, LTC_ARGCHK, and XMEMCPY.

```
28 {
29    LTC_ARGCHK(IV  != NULL);
30    LTC_ARGCHK(len != NULL);
31    LTC_ARGCHK(cfb != NULL);
32    if ((unsigned long)cfb->blocklen > *len) {
33       *len = cfb->blocklen;
34       return CRYPT_BUFFER_OVERFLOW;
35    }
36    XMEMCPY(IV, cfb->IV, cfb->blocklen);
37    *len = cfb->blocklen;
38 
39    return CRYPT_OK;
40 }
```

5.172 modes/cfb/cfb_setiv.c File Reference

5.172.1 Detailed Description

CFB implementation, set IV, Tom St Denis.

Definition in file cfb_setiv.c.

`#include "tomcrypt.h"`

Include dependency graph for cfb_setiv.c:

Functions

- int cfb_setiv (const unsigned char ∗IV, unsigned long len, symmetric_CFB ∗cfb)

    Set an initial vector.

5.172.2 Function Documentation

5.172.2.1 int cfb_setiv (const unsigned char ∗ *IV*, unsigned long *len*, symmetric_CFB ∗ *cfb*)

Set an initial vector.

Parameters:

IV The initial vector

len The length of the vector (in octets)

cfb The CFB state

Returns:

CRYPT_OK if successful

Definition at line 27 of file cfb_setiv.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_-descriptor::ecb_encrypt, and LTC_ARGCHK.

```
{
   int err;

   LTC_ARGCHK(IV  != NULL);
   LTC_ARGCHK(cfb != NULL);

   if ((err = cipher_is_valid(cfb->cipher)) != CRYPT_OK) {
       return err;
   }

   if (len != (unsigned long)cfb->blocklen) {
      return CRYPT_INVALID_ARG;
   }

   /* force next block */
   cfb->padlen = 0;
   return cipher_descriptor[cfb->cipher].ecb_encrypt(IV, cfb->IV, &cfb->key);
}
```

Here is the call graph for this function:

5.173 modes/cfb/cfb_start.c File Reference

5.173.1 Detailed Description

CFB implementation, start chain, Tom St Denis.

Definition in file cfb_start.c.

`#include "tomcrypt.h"`

Include dependency graph for cfb_start.c:

Functions

- int cfb_start (int cipher, const unsigned char ∗IV, const unsigned char ∗key, int keylen, int num_rounds, symmetric_CFB ∗cfb)

  Initialize a CFB context.

5.173.2 Function Documentation

5.173.2.1 int cfb_start (int *cipher*, const unsigned char ∗ *IV*, const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_CFB ∗ *cfb*)

Initialize a CFB context.

Parameters:

cipher The index of the cipher desired

IV The initial vector

key The secret key

keylen The length of the secret key (octets)

num_rounds Number of rounds in the cipher desired (0 for default)

cfb The CFB state to initialize

Returns:

CRYPT_OK if successful

Definition at line 31 of file cfb_start.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_OK, and LTC_ARGCHK.

```
33 {
34    int x, err;
35
36    LTC_ARGCHK(IV != NULL);
37    LTC_ARGCHK(key != NULL);
38    LTC_ARGCHK(cfb != NULL);
39
40    if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
41       return err;
42    }
43
44
```

```
   /* copy data */
   cfb->cipher = cipher;
   cfb->blocklen = cipher_descriptor[cipher].block_length;
   for (x = 0; x < cfb->blocklen; x++)
       cfb->IV[x] = IV[x];

   /* init the cipher */
   if ((err = cipher_descriptor[cipher].setup(key, keylen, num_rounds, &cfb->key)) != CRYPT_OK) {
      return err;
   }

   /* encrypt the IV */
   cfb->padlen = 0;
   return cipher_descriptor[cfb->cipher].ecb_encrypt(cfb->IV, cfb->IV, &cfb->key);
}
```

Here is the call graph for this function:

5.174 modes/ctr/ctr_decrypt.c File Reference

5.174.1 Detailed Description

CTR implementation, decrypt data, Tom St Denis.

Definition in file ctr_decrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for ctr_decrypt.c:

Functions

- int ctr_decrypt (const unsigned char ∗ct, unsigned char ∗pt, unsigned long len, symmetric_CTR ∗ctr)

  CTR decrypt.

5.174.2 Function Documentation

5.174.2.1 int ctr_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, unsigned long *len*, symmetric_CTR ∗ *ctr*)

CTR decrypt.

Parameters:

ct Ciphertext

pt [out] Plaintext

len Length of ciphertext (octets)

ctr CTR state

Returns:

CRYPT_OK if successful

Definition at line 28 of file ctr_decrypt.c.

References ctr_encrypt(), and LTC_ARGCHK.

Referenced by eax_decrypt().

```
{
   LTC_ARGCHK(pt != NULL);
   LTC_ARGCHK(ct != NULL);
   LTC_ARGCHK(ctr != NULL);

   return ctr_encrypt(ct, pt, len, ctr);
}
```

Here is the call graph for this function:

5.175 modes/ctr/ctr_done.c File Reference

5.175.1 Detailed Description

CTR implementation, finish chain, Tom St Denis.

Definition in file ctr_done.c.

`#include "tomcrypt.h"`

Include dependency graph for ctr_done.c:

Functions

- int ctr_done (symmetric_CTR ∗ctr)

  Terminate the chain.

5.175.2 Function Documentation

5.175.2.1 int ctr_done (symmetric_CTR ∗ *ctr*)

Terminate the chain.

Parameters:

ctr The CTR chain to terminate

Returns:

CRYPT_OK on success

Definition at line 24 of file ctr_done.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_OK, ltc_cipher_descriptor::done, and LTC_-ARGCHK.

Referenced by eax_done(), and yarrow_done().

```
{
   int err;
   LTC_ARGCHK(ctr != NULL);

   if ((err = cipher_is_valid(ctr->cipher)) != CRYPT_OK) {
      return err;
   }
   cipher_descriptor[ctr->cipher].done(&ctr->key);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.176 modes/ctr/ctr_encrypt.c File Reference

5.176.1 Detailed Description

CTR implementation, encrypt data, Tom St Denis.

Definition in file ctr_encrypt.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ctr_encrypt.c:

Functions

- int ctr_encrypt (const unsigned char ∗pt, unsigned char ∗ct, unsigned long len, symmetric_CTR ∗ctr)

  CTR encrypt.

5.176.2 Function Documentation

5.176.2.1 int ctr_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, unsigned long *len*, symmetric_CTR ∗ *ctr*)

CTR encrypt.

Parameters:

pt Plaintext

ct [out] Ciphertext

len Length of plaintext (octets)

ctr CTR state

Returns:

CRYPT_OK if successful

Definition at line 29 of file ctr_encrypt.c.

References ltc_cipher_descriptor::accel_ctr_encrypt, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, and LTC_ARGCHK.

Referenced by ctr_decrypt(), eax_encrypt(), and yarrow_read().

```
30 {
31    int x, err;
32
33    LTC_ARGCHK(pt != NULL);
34    LTC_ARGCHK(ct != NULL);
35    LTC_ARGCHK(ctr != NULL);
36
37    if ((err = cipher_is_valid(ctr->cipher)) != CRYPT_OK) {
38        return err;
39    }
40
41    /* is blocklen/padlen valid? */
42    if (ctr->blocklen < 1 || ctr->blocklen > (int)sizeof(ctr->ctr) ||
```

```
       ctr->padlen   < 0 || ctr->padlen   > (int)sizeof(ctr->pad)) {
      return CRYPT_INVALID_ARG;
   }

#ifdef LTC_FAST
   if (ctr->blocklen % sizeof(LTC_FAST_TYPE)) {
      return CRYPT_INVALID_ARG;
   }
#endif

   /* handle acceleration only if pad is empty, accelerator is present and length is >= a block size */
   if ((ctr->padlen == ctr->blocklen) && cipher_descriptor[ctr->cipher].accel_ctr_encrypt != NULL && (l
      if ((err = cipher_descriptor[ctr->cipher].accel_ctr_encrypt(pt, ct, len/ctr->blocklen, ctr->ctr,
         return err;
      }
      len %= ctr->blocklen;
   }

   while (len) {
      /* is the pad empty? */
      if (ctr->padlen == ctr->blocklen) {
         /* increment counter */
         if (ctr->mode == CTR_COUNTER_LITTLE_ENDIAN) {
            /* little-endian */
            for (x = 0; x < ctr->blocklen; x++) {
               ctr->ctr[x] = (ctr->ctr[x] + (unsigned char)1) & (unsigned char)255;
               if (ctr->ctr[x] != (unsigned char)0) {
                  break;
               }
            }
         } else {
            /* big-endian */
            for (x = ctr->blocklen-1; x >= 0; x--) {
               ctr->ctr[x] = (ctr->ctr[x] + (unsigned char)1) & (unsigned char)255;
               if (ctr->ctr[x] != (unsigned char)0) {
                  break;
               }
            }
         }

         /* encrypt it */
         if ((err = cipher_descriptor[ctr->cipher].ecb_encrypt(ctr->ctr, ctr->pad, &ctr->key)) != CRYPT
            return err;
         }
         ctr->padlen = 0;
      }
#ifdef LTC_FAST
      if (ctr->padlen == 0 && len >= (unsigned long)ctr->blocklen) {
         for (x = 0; x < ctr->blocklen; x += sizeof(LTC_FAST_TYPE)) {
            *((LTC_FAST_TYPE*)((unsigned char *)ct + x)) = *((LTC_FAST_TYPE*)((unsigned char *)pt + x))
                                                           *((LTC_FAST_TYPE*)((unsigned char *)ctr->pad
         }
        pt         += ctr->blocklen;
        ct         += ctr->blocklen;
        len        -= ctr->blocklen;
        ctr->padlen = ctr->blocklen;
        continue;
       }
#endif
      *ct++ = *pt++ ^ ctr->pad[ctr->padlen++];
      --len;
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.177 modes/ctr/ctr_getiv.c File Reference

5.177.1 Detailed Description

CTR implementation, get IV, Tom St Denis.

Definition in file ctr_getiv.c.

`#include "tomcrypt.h"`

Include dependency graph for ctr_getiv.c:

Functions

- int ctr_getiv (unsigned char ∗IV, unsigned long ∗len, symmetric_CTR ∗ctr)

  Get the current initial vector.

5.177.2 Function Documentation

5.177.2.1 int ctr_getiv (unsigned char ∗ *IV*, unsigned long ∗ *len*, symmetric_CTR ∗ *ctr*)

Get the current initial vector.

Parameters:

IV [out] The destination of the initial vector

len [in/out] The max size and resulting size of the initial vector

ctr The CTR state

Returns:

CRYPT_OK if successful

Definition at line 27 of file ctr_getiv.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, LTC_ARGCHK, and XMEMCPY.

```
{
   LTC_ARGCHK(IV  != NULL);
   LTC_ARGCHK(len != NULL);
   LTC_ARGCHK(ctr != NULL);
   if ((unsigned long)ctr->blocklen > *len) {
      *len = ctr->blocklen;
      return CRYPT_BUFFER_OVERFLOW;
   }
   XMEMCPY(IV, ctr->ctr, ctr->blocklen);
   *len = ctr->blocklen;

   return CRYPT_OK;
}
```

5.178 modes/ctr/ctr_setiv.c File Reference

5.178.1 Detailed Description

CTR implementation, set IV, Tom St Denis.

Definition in file ctr_setiv.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ctr_setiv.c:

Functions

- int ctr_setiv (const unsigned char ∗IV, unsigned long len, symmetric_CTR ∗ctr)
  Set an initial vector.

5.178.2 Function Documentation

5.178.2.1 int ctr_setiv (const unsigned char ∗ *IV*, unsigned long *len*, symmetric_CTR ∗ *ctr*)

Set an initial vector.

Parameters:

IV The initial vector

len The length of the vector (in octets)

ctr The CTR state

Returns:

CRYPT_OK if successful

Definition at line 27 of file ctr_setiv.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_-descriptor::ecb_encrypt, LTC_ARGCHK, and XMEMCPY.

```
28 {
29    int err;
30
31    LTC_ARGCHK(IV  != NULL);
32    LTC_ARGCHK(ctr != NULL);
33
34    /* bad param? */
35    if ((err = cipher_is_valid(ctr->cipher)) != CRYPT_OK) {
36       return err;
37    }
38
39    if (len != (unsigned long)ctr->blocklen) {
40       return CRYPT_INVALID_ARG;
41    }
42
43    /* set IV */
44    XMEMCPY(ctr->ctr, IV, len);
45
46    /* force next block */
47    ctr->padlen = 0;
```

```
   return cipher_descriptor[ctr->cipher].ecb_encrypt(IV, ctr->pad, &ctr->key);
}
```

Here is the call graph for this function:

5.179 modes/ctr/ctr_start.c File Reference

5.179.1 Detailed Description

CTR implementation, start chain, Tom St Denis.

Definition in file ctr_start.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ctr_start.c:

Functions

- int ctr_start (int cipher, const unsigned char ∗IV, const unsigned char ∗key, int keylen, int num_-rounds, int ctr_mode, symmetric_CTR ∗ctr)

  Initialize a CTR context.

5.179.2 Function Documentation

5.179.2.1 int ctr_start (int *cipher*, const unsigned char ∗ *IV*, const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, int *ctr_mode*, symmetric_CTR ∗ *ctr*)

Initialize a CTR context.

Parameters:

cipher The index of the cipher desired

IV The initial vector

key The secret key

keylen The length of the secret key (octets)

num_rounds Number of rounds in the cipher desired (0 for default)

ctr_mode The counter mode (CTR_COUNTER_LITTLE_ENDIAN or CTR_COUNTER_BIG_-ENDIAN)

ctr The CTR state to initialize

Returns:

CRYPT_OK if successful

Definition at line 32 of file ctr_start.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_OK, and LTC_ARGCHK.

Referenced by ctr_test(), eax_init(), and yarrow_ready().

```
{
   int x, err;

   LTC_ARGCHK(IV  != NULL);
   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(ctr != NULL);

```

```
   /* bad param? */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

   /* setup cipher */
   if ((err = cipher_descriptor[cipher].setup(key, keylen, num_rounds, &ctr->key)) != CRYPT_OK) {
      return err;
   }

   /* copy ctr */
   ctr->blocklen = cipher_descriptor[cipher].block_length;
   ctr->cipher   = cipher;
   ctr->padlen   = 0;
   ctr->mode     = ctr_mode & 1;
   for (x = 0; x < ctr->blocklen; x++) {
       ctr->ctr[x] = IV[x];
   }

   if (ctr_mode & LTC_CTR_RFC3686) {
      /* increment the IV as per RFC 3686 */
      if (ctr->mode == CTR_COUNTER_LITTLE_ENDIAN) {
         /* little-endian */
         for (x = 0; x < ctr->blocklen; x++) {
             ctr->ctr[x] = (ctr->ctr[x] + (unsigned char)1) & (unsigned char)255;
             if (ctr->ctr[x] != (unsigned char)0) {
                break;
             }
         }
      } else {
         /* big-endian */
         for (x = ctr->blocklen-1; x >= 0; x--) {
             ctr->ctr[x] = (ctr->ctr[x] + (unsigned char)1) & (unsigned char)255;
             if (ctr->ctr[x] != (unsigned char)0) {
                break;
             }
         }
      }
   }

   return cipher_descriptor[ctr->cipher].ecb_encrypt(ctr->ctr, ctr->pad, &ctr->key);
}
```

Here is the call graph for this function:

5.180 modes/ctr/ctr_test.c File Reference

5.180.1 Detailed Description

CTR implementation, Tests again RFC 3686, Tom St Denis.

Definition in file ctr_test.c.

`#include "tomcrypt.h"`

Include dependency graph for ctr_test.c:

Functions

- int ctr_test (void)

5.180.2 Function Documentation

5.180.2.1 int ctr_test (void)

Definition at line 20 of file ctr_test.c.

References CRYPT_NOP, CRYPT_OK, ctr_start(), and find_cipher().

```
21 {
22 #ifdef LTC_NO_TEST
23    return CRYPT_NOP;
24 #else
25    static const struct {
26       int keylen, msglen;
27       unsigned char key[32], IV[16], pt[64], ct[64];
28    } tests[] = {
29 /* 128-bit key, 16-byte pt */
30 {
31    16, 16,
32    {0xAE,0x68,0x52,0xF8,0x12,0x10,0x67,0xCC,0x4B,0xF7,0xA5,0x76,0x55,0x77,0xF3,0x9E },
33    {0x00,0x00,0x00,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00 },
34    {0x53,0x69,0x6E,0x67,0x6C,0x65,0x20,0x62,0x6C,0x6F,0x63,0x6B,0x20,0x6D,0x73,0x67 },
35    {0xE4,0x09,0x5D,0x4F,0xB7,0xA7,0xB3,0x79,0x2D,0x61,0x75,0xA3,0x26,0x13,0x11,0xB8 },
36 },
37
38 /* 128-bit key, 36-byte pt */
39 {
40    16, 36,
41    {0x76,0x91,0xBE,0x03,0x5E,0x50,0x20,0xA8,0xAC,0x6E,0x61,0x85,0x29,0xF9,0xA0,0xDC },
42    {0x00,0xE0,0x01,0x7B,0x27,0x77,0x7F,0x3F,0x4A,0x17,0x86,0xF0,0x00,0x00,0x00,0x00 },
43    {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,
44     0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F,
45     0x20,0x21,0x22,0x23},
46    {0xC1,0xCF,0x48,0xA8,0x9F,0x2F,0xFD,0xD9,0xCF,0x46,0x52,0xE9,0xEF,0xDB,0x72,0xD7,
47     0x45,0x40,0xA4,0x2B,0xDE,0x6D,0x78,0x36,0xD5,0x9A,0x5C,0xEA,0xAE,0xF3,0x10,0x53,
48     0x25,0xB2,0x07,0x2F },
49 },
50 };
51   int idx, err, x;
52   unsigned char buf[64];
53   symmetric_CTR ctr;
54
55   /* AES can be under rijndael or aes... try to find it */
56   if ((idx = find_cipher("aes")) == -1) {
57      if ((idx = find_cipher("rijndael")) == -1) {
```

```
      return CRYPT_NOP;
    }
  }

  for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
    if ((err = ctr_start(idx, tests[x].IV, tests[x].key, tests[x].keylen, 0, CTR_COUNTER_BIG_ENDIAN|LT
      return err;
    }
    if ((err = ctr_encrypt(tests[x].pt, buf, tests[x].msglen, &ctr)) != CRYPT_OK) {
      return err;
    }
    ctr_done(&ctr);
    if (XMEMCMP(buf, tests[x].ct, tests[x].msglen)) {
      return CRYPT_FAIL_TESTVECTOR;
    }
  }
  return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.181 modes/ecb/ecb_decrypt.c File Reference

5.181.1 Detailed Description

ECB implementation, decrypt a block, Tom St Denis.

Definition in file ecb_decrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for ecb_decrypt.c:

Functions

- int ecb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, unsigned long len, symmetric_ECB ∗ecb)

  ECB decrypt.

5.181.2 Function Documentation

5.181.2.1 int ecb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, unsigned long *len*, symmetric_ECB ∗ *ecb*)

ECB decrypt.

Parameters:

ct Ciphertext

pt [out] Plaintext

len The number of octets to process (must be multiple of the cipher block size)

ecb ECB state

Returns:

CRYPT_OK if successful

Definition at line 28 of file ecb_decrypt.c.

References ltc_cipher_descriptor::accel_ecb_decrypt, ltc_cipher_descriptor::block_length, cipher_-descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_decrypt, and LTC_ARGCHK.

```
{
   int err;
   LTC_ARGCHK(pt != NULL);
   LTC_ARGCHK(ct != NULL);
   LTC_ARGCHK(ecb != NULL);
   if ((err = cipher_is_valid(ecb->cipher)) != CRYPT_OK) {
       return err;
   }
   if (len % cipher_descriptor[ecb->cipher].block_length) {
      return CRYPT_INVALID_ARG;
   }

   /* check for accel */
   if (cipher_descriptor[ecb->cipher].accel_ecb_decrypt != NULL) {
```

```
      return cipher_descriptor[ecb->cipher].accel_ecb_decrypt(ct, pt, len / cipher_descriptor[ecb->ciph
   } else {
      while (len) {
         if ((err = cipher_descriptor[ecb->cipher].ecb_decrypt(ct, pt, &ecb->key)) != CRYPT_OK) {
            return err;
         }
         pt  += cipher_descriptor[ecb->cipher].block_length;
         ct  += cipher_descriptor[ecb->cipher].block_length;
         len -= cipher_descriptor[ecb->cipher].block_length;
      }
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.182 modes/ecb/ecb_done.c File Reference

5.182.1 Detailed Description

ECB implementation, finish chain, Tom St Denis.

Definition in file ecb_done.c.

`#include "tomcrypt.h"`

Include dependency graph for ecb_done.c:

Functions

- int ecb_done (symmetric_ECB ∗ecb)

  Terminate the chain.

5.182.2 Function Documentation

5.182.2.1 int ecb_done (symmetric_ECB ∗ *ecb*)

Terminate the chain.

Parameters:

ecb The ECB chain to terminate

Returns:

CRYPT_OK on success

Definition at line 24 of file ecb_done.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_OK, ltc_cipher_descriptor::done, and LTC_-ARGCHK.

```
{
   int err;
   LTC_ARGCHK(ecb != NULL);

   if ((err = cipher_is_valid(ecb->cipher)) != CRYPT_OK) {
      return err;
   }
   cipher_descriptor[ecb->cipher].done(&ecb->key);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.183 modes/ecb/ecb_encrypt.c File Reference

5.183.1 Detailed Description

ECB implementation, encrypt a block, Tom St Denis.

Definition in file ecb_encrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for ecb_encrypt.c:

Functions

- int ecb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, unsigned long len, symmetric_ECB ∗ecb)

  ECB encrypt.

5.183.2 Function Documentation

5.183.2.1 int ecb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, unsigned long *len*, symmetric_ECB ∗ *ecb*)

ECB encrypt.

Parameters:

pt Plaintext

ct [out] Ciphertext

len The number of octets to process (must be multiple of the cipher block size)

ecb ECB state

Returns:

CRYPT_OK if successful

Definition at line 28 of file ecb_encrypt.c.

References ltc_cipher_descriptor::accel_ecb_encrypt, ltc_cipher_descriptor::block_length, cipher_-descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, and LTC_ARGCHK.

Referenced by gcm_init(), and omac_init().

```
29 {
30    int err;
31    LTC_ARGCHK(pt != NULL);
32    LTC_ARGCHK(ct != NULL);
33    LTC_ARGCHK(ecb != NULL);
34    if ((err = cipher_is_valid(ecb->cipher)) != CRYPT_OK) {
35        return err;
36    }
37    if (len % cipher_descriptor[ecb->cipher].block_length) {
38       return CRYPT_INVALID_ARG;
39    }
40
```

```
   /* check for accel */
   if (cipher_descriptor[ecb->cipher].accel_ecb_encrypt != NULL) {
      return cipher_descriptor[ecb->cipher].accel_ecb_encrypt(pt, ct, len / cipher_descriptor[ecb->ciph
   } else {
      while (len) {
         if ((err = cipher_descriptor[ecb->cipher].ecb_encrypt(pt, ct, &ecb->key)) != CRYPT_OK) {
            return err;
         }
         pt  += cipher_descriptor[ecb->cipher].block_length;
         ct  += cipher_descriptor[ecb->cipher].block_length;
         len -= cipher_descriptor[ecb->cipher].block_length;
      }
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.184 modes/ecb/ecb_start.c File Reference

5.184.1 Detailed Description

ECB implementation, start chain, Tom St Denis.

Definition in file ecb_start.c.

`#include "tomcrypt.h"`

Include dependency graph for ecb_start.c:

Functions

- int ecb_start (int cipher, const unsigned char ∗key, int keylen, int num_rounds, symmetric_ECB ∗ecb)

    Initialize a ECB context.

5.184.2 Function Documentation

5.184.2.1 int ecb_start (int *cipher*, const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_ECB ∗ *ecb*)

Initialize a ECB context.

Parameters:

cipher The index of the cipher desired

key The secret key

keylen The length of the secret key (octets)

num_rounds Number of rounds in the cipher desired (0 for default)

ecb The ECB state to initialize

Returns:

CRYPT_OK if successful

Definition at line 30 of file ecb_start.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_OK, LTC_-ARGCHK, and ltc_cipher_descriptor::setup.

```
{
   int err;
   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(ecb != NULL);

   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }
   ecb->cipher = cipher;
   ecb->blocklen = cipher_descriptor[cipher].block_length;
   return cipher_descriptor[cipher].setup(key, keylen, num_rounds, &ecb->key);
}
```

Here is the call graph for this function:

5.185 modes/f8/f8_decrypt.c File Reference

5.185.1 Detailed Description

F8 implementation, decrypt data, Tom St Denis.

Definition in file f8_decrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for f8_decrypt.c:

Functions

- int f8_decrypt (const unsigned char ∗ct, unsigned char ∗pt, unsigned long len, symmetric_F8 ∗f8)
  F8 decrypt.

5.185.2 Function Documentation

5.185.2.1 int f8_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, unsigned long *len*, symmetric_F8 ∗ *f8*)

F8 decrypt.

Parameters:

ct Ciphertext

pt [out] Plaintext

len Length of ciphertext (octets)

f8 F8 state

Returns:

CRYPT_OK if successful

Definition at line 28 of file f8_decrypt.c.

References f8_encrypt(), and LTC_ARGCHK.

```
{
   LTC_ARGCHK(pt != NULL);
   LTC_ARGCHK(ct != NULL);
   LTC_ARGCHK(f8 != NULL);
   return f8_encrypt(ct, pt, len, f8);
}
```

Here is the call graph for this function:

5.186 modes/f8/f8_done.c File Reference

5.186.1 Detailed Description

F8 implementation, finish chain, Tom St Denis.

Definition in file f8_done.c.

`#include "tomcrypt.h"`

Include dependency graph for f8_done.c:

Functions

- int f8_done (symmetric_F8 ∗f8)

  Terminate the chain.

5.186.2 Function Documentation

5.186.2.1 int f8_done (symmetric_F8 ∗ *f8*)

Terminate the chain.

Parameters:

f8 The F8 chain to terminate

Returns:

CRYPT_OK on success

Definition at line 24 of file f8_done.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_OK, ltc_cipher_descriptor::done, and LTC_-ARGCHK.

Referenced by f8_test_mode().

```
{
   int err;
   LTC_ARGCHK(f8 != NULL);

   if ((err = cipher_is_valid(f8->cipher)) != CRYPT_OK) {
      return err;
   }
   cipher_descriptor[f8->cipher].done(&f8->key);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.187 modes/f8/f8_encrypt.c File Reference

5.187.1 Detailed Description

F8 implementation, encrypt data, Tom St Denis.

Definition in file f8_encrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for f8_encrypt.c:

Functions

- int f8_encrypt (const unsigned char *pt, unsigned char *ct, unsigned long len, symmetric_F8 *f8)

  F8 encrypt.

5.187.2 Function Documentation

5.187.2.1 int f8_encrypt (const unsigned char * *pt*, unsigned char * *ct*, unsigned long *len*, symmetric_F8 * *f8*)

F8 encrypt.

Parameters:

pt Plaintext

ct [out] Ciphertext

len Length of plaintext (octets)

f8 F8 state

Returns:

CRYPT_OK if successful

Definition at line 28 of file f8_encrypt.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_descriptor::ecb_encrypt, LTC_ARGCHK, MAXBLOCKSIZE, and zeromem().

Referenced by f8_decrypt(), and f8_test_mode().

```
{
   int           err, x;
   unsigned char buf[MAXBLOCKSIZE];
   LTC_ARGCHK(pt != NULL);
   LTC_ARGCHK(ct != NULL);
   LTC_ARGCHK(f8 != NULL);
   if ((err = cipher_is_valid(f8->cipher)) != CRYPT_OK) {
       return err;
   }

   /* is blocklen/padlen valid? */
   if (f8->blocklen < 0 || f8->blocklen > (int)sizeof(f8->IV) ||
       f8->padlen   < 0 || f8->padlen   > (int)sizeof(f8->IV)) {
      return CRYPT_INVALID_ARG;
```

```
   }

   zeromem(buf, sizeof(buf));

   /* make sure the pad is empty */
   if (f8->padlen == f8->blocklen) {
      /* xor of IV, MIV and blockcnt == what goes into cipher */
      STORE32H(f8->blockcnt, (buf+(f8->blocklen-4)));
      ++(f8->blockcnt);
      for (x = 0; x < f8->blocklen; x++) {
          f8->IV[x] ^= f8->MIV[x] ^ buf[x];
      }
      if ((err = cipher_descriptor[f8->cipher].ecb_encrypt(f8->IV, f8->IV, &f8->key)) != CRYPT_OK) {
         return err;
      }
      f8->padlen = 0;
   }

#ifdef LTC_FAST
   if (f8->padlen == 0) {
      while (len >= (unsigned long)f8->blocklen) {
         STORE32H(f8->blockcnt, (buf+(f8->blocklen-4)));
         ++(f8->blockcnt);
         for (x = 0; x < f8->blocklen; x += sizeof(LTC_FAST_TYPE)) {
             *((LTC_FAST_TYPE*)(&ct[x])) = *((LTC_FAST_TYPE*)(&pt[x])) ^ *((LTC_FAST_TYPE*)(&f8->IV[x]))
             *((LTC_FAST_TYPE*)(&f8->IV[x])) ^= *((LTC_FAST_TYPE*)(&f8->MIV[x])) ^ *((LTC_FAST_TYPE*)(&
         }
         if ((err = cipher_descriptor[f8->cipher].ecb_encrypt(f8->IV, f8->IV, &f8->key)) != CRYPT_OK) {
            return err;
         }
         len -= x;
         pt  += x;
         ct  += x;
      }
   }
#endif

   while (len > 0) {
       if (f8->padlen == f8->blocklen) {
          /* xor of IV, MIV and blockcnt == what goes into cipher */
          STORE32H(f8->blockcnt, (buf+(f8->blocklen-4)));
          ++(f8->blockcnt);
          for (x = 0; x < f8->blocklen; x++) {
              f8->IV[x] ^= f8->MIV[x] ^ buf[x];
          }
          if ((err = cipher_descriptor[f8->cipher].ecb_encrypt(f8->IV, f8->IV, &f8->key)) != CRYPT_OK)
             return err;
          }
          f8->padlen = 0;
       }
       *ct++ = *pt++ ^ f8->IV[f8->padlen++];
       --len;
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.188 modes/f8/f8_getiv.c File Reference

`#include "tomcrypt.h"`

Include dependency graph for f8_getiv.c:

Functions

- int f8_getiv (unsigned char ∗IV, unsigned long ∗len, symmetric_F8 ∗f8)

  Get the current initial vector.

5.188.1 Function Documentation

5.188.1.1 int f8_getiv (unsigned char ∗ *IV*, unsigned long ∗ *len*, symmetric_F8 ∗ *f8*)

Get the current initial vector.

Parameters:

IV [out] The destination of the initial vector

len [in/out] The max size and resulting size of the initial vector

f8 The F8 state

Returns:

CRYPT_OK if successful

Definition at line 27 of file f8_getiv.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, LTC_ARGCHK, and XMEMCPY.

```
{
   LTC_ARGCHK(IV  != NULL);
   LTC_ARGCHK(len != NULL);
   LTC_ARGCHK(f8  != NULL);
   if ((unsigned long)f8->blocklen > *len) {
      *len = f8->blocklen;
      return CRYPT_BUFFER_OVERFLOW;
   }
   XMEMCPY(IV, f8->IV, f8->blocklen);
   *len = f8->blocklen;

   return CRYPT_OK;
}
```

5.189 modes/f8/f8_setiv.c File Reference

5.189.1 Detailed Description

F8 implementation, set IV, Tom St Denis.

Definition in file f8_setiv.c.

`#include "tomcrypt.h"`

Include dependency graph for f8_setiv.c:

Functions

- int f8_setiv (const unsigned char ∗IV, unsigned long len, symmetric_F8 ∗f8)

  Set an initial vector.

5.189.2 Function Documentation

5.189.2.1 int f8_setiv (const unsigned char ∗ *IV*, unsigned long *len*, symmetric_F8 ∗ *f8*)

Set an initial vector.

Parameters:

IV The initial vector

len The length of the vector (in octets)

f8 The F8 state

Returns:

CRYPT_OK if successful

Definition at line 27 of file f8_setiv.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_-descriptor::ecb_encrypt, and LTC_ARGCHK.

```
{
   int err;

   LTC_ARGCHK(IV != NULL);
   LTC_ARGCHK(f8 != NULL);

   if ((err = cipher_is_valid(f8->cipher)) != CRYPT_OK) {
       return err;
   }

   if (len != (unsigned long)f8->blocklen) {
      return CRYPT_INVALID_ARG;
   }

   /* force next block */
   f8->padlen = 0;
   return cipher_descriptor[f8->cipher].ecb_encrypt(IV, f8->IV, &f8->key);
}
```

Here is the call graph for this function:

5.190 modes/f8/f8_start.c File Reference

5.190.1 Detailed Description

F8 implementation, start chain, Tom St Denis.

Definition in file f8_start.c.

`#include "tomcrypt.h"`

Include dependency graph for f8_start.c:

Functions

- int f8_start (int cipher, const unsigned char ∗IV, const unsigned char ∗key, int keylen, const unsigned char ∗salt_key, int skeylen, int num_rounds, symmetric_F8 ∗f8)

  Initialize an F8 context.

5.190.2 Function Documentation

5.190.2.1 int f8_start (int *cipher*, const unsigned char ∗ *IV*, const unsigned char ∗ *key*, int *keylen*, const unsigned char ∗ *salt_key*, int *skeylen*, int *num_rounds*, symmetric_F8 ∗ *f8*)

Initialize an F8 context.

Parameters:

cipher The index of the cipher desired

IV The initial vector

key The secret key

keylen The length of the secret key (octets)

salt_key The salting key for the IV

skeylen The length of the salting key (octets)

num_rounds Number of rounds in the cipher desired (0 for default)

f8 The F8 state to initialize

Returns:

CRYPT_OK if successful

Definition at line 33 of file f8_start.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_-INVALID_ARG, CRYPT_OK, LTC_ARGCHK, MAXBLOCKSIZE, and zeromem().

Referenced by f8_test_mode().

```
{
   int           x, err;
   unsigned char tkey[MAXBLOCKSIZE];

   LTC_ARGCHK(IV       != NULL);
   LTC_ARGCHK(key      != NULL);
   LTC_ARGCHK(salt_key != NULL);
```

```
   LTC_ARGCHK(f8       != NULL);

   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

#ifdef LTC_FAST
   if (cipher_descriptor[cipher].block_length % sizeof(LTC_FAST_TYPE)) {
      return CRYPT_INVALID_ARG;
   }
#endif

   /* copy details */
   f8->blockcnt = 0;
   f8->cipher   = cipher;
   f8->blocklen = cipher_descriptor[cipher].block_length;
   f8->padlen   = f8->blocklen;

   /* now get key ^ salt_key [extend salt_ket with 0x55 as required to match length] */
   zeromem(tkey, sizeof(tkey));
   for (x = 0; x < keylen && x < (int)sizeof(tkey); x++) {
       tkey[x] = key[x];
   }
   for (x = 0; x < skeylen && x < (int)sizeof(tkey); x++) {
       tkey[x] ^= salt_key[x];
   }
   for (; x < keylen && x < (int)sizeof(tkey); x++) {
       tkey[x] ^= 0x55;
   }

   /* now encrypt with tkey[0..keylen-1] the IV and use that as the IV */
   if ((err = cipher_descriptor[cipher].setup(tkey, keylen, num_rounds, &f8->key)) != CRYPT_OK) {
      return err;
   }

   /* encrypt IV */
   if ((err = cipher_descriptor[f8->cipher].ecb_encrypt(IV, f8->MIV, &f8->key)) != CRYPT_OK) {
      cipher_descriptor[f8->cipher].done(&f8->key);
      return err;
   }
   zeromem(tkey, sizeof(tkey));
   zeromem(f8->IV, sizeof(f8->IV));

   /* terminate this cipher */
   cipher_descriptor[f8->cipher].done(&f8->key);

   /* init the cipher */
   return cipher_descriptor[cipher].setup(key, keylen, num_rounds, &f8->key);
}
```

Here is the call graph for this function:

5.191 modes/f8/f8_test_mode.c File Reference

5.191.1 Detailed Description

F8 implementation, test, Tom St Denis.

Definition in file f8_test_mode.c.

`#include "tomcrypt.h"`

Include dependency graph for f8_test_mode.c:

Functions

- int f8_test_mode (void)

5.191.2 Function Documentation

5.191.2.1 int f8_test_mode (void)

Definition at line 21 of file f8_test_mode.c.

References CRYPT_FAIL_TESTVECTOR, CRYPT_NOP, CRYPT_OK, f8_done(), f8_encrypt(), f8_-start(), find_cipher(), and XMEMCMP.

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   static const unsigned char key[16] = { 0x23, 0x48, 0x29, 0x00, 0x84, 0x67, 0xbe, 0x18,
                                          0x6c, 0x3d, 0xe1, 0x4a, 0xae, 0x72, 0xd6, 0x2c };
   static const unsigned char salt[4] = { 0x32, 0xf2, 0x87, 0x0d };
   static const unsigned char IV[16]  = { 0x00, 0x6e, 0x5c, 0xba, 0x50, 0x68, 0x1d, 0xe5,
                                          0x5c, 0x62, 0x15, 0x99, 0xd4, 0x62, 0x56, 0x4a };
   static const unsigned char pt[39]  = { 0x70, 0x73, 0x65, 0x75, 0x64, 0x6f, 0x72, 0x61,
                                          0x6e, 0x64, 0x6f, 0x6d, 0x6e, 0x65, 0x73, 0x73,
                                          0x20, 0x69, 0x73, 0x20, 0x74, 0x68, 0x65, 0x20,
                                          0x6e, 0x65, 0x78, 0x74, 0x20, 0x62, 0x65, 0x73,
                                          0x74, 0x20, 0x74, 0x68, 0x69, 0x6e, 0x67       };
   static const unsigned char ct[39]  = { 0x01, 0x9c, 0xe7, 0xa2, 0x6e, 0x78, 0x54, 0x01,
                                          0x4a, 0x63, 0x66, 0xaa, 0x95, 0xd4, 0xee, 0xfd,
                                          0x1a, 0xd4, 0x17, 0x2a, 0x14, 0xf9, 0xfa, 0xf4,
                                          0x55, 0xb7, 0xf1, 0xd4, 0xb6, 0x2b, 0xd0, 0x8f,
                                          0x56, 0x2c, 0x0e, 0xef, 0x7c, 0x48, 0x02       };
   unsigned char buf[39];
   symmetric_F8  f8;
   int           err, idx;

   idx = find_cipher("aes");
   if (idx == -1) {
      idx = find_cipher("rijndael");
      if (idx == -1) return CRYPT_NOP;
   }

   /* initialize the context */
   if ((err = f8_start(idx, IV, key, sizeof(key), salt, sizeof(salt), 0, &f8)) != CRYPT_OK) {
      return err;
   }

   /* encrypt block */
   if ((err = f8_encrypt(pt, buf, sizeof(pt), &f8)) != CRYPT_OK) {
```

```
      f8_done(&f8);
      return err;
   }
   f8_done(&f8);

   /* compare */
   if (XMEMCMP(buf, ct, sizeof(ct))) {
      return CRYPT_FAIL_TESTVECTOR;
   }

   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.192 modes/lrw/lrw_decrypt.c File Reference

5.192.1 Detailed Description

LRW_MODE implementation, Decrypt blocks, Tom St Denis.

Definition in file lrw_decrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for lrw_decrypt.c:

Functions

- int lrw_decrypt (const unsigned char ∗ct, unsigned char ∗pt, unsigned long len, symmetric_LRW ∗lrw)

    LRW decrypt blocks.

5.192.2 Function Documentation

5.192.2.1 int lrw_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, unsigned long *len*, symmetric_LRW ∗ *lrw*)

LRW decrypt blocks.

Parameters:

ct The ciphertext

pt [out] The plaintext

len The length in octets, must be a multiple of 16

lrw The LRW state

Definition at line 27 of file lrw_decrypt.c.

References ltc_cipher_descriptor::accel_lrw_decrypt, cipher_descriptor, cipher_is_valid(), CRYPT_OK, lrw_process(), and LTC_ARGCHK.

```
{
   int err;

   LTC_ARGCHK(pt  != NULL);
   LTC_ARGCHK(ct  != NULL);
   LTC_ARGCHK(lrw != NULL);

   if ((err = cipher_is_valid(lrw->cipher)) != CRYPT_OK) {
      return err;
   }

   if (cipher_descriptor[lrw->cipher].accel_lrw_decrypt != NULL) {
      return cipher_descriptor[lrw->cipher].accel_lrw_decrypt(ct, pt, len, lrw->IV, lrw->tweak, &lrw->k
   }

   return lrw_process(ct, pt, len, LRW_DECRYPT, lrw);
}
```

Here is the call graph for this function:

5.193 modes/lrw/lrw_done.c File Reference

5.193.1 Detailed Description

LRW_MODE implementation, Free resources, Tom St Denis.

Definition in file lrw_done.c.

`#include "tomcrypt.h"`

Include dependency graph for lrw_done.c:

Functions

- int lrw_done (symmetric_LRW ∗lrw)

  Terminate a LRW state.

5.193.2 Function Documentation

5.193.2.1 int lrw_done (symmetric_LRW ∗ *lrw*)

Terminate a LRW state.

Parameters:

lrw The state to terminate

Returns:

CRYPT_OK if successful

Definition at line 25 of file lrw_done.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_OK, ltc_cipher_descriptor::done, and LTC_-ARGCHK.

```
{
   int err;

   LTC_ARGCHK(lrw != NULL);

   if ((err = cipher_is_valid(lrw->cipher)) != CRYPT_OK) {
      return err;
   }
   cipher_descriptor[lrw->cipher].done(&lrw->key);

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.194 modes/lrw/lrw_encrypt.c File Reference

5.194.1 Detailed Description

LRW_MODE implementation, Encrypt blocks, Tom St Denis.

Definition in file lrw_encrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for lrw_encrypt.c:

Functions

- int lrw_encrypt (const unsigned char ∗pt, unsigned char ∗ct, unsigned long len, symmetric_LRW ∗lrw)

  LRW encrypt blocks.

5.194.2 Function Documentation

5.194.2.1 int lrw_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, unsigned long *len*, symmetric_LRW ∗ *lrw*)

LRW encrypt blocks.

Parameters:

pt The plaintext

ct [out] The ciphertext

len The length in octets, must be a multiple of 16

lrw The LRW state

Definition at line 27 of file lrw_encrypt.c.

References ltc_cipher_descriptor::accel_lrw_encrypt, cipher_descriptor, cipher_is_valid(), CRYPT_OK, lrw_process(), and LTC_ARGCHK.

```
{
   int err;

   LTC_ARGCHK(pt  != NULL);
   LTC_ARGCHK(ct  != NULL);
   LTC_ARGCHK(lrw != NULL);

   if ((err = cipher_is_valid(lrw->cipher)) != CRYPT_OK) {
      return err;
   }

   if (cipher_descriptor[lrw->cipher].accel_lrw_encrypt != NULL) {
      return cipher_descriptor[lrw->cipher].accel_lrw_encrypt(pt, ct, len, lrw->IV, lrw->tweak, &lrw->k
   }

   return lrw_process(pt, ct, len, LRW_ENCRYPT, lrw);
}
```

Here is the call graph for this function:

5.195 modes/lrw/lrw_getiv.c File Reference

5.195.1 Detailed Description

LRW_MODE implementation, Retrieve the current IV, Tom St Denis.

Definition in file lrw_getiv.c.

`#include "tomcrypt.h"`

Include dependency graph for lrw_getiv.c:

Functions

- int lrw_getiv (unsigned char ∗IV, unsigned long ∗len, symmetric_LRW ∗lrw)

  Get the IV for LRW.

5.195.2 Function Documentation

5.195.2.1 int lrw_getiv (unsigned char ∗ *IV*, unsigned long ∗ *len*, symmetric_LRW ∗ *lrw*)

Get the IV for LRW.

Parameters:

IV [out] The IV, must be 16 octets

len Length ... must be at least 16 :-)

lrw The LRW state to read

Returns:

CRYPT_OK if successful

Definition at line 27 of file lrw_getiv.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, LTC_ARGCHK, and XMEMCPY.

```
{
   LTC_ARGCHK(IV != NULL);
   LTC_ARGCHK(len != NULL);
   LTC_ARGCHK(lrw != NULL);
   if (*len < 16) {
       *len = 16;
       return CRYPT_BUFFER_OVERFLOW;
   }

   XMEMCPY(IV, lrw->IV, 16);
   *len = 16;
   return CRYPT_OK;
}
```

5.196 modes/lrw/lrw_process.c File Reference

5.196.1 Detailed Description

LRW_MODE implementation, Encrypt/decrypt blocks, Tom St Denis.

Definition in file lrw_process.c.

`#include "tomcrypt.h"`

Include dependency graph for lrw_process.c:

Functions

- int lrw_process (const unsigned char ∗pt, unsigned char ∗ct, unsigned long len, int mode, symmetric_LRW ∗lrw)

  Process blocks with LRW, since decrypt/encrypt are largely the same they share this code.

5.196.2 Function Documentation

5.196.2.1 int lrw_process (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, unsigned long *len*, int *mode*, symmetric_LRW ∗ *lrw*)

Process blocks with LRW, since decrypt/encrypt are largely the same they share this code.

Parameters:

pt The "input" data

ct [out] The "output" data

len The length of the input, must be a multiple of 128-bits (16 octets)

mode LRW_ENCRYPT or LRW_DECRYPT

lrw The LRW state

Returns:

CRYPT_OK if successful

Definition at line 29 of file lrw_process.c.

References CRYPT_INVALID_ARG, LTC_ARGCHK, and XMEMCPY.

Referenced by lrw_decrypt(), and lrw_encrypt().

```
{
   unsigned char prod[16];
   int           x, err;
#ifdef LRW_TABLES
   int           y;
#endif

   LTC_ARGCHK(pt  != NULL);
   LTC_ARGCHK(ct  != NULL);
   LTC_ARGCHK(lrw != NULL);

   if (len & 15) {
      return CRYPT_INVALID_ARG;
```

```
   }

   while (len) {
      /* copy pad */
      XMEMCPY(prod, lrw->pad, 16);

      /* increment IV */
      for (x = 15; x >= 0; x--) {
          lrw->IV[x] = (lrw->IV[x] + 1) & 255;
          if (lrw->IV[x]) {
              break;
          }
      }

      /* update pad */
#ifdef LRW_TABLES
      /* for each byte changed we undo it's affect on the pad then add the new product */
      for (; x < 16; x++) {
#ifdef LTC_FAST
          for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
              *((LTC_FAST_TYPE *)(lrw->pad + y)) ^= *((LTC_FAST_TYPE *)(&lrw->PC[x][lrw->IV[x]][y])) ^
          }
#else
          for (y = 0; y < 16; y++) {
              lrw->pad[y] ^= lrw->PC[x][lrw->IV[x]][y] ^ lrw->PC[x][(lrw->IV[x]-1)&255][y];
          }
#endif
      }
#else
      gcm_gf_mult(lrw->tweak, lrw->IV, lrw->pad);
#endif

      /* xor prod */
#ifdef LTC_FAST
      for (x = 0; x < 16; x += sizeof(LTC_FAST_TYPE)) {
           *((LTC_FAST_TYPE *)(ct + x)) = *((LTC_FAST_TYPE *)(pt + x)) ^ *((LTC_FAST_TYPE *)(prod + x))
      }
#else
      for (x = 0; x < 16; x++) {
         ct[x] = pt[x] ^ prod[x];
      }
#endif

      /* send through cipher */
      if (mode == LRW_ENCRYPT) {
         if ((err = cipher_descriptor[lrw->cipher].ecb_encrypt(ct, ct, &lrw->key)) != CRYPT_OK) {
            return err;
         }
      } else {
         if ((err = cipher_descriptor[lrw->cipher].ecb_decrypt(ct, ct, &lrw->key)) != CRYPT_OK) {
            return err;
         }
      }

      /* xor prod */
#ifdef LTC_FAST
      for (x = 0; x < 16; x += sizeof(LTC_FAST_TYPE)) {
           *((LTC_FAST_TYPE *)(ct + x)) = *((LTC_FAST_TYPE *)(ct + x)) ^ *((LTC_FAST_TYPE *)(prod + x
      }
#else
      for (x = 0; x < 16; x++) {
         ct[x] = ct[x] ^ prod[x];
      }
#endif

      /* move to next */
      pt  += 16;
```

```
      ct  += 16;
      len -= 16;
   }

   return CRYPT_OK;
}
```

5.197 modes/lrw/lrw_setiv.c File Reference

5.197.1 Detailed Description

LRW_MODE implementation, Set the current IV, Tom St Denis.

Definition in file lrw_setiv.c.

`#include "tomcrypt.h"`

Include dependency graph for lrw_setiv.c:

Functions

- int lrw_setiv (const unsigned char ∗IV, unsigned long len, symmetric_LRW ∗lrw)

  Set the IV for LRW.

5.197.2 Function Documentation

5.197.2.1 int lrw_setiv (const unsigned char ∗ *IV*, unsigned long *len*, symmetric_LRW ∗ *lrw*)

Set the IV for LRW.

Parameters:

IV The IV, must be 16 octets

len Length ... must be 16 :-)

lrw The LRW state to update

Returns:

CRYPT_OK if successful

Definition at line 27 of file lrw_setiv.c.

References ltc_cipher_descriptor::accel_lrw_decrypt, ltc_cipher_descriptor::accel_lrw_encrypt, cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, and XMEMCPY.

```
28 {
29    int           err;
30 #ifdef LRW_TABLES
31    unsigned char T[16];
32    int           x, y;
33 #endif
34    LTC_ARGCHK(IV != NULL);
35    LTC_ARGCHK(lrw != NULL);
36 
37    if (len != 16) {
38       return CRYPT_INVALID_ARG;
39    }
40 
41    if ((err = cipher_is_valid(lrw->cipher)) != CRYPT_OK) {
42       return err;
43    }
44 
45    /* copy the IV */
46    XMEMCPY(lrw->IV, IV, 16);
```

```

   /* check if we have to actually do work */
   if (cipher_descriptor[lrw->cipher].accel_lrw_encrypt != NULL && cipher_descriptor[lrw->cipher].accel
       /* we have accelerators, let's bail since they don't use lrw->pad anyways */
       return CRYPT_OK;
   }

#ifdef LRW_TABLES
   XMEMCPY(T, &lrw->PC[0][IV[0]][0], 16);
   for (x = 1; x < 16; x++) {
#ifdef LTC_FAST
       for (y = 0; y < 16; y += sizeof(LTC_FAST_TYPE)) {
           *((LTC_FAST_TYPE *)(T + y)) ^= *((LTC_FAST_TYPE *)(&lrw->PC[x][IV[x]][y]));
       }
#else
       for (y = 0; y < 16; y++) {
           T[y] ^= lrw->PC[x][IV[x]][y];
       }
#endif
   }
   XMEMCPY(lrw->pad, T, 16);
#else
   gcm_gf_mult(lrw->tweak, IV, lrw->pad);
#endif

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.198 modes/lrw/lrw_start.c File Reference

5.198.1 Detailed Description

LRW_MODE implementation, start mode, Tom St Denis.

Definition in file lrw_start.c.

`#include "tomcrypt.h"`

Include dependency graph for lrw_start.c:

Functions

- int lrw_start (int cipher, const unsigned char ∗IV, const unsigned char ∗key, int keylen, const unsigned char ∗tweak, int num_rounds, symmetric_LRW ∗lrw)

  Initialize the LRW context.

5.198.2 Function Documentation

5.198.2.1 int lrw_start (int *cipher*, const unsigned char ∗ *IV*, const unsigned char ∗ *key*, int *keylen*, const unsigned char ∗ *tweak*, int *num_rounds*, symmetric_LRW ∗ *lrw*)

Initialize the LRW context.

Parameters:

cipher The cipher desired, must be a 128-bit block cipher

IV The index value, must be 128-bits

key The cipher key

keylen The length of the cipher key in octets

tweak The tweak value (second key), must be 128-bits

num_rounds The number of rounds for the cipher (0 == default)

lrw [out] The LRW state

Returns:

CRYPT_OK on success.

Definition at line 31 of file lrw_start.c.

References B, cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_INVALID_-CIPHER, CRYPT_OK, gcm_gf_mult(), LTC_ARGCHK, XMEMCPY, and zeromem().

Referenced by lrw_test().

```
37 {
38    int           err;
39 #ifdef LRW_TABLES
40    unsigned char B[16];
41    int           x, y, z, t;
42 #endif
43
44   LTC_ARGCHK(IV    != NULL);
```

```
  LTC_ARGCHK(key   != NULL);
  LTC_ARGCHK(tweak != NULL);
  LTC_ARGCHK(lrw   != NULL);

#ifdef LTC_FAST
   if (16 % sizeof(LTC_FAST_TYPE)) {
      return CRYPT_INVALID_ARG;
   }
#endif

   /* is cipher valid? */
   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }
   if (cipher_descriptor[cipher].block_length != 16) {
      return CRYPT_INVALID_CIPHER;
   }

   /* schedule key */
   if ((err = cipher_descriptor[cipher].setup(key, keylen, num_rounds, &lrw->key)) != CRYPT_OK) {
      return err;
   }
   lrw->cipher = cipher;

   /* copy the IV and tweak */
   XMEMCPY(lrw->tweak, tweak, 16);

#ifdef LRW_TABLES
   /* setup tables */
   /* generate the first table as it has no shifting (from which we make the other tables) */
   zeromem(B, 16);
   for (y = 0; y < 256; y++) {
        B[0] = y;
        gcm_gf_mult(tweak, B, &lrw->PC[0][y][0]);
   }

   /* now generate the rest of the tables based the previous table */
   for (x = 1; x < 16; x++) {
      for (y = 0; y < 256; y++) {
         /* now shift it right by 8 bits */
         t = lrw->PC[x-1][y][15];
         for (z = 15; z > 0; z--) {
             lrw->PC[x][y][z] = lrw->PC[x-1][y][z-1];
         }
         lrw->PC[x][y][0]  = gcm_shift_table[t<<1];
         lrw->PC[x][y][1] ^= gcm_shift_table[(t<<1)+1];
     }
  }
#endif

   /* generate first pad */
   return lrw_setiv(IV, 16, lrw);
}
```

Here is the call graph for this function:

5.199 modes/lrw/lrw_test.c File Reference

5.199.1 Detailed Description

LRW_MODE implementation, test LRW, Tom St Denis.

Definition in file lrw_test.c.

`#include "tomcrypt.h"`

Include dependency graph for lrw_test.c:

Functions

- int lrw_test (void)

  Test LRW against specs.

5.199.2 Function Documentation

5.199.2.1 int lrw_test (void)

Test LRW against specs.

Returns:

CRYPT_OK if goodly

Definition at line 24 of file lrw_test.c.

References CRYPT_NOP, CRYPT_OK, find_cipher(), and lrw_start().

```
{
#ifndef  LTC_TEST
   return CRYPT_NOP;
#else
   static const struct {
      unsigned char key[16], tweak[16], IV[16], P[16], expected_tweak[16], C[16];
   } tests[] = {

{
{ 0x45, 0x62, 0xac, 0x25, 0xf8, 0x28, 0x17, 0x6d, 0x4c, 0x26, 0x84, 0x14, 0xb5, 0x68, 0x01, 0x85 },
{ 0x25, 0x8e, 0x2a, 0x05, 0xe7, 0x3e, 0x9d, 0x03, 0xee, 0x5a, 0x83, 0x0c, 0xcc, 0x09, 0x4c, 0x87 },
{ 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
{ 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46 },
{ 0x25, 0x8e, 0x2a, 0x05, 0xe7, 0x3e, 0x9d, 0x03, 0xee, 0x5a, 0x83, 0x0c, 0xcc, 0x09, 0x4c, 0x87 },
{ 0xf1, 0xb2, 0x73, 0xcd, 0x65, 0xa3, 0xdf, 0x5f, 0xe9, 0x5d, 0x48, 0x92, 0x54, 0x63, 0x4e, 0xb8 }
},

{
{ 0x59, 0x70, 0x47, 0x14, 0xf5, 0x57, 0x47, 0x8c, 0xd7, 0x79, 0xe8, 0x0f, 0x54, 0x88, 0x79, 0x44 },
{ 0x35, 0x23, 0xc2, 0xde, 0xc5, 0x69, 0x4f, 0xa8, 0x72, 0xa9, 0xac, 0xa7, 0x0b, 0x2b, 0xee, 0xbc },
{ 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
{ 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46 },
{ 0x1a, 0x91, 0xe1, 0x6f, 0x62, 0xb4, 0xa7, 0xd4, 0x39, 0x54, 0xd6, 0x53, 0x85, 0x95, 0xf7, 0x5e },
{ 0x00, 0xc8, 0x2b, 0xae, 0x95, 0xbb, 0xcd, 0xe5, 0x27, 0x4f, 0x07, 0x69, 0xb2, 0x60, 0xe1, 0x36 },
},

{
{ 0x59, 0x70, 0x47, 0x14, 0xf5, 0x57, 0x47, 0x8c, 0xd7, 0x79, 0xe8, 0x0f, 0x54, 0x88, 0x79, 0x44 },
```

```
{ 0x67, 0x53, 0xc9, 0x0c, 0xb7, 0xd8, 0xcd, 0xe5, 0x06, 0xa0, 0x47, 0x78, 0x1a, 0xad, 0x85, 0x11 },
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02 },
{ 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46 },
{ 0x1a, 0x91, 0xe1, 0x6f, 0x62, 0xb4, 0xa7, 0xd4, 0x39, 0x54, 0xd6, 0x53, 0x85, 0x95, 0xf7, 0x5e },
{ 0x00, 0xc8, 0x2b, 0xae, 0x95, 0xbb, 0xcd, 0xe5, 0x27, 0x4f, 0x07, 0x69, 0xb2, 0x60, 0xe1, 0x36 },
},

{

{ 0xd8, 0x2a, 0x91, 0x34, 0xb2, 0x6a, 0x56, 0x50, 0x30, 0xfe, 0x69, 0xe2, 0x37, 0x7f, 0x98, 0x47 },
{ 0x4e, 0xb5, 0x5d, 0x31, 0x05, 0x97, 0x3a, 0x3f, 0x5e, 0x23, 0xda, 0xfb, 0x5a, 0x45, 0xd6, 0xc0 },
{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00 },
{ 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46 },
{ 0x18, 0xc9, 0x1f, 0x6d, 0x60, 0x1a, 0x1a, 0x37, 0x5d, 0x0b, 0x0e, 0xf7, 0x3a, 0xd5, 0x74, 0xc4 },
{ 0x76, 0x32, 0x21, 0x83, 0xed, 0x8f, 0xf1, 0x82, 0xf9, 0x59, 0x62, 0x03, 0x69, 0x0e, 0x5e, 0x01 },

}
};

   int idx, err, x;
   symmetric_LRW lrw;
   unsigned char buf[2][16];

   idx = find_cipher("aes");
   if (idx == -1) {
      idx = find_cipher("rijndael");
      if (idx == -1) {
         return CRYPT_NOP;
      }
   }

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
      /* schedule it */
      if ((err = lrw_start(idx, tests[x].IV, tests[x].key, 16, tests[x].tweak, 0, &lrw)) != CRYPT_OK) {
         return err;
      }

      /* check pad against expected tweak */
      if (XMEMCMP(tests[x].expected_tweak, lrw.pad, 16)) {
         lrw_done(&lrw);
         return CRYPT_FAIL_TESTVECTOR;
      }

      /* process block */
      if ((err = lrw_encrypt(tests[x].P, buf[0], 16, &lrw)) != CRYPT_OK) {
         lrw_done(&lrw);
         return err;
      }

      if (XMEMCMP(buf[0], tests[x].C, 16)) {
         lrw_done(&lrw);
         return CRYPT_FAIL_TESTVECTOR;
      }

      /* process block */
      if ((err = lrw_setiv(tests[x].IV, 16, &lrw)) != CRYPT_OK) {
         lrw_done(&lrw);
         return err;
      }

      if ((err = lrw_decrypt(buf[0], buf[1], 16, &lrw)) != CRYPT_OK) {
         lrw_done(&lrw);
         return err;
      }

      if (XMEMCMP(buf[1], tests[x].P, 16)) {
         lrw_done(&lrw);
```

```
         return CRYPT_FAIL_TESTVECTOR;
      }
      if ((err = lrw_done(&lrw)) != CRYPT_OK) {
         return err;
      }
   }
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.200 modes/ofb/ofb_decrypt.c File Reference

5.200.1 Detailed Description

OFB implementation, decrypt data, Tom St Denis.

Definition in file ofb_decrypt.c.

`#include "tomcrypt.h"`

Include dependency graph for ofb_decrypt.c:

Functions

- int ofb_decrypt (const unsigned char ∗ct, unsigned char ∗pt, unsigned long len, symmetric_OFB ∗ofb)

  OFB decrypt.

5.200.2 Function Documentation

5.200.2.1 int ofb_decrypt (const unsigned char ∗ *ct*, unsigned char ∗ *pt*, unsigned long *len*, symmetric_OFB ∗ *ofb*)

OFB decrypt.

Parameters:

ct Ciphertext

pt [out] Plaintext

len Length of ciphertext (octets)

ofb OFB state

Returns:

CRYPT_OK if successful

Definition at line 28 of file ofb_decrypt.c.

References LTC_ARGCHK, and ofb_encrypt().

```
{
   LTC_ARGCHK(pt != NULL);
   LTC_ARGCHK(ct != NULL);
   LTC_ARGCHK(ofb != NULL);
   return ofb_encrypt(ct, pt, len, ofb);
}
```

Here is the call graph for this function:

5.201 modes/ofb/ofb_done.c File Reference

5.201.1 Detailed Description

OFB implementation, finish chain, Tom St Denis.

Definition in file ofb_done.c.

`#include "tomcrypt.h"`

Include dependency graph for ofb_done.c:

Functions

- int ofb_done (symmetric_OFB ∗ofb)

  Terminate the chain.

5.201.2 Function Documentation

5.201.2.1 int ofb_done (symmetric_OFB ∗ *ofb*)

Terminate the chain.

Parameters:

ofb The OFB chain to terminate

Returns:

CRYPT_OK on success

Definition at line 24 of file ofb_done.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_OK, ltc_cipher_descriptor::done, and LTC_-ARGCHK.

```
{
   int err;
   LTC_ARGCHK(ofb != NULL);

   if ((err = cipher_is_valid(ofb->cipher)) != CRYPT_OK) {
      return err;
   }
   cipher_descriptor[ofb->cipher].done(&ofb->key);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.202 modes/ofb/ofb_encrypt.c File Reference

5.202.1 Detailed Description

OFB implementation, encrypt data, Tom St Denis.

Definition in file ofb_encrypt.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ofb_encrypt.c:

Functions

- int ofb_encrypt (const unsigned char ∗pt, unsigned char ∗ct, unsigned long len, symmetric_OFB ∗ofb)

  OFB encrypt.

5.202.2 Function Documentation

5.202.2.1 int ofb_encrypt (const unsigned char ∗ *pt*, unsigned char ∗ *ct*, unsigned long *len*, symmetric_OFB ∗ *ofb*)

OFB encrypt.

Parameters:

pt Plaintext

ct [out] Ciphertext

len Length of plaintext (octets)

ofb OFB state

Returns:

CRYPT_OK if successful

Definition at line 28 of file ofb_encrypt.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_-descriptor::ecb_encrypt, and LTC_ARGCHK.

Referenced by ofb_decrypt().

```
{
   int err;
   LTC_ARGCHK(pt != NULL);
   LTC_ARGCHK(ct != NULL);
   LTC_ARGCHK(ofb != NULL);
   if ((err = cipher_is_valid(ofb->cipher)) != CRYPT_OK) {
       return err;
   }

   /* is blocklen/padlen valid? */
   if (ofb->blocklen < 0 || ofb->blocklen > (int)sizeof(ofb->IV) ||
       ofb->padlen   < 0 || ofb->padlen   > (int)sizeof(ofb->IV)) {
      return CRYPT_INVALID_ARG;
```

```
   }

   while (len-- > 0) {
       if (ofb->padlen == ofb->blocklen) {
          if ((err = cipher_descriptor[ofb->cipher].ecb_encrypt(ofb->IV, ofb->IV, &ofb->key)) != CRYPT_
             return err;
          }
          ofb->padlen = 0;
       }
       *ct++ = *pt++ ^ ofb->IV[ofb->padlen++];
   }
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.203 modes/ofb/ofb_getiv.c File Reference

5.203.1 Detailed Description

F8 implementation, get IV, Tom St Denis.

Definition in file ofb_getiv.c.

`#include "tomcrypt.h"`

Include dependency graph for ofb_getiv.c:

Functions

- int ofb_getiv (unsigned char ∗IV, unsigned long ∗len, symmetric_OFB ∗ofb)

    Get the current initial vector.

5.203.2 Function Documentation

5.203.2.1 int ofb_getiv (unsigned char ∗ *IV*, unsigned long ∗ *len*, symmetric_OFB ∗ *ofb*)

Get the current initial vector.

Parameters:

IV [out] The destination of the initial vector

len [in/out] The max size and resulting size of the initial vector

ofb The OFB state

Returns:

CRYPT_OK if successful

Definition at line 27 of file ofb_getiv.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, LTC_ARGCHK, and XMEMCPY.

```
{
   LTC_ARGCHK(IV  != NULL);
   LTC_ARGCHK(len != NULL);
   LTC_ARGCHK(ofb != NULL);
   if ((unsigned long)ofb->blocklen > *len) {
      *len = ofb->blocklen;
      return CRYPT_BUFFER_OVERFLOW;
   }
   XMEMCPY(IV, ofb->IV, ofb->blocklen);
   *len = ofb->blocklen;

   return CRYPT_OK;
}
```

5.204 modes/ofb/ofb_setiv.c File Reference

5.204.1 Detailed Description

OFB implementation, set IV, Tom St Denis.

Definition in file ofb_setiv.c.

`#include "tomcrypt.h"`

Include dependency graph for ofb_setiv.c:

Functions

- int ofb_setiv (const unsigned char ∗IV, unsigned long len, symmetric_OFB ∗ofb)
  Set an initial vector.

5.204.2 Function Documentation

5.204.2.1 int ofb_setiv (const unsigned char ∗ *IV*, unsigned long *len*, symmetric_OFB ∗ *ofb*)

Set an initial vector.

Parameters:

IV The initial vector

len The length of the vector (in octets)

ofb The OFB state

Returns:

CRYPT_OK if successful

Definition at line 27 of file ofb_setiv.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_INVALID_ARG, CRYPT_OK, ltc_cipher_-descriptor::ecb_encrypt, and LTC_ARGCHK.

```
{
   int err;

   LTC_ARGCHK(IV  != NULL);
   LTC_ARGCHK(ofb != NULL);

   if ((err = cipher_is_valid(ofb->cipher)) != CRYPT_OK) {
       return err;
   }

   if (len != (unsigned long)ofb->blocklen) {
      return CRYPT_INVALID_ARG;
   }

   /* force next block */
   ofb->padlen = 0;
   return cipher_descriptor[ofb->cipher].ecb_encrypt(IV, ofb->IV, &ofb->key);
}
```

Here is the call graph for this function:

5.205 modes/ofb/ofb_start.c File Reference

5.205.1 Detailed Description

OFB implementation, start chain, Tom St Denis.

Definition in file ofb_start.c.

`#include "tomcrypt.h"`

Include dependency graph for ofb_start.c:

Functions

- int ofb_start (int cipher, const unsigned char ∗IV, const unsigned char ∗key, int keylen, int num_rounds, symmetric_OFB ∗ofb)

  Initialize a OFB context.

5.205.2 Function Documentation

5.205.2.1 int ofb_start (int *cipher*, const unsigned char ∗ *IV*, const unsigned char ∗ *key*, int *keylen*, int *num_rounds*, symmetric_OFB ∗ *ofb*)

Initialize a OFB context.

Parameters:

cipher The index of the cipher desired

IV The initial vector

key The secret key

keylen The length of the secret key (octets)

num_rounds Number of rounds in the cipher desired (0 for default)

ofb The OFB state to initialize

Returns:

CRYPT_OK if successful

Definition at line 31 of file ofb_start.c.

References ltc_cipher_descriptor::block_length, cipher_descriptor, cipher_is_valid(), CRYPT_OK, and LTC_ARGCHK.

```
{
   int x, err;

   LTC_ARGCHK(IV != NULL);
   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(ofb != NULL);

   if ((err = cipher_is_valid(cipher)) != CRYPT_OK) {
      return err;
   }

   /* copy details */
```

```
    ofb->cipher = cipher;
    ofb->blocklen = cipher_descriptor[cipher].block_length;
    for (x = 0; x < ofb->blocklen; x++) {
        ofb->IV[x] = IV[x];
    }

    /* init the cipher */
    ofb->padlen = ofb->blocklen;
    return cipher_descriptor[cipher].setup(key, keylen, num_rounds, &ofb->key);
}
```

Here is the call graph for this function:

5.206 pk/asn1/der/bit/der_decode_bit_string.c File Reference

5.206.1 Detailed Description

ASN.1 DER, encode a BIT STRING, Tom St Denis.

Definition in file der_decode_bit_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_bit_string.c:

Functions

- int der_decode_bit_string (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

    Store a BIT STRING.

5.206.2 Function Documentation

5.206.2.1 int der_decode_bit_string (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store a BIT STRING.

Parameters:

in The DER encoded BIT STRING

inlen The size of the DER BIT STRING

out [out] The array of bits stored (one per char)

outlen [in/out] The number of bits stored

Returns:

CRYPT_OK if successful

Definition at line 29 of file der_decode_bit_string.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_INVALID_PACKET, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi().

```
{
   unsigned long dlen, blen, x, y;

   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* packet must be at least 4 bytes */
   if (inlen < 4) {
       return CRYPT_INVALID_ARG;
   }

   /* check for 0x03 */
```

```
   if ((in[0]&0x1F) != 0x03) {
      return CRYPT_INVALID_PACKET;
   }

    /* offset in the data */
    x = 1;

   /* get the length of the data */
   if (in[x] & 0x80) {
      /* long format get number of length bytes */
      y = in[x++] & 0x7F;

      /* invalid if 0 or > 2 */
      if (y == 0 || y > 2) {
         return CRYPT_INVALID_PACKET;
      }

      /* read the data len */
      dlen = 0;
      while (y--) {
         dlen = (dlen << 8) | (unsigned long)in[x++];
      }
   } else {
      /* short format */
      dlen = in[x++] & 0x7F;
   }

   /* is the data len too long or too short? */
   if ((dlen == 0) || (dlen + x > inlen)) {
       return CRYPT_INVALID_PACKET;
   }

   /* get padding count */
   blen = ((dlen - 1) << 3) - (in[x++] & 7);

   /* too many bits? */
   if (blen > *outlen) {
      *outlen = blen;
      return CRYPT_BUFFER_OVERFLOW;
   }

   /* decode/store the bits */
   for (y = 0; y < blen; y++) {
       out[y] = (in[x] & (1 << (7 - (y & 7)))) ? 1 : 0;
       if ((y & 7) == 7) {
          ++x;
       }
   }

   /* we done */
   *outlen = blen;
   return CRYPT_OK;
}
```

5.207 pk/asn1/der/bit/der_encode_bit_string.c File Reference

5.207.1 Detailed Description

ASN.1 DER, encode a BIT STRING, Tom St Denis.

Definition in file der_encode_bit_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_encode_bit_string.c:

Functions

- int der_encode_bit_string (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Store a BIT STRING.

5.207.2 Function Documentation

5.207.2.1 int der_encode_bit_string (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store a BIT STRING.

Parameters:

in The array of bits to store (one per char)

inlen The number of bits tostore

out [out] The destination for the DER encoded BIT STRING

outlen [in/out] The max size and resulting size of the DER BIT STRING

Returns:

CRYPT_OK if successful

Definition at line 29 of file der_encode_bit_string.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, der_length_bit_string(), len, and LTC_-ARGCHK.

```
{
   unsigned long len, x, y, buf;
   int           err;

   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* avoid overflows */
   if ((err = der_length_bit_string(inlen, &len)) != CRYPT_OK) {
      return err;
   }

   if (len > *outlen) {
      *outlen = len;
```

```
      return CRYPT_BUFFER_OVERFLOW;
   }

   /* store header (include bit padding count in length) */
   x = 0;
   y = (inlen >> 3) + ((inlen&7) ? 1 : 0) + 1;

   out[x++] = 0x03;
   if (y < 128) {
      out[x++] = y;
   } else if (y < 256) {
      out[x++] = 0x81;
      out[x++] = y;
   } else if (y < 65536) {
      out[x++] = 0x82;
      out[x++] = (y>>8)&255;
      out[x++] = y&255;
   }

   /* store number of zero padding bits */
   out[x++] = (8 - inlen) & 7;

   /* store the bits in big endian format */
   for (y = buf = 0; y < inlen; y++) {
       buf |= (in[y] ? 1 : 0) << (7 - (y & 7));
       if ((y & 7) == 7) {
          out[x++] = buf;
          buf      = 0;
       }
   }
   /* store last byte */
   if (inlen & 7) {
      out[x++] = buf;
   }
   *outlen = x;
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.208 pk/asn1/der/bit/der_length_bit_string.c File Reference

5.208.1 Detailed Description

ASN.1 DER, get length of BIT STRING, Tom St Denis.

Definition in file der_length_bit_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_bit_string.c:

Functions

- int der_length_bit_string (unsigned long nbits, unsigned long ∗outlen)
  Gets length of DER encoding of BIT STRING.

5.208.2 Function Documentation

5.208.2.1 int der_length_bit_string (unsigned long *nbits*, unsigned long ∗ *outlen*)

Gets length of DER encoding of BIT STRING.

Parameters:

nbits The number of bits in the string to encode

outlen [out] The length of the DER encoding for the given string

Returns:

CRYPT_OK if successful

Definition at line 25 of file der_length_bit_string.c.

References CRYPT_INVALID_ARG, CRYPT_OK, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi(), and der_encode_bit_string().

```
{
   unsigned long nbytes;
   LTC_ARGCHK(outlen != NULL);

   /* get the number of the bytes */
   nbytes = (nbits >> 3) + ((nbits & 7) ? 1 : 0) + 1;

   if (nbytes < 128) {
      /* 03 LL PP DD DD DD ... */
      *outlen = 2 + nbytes;
   } else if (nbytes < 256) {
      /* 03 81 LL PP DD DD DD ... */
      *outlen = 3 + nbytes;
   } else if (nbytes < 65536) {
      /* 03 82 LL LL PP DD DD DD ... */
      *outlen = 4 + nbytes;
   } else {
      return CRYPT_INVALID_ARG;
   }

   return CRYPT_OK;
}
```

5.209 pk/asn1/der/boolean/der_decode_boolean.c File Reference

5.209.1 Detailed Description

ASN.1 DER, decode a BOOLEAN, Tom St Denis.

Definition in file der_decode_boolean.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_boolean.c:

Functions

- int der_decode_boolean (const unsigned char ∗in, unsigned long inlen, int ∗out)

    Read a BOOLEAN.

5.209.2 Function Documentation

5.209.2.1 int der_decode_boolean (const unsigned char ∗ *in*, unsigned long *inlen*, int ∗ *out*)

Read a BOOLEAN.

Parameters:

in The destination for the DER encoded BOOLEAN

inlen The size of the DER BOOLEAN

out [out] The boolean to decode

Returns:

CRYPT_OK if successful

Definition at line 28 of file der_decode_boolean.c.

References CRYPT_INVALID_ARG, CRYPT_OK, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi().

```
{
   LTC_ARGCHK(in  != NULL);
   LTC_ARGCHK(out != NULL);

   if (inlen != 3 || in[0] != 0x01 || in[1] != 0x01 || (in[2] != 0x00 && in[2] != 0xFF)) {
      return CRYPT_INVALID_ARG;
   }

   *out = (in[2]==0xFF) ? 1 : 0;

   return CRYPT_OK;
}
```

5.210 pk/asn1/der/boolean/der_encode_boolean.c File Reference

5.210.1 Detailed Description

ASN.1 DER, encode a BOOLEAN, Tom St Denis.

Definition in file der_encode_boolean.c.

```
#include "tomcrypt.h"
```

Include dependency graph for der_encode_boolean.c:

Functions

- int der_encode_boolean (int in, unsigned char ∗out, unsigned long ∗outlen)

    Store a BOOLEAN.

5.210.2 Function Documentation

5.210.2.1 int der_encode_boolean (int *in*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store a BOOLEAN.

Parameters:

in The boolean to encode

out [out] The destination for the DER encoded BOOLEAN

outlen [in/out] The max size and resulting size of the DER BOOLEAN

Returns:

CRYPT_OK if successful

Definition at line 28 of file der_encode_boolean.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(out    != NULL);

   if (*outlen < 3) {
       *outlen = 3;
       return CRYPT_BUFFER_OVERFLOW;
   }

   *outlen = 3;
   out[0] = 0x01;
   out[1] = 0x01;
   out[2] = in ? 0xFF : 0x00;

   return CRYPT_OK;
}
```

5.211 pk/asn1/der/boolean/der_length_boolean.c File Reference

5.211.1 Detailed Description

ASN.1 DER, get length of a BOOLEAN, Tom St Denis.

Definition in file der_length_boolean.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_boolean.c:

Functions

- int der_length_boolean (unsigned long ∗outlen)

  Gets length of DER encoding of a BOOLEAN.

5.211.2 Function Documentation

5.211.2.1 int der_length_boolean (unsigned long ∗ *outlen*)

Gets length of DER encoding of a BOOLEAN.

Parameters:

outlen [out] The length of the DER encoding

Returns:

CRYPT_OK if successful

Definition at line 24 of file der_length_boolean.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi().

```
{
   LTC_ARGCHK(outlen != NULL);
   *outlen = 3;
   return CRYPT_OK;
}
```

5.212 pk/asn1/der/choice/der_decode_choice.c File Reference

5.212.1 Detailed Description

ASN.1 DER, decode a CHOICE, Tom St Denis.

Definition in file der_decode_choice.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_choice.c:

Functions

- int der_decode_choice (const unsigned char ∗in, unsigned long ∗inlen, ltc_asn1_list ∗list, unsigned long outlen)

  Decode a CHOICE.

5.212.2 Function Documentation

5.212.2.1 int der_decode_choice (const unsigned char ∗ *in*, unsigned long ∗ *inlen*, ltc_asn1_list ∗ *list*, unsigned long *outlen*)

Decode a CHOICE.

Parameters:

in The DER encoded input

inlen [in/out] The size of the input and resulting size of read type

list The list of items to decode

outlen The number of items in the list

Returns:

CRYPT_OK on success

Definition at line 28 of file der_decode_choice.c.

References CRYPT_INVALID_PACKET, and LTC_ARGCHK.

```
{
   unsigned long size, x, z;
   void          *data;

   LTC_ARGCHK(in    != NULL);
   LTC_ARGCHK(inlen != NULL);
   LTC_ARGCHK(list  != NULL);

   /* get blk size */
   if (*inlen < 2) {
      return CRYPT_INVALID_PACKET;
   }

   /* set all of the "used" flags to zero */
   for (x = 0; x < outlen; x++) {
       list[x].used = 0;
   }
```

```

    /* now scan until we have a winner */
    for (x = 0; x < outlen; x++) {
        size = list[x].size;
        data = list[x].data;

        switch (list[x].type) {
            case LTC_ASN1_INTEGER:
                if (der_decode_integer(in, *inlen, data) == CRYPT_OK) {
                   if (der_length_integer(data, &z) == CRYPT_OK) {
                       list[x].used = 1;
                       *inlen       = z;
                       return CRYPT_OK;
                   }
                }
                break;

            case LTC_ASN1_SHORT_INTEGER:
                if (der_decode_short_integer(in, *inlen, data) == CRYPT_OK) {
                   if (der_length_short_integer(size, &z) == CRYPT_OK) {
                       list[x].used = 1;
                       *inlen       = z;
                       return CRYPT_OK;
                   }
                }
                break;

            case LTC_ASN1_BIT_STRING:
                if (der_decode_bit_string(in, *inlen, data, &size) == CRYPT_OK) {
                   if (der_length_bit_string(size, &z) == CRYPT_OK) {
                      list[x].used = 1;
                      list[x].size = size;
                      *inlen       = z;
                      return CRYPT_OK;
                   }
                }
                break;

            case LTC_ASN1_OCTET_STRING:
                if (der_decode_octet_string(in, *inlen, data, &size) == CRYPT_OK) {
                   if (der_length_octet_string(size, &z) == CRYPT_OK) {
                      list[x].used = 1;
                      list[x].size = size;
                      *inlen       = z;
                      return CRYPT_OK;
                   }
                }
                break;

            case LTC_ASN1_NULL:
                if (*inlen == 2 && in[x] == 0x05 && in[x+1] == 0x00) {
                   *inlen = 2;
                   list[x].used   = 1;
                   return CRYPT_OK;
                }
                break;

            case LTC_ASN1_OBJECT_IDENTIFIER:
                if (der_decode_object_identifier(in, *inlen, data, &size) == CRYPT_OK) {
                   if (der_length_object_identifier(data, size, &z) == CRYPT_OK) {
                      list[x].used = 1;
                      list[x].size = size;
                      *inlen       = z;
                      return CRYPT_OK;
                   }
                }
                break;
```

```

         case LTC_ASN1_IA5_STRING:
             if (der_decode_ia5_string(in, *inlen, data, &size) == CRYPT_OK) {
                if (der_length_ia5_string(data, size, &z) == CRYPT_OK) {
                   list[x].used = 1;
                   list[x].size = size;
                   *inlen       = z;
                   return CRYPT_OK;
                }
             }
             break;


         case LTC_ASN1_PRINTABLE_STRING:
             if (der_decode_printable_string(in, *inlen, data, &size) == CRYPT_OK) {
                if (der_length_printable_string(data, size, &z) == CRYPT_OK) {
                   list[x].used = 1;
                   list[x].size = size;
                   *inlen       = z;
                   return CRYPT_OK;
                }
             }
             break;

         case LTC_ASN1_UTCTIME:
             z = *inlen;
             if (der_decode_utctime(in, &z, data) == CRYPT_OK) {
                list[x].used = 1;
                *inlen       = z;
                return CRYPT_OK;
             }
             break;

         case LTC_ASN1_SET:
         case LTC_ASN1_SETOF:
         case LTC_ASN1_SEQUENCE:
             if (der_decode_sequence(in, *inlen, data, size) == CRYPT_OK) {
                if (der_length_sequence(data, size, &z) == CRYPT_OK) {
                   list[x].used = 1;
                   *inlen       = z;
                   return CRYPT_OK;
                }
             }
             break;

         default:
             return CRYPT_INVALID_ARG;
     }
  }

  return CRYPT_INVALID_PACKET;
}
```

5.213 pk/asn1/der/ia5/der_decode_ia5_string.c File Reference

5.213.1 Detailed Description

ASN.1 DER, encode a IA5 STRING, Tom St Denis.

Definition in file der_decode_ia5_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_ia5_string.c:

Functions

- int der_decode_ia5_string (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Store a IA5 STRING.

5.213.2 Function Documentation

5.213.2.1 int der_decode_ia5_string (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store a IA5 STRING.

Parameters:

in The DER encoded IA5 STRING

inlen The size of the DER IA5 STRING

out [out] The array of octets stored (one per char)

outlen [in/out] The number of octets stored

Returns:

CRYPT_OK if successful

Definition at line 29 of file der_decode_ia5_string.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_INVALID_PACKET, der_ia5_value_decode(), len, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi().

```
{
   unsigned long x, y, len;
   int           t;

   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* must have header at least */
   if (inlen < 2) {
      return CRYPT_INVALID_PACKET;
   }

```

```
   /* check for 0x16 */
   if ((in[0] & 0x1F) != 0x16) {
      return CRYPT_INVALID_PACKET;
   }
   x = 1;

   /* decode the length */
   if (in[x] & 0x80) {
      /* valid # of bytes in length are 1,2,3 */
      y = in[x] & 0x7F;
      if ((y == 0) || (y > 3) || ((x + y) > inlen)) {
         return CRYPT_INVALID_PACKET;
      }

      /* read the length in */
      len = 0;
      ++x;
      while (y--) {
         len = (len << 8) | in[x++];
      }
   } else {
      len = in[x++] & 0x7F;
   }

   /* is it too long? */
   if (len > *outlen) {
      *outlen = len;
      return CRYPT_BUFFER_OVERFLOW;
   }

   if (len + x > inlen) {
      return CRYPT_INVALID_PACKET;
   }

   /* read the data */
   for (y = 0; y < len; y++) {
       t = der_ia5_value_decode(in[x++]);
       if (t == -1) {
           return CRYPT_INVALID_ARG;
       }
       out[y] = t;
   }

   *outlen = y;

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.214 pk/asn1/der/ia5/der_encode_ia5_string.c File Reference

5.214.1 Detailed Description

ASN.1 DER, encode a IA5 STRING, Tom St Denis.

Definition in file der_encode_ia5_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_encode_ia5_string.c:

Functions

- int der_encode_ia5_string (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Store an IA5 STRING.

5.214.2 Function Documentation

5.214.2.1 int der_encode_ia5_string (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store an IA5 STRING.

Parameters:

in The array of IA5 to store (one per char)

inlen The number of IA5 to store

out [out] The destination for the DER encoded IA5 STRING

outlen [in/out] The max size and resulting size of the DER IA5 STRING

Returns:

CRYPT_OK if successful

Definition at line 28 of file der_encode_ia5_string.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_OK, der_ia5_char_-encode(), der_length_ia5_string(), len, and LTC_ARGCHK.

```
{
   unsigned long x, y, len;
   int           err;

   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* get the size */
   if ((err = der_length_ia5_string(in, inlen, &len)) != CRYPT_OK) {
      return err;
   }

   /* too big? */
   if (len > *outlen) {
```

```
      *outlen = len;
      return CRYPT_BUFFER_OVERFLOW;
   }

   /* encode the header+len */
   x = 0;
   out[x++] = 0x16;
   if (inlen < 128) {
      out[x++] = inlen;
   } else if (inlen < 256) {
      out[x++] = 0x81;
      out[x++] = inlen;
   } else if (inlen < 65536UL) {
      out[x++] = 0x82;
      out[x++] = (inlen>>8)&255;
      out[x++] = inlen&255;
   } else if (inlen < 16777216UL) {
      out[x++] = 0x83;
      out[x++] = (inlen>>16)&255;
      out[x++] = (inlen>>8)&255;
      out[x++] = inlen&255;
   } else {
      return CRYPT_INVALID_ARG;
   }

   /* store octets */
   for (y = 0; y < inlen; y++) {
       out[x++] = der_ia5_char_encode(in[y]);
   }

   /* retun length */
   *outlen = x;

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.215 pk/asn1/der/ia5/der_length_ia5_string.c File Reference

5.215.1 Detailed Description

ASN.1 DER, get length of IA5 STRING, Tom St Denis.

Definition in file der_length_ia5_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_ia5_string.c:

Functions

- int der_ia5_char_encode (int c)
- int der_ia5_value_decode (int v)
- int der_length_ia5_string (const unsigned char *octets, unsigned long noctets, unsigned long *outlen)

  Gets length of DER encoding of IA5 STRING.

Variables

- struct {
  int len
  unsigned char poly_div [MAXBLOCKSIZE]
  unsigned char poly_mul [MAXBLOCKSIZE]
  int code
  int value
  } ia5_table [ ]

5.215.2 Function Documentation

5.215.2.1 int der_ia5_char_encode (int *c*)

Definition at line 127 of file der_length_ia5_string.c.

References code, ia5_table, and value.

Referenced by der_encode_ia5_string(), der_encode_utctime(), and der_length_ia5_string().

```
{
   int x;
   for (x = 0; x < (int)(sizeof(ia5_table)/sizeof(ia5_table[0])); x++) {
       if (ia5_table[x].code == c) {
          return ia5_table[x].value;
       }
   }
   return -1;
}
```

5.215.2.2 int der_ia5_value_decode (int *v*)

Definition at line 138 of file der_length_ia5_string.c.

References code, ia5_table, and value.

Referenced by der_decode_ia5_string(), and der_decode_utctime().

```
{
   int x;
   for (x = 0; x < (int)(sizeof(ia5_table)/sizeof(ia5_table[0])); x++) {
       if (ia5_table[x].value == v) {
          return ia5_table[x].code;
       }
   }
   return -1;
}
```

5.215.2.3 int der_length_ia5_string (const unsigned char ∗ *octets*, unsigned long *noctets*, unsigned long ∗ *outlen*)

Gets length of DER encoding of IA5 STRING.

Parameters:

octets The values you want to encode

noctets The number of octets in the string to encode

outlen [out] The length of the DER encoding for the given string

Returns:

CRYPT_OK if successful

Definition at line 156 of file der_length_ia5_string.c.

References CRYPT_INVALID_ARG, der_ia5_char_encode(), and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi(), and der_encode_ia5_string().

```
{
   unsigned long x;

   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(octets != NULL);

   /* scan string for validity */
   for (x = 0; x < noctets; x++) {
       if (der_ia5_char_encode(octets[x]) == -1) {
          return CRYPT_INVALID_ARG;
       }
   }

   if (noctets < 128) {
      /* 16 LL DD DD DD ... */
      *outlen = 2 + noctets;
   } else if (noctets < 256) {
      /* 16 81 LL DD DD DD ... */
      *outlen = 3 + noctets;
   } else if (noctets < 65536UL) {
      /* 16 82 LL LL DD DD DD ... */
      *outlen = 4 + noctets;
```

```
   } else if (noctets < 16777216UL) {
      /* 16 83 LL LL LL DD DD DD ... */
      *outlen = 5 + noctets;
   } else {
      return CRYPT_INVALID_ARG;
   }

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.215.3 Variable Documentation

5.215.3.1 int code

Definition at line 21 of file der_length_ia5_string.c.

Referenced by der_ia5_char_encode(), der_ia5_value_decode(), der_printable_char_encode(), and der_-printable_value_decode().

5.215.3.2 const { ... } ia5_table[ ] [static]

Referenced by der_ia5_char_encode(), and der_ia5_value_decode().

5.215.3.3 int value

Definition at line 21 of file der_length_ia5_string.c.

Referenced by der_ia5_char_encode(), der_ia5_value_decode(), der_printable_char_encode(), and der_-printable_value_decode().

5.216 pk/asn1/der/integer/der_decode_integer.c File Reference

5.216.1 Detailed Description

ASN.1 DER, decode an integer, Tom St Denis.

Definition in file der_decode_integer.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_integer.c:

Functions

- int der_decode_integer (const unsigned char ∗in, unsigned long inlen, void ∗num)

  Read a mp_int integer.

5.216.2 Function Documentation

5.216.2.1 int der_decode_integer (const unsigned char ∗ *in*, unsigned long *inlen*, void ∗ *num*)

Read a mp_int integer.

Parameters:

in The DER encoded data

inlen Size of DER encoded data

num The first mp_int to decode

Returns:

CRYPT_OK if successful

Definition at line 28 of file der_decode_integer.c.

References CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi().

```
{
   unsigned long x, y, z;
   int           err;

   LTC_ARGCHK(num    != NULL);
   LTC_ARGCHK(in     != NULL);

   /* min DER INTEGER is 0x02 01 00 == 0 */
   if (inlen < (1 + 1 + 1)) {
      return CRYPT_INVALID_PACKET;
   }

   /* ok expect 0x02 when we AND with 0001 1111 [1F] */
   x = 0;
   if ((in[x++] & 0x1F) != 0x02) {
      return CRYPT_INVALID_PACKET;
   }

   /* now decode the len stuff */
```

```
   z = in[x++];

   if ((z & 0x80) == 0x00) {
      /* short form */

      /* will it overflow? */
      if (x + z > inlen) {
         return CRYPT_INVALID_PACKET;
      }

      /* no so read it */
      if ((err = mp_read_unsigned_bin(num, (unsigned char *)in + x, z)) != CRYPT_OK) {
         return err;
      }
   } else {
      /* long form */
      z &= 0x7F;

      /* will number of length bytes overflow? (or > 4) */
      if (((x + z) > inlen) || (z > 4) || (z == 0)) {
         return CRYPT_INVALID_PACKET;
      }

      /* now read it in */
      y = 0;
      while (z--) {
         y = ((unsigned long)(in[x++])) | (y << 8);
      }

      /* now will reading y bytes overrun? */
      if ((x + y) > inlen) {
         return CRYPT_INVALID_PACKET;
      }

      /* no so read it */
      if ((err = mp_read_unsigned_bin(num, (unsigned char *)in + x, y)) != CRYPT_OK) {
         return err;
      }
   }

   /* see if it's negative */
   if (in[x] & 0x80) {
      void *tmp;
      if (mp_init(&tmp) != CRYPT_OK) {
         return CRYPT_MEM;
      }

      if (mp_2expt(tmp, mp_count_bits(num)) != CRYPT_OK || mp_sub(num, tmp, num) != CRYPT_OK) {
         mp_clear(tmp);
         return CRYPT_MEM;
      }
      mp_clear(tmp);
   }

   return CRYPT_OK;

}
```

5.217 pk/asn1/der/integer/der_encode_integer.c File Reference

5.217.1 Detailed Description

ASN.1 DER, encode an integer, Tom St Denis.

Definition in file der_encode_integer.c.

```
#include "tomcrypt.h"
```

Include dependency graph for der_encode_integer.c:

Functions

- int der_encode_integer (void ∗num, unsigned char ∗out, unsigned long ∗outlen)

    Store a mp_int integer.

5.217.2 Function Documentation

5.217.2.1 int der_encode_integer (void ∗ *num*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store a mp_int integer.

Parameters:

num The first mp_int to encode

out [out] The destination for the DER encoded integers

outlen [in/out] The max size and resulting size of the DER encoded integers

Returns:

CRYPT_OK if successful

Definition at line 29 of file der_encode_integer.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_MEM, CRYPT_OK, der_length_integer(), LTC_ARGCHK, LTC_MP_GT, LTC_MP_LT, and LTC_MP_YES.

```
{
   unsigned long tmplen, y;
   int           err, leading_zero;

   LTC_ARGCHK(num    != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* find out how big this will be */
   if ((err = der_length_integer(num, &tmplen)) != CRYPT_OK) {
      return err;
   }

   if (*outlen < tmplen) {
      *outlen = tmplen;
      return CRYPT_BUFFER_OVERFLOW;
   }

   if (mp_cmp_d(num, 0) != LTC_MP_LT) {
```

```
      /* we only need a leading zero if the msb of the first byte is one */
      if ((mp_count_bits(num) & 7) == 0 || mp_iszero(num) == LTC_MP_YES) {
         leading_zero = 1;
      } else {
         leading_zero = 0;
      }

      /* get length of num in bytes (plus 1 since we force the msbyte to zero) */
      y = mp_unsigned_bin_size(num) + leading_zero;
   } else {
      leading_zero = 0;
      y            = mp_count_bits(num);
      y            = y + (8 - (y & 7));
      y            = y >> 3;
      if (((mp_cnt_lsb(num)+1)==mp_count_bits(num)) && ((mp_count_bits(num)&7)==0)) --y;
   }

   /* now store initial data */
   *out++ = 0x02;
   if (y < 128) {
      /* short form */
      *out++ = (unsigned char)y;
   } else if (y < 256) {
      *out++ = 0x81;
      *out++ = y;
   } else if (y < 65536UL) {
      *out++ = 0x82;
      *out++ = (y>>8)&255;
      *out++ = y;
   } else if (y < 16777216UL) {
      *out++ = 0x83;
      *out++ = (y>>16)&255;
      *out++ = (y>>8)&255;
      *out++ = y;
   } else {
      return CRYPT_INVALID_ARG;
   }

   /* now store msbyte of zero if num is non-zero */
   if (leading_zero) {
      *out++ = 0x00;
   }

   /* if it's not zero store it as big endian */
   if (mp_cmp_d(num, 0) == LTC_MP_GT) {
      /* now store the mpint */
      if ((err = mp_to_unsigned_bin(num, out)) != CRYPT_OK) {
          return err;
      }
   } else if (mp_iszero(num) != LTC_MP_YES) {
      void *tmp;

       /* negative */
       if (mp_init(&tmp) != CRYPT_OK) {
          return CRYPT_MEM;
       }

       /* 2^roundup and subtract */
       y = mp_count_bits(num);
       y = y + (8 - (y & 7));
       if (((mp_cnt_lsb(num)+1)==mp_count_bits(num)) && ((mp_count_bits(num)&7)==0)) y -= 8;
       if (mp_2expt(tmp, y) != CRYPT_OK || mp_add(tmp, num, tmp) != CRYPT_OK) {
          mp_clear(tmp);
          return CRYPT_MEM;
       }
       if ((err = mp_to_unsigned_bin(tmp, out)) != CRYPT_OK) {
          mp_clear(tmp);
```

```
            return err;
         }
         mp_clear(tmp);
      }

      /* we good */
      *outlen = tmplen;
      return CRYPT_OK;
}
```

Here is the call graph for this function:

5.218 pk/asn1/der/integer/der_length_integer.c File Reference

5.218.1 Detailed Description

ASN.1 DER, get length of encoding, Tom St Denis.

Definition in file der_length_integer.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_integer.c:

Functions

- int der_length_integer (void ∗num, unsigned long ∗outlen)
  Gets length of DER encoding of num.

5.218.2 Function Documentation

5.218.2.1 int der_length_integer (void ∗ *num*, unsigned long ∗ *outlen*)

Gets length of DER encoding of num.

Parameters:

num The int to get the size of

outlen [out] The length of the DER encoding for the given integer

Returns:

CRYPT_OK if successful

Definition at line 26 of file der_length_integer.c.

References CRYPT_OK, len, LTC_ARGCHK, LTC_MP_LT, and LTC_MP_YES.

Referenced by der_decode_sequence_flexi(), and der_encode_integer().

```
{
   unsigned long z, len;
   int           leading_zero;

   LTC_ARGCHK(num     != NULL);
   LTC_ARGCHK(outlen  != NULL);

   if (mp_cmp_d(num, 0) != LTC_MP_LT) {
      /* positive */

      /* we only need a leading zero if the msb of the first byte is one */
      if ((mp_count_bits(num) & 7) == 0 || mp_iszero(num) == LTC_MP_YES) {
         leading_zero = 1;
      } else {
         leading_zero = 0;
      }

      /* size for bignum */
      z = len = leading_zero + mp_unsigned_bin_size(num);
   } else {
```

```
      /* it's negative */
      /* find power of 2 that is a multiple of eight and greater than count bits */
      leading_zero = 0;
      z = mp_count_bits(num);
      z = z + (8 - (z & 7));
      if (((mp_cnt_lsb(num)+1)==mp_count_bits(num)) && ((mp_count_bits(num)&7)==0)) --z;
      len = z = z >> 3;
   }

   /* now we need a length */
   if (z < 128) {
      /* short form */
      ++len;
   } else {
      /* long form (relies on z != 0), assumes length bytes < 128 */
      ++len;

      while (z) {
         ++len;
         z >>= 8;
      }
   }

   /* we need a 0x02 to indicate it's INTEGER */
   ++len;

   /* return length */
   *outlen = len;
   return CRYPT_OK;
}
```

5.219 pk/asn1/der/object_identifier/der_decode_object_identifier.c File Reference

5.219.1 Detailed Description

ASN.1 DER, Decode Object Identifier, Tom St Denis.

Definition in file der_decode_object_identifier.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_object_identifier.c:

Functions

- int der_decode_object_identifier (const unsigned char ∗in, unsigned long inlen, unsigned long ∗words, unsigned long ∗outlen)

  Decode OID data and store the array of integers in words.

5.219.2 Function Documentation

5.219.2.1 int der_decode_object_identifier (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned long ∗ *words*, unsigned long ∗ *outlen*)

Decode OID data and store the array of integers in words.

Parameters:

in The OID DER encoded data

inlen The length of the OID data

words [out] The destination of the OID words

outlen [in/out] The number of OID words

Returns:

CRYPT_OK if successful

Definition at line 27 of file der_decode_object_identifier.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_PACKET, CRYPT_OK, len, and LTC_-ARGCHK.

Referenced by der_decode_sequence_flexi().

```
29 {
30    unsigned long x, y, t, len;
31
32    LTC_ARGCHK(in     != NULL);
33    LTC_ARGCHK(words  != NULL);
34    LTC_ARGCHK(outlen != NULL);
35
36    /* header is at least 3 bytes */
37    if (inlen < 3) {
38       return CRYPT_INVALID_PACKET;
39    }
```

```

   /* must be room for at least two words */
   if (*outlen < 2) {
      return CRYPT_BUFFER_OVERFLOW;
   }

   /* decode the packet header */
   x = 0;
   if ((in[x++] & 0x1F) != 0x06) {
      return CRYPT_INVALID_PACKET;
   }

   /* get the length */
   if (in[x] < 128) {
      len = in[x++];
   } else {
       if (in[x] < 0x81 || in[x] > 0x82) {
          return CRYPT_INVALID_PACKET;
       }
       y   = in[x++] & 0x7F;
       len = 0;
       while (y--) {
          len = (len << 8) | (unsigned long)in[x++];
       }
   }

   if (len < 1 || (len + x) > inlen) {
      return CRYPT_INVALID_PACKET;
   }

   /* decode words */
   y = 0;
   t = 0;
   while (len--) {
       t = (t << 7) | (in[x] & 0x7F);
       if (!(in[x++] & 0x80)) {
           /* store t */
           if (y >= *outlen) {
              return CRYPT_BUFFER_OVERFLOW;
           }
       if (y == 0) {
          words[0] = t / 40;
          words[1] = t % 40;
          y = 2;
       } else {
              words[y++] = t;
       }
           t          = 0;
       }
   }

   *outlen = y;
   return CRYPT_OK;
}
```

5.220 pk/asn1/der/object_identifier/der_encode_object_identifier.c File Reference

5.220.1 Detailed Description

ASN.1 DER, Encode Object Identifier, Tom St Denis.

Definition in file der_encode_object_identifier.c.

`#include "tomcrypt.h"`

Include dependency graph for der_encode_object_identifier.c:

Functions

- int der_encode_object_identifier (unsigned long ∗words, unsigned long nwords, unsigned char ∗out, unsigned long ∗outlen)

    Encode an OID.

5.220.2 Function Documentation

5.220.2.1 int der_encode_object_identifier (unsigned long ∗ *words*, unsigned long *nwords*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Encode an OID.

Parameters:

words The words to encode (upto 32-bits each)

nwords The number of words in the OID

out [out] Destination of OID data

outlen [in/out] The max and resulting size of the OID

Returns:

CRYPT_OK if successful

Definition at line 27 of file der_encode_object_identifier.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, der_length_object_identifier(), der_object_identifier_bits(), LTC_ARGCHK, and mask.

```
{
   unsigned long i, x, y, z, t, mask, wordbuf;
   int           err;

   LTC_ARGCHK(words  != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* check length */
   if ((err = der_length_object_identifier(words, nwords, &x)) != CRYPT_OK) {
      return err;
   }
   if (x > *outlen) {
```

```
      *outlen = x;
      return CRYPT_BUFFER_OVERFLOW;
   }

   /* compute length to store OID data */
   z = 0;
   wordbuf = words[0] * 40 + words[1];
   for (y = 1; y < nwords; y++) {
       t = der_object_identifier_bits(wordbuf);
       z += t/7 + ((t%7) ? 1 : 0) + (wordbuf == 0 ? 1 : 0);
       if (y < nwords - 1) {
          wordbuf = words[y + 1];
       }
   }

   /* store header + length */
   x = 0;
   out[x++] = 0x06;
   if (z < 128) {
      out[x++] = z;
   } else if (z < 256) {
      out[x++] = 0x81;
      out[x++] = z;
   } else if (z < 65536UL) {
      out[x++] = 0x82;
      out[x++] = (z>>8)&255;
      out[x++] = z&255;
   } else {
      return CRYPT_INVALID_ARG;
   }

   /* store first byte */
    wordbuf = words[0] * 40 + words[1];
    for (i = 1; i < nwords; i++) {
        /* store 7 bit words in little endian */
        t    = wordbuf & 0xFFFFFFFF;
        if (t) {
           y    = x;
           mask = 0;
           while (t) {
               out[x++] = (t & 0x7F) | mask;
               t    >>= 7;
               mask  |= 0x80;  /* upper bit is set on all but the last byte */
           }
           /* now swap bytes y...x-1 */
           z = x - 1;
           while (y < z) {
               t = out[y]; out[y] = out[z]; out[z] = t;
               ++y;
               --z;
           }
       } else {
          /* zero word */
          out[x++] = 0x00;
       }

       if (i < nwords - 1) {
          wordbuf = words[i + 1];
        }
    }

    *outlen = x;
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.221 pk/asn1/der/object_identifier/der_length_object_identifier.c File Reference

5.221.1 Detailed Description

ASN.1 DER, get length of Object Identifier, Tom St Denis.

Definition in file der_length_object_identifier.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_object_identifier.c:

Functions

- unsigned long der_object_identifier_bits (unsigned long x)
- int der_length_object_identifier (unsigned long ∗words, unsigned long nwords, unsigned long ∗outlen)

  Gets length of DER encoding of Object Identifier.

5.221.2 Function Documentation

5.221.2.1 int der_length_object_identifier (unsigned long ∗ *words*, unsigned long *nwords*, unsigned long ∗ *outlen*)

Gets length of DER encoding of Object Identifier.

Parameters:

nwords The number of OID words

words The actual OID words to get the size of

outlen [out] The length of the DER encoding for the given string

Returns:

CRYPT_OK if successful

Definition at line 40 of file der_length_object_identifier.c.

References CRYPT_INVALID_ARG, der_object_identifier_bits(), and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi(), and der_encode_object_identifier().

```
{
   unsigned long y, z, t, wordbuf;

   LTC_ARGCHK(words  != NULL);
   LTC_ARGCHK(outlen != NULL);


   /* must be >= 2 words */
   if (nwords < 2) {
      return CRYPT_INVALID_ARG;
   }

   /* word1 = 0,1,2,3 and word2 0..39 */
```

```
   if (words[0] > 3 || (words[0] < 2 && words[1] > 39)) {
      return CRYPT_INVALID_ARG;
   }

   /* leading word is the first two */
   z = 0;
   wordbuf = words[0] * 40 + words[1];
   for (y = 1; y < nwords; y++) {
       t = der_object_identifier_bits(wordbuf);
       z += t/7 + ((t%7) ? 1 : 0) + (wordbuf == 0 ? 1 : 0);
       if (y < nwords - 1) {
          /* grab next word */
          wordbuf = words[y+1];
       }
   }

   /* now depending on the length our length encoding changes */
   if (z < 128) {
      z += 2;
   } else if (z < 256) {
      z += 3;
   } else if (z < 65536UL) {
      z += 4;
   } else {
      return CRYPT_INVALID_ARG;
   }

   *outlen = z;
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.221.2.2 unsigned long der_object_identifier_bits (unsigned long *x*)

Definition at line 20 of file der_length_object_identifier.c.

References c.

Referenced by der_encode_object_identifier(), and der_length_object_identifier().

```
{
   unsigned long c;
   x &= 0xFFFFFFFF;
   c  = 0;
   while (x) {
     ++c;
     x >>= 1;
   }
   return c;
}
```

5.222 pk/asn1/der/octet/der_decode_octet_string.c File Reference

5.222.1 Detailed Description

ASN.1 DER, encode a OCTET STRING, Tom St Denis.

Definition in file der_decode_octet_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_octet_string.c:

Functions

- int der_decode_octet_string (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Store a OCTET STRING.

5.222.2 Function Documentation

5.222.2.1 int der_decode_octet_string (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store a OCTET STRING.

Parameters:

in The DER encoded OCTET STRING

inlen The size of the DER OCTET STRING

out [out] The array of octets stored (one per char)

outlen [in/out] The number of octets stored

Returns:

CRYPT_OK if successful

Definition at line 29 of file der_decode_octet_string.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_PACKET, len, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi().

```
{
   unsigned long x, y, len;

   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* must have header at least */
   if (inlen < 2) {
      return CRYPT_INVALID_PACKET;
   }

   /* check for 0x04 */
   if ((in[0] & 0x1F) != 0x04) {
      return CRYPT_INVALID_PACKET;
```

```
   }
   x = 1;

   /* decode the length */
   if (in[x] & 0x80) {
      /* valid # of bytes in length are 1,2,3 */
      y = in[x] & 0x7F;
      if ((y == 0) || (y > 3) || ((x + y) > inlen)) {
         return CRYPT_INVALID_PACKET;
      }

      /* read the length in */
      len = 0;
      ++x;
      while (y--) {
         len = (len << 8) | in[x++];
      }
   } else {
      len = in[x++] & 0x7F;
   }

   /* is it too long? */
   if (len > *outlen) {
      *outlen = len;
      return CRYPT_BUFFER_OVERFLOW;
   }

   if (len + x > inlen) {
      return CRYPT_INVALID_PACKET;
   }

   /* read the data */
   for (y = 0; y < len; y++) {
       out[y] = in[x++];
   }

   *outlen = y;

   return CRYPT_OK;
}
```

5.223 pk/asn1/der/octet/der_encode_octet_string.c File Reference

5.223.1 Detailed Description

ASN.1 DER, encode a OCTET STRING, Tom St Denis.

Definition in file der_encode_octet_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_encode_octet_string.c:

Functions

- int der_encode_octet_string (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

    Store an OCTET STRING.

5.223.2 Function Documentation

5.223.2.1 int der_encode_octet_string (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store an OCTET STRING.

Parameters:

in The array of OCTETS to store (one per char)

inlen The number of OCTETS to store

out [out] The destination for the DER encoded OCTET STRING

outlen [in/out] The max size and resulting size of the DER OCTET STRING

Returns:

CRYPT_OK if successful

Definition at line 29 of file der_encode_octet_string.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_OK, der_length_octet_string(), len, and LTC_ARGCHK.

```
{
   unsigned long x, y, len;
   int           err;

   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* get the size */
   if ((err = der_length_octet_string(inlen, &len)) != CRYPT_OK) {
      return err;
   }

   /* too big? */
   if (len > *outlen) {
```

```
      *outlen = len;
      return CRYPT_BUFFER_OVERFLOW;
   }

   /* encode the header+len */
   x = 0;
   out[x++] = 0x04;
   if (inlen < 128) {
      out[x++] = inlen;
   } else if (inlen < 256) {
      out[x++] = 0x81;
      out[x++] = inlen;
   } else if (inlen < 65536UL) {
      out[x++] = 0x82;
      out[x++] = (inlen>>8)&255;
      out[x++] = inlen&255;
   } else if (inlen < 16777216UL) {
      out[x++] = 0x83;
      out[x++] = (inlen>>16)&255;
      out[x++] = (inlen>>8)&255;
      out[x++] = inlen&255;
   } else {
      return CRYPT_INVALID_ARG;
   }

   /* store octets */
   for (y = 0; y < inlen; y++) {
       out[x++] = in[y];
   }

   /* retun length */
   *outlen = x;

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.224 pk/asn1/der/octet/der_length_octet_string.c File Reference

5.224.1 Detailed Description

ASN.1 DER, get length of OCTET STRING, Tom St Denis.

Definition in file der_length_octet_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_octet_string.c:

Functions

- int der_length_octet_string (unsigned long noctets, unsigned long ∗outlen)

  Gets length of DER encoding of OCTET STRING.

5.224.2 Function Documentation

5.224.2.1 int der_length_octet_string (unsigned long *noctets*, unsigned long ∗ *outlen*)

Gets length of DER encoding of OCTET STRING.

Parameters:

noctets The number of octets in the string to encode

outlen [out] The length of the DER encoding for the given string

Returns:

CRYPT_OK if successful

Definition at line 25 of file der_length_octet_string.c.

References CRYPT_INVALID_ARG, CRYPT_OK, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi(), and der_encode_octet_string().

```
{
   LTC_ARGCHK(outlen != NULL);

   if (noctets < 128) {
      /* 04 LL DD DD DD ... */
      *outlen = 2 + noctets;
   } else if (noctets < 256) {
      /* 04 81 LL DD DD DD ... */
      *outlen = 3 + noctets;
   } else if (noctets < 65536UL) {
      /* 04 82 LL LL DD DD DD ... */
      *outlen = 4 + noctets;
   } else if (noctets < 16777216UL) {
      /* 04 83 LL LL LL DD DD DD ... */
      *outlen = 5 + noctets;
   } else {
      return CRYPT_INVALID_ARG;
   }

   return CRYPT_OK;
}
```

5.225 pk/asn1/der/printable_string/der_decode_printable_string.c File Reference

5.225.1 Detailed Description

ASN.1 DER, encode a printable STRING, Tom St Denis.

Definition in file der_decode_printable_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_printable_string.c:

Functions

- int der_decode_printable_string (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Store a printable STRING.

5.225.2 Function Documentation

5.225.2.1 int der_decode_printable_string (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store a printable STRING.

Parameters:

in The DER encoded printable STRING

inlen The size of the DER printable STRING

out [out] The array of octets stored (one per char)

outlen [in/out] The number of octets stored

Returns:

CRYPT_OK if successful

Definition at line 29 of file der_decode_printable_string.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_INVALID_PACKET, der_printable_value_decode(), len, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi().

```
{
   unsigned long x, y, len;
   int           t;

   LTC_ARGCHK(in     != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* must have header at least */
   if (inlen < 2) {
      return CRYPT_INVALID_PACKET;
```

```
   }

   /* check for 0x13 */
   if ((in[0] & 0x1F) != 0x13) {
      return CRYPT_INVALID_PACKET;
   }
   x = 1;

   /* decode the length */
   if (in[x] & 0x80) {
      /* valid # of bytes in length are 1,2,3 */
      y = in[x] & 0x7F;
      if ((y == 0) || (y > 3) || ((x + y) > inlen)) {
         return CRYPT_INVALID_PACKET;
      }

      /* read the length in */
      len = 0;
      ++x;
      while (y--) {
         len = (len << 8) | in[x++];
      }
   } else {
      len = in[x++] & 0x7F;
   }

   /* is it too long? */
   if (len > *outlen) {
      *outlen = len;
      return CRYPT_BUFFER_OVERFLOW;
   }

   if (len + x > inlen) {
      return CRYPT_INVALID_PACKET;
   }

   /* read the data */
   for (y = 0; y < len; y++) {
       t = der_printable_value_decode(in[x++]);
       if (t == -1) {
           return CRYPT_INVALID_ARG;
       }
       out[y] = t;
   }

   *outlen = y;

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.226 pk/asn1/der/printable_string/der_encode_printable_string.c File Reference

5.226.1 Detailed Description

ASN.1 DER, encode a printable STRING, Tom St Denis.

Definition in file der_encode_printable_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_encode_printable_string.c:

Functions

- int der_encode_printable_string (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

    Store an printable STRING.

5.226.2 Function Documentation

5.226.2.1 int der_encode_printable_string (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store an printable STRING.

Parameters:

in The array of printable to store (one per char)

inlen The number of printable to store

out [out] The destination for the DER encoded printable STRING

outlen [in/out] The max size and resulting size of the DER printable STRING

Returns:

CRYPT_OK if successful

Definition at line 28 of file der_encode_printable_string.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_OK, der_length_printable_string(), der_printable_char_encode(), len, and LTC_ARGCHK.

```
30 {
31    unsigned long x, y, len;
32    int           err;
33
34    LTC_ARGCHK(in     != NULL);
35    LTC_ARGCHK(out    != NULL);
36    LTC_ARGCHK(outlen != NULL);
37
38    /* get the size */
39    if ((err = der_length_printable_string(in, inlen, &len)) != CRYPT_OK) {
40       return err;
41    }
42
```

```
   /* too big? */
   if (len > *outlen) {
      *outlen = len;
      return CRYPT_BUFFER_OVERFLOW;
   }

   /* encode the header+len */
   x = 0;
   out[x++] = 0x13;
   if (inlen < 128) {
      out[x++] = inlen;
   } else if (inlen < 256) {
      out[x++] = 0x81;
      out[x++] = inlen;
   } else if (inlen < 65536UL) {
      out[x++] = 0x82;
      out[x++] = (inlen>>8)&255;
      out[x++] = inlen&255;
   } else if (inlen < 16777216UL) {
      out[x++] = 0x83;
      out[x++] = (inlen>>16)&255;
      out[x++] = (inlen>>8)&255;
      out[x++] = inlen&255;
   } else {
      return CRYPT_INVALID_ARG;
   }

   /* store octets */
   for (y = 0; y < inlen; y++) {
       out[x++] = der_printable_char_encode(in[y]);
   }

   /* retun length */
   *outlen = x;

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.227 pk/asn1/der/printable_string/der_length_printable_string.c File Reference

5.227.1 Detailed Description

ASN.1 DER, get length of Printable STRING, Tom St Denis.

Definition in file der_length_printable_string.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_printable_string.c:

Functions

- int der_printable_char_encode (int c)
- int der_printable_value_decode (int v)
- int der_length_printable_string (const unsigned char *octets, unsigned long noctets, unsigned long *outlen)

    Gets length of DER encoding of Printable STRING.

Variables

- struct {
    int len
    unsigned char poly_div [MAXBLOCKSIZE]
    unsigned char poly_mul [MAXBLOCKSIZE]
    int code
    int value
  } printable_table [ ]

5.227.2 Function Documentation

5.227.2.1 int der_length_printable_string (const unsigned char * *octets*, unsigned long *noctets*, unsigned long * *outlen*)

Gets length of DER encoding of Printable STRING.

Parameters:

octets The values you want to encode

noctets The number of octets in the string to encode

outlen [out] The length of the DER encoding for the given string

Returns:

CRYPT_OK if successful

Definition at line 128 of file der_length_printable_string.c.

References CRYPT_INVALID_ARG, der_printable_char_encode(), and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi(), and der_encode_printable_string().

```
{
   unsigned long x;

   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(octets != NULL);

   /* scan string for validity */
   for (x = 0; x < noctets; x++) {
       if (der_printable_char_encode(octets[x]) == -1) {
          return CRYPT_INVALID_ARG;
       }
   }

   if (noctets < 128) {
      /* 16 LL DD DD DD ... */
      *outlen = 2 + noctets;
   } else if (noctets < 256) {
      /* 16 81 LL DD DD DD ... */
      *outlen = 3 + noctets;
   } else if (noctets < 65536UL) {
      /* 16 82 LL LL DD DD DD ... */
      *outlen = 4 + noctets;
   } else if (noctets < 16777216UL) {
      /* 16 83 LL LL LL DD DD DD ... */
      *outlen = 5 + noctets;
   } else {
      return CRYPT_INVALID_ARG;
   }

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.227.2.2 int der_printable_char_encode (int *c*)

Definition at line 99 of file der_length_printable_string.c.

References code, printable_table, and value.

Referenced by der_encode_printable_string(), and der_length_printable_string().

```
{
   int x;
   for (x = 0; x < (int)(sizeof(printable_table)/sizeof(printable_table[0])); x++) {
       if (printable_table[x].code == c) {
          return printable_table[x].value;
       }
   }
   return -1;
}
```

5.227.2.3 int der_printable_value_decode (int *v*)

Definition at line 110 of file der_length_printable_string.c.

References code, printable_table, and value.

Referenced by der_decode_printable_string().

```
{
```

```
    int x;
    for (x = 0; x < (int)(sizeof(printable_table)/sizeof(printable_table[0])); x++) {
        if (printable_table[x].value == v) {
           return printable_table[x].code;
        }
    }
    return -1;
}
```

5.227.3 Variable Documentation

5.227.3.1 int code

Definition at line 21 of file der_length_printable_string.c.

5.227.3.2 const { ... } printable_table[ ] [static]

Referenced by der_printable_char_encode(), and der_printable_value_decode().

5.227.3.3 int value

Definition at line 21 of file der_length_printable_string.c.

5.228 pk/asn1/der/sequence/der_decode_sequence_ex.c File Reference

5.228.1 Detailed Description

ASN.1 DER, decode a SEQUENCE, Tom St Denis.

Definition in file der_decode_sequence_ex.c.

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for der_decode_sequence_ex.c:

Functions

- int der_decode_sequence_ex (const unsigned char ∗in, unsigned long inlen, ltc_asn1_list ∗list, unsigned long outlen, int ordered)

    Decode a SEQUENCE.

5.228.2 Function Documentation

5.228.2.1 int der_decode_sequence_ex (const unsigned char ∗ *in*, unsigned long *inlen*, ltc_asn1_list ∗ *list*, unsigned long *outlen*, int *ordered*)

Decode a SEQUENCE.

Parameters:

in The DER encoded input

inlen The size of the input

list The list of items to decode

outlen The number of items in the list

ordered Search an unordeded or ordered list

Returns:

CRYPT_OK on success

Definition at line 31 of file der_decode_sequence_ex.c.

References CRYPT_INVALID_PACKET, and LTC_ARGCHK.

```
{
   int           err, type;
   unsigned long size, x, y, z, i, blksize;
   void          *data;

   LTC_ARGCHK(in   != NULL);
   LTC_ARGCHK(list != NULL);

   /* get blk size */
   if (inlen < 2) {
      return CRYPT_INVALID_PACKET;
```

```
   }

   /* sequence type? We allow 0x30 SEQUENCE and 0x31 SET since fundamentally they're the same structure
   x = 0;
   if (in[x] != 0x30 && in[x] != 0x31) {
      return CRYPT_INVALID_PACKET;
   }
   ++x;

   if (in[x] < 128) {
      blksize = in[x++];
   } else if (in[x] & 0x80) {
      if (in[x] < 0x81 || in[x] > 0x83) {
         return CRYPT_INVALID_PACKET;
      }
      y = in[x++] & 0x7F;

      /* would reading the len bytes overrun? */
      if (x + y > inlen) {
         return CRYPT_INVALID_PACKET;
      }

      /* read len */
      blksize = 0;
      while (y--) {
          blksize = (blksize << 8) | (unsigned long)in[x++];
      }
  }

  /* would this blksize overflow? */
  if (x + blksize > inlen) {
     return CRYPT_INVALID_PACKET;
  }

   /* mark all as unused */
   for (i = 0; i < outlen; i++) {
       list[i].used = 0;
   }

  /* ok read data */
   inlen = blksize;
   for (i = 0; i < outlen; i++) {
       z    = 0;
       type = list[i].type;
       size = list[i].size;
       data = list[i].data;
       if (!ordered && list[i].used == 1) { continue; }

       if (type == LTC_ASN1_EOL) {
          break;
       }

       switch (type) {
           case LTC_ASN1_BOOLEAN:
                z = inlen;
                if ((err = der_decode_boolean(in + x, z, ((int *)data))) != CRYPT_OK) {
                     goto LBL_ERR;
                 }
                 if ((err = der_length_boolean(&z)) != CRYPT_OK) {
                     goto LBL_ERR;
                  }
                  break;

            case LTC_ASN1_INTEGER:
                z = inlen;
                if ((err = der_decode_integer(in + x, z, data)) != CRYPT_OK) {
                   if (!ordered) {  continue; }
```

```
                goto LBL_ERR;
             }
             if ((err = der_length_integer(data, &z)) != CRYPT_OK) {
                goto LBL_ERR;
             }
             break;

         case LTC_ASN1_SHORT_INTEGER:
             z = inlen;
             if ((err = der_decode_short_integer(in + x, z, data)) != CRYPT_OK) {
                if (!ordered) { continue; }
                goto LBL_ERR;
             }
             if ((err = der_length_short_integer(((unsigned long*)data)[0], &z)) != CRYPT_OK) {
                goto LBL_ERR;
             }

             break;

         case LTC_ASN1_BIT_STRING:
             z = inlen;
             if ((err = der_decode_bit_string(in + x, z, data, &size)) != CRYPT_OK) {
                if (!ordered) { continue; }
                goto LBL_ERR;
             }
             list[i].size = size;
             if ((err = der_length_bit_string(size, &z)) != CRYPT_OK) {
                goto LBL_ERR;
             }
             break;

         case LTC_ASN1_OCTET_STRING:
             z = inlen;
             if ((err = der_decode_octet_string(in + x, z, data, &size)) != CRYPT_OK) {
                if (!ordered) { continue; }
                goto LBL_ERR;
             }
             list[i].size = size;
             if ((err = der_length_octet_string(size, &z)) != CRYPT_OK) {
                goto LBL_ERR;
             }
             break;

         case LTC_ASN1_NULL:
             if (inlen < 2 || in[x] != 0x05 || in[x+1] != 0x00) {
                if (!ordered) { continue; }
                err = CRYPT_INVALID_PACKET;
                goto LBL_ERR;
             }
             z = 2;
             break;

         case LTC_ASN1_OBJECT_IDENTIFIER:
             z = inlen;
             if ((err = der_decode_object_identifier(in + x, z, data, &size)) != CRYPT_OK) {
                if (!ordered) { continue; }
                goto LBL_ERR;
             }
             list[i].size = size;
             if ((err = der_length_object_identifier(data, size, &z)) != CRYPT_OK) {
                goto LBL_ERR;
             }
             break;

         case LTC_ASN1_IA5_STRING:
             z = inlen;
             if ((err = der_decode_ia5_string(in + x, z, data, &size)) != CRYPT_OK) {
```

```
                   if (!ordered) { continue; }
                   goto LBL_ERR;
                }
                list[i].size = size;
                if ((err = der_length_ia5_string(data, size, &z)) != CRYPT_OK) {
                   goto LBL_ERR;
                }
                break;


            case LTC_ASN1_PRINTABLE_STRING:
                z = inlen;
                if ((err = der_decode_printable_string(in + x, z, data, &size)) != CRYPT_OK) {
                   if (!ordered) { continue; }
                   goto LBL_ERR;
                }
                list[i].size = size;
                if ((err = der_length_printable_string(data, size, &z)) != CRYPT_OK) {
                   goto LBL_ERR;
                }
                break;

            case LTC_ASN1_UTCTIME:
                z = inlen;
                if ((err = der_decode_utctime(in + x, &z, data)) != CRYPT_OK) {
                   if (!ordered) { continue; }
                   goto LBL_ERR;
                }
                break;

            case LTC_ASN1_SET:
                z = inlen;
                if ((err = der_decode_set(in + x, z, data, size)) != CRYPT_OK) {
                   if (!ordered) { continue; }
                   goto LBL_ERR;
                }
                if ((err = der_length_sequence(data, size, &z)) != CRYPT_OK) {
                   goto LBL_ERR;
                }
                break;

            case LTC_ASN1_SETOF:
            case LTC_ASN1_SEQUENCE:
                /* detect if we have the right type */
                if ((type == LTC_ASN1_SETOF && (in[x] & 0x3F) != 0x31) || (type == LTC_ASN1_SEQUENCE &&
                   err = CRYPT_INVALID_PACKET;
                   goto LBL_ERR;
                }

                z = inlen;
                if ((err = der_decode_sequence(in + x, z, data, size)) != CRYPT_OK) {
                   if (!ordered) { continue; }
                   goto LBL_ERR;
                }
                if ((err = der_length_sequence(data, size, &z)) != CRYPT_OK) {
                   goto LBL_ERR;
                }
                break;


            case LTC_ASN1_CHOICE:
                z = inlen;
                if ((err = der_decode_choice(in + x, &z, data, size)) != CRYPT_OK) {
                   if (!ordered) { continue; }
                   goto LBL_ERR;
                }
                break;
```

```

            default:
                err = CRYPT_INVALID_ARG;
                goto LBL_ERR;
        }
        x           += z;
        inlen       -= z;
        list[i].used = 1;
        if (!ordered) {
           /* restart the decoder */
           i = -1;
        }
    }

    for (i = 0; i < outlen; i++) {
       if (list[i].used == 0) {
           err = CRYPT_INVALID_PACKET;
           goto LBL_ERR;
       }
    }
    err = CRYPT_OK;

LBL_ERR:
    return err;
}
```

5.229 pk/asn1/der/sequence/der_decode_sequence_flexi.c File Reference

5.229.1 Detailed Description

ASN.1 DER, decode an array of ASN.1 types with a flexi parser, Tom St Denis.

Definition in file der_decode_sequence_flexi.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_sequence_flexi.c:

Functions

- static unsigned long fetch_length (const unsigned char ∗in, unsigned long inlen)
- int der_decode_sequence_flexi (const unsigned char ∗in, unsigned long ∗inlen, ltc_asn1_list ∗∗out)

  ASN.1 DER Flexi(ble) decoder will decode arbitrary DER packets and create a linked list of the decoded elements.

5.229.2 Function Documentation

5.229.2.1 int der_decode_sequence_flexi (const unsigned char ∗ *in*, unsigned long ∗ *inlen*, ltc_asn1_list ∗∗ *out*)

ASN.1 DER Flexi(ble) decoder will decode arbitrary DER packets and create a linked list of the decoded elements.

Parameters:

in The input buffer

inlen [in/out] The length of the input buffer and on output the amount of decoded data

out [out] A pointer to the linked list

Returns:

CRYPT_OK on success.

Definition at line 63 of file der_decode_sequence_flexi.c.

References CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, der_decode_bit_string(), der_decode_boolean(), der_decode_ia5_string(), der_decode_integer(), der_decode_object_identifier(), der_decode_octet_string(), der_decode_printable_string(), der_decode_utctime(), der_length_bit_string(), der_length_boolean(), der_length_ia5_string(), der_length_integer(), der_length_object_identifier(), der_length_octet_string(), der_length_printable_string(), der_length_utctime(), der_sequence_free(), fetch_length(), len, LTC_ARGCHK, XCALLOC, XFREE, and XREALLOC.

```
{
   ltc_asn1_list *l;
   unsigned long err, type, len, totlen, x, y;
   void          *realloc_tmp;

   LTC_ARGCHK(in    != NULL);
   LTC_ARGCHK(inlen != NULL);
```

```
   LTC_ARGCHK(out   != NULL);

   l = NULL;
   totlen = 0;

   /* scan the input and and get lengths and what not */
   while (*inlen) {
      /* read the type byte */
      type = *in;

      /* fetch length */
      len = fetch_length(in, *inlen);
      if (len > *inlen) {
         err = CRYPT_INVALID_PACKET;
         goto error;
      }

      /* alloc new link */
      if (l == NULL) {
         l = XCALLOC(1, sizeof(*l));
         if (l == NULL) {
            err = CRYPT_MEM;
            goto error;
         }
      } else {
         l->next = XCALLOC(1, sizeof(*l));
         if (l->next == NULL) {
            err = CRYPT_MEM;
            goto error;
          }
          l->next->prev = l;
          l = l->next;
       }

       /* now switch on type */
       switch (type) {
          case 0x01: /* BOOLEAN */
              l->type = LTC_ASN1_BOOLEAN;
              l->size = 1;
              l->data = XCALLOC(1, sizeof(int));

              if ((err = der_decode_boolean(in, *inlen, l->data)) != CRYPT_OK) {
                 goto error;
             }

              if ((err = der_length_boolean(&len)) != CRYPT_OK) {
                 goto error;
             }
              break;

          case 0x02: /* INTEGER */
              /* init field */
              l->type = LTC_ASN1_INTEGER;
              l->size = 1;
              if ((err = mp_init(&l->data)) != CRYPT_OK) {
                  goto error;
              }

              /* decode field */
              if ((err = der_decode_integer(in, *inlen, l->data)) != CRYPT_OK) {
                  goto error;
              }

              /* calc length of object */
              if ((err = der_length_integer(l->data, &len)) != CRYPT_OK) {
                  goto error;
              }
```

```
             break;

         case 0x03: /* BIT */
            /* init field */
            l->type = LTC_ASN1_BIT_STRING;
            l->size = len * 8; /* *8 because we store decoded bits one per char and they are encoded 8

            if ((l->data = XCALLOC(1, l->size)) == NULL) {
               err = CRYPT_MEM;
               goto error;
            }

            if ((err = der_decode_bit_string(in, *inlen, l->data, &l->size)) != CRYPT_OK) {
               goto error;
            }

            if ((err = der_length_bit_string(l->size, &len)) != CRYPT_OK) {
               goto error;
            }
            break;

         case 0x04: /* OCTET */

            /* init field */
            l->type = LTC_ASN1_OCTET_STRING;
            l->size = len;

            if ((l->data = XCALLOC(1, l->size)) == NULL) {
               err = CRYPT_MEM;
               goto error;
            }

            if ((err = der_decode_octet_string(in, *inlen, l->data, &l->size)) != CRYPT_OK) {
               goto error;
            }

            if ((err = der_length_octet_string(l->size, &len)) != CRYPT_OK) {
               goto error;
            }
            break;

         case 0x05: /* NULL */

            /* valid NULL is 0x05 0x00 */
            if (in[0] != 0x05 || in[1] != 0x00) {
               err = CRYPT_INVALID_PACKET;
               goto error;
            }

            /* simple to store ;-) */
            l->type = LTC_ASN1_NULL;
            l->data = NULL;
            l->size = 0;
            len     = 2;

            break;

         case 0x06: /* OID */

            /* init field */
            l->type = LTC_ASN1_OBJECT_IDENTIFIER;
            l->size = len;

            if ((l->data = XCALLOC(len, sizeof(unsigned long))) == NULL) {
               err = CRYPT_MEM;
               goto error;
            }
```

```
            if ((err = der_decode_object_identifier(in, *inlen, l->data, &l->size)) != CRYPT_OK) {
               goto error;
            }

            if ((err = der_length_object_identifier(l->data, l->size, &len)) != CRYPT_OK) {
               goto error;
            }

            /* resize it to save a bunch of mem */
            if ((realloc_tmp = XREALLOC(l->data, l->size * sizeof(unsigned long))) == NULL) {
               /* out of heap but this is not an error */
               break;
            }
            l->data = realloc_tmp;
            break;


         case 0x13: /* PRINTABLE */

            /* init field */
            l->type = LTC_ASN1_PRINTABLE_STRING;
            l->size = len;

            if ((l->data = XCALLOC(1, l->size)) == NULL) {
               err = CRYPT_MEM;
               goto error;
            }

            if ((err = der_decode_printable_string(in, *inlen, l->data, &l->size)) != CRYPT_OK) {
               goto error;
            }

            if ((err = der_length_printable_string(l->data, l->size, &len)) != CRYPT_OK) {
               goto error;
            }
            break;

         case 0x16: /* IA5 */

            /* init field */
            l->type = LTC_ASN1_IA5_STRING;
            l->size = len;

            if ((l->data = XCALLOC(1, l->size)) == NULL) {
               err = CRYPT_MEM;
               goto error;
            }

            if ((err = der_decode_ia5_string(in, *inlen, l->data, &l->size)) != CRYPT_OK) {
               goto error;
            }

            if ((err = der_length_ia5_string(l->data, l->size, &len)) != CRYPT_OK) {
               goto error;
            }
            break;

         case 0x17: /* UTC TIME */

            /* init field */
            l->type = LTC_ASN1_UTCTIME;
            l->size = 1;

            if ((l->data = XCALLOC(1, sizeof(ltc_utctime))) == NULL) {
               err = CRYPT_MEM;
               goto error;
```

```
            }

            len = *inlen;
            if ((err = der_decode_utctime(in, &len, l->data)) != CRYPT_OK) {
               goto error;
            }

            if ((err = der_length_utctime(l->data, &len)) != CRYPT_OK) {
               goto error;
            }
            break;

         case 0x30: /* SEQUENCE */
         case 0x31: /* SET */

             /* init field */
             l->type = (type == 0x30) ? LTC_ASN1_SEQUENCE : LTC_ASN1_SET;

             /* we have to decode the SEQUENCE header and get it's length */

                /* move past type */
                ++in; --(*inlen);

                /* read length byte */
                x = *in++; --(*inlen);

                /* smallest SEQUENCE/SET header */
                y = 2;

                /* now if it's > 127 the next bytes are the length of the length */
                if (x > 128) {
                   x      &= 0x7F;
                   in     += x;
                   *inlen -= x;

                   /* update sequence header len */
                   y      += x;
                }

             /* Sequence elements go as child */
             len = len - y;
             if ((err = der_decode_sequence_flexi(in, &len, &(l->child))) != CRYPT_OK) {
                goto error;
             }

             /* len update */
             totlen += y;

             /* link them up y0 */
             l->child->parent = l;

             break;
         default:
           /* invalid byte ... this is a soft error */
           /* remove link */
           l       = l->prev;
           XFREE(l->next);
           l->next = NULL;
           goto outside;
      }

      /* advance pointers */
      totlen  += len;
      in      += len;
      *inlen  -= len;
   }

```

```
outside:

   /* rewind l please */
   while (l->prev != NULL || l->parent != NULL) {
      if (l->parent != NULL) {
         l = l->parent;
      } else {
         l = l->prev;
      }
   }

   /* return */
   *out   = l;
   *inlen = totlen;
   return CRYPT_OK;

error:
   /* free list */
   der_sequence_free(l);

   return err;
}
```

Here is the call graph for this function:

5.229.2.2 static unsigned long fetch_length (const unsigned char * *in*, unsigned long *inlen*) [static]

Definition at line 20 of file der_decode_sequence_flexi.c.

Referenced by der_decode_sequence_flexi().

```
{
   unsigned long x, y, z;

   y = 0;

   /* skip type and read len */
   if (inlen < 2) {
      return 0xFFFFFFFF;
   }
   ++in; ++y;

   /* read len */
   x = *in++; ++y;

   /* <128 means literal */
   if (x < 128) {
      return x+y;
   }
   x     &= 0x7F; /* the lower 7 bits are the length of the length */
   inlen -= 2;

   /* len means len of len! */
   if (x == 0 || x > 4 || x > inlen) {
      return 0xFFFFFFFF;
   }

   y += x;
   z = 0;
   while (x--) {
      z = (z<<8) | ((unsigned long)*in);
      ++in;
   }
```

```
   return z+y;
}
```

5.230 pk/asn1/der/sequence/der_decode_sequence_multi.c File Reference

5.230.1 Detailed Description

ASN.1 DER, decode a SEQUENCE, Tom St Denis.

Definition in file der_decode_sequence_multi.c.

```
#include "tomcrypt.h"
#include <stdarg.h>
```

Include dependency graph for der_decode_sequence_multi.c:

Functions

- int der_decode_sequence_multi (const unsigned char ∗in, unsigned long inlen,...)
  Decode a SEQUENCE type using a VA list.

5.230.2 Function Documentation

5.230.2.1 int der_decode_sequence_multi (const unsigned char ∗ *in*, unsigned long *inlen*, ...)

Decode a SEQUENCE type using a VA list.

Parameters:
- ***in*** Input buffer
- ***inlen*** Length of input in octets

Remarks:
<...> is of the form <type, size, data> (int, unsigned long, void∗)

Returns:
CRYPT_OK on success

Definition at line 29 of file der_decode_sequence_multi.c.

References CRYPT_INVALID_ARG, and LTC_ARGCHK.

Referenced by dsa_import(), dsa_verify_hash(), ecc_import(), and ecc_verify_hash().

```
{
   int           err, type;
   unsigned long size, x;
   void          *data;
   va_list       args;
   ltc_asn1_list *list;

   LTC_ARGCHK(in    != NULL);

   /* get size of output that will be required */
   va_start(args, inlen);
   x = 0;
   for (;;) {
```

```
        type = va_arg(args, int);
        size = va_arg(args, unsigned long);
        data = va_arg(args, void*);

        if (type == LTC_ASN1_EOL) {
           break;
        }

        switch (type) {
            case LTC_ASN1_BOOLEAN:
            case LTC_ASN1_INTEGER:
            case LTC_ASN1_SHORT_INTEGER:
            case LTC_ASN1_BIT_STRING:
            case LTC_ASN1_OCTET_STRING:
            case LTC_ASN1_NULL:
            case LTC_ASN1_OBJECT_IDENTIFIER:
            case LTC_ASN1_IA5_STRING:
            case LTC_ASN1_PRINTABLE_STRING:
            case LTC_ASN1_UTCTIME:
            case LTC_ASN1_SET:
            case LTC_ASN1_SETOF:
            case LTC_ASN1_SEQUENCE:
            case LTC_ASN1_CHOICE:
                 ++x;
                 break;

            default:
                va_end(args);
                return CRYPT_INVALID_ARG;
        }
    }
    va_end(args);

    /* allocate structure for x elements */
    if (x == 0) {
       return CRYPT_NOP;
    }

    list = XCALLOC(sizeof(*list), x);
    if (list == NULL) {
       return CRYPT_MEM;
    }

    /* fill in the structure */
    va_start(args, inlen);
    x = 0;
    for (;;) {
        type = va_arg(args, int);
        size = va_arg(args, unsigned long);
        data = va_arg(args, void*);

        if (type == LTC_ASN1_EOL) {
           break;
        }

        switch (type) {
            case LTC_ASN1_BOOLEAN:
            case LTC_ASN1_INTEGER:
            case LTC_ASN1_SHORT_INTEGER:
            case LTC_ASN1_BIT_STRING:
            case LTC_ASN1_OCTET_STRING:
            case LTC_ASN1_NULL:
            case LTC_ASN1_OBJECT_IDENTIFIER:
            case LTC_ASN1_IA5_STRING:
            case LTC_ASN1_PRINTABLE_STRING:
            case LTC_ASN1_UTCTIME:
            case LTC_ASN1_SEQUENCE:
```

```
            case LTC_ASN1_SET:
            case LTC_ASN1_SETOF:
            case LTC_ASN1_CHOICE:
                 list[x].type   = type;
                 list[x].size   = size;
                 list[x++].data = data;
                 break;

            default:
                va_end(args);
                err = CRYPT_INVALID_ARG;
                goto LBL_ERR;
        }
    }
    va_end(args);

    err = der_decode_sequence(in, inlen, list, x);
LBL_ERR:
    XFREE(list);
    return err;
}
```

5.231 pk/asn1/der/sequence/der_encode_sequence_ex.c File Reference

5.231.1 Detailed Description

ASN.1 DER, encode a SEQUENCE, Tom St Denis.

Definition in file der_encode_sequence_ex.c.

`#include "tomcrypt.h"`

`#include <stdarg.h>`

Include dependency graph for der_encode_sequence_ex.c:

Functions

- int der_encode_sequence_ex (ltc_asn1_list ∗list, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, int type_of)

  Encode a SEQUENCE.

5.231.2 Function Documentation

5.231.2.1 int der_encode_sequence_ex (ltc_asn1_list ∗ *list*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, int *type_of*)

Encode a SEQUENCE.

Parameters:

list The list of items to encode

inlen The number of items in the list

out [out] The destination

outlen [in/out] The size of the output

type_of LTC_ASN1_SEQUENCE or LTC_ASN1_SET/LTC_ASN1_SETOF

Returns:

CRYPT_OK on success

Definition at line 31 of file der_encode_sequence_ex.c.

References LTC_ARGCHK.

```
{
   int           err, type;
   unsigned long size, x, y, z, i;
   void          *data;

   LTC_ARGCHK(list    != NULL);
   LTC_ARGCHK(out     != NULL);
   LTC_ARGCHK(outlen  != NULL);

   /* get size of output that will be required */
   y = 0;
```

```
   for (i = 0; i < inlen; i++) {
       type = list[i].type;
       size = list[i].size;
       data = list[i].data;

       if (type == LTC_ASN1_EOL) {
          break;
       }

       switch (type) {
            case LTC_ASN1_BOOLEAN:
                if ((err = der_length_boolean(&x)) != CRYPT_OK) {
                   goto LBL_ERR;
                }
                y += x;
                break;

           case LTC_ASN1_INTEGER:
               if ((err = der_length_integer(data, &x)) != CRYPT_OK) {
                  goto LBL_ERR;
               }
               y += x;
               break;

           case LTC_ASN1_SHORT_INTEGER:
               if ((err = der_length_short_integer(*((unsigned long*)data), &x)) != CRYPT_OK) {
                  goto LBL_ERR;
               }
               y += x;
               break;

           case LTC_ASN1_BIT_STRING:
               if ((err = der_length_bit_string(size, &x)) != CRYPT_OK) {
                  goto LBL_ERR;
               }
               y += x;
               break;

           case LTC_ASN1_OCTET_STRING:
               if ((err = der_length_octet_string(size, &x)) != CRYPT_OK) {
                  goto LBL_ERR;
               }
               y += x;
               break;

           case LTC_ASN1_NULL:
               y += 2;
               break;

           case LTC_ASN1_OBJECT_IDENTIFIER:
               if ((err = der_length_object_identifier(data, size, &x)) != CRYPT_OK) {
                  goto LBL_ERR;
               }
               y += x;
               break;

            case LTC_ASN1_IA5_STRING:
                if ((err = der_length_ia5_string(data, size, &x)) != CRYPT_OK) {
                   goto LBL_ERR;
                }
                y += x;
                break;

            case LTC_ASN1_PRINTABLE_STRING:
                if ((err = der_length_printable_string(data, size, &x)) != CRYPT_OK) {
                   goto LBL_ERR;
                }
```

```
                y += x;
                break;

            case LTC_ASN1_UTCTIME:
                if ((err = der_length_utctime(data, &x)) != CRYPT_OK) {
                   goto LBL_ERR;
                }
                y += x;
                break;

            case LTC_ASN1_SET:
            case LTC_ASN1_SETOF:
            case LTC_ASN1_SEQUENCE:
                if ((err = der_length_sequence(data, size, &x)) != CRYPT_OK) {
                   goto LBL_ERR;
                }
                y += x;
                break;

            default:
                err = CRYPT_INVALID_ARG;
                goto LBL_ERR;
        }
    }

    /* calc header size */
    z = y;
    if (y < 128) {
       y += 2;
    } else if (y < 256) {
       /* 0x30 0x81 LL */
       y += 3;
    } else if (y < 65536UL) {
       /* 0x30 0x82 LL LL */
       y += 4;
    } else if (y < 16777216UL) {
       /* 0x30 0x83 LL LL LL */
       y += 5;
    } else {
       err = CRYPT_INVALID_ARG;
       goto LBL_ERR;
    }

    /* too big ? */
    if (*outlen < y) {
       *outlen = y;
       err = CRYPT_BUFFER_OVERFLOW;
       goto LBL_ERR;
    }

    /* store header */
    x = 0;
    out[x++] = (type_of == LTC_ASN1_SEQUENCE) ? 0x30 : 0x31;

    if (z < 128) {
       out[x++] = z;
    } else if (z < 256) {
       out[x++] = 0x81;
       out[x++] = z;
    } else if (z < 65536UL) {
       out[x++] = 0x82;
       out[x++] = (z>>8UL)&255;
       out[x++] = z&255;
    } else if (z < 16777216UL) {
       out[x++] = 0x83;
       out[x++] = (z>>16UL)&255;
       out[x++] = (z>>8UL)&255;
```

```
   out[x++] = z&255;
}

/* store data */
*outlen -= x;
for (i = 0; i < inlen; i++) {
    type = list[i].type;
    size = list[i].size;
    data = list[i].data;

    if (type == LTC_ASN1_EOL) {
       break;
    }

    switch (type) {
         case LTC_ASN1_BOOLEAN:
             z = *outlen;
             if ((err = der_encode_boolean(*((int *)data), out + x, &z)) != CRYPT_OK) {
                goto LBL_ERR;
             }
             x       += z;
             *outlen -= z;
             break;

        case LTC_ASN1_INTEGER:
            z = *outlen;
            if ((err = der_encode_integer(data, out + x, &z)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            x       += z;
            *outlen -= z;
            break;

        case LTC_ASN1_SHORT_INTEGER:
            z = *outlen;
            if ((err = der_encode_short_integer(*((unsigned long*)data), out + x, &z)) != CRYPT_OK)
               goto LBL_ERR;
            }
            x       += z;
            *outlen -= z;
            break;

        case LTC_ASN1_BIT_STRING:
            z = *outlen;
            if ((err = der_encode_bit_string(data, size, out + x, &z)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            x       += z;
            *outlen -= z;
            break;

        case LTC_ASN1_OCTET_STRING:
            z = *outlen;
            if ((err = der_encode_octet_string(data, size, out + x, &z)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            x       += z;
            *outlen -= z;
            break;

        case LTC_ASN1_NULL:
            out[x++] = 0x05;
            out[x++] = 0x00;
            *outlen -= 2;
            break;

        case LTC_ASN1_OBJECT_IDENTIFIER:
```

```
              z = *outlen;
              if ((err = der_encode_object_identifier(data, size, out + x, &z)) != CRYPT_OK) {
                 goto LBL_ERR;
              }
              x       += z;
              *outlen -= z;
              break;

          case LTC_ASN1_IA5_STRING:
              z = *outlen;
              if ((err = der_encode_ia5_string(data, size, out + x, &z)) != CRYPT_OK) {
                 goto LBL_ERR;
              }
              x       += z;
              *outlen -= z;
              break;

          case LTC_ASN1_PRINTABLE_STRING:
              z = *outlen;
              if ((err = der_encode_printable_string(data, size, out + x, &z)) != CRYPT_OK) {
                 goto LBL_ERR;
              }
              x       += z;
              *outlen -= z;
              break;

          case LTC_ASN1_UTCTIME:
              z = *outlen;
              if ((err = der_encode_utctime(data, out + x, &z)) != CRYPT_OK) {
                 goto LBL_ERR;
              }
              x       += z;
              *outlen -= z;
              break;

          case LTC_ASN1_SET:
              z = *outlen;
              if ((err = der_encode_set(data, size, out + x, &z)) != CRYPT_OK) {
                 goto LBL_ERR;
              }
              x       += z;
              *outlen -= z;
              break;

          case LTC_ASN1_SETOF:
              z = *outlen;
              if ((err = der_encode_setof(data, size, out + x, &z)) != CRYPT_OK) {
                 goto LBL_ERR;
              }
              x       += z;
              *outlen -= z;
              break;

          case LTC_ASN1_SEQUENCE:
              z = *outlen;
              if ((err = der_encode_sequence_ex(data, size, out + x, &z, type)) != CRYPT_OK) {
                 goto LBL_ERR;
              }
              x       += z;
              *outlen -= z;
              break;

          default:
              err = CRYPT_INVALID_ARG;
              goto LBL_ERR;
      }
   }
```

```
   *outlen = x;
   err = CRYPT_OK;

LBL_ERR:
   return err;
}
```

5.232 pk/asn1/der/sequence/der_encode_sequence_multi.c File Reference

5.232.1 Detailed Description

ASN.1 DER, encode a SEQUENCE, Tom St Denis.

Definition in file der_encode_sequence_multi.c.

```
#include "tomcrypt.h"
#include <stdarg.h>
```

Include dependency graph for der_encode_sequence_multi.c:

Functions

- int der_encode_sequence_multi (unsigned char ∗out, unsigned long ∗outlen,...)
  Encode a SEQUENCE type using a VA list.

5.232.2 Function Documentation

5.232.2.1 int der_encode_sequence_multi (unsigned char ∗ *out*, unsigned long ∗ *outlen*, ...)

Encode a SEQUENCE type using a VA list.

Parameters:

out [out] Destination for data

outlen [in/out] Length of buffer and resulting length of output

Remarks:

<...> is of the form <type, size, data> (int, unsigned long, void∗)

Returns:

CRYPT_OK on success

Definition at line 29 of file der_encode_sequence_multi.c.

References CRYPT_INVALID_ARG, and LTC_ARGCHK.

Referenced by dsa_export(), dsa_sign_hash(), ecc_export(), and rsa_export().

```
{
   int           err, type;
   unsigned long size, x;
   void          *data;
   va_list       args;
   ltc_asn1_list *list;

   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* get size of output that will be required */
   va_start(args, outlen);
   x = 0;
```

```
   for (;;) {
       type = va_arg(args, int);
       size = va_arg(args, unsigned long);
       data = va_arg(args, void*);

       if (type == LTC_ASN1_EOL) {
          break;
       }

       switch (type) {
           case LTC_ASN1_BOOLEAN:
           case LTC_ASN1_INTEGER:
           case LTC_ASN1_SHORT_INTEGER:
           case LTC_ASN1_BIT_STRING:
           case LTC_ASN1_OCTET_STRING:
           case LTC_ASN1_NULL:
           case LTC_ASN1_OBJECT_IDENTIFIER:
           case LTC_ASN1_IA5_STRING:
           case LTC_ASN1_PRINTABLE_STRING:
           case LTC_ASN1_UTCTIME:
           case LTC_ASN1_SEQUENCE:
           case LTC_ASN1_SET:
           case LTC_ASN1_SETOF:
                ++x;
                break;

           default:
               va_end(args);
               return CRYPT_INVALID_ARG;
       }
   }
   va_end(args);

   /* allocate structure for x elements */
   if (x == 0) {
      return CRYPT_NOP;
   }

   list = XCALLOC(sizeof(*list), x);
   if (list == NULL) {
      return CRYPT_MEM;
   }

   /* fill in the structure */
   va_start(args, outlen);
   x = 0;
   for (;;) {
       type = va_arg(args, int);
       size = va_arg(args, unsigned long);
       data = va_arg(args, void*);

       if (type == LTC_ASN1_EOL) {
          break;
       }

       switch (type) {
           case LTC_ASN1_BOOLEAN:
           case LTC_ASN1_INTEGER:
           case LTC_ASN1_SHORT_INTEGER:
           case LTC_ASN1_BIT_STRING:
           case LTC_ASN1_OCTET_STRING:
           case LTC_ASN1_NULL:
           case LTC_ASN1_OBJECT_IDENTIFIER:
           case LTC_ASN1_IA5_STRING:
           case LTC_ASN1_PRINTABLE_STRING:
           case LTC_ASN1_UTCTIME:
           case LTC_ASN1_SEQUENCE:
```

```
           case LTC_ASN1_SET:
           case LTC_ASN1_SETOF:
                list[x].type   = type;
                list[x].size   = size;
                list[x++].data = data;
                break;

           default:
               va_end(args);
               err = CRYPT_INVALID_ARG;
               goto LBL_ERR;
       }
   }
   va_end(args);

   err = der_encode_sequence(list, x, out, outlen);
LBL_ERR:
   XFREE(list);
   return err;
}
```

5.233 pk/asn1/der/sequence/der_length_sequence.c File Reference

5.233.1 Detailed Description

ASN.1 DER, length a SEQUENCE, Tom St Denis.

Definition in file der_length_sequence.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_sequence.c:

Functions

- int der_length_sequence (ltc_asn1_list ∗list, unsigned long inlen, unsigned long ∗outlen)

  Get the length of a DER sequence.

5.233.2 Function Documentation

5.233.2.1 int der_length_sequence (ltc_asn1_list ∗ *list*, unsigned long *inlen*, unsigned long ∗ *outlen*)

Get the length of a DER sequence.

Parameters:

list The sequences of items in the SEQUENCE

inlen The number of items

outlen [out] The length required in octets to store it

Returns:

CRYPT_OK on success

Definition at line 27 of file der_length_sequence.c.

References LTC_ARGCHK.

```
{
   int           err, type;
   unsigned long size, x, y, z, i;
   void          *data;

   LTC_ARGCHK(list    != NULL);
   LTC_ARGCHK(outlen  != NULL);

   /* get size of output that will be required */
   y = 0;
   for (i = 0; i < inlen; i++) {
       type = list[i].type;
       size = list[i].size;
       data = list[i].data;

       if (type == LTC_ASN1_EOL) {
          break;
       }

       switch (type) {
```

```
         case LTC_ASN1_BOOLEAN:
            if ((err = der_length_boolean(&x)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            y += x;
            break;

         case LTC_ASN1_INTEGER:
            if ((err = der_length_integer(data, &x)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            y += x;
            break;

         case LTC_ASN1_SHORT_INTEGER:
            if ((err = der_length_short_integer(*((unsigned long *)data), &x)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            y += x;
            break;

         case LTC_ASN1_BIT_STRING:
            if ((err = der_length_bit_string(size, &x)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            y += x;
            break;

         case LTC_ASN1_OCTET_STRING:
            if ((err = der_length_octet_string(size, &x)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            y += x;
            break;

         case LTC_ASN1_NULL:
            y += 2;
            break;

         case LTC_ASN1_OBJECT_IDENTIFIER:
            if ((err = der_length_object_identifier(data, size, &x)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            y += x;
            break;

         case LTC_ASN1_IA5_STRING:
            if ((err = der_length_ia5_string(data, size, &x)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            y += x;
            break;

         case LTC_ASN1_PRINTABLE_STRING:
            if ((err = der_length_printable_string(data, size, &x)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            y += x;
            break;

         case LTC_ASN1_UTCTIME:
            if ((err = der_length_utctime(data, &x)) != CRYPT_OK) {
               goto LBL_ERR;
            }
            y += x;
            break;

```

```
         case LTC_ASN1_SET:
         case LTC_ASN1_SETOF:
         case LTC_ASN1_SEQUENCE:
             if ((err = der_length_sequence(data, size, &x)) != CRYPT_OK) {
                goto LBL_ERR;
             }
             y += x;
             break;


         default:
             err = CRYPT_INVALID_ARG;
             goto LBL_ERR;
       }
   }

   /* calc header size */
   z = y;
   if (y < 128) {
      y += 2;
   } else if (y < 256) {
      /* 0x30 0x81 LL */
      y += 3;
   } else if (y < 65536UL) {
      /* 0x30 0x82 LL LL */
      y += 4;
   } else if (y < 16777216UL) {
      /* 0x30 0x83 LL LL LL */
      y += 5;
   } else {
      err = CRYPT_INVALID_ARG;
      goto LBL_ERR;
   }

   /* store size */
   *outlen = y;
   err     = CRYPT_OK;

LBL_ERR:
   return err;
}
```

5.234 pk/asn1/der/sequence/der_sequence_free.c File Reference

5.234.1 Detailed Description

ASN.1 DER, free's a structure allocated by der_decode_sequence_flexi(), Tom St Denis.

Definition in file der_sequence_free.c.

`#include "tomcrypt.h"`

Include dependency graph for der_sequence_free.c:

Functions

- void der_sequence_free (ltc_asn1_list ∗in)

  Free memory allocated by der_decode_sequence_flexi().

5.234.2 Function Documentation

5.234.2.1 void der_sequence_free (ltc_asn1_list ∗ *in*)

Free memory allocated by der_decode_sequence_flexi().

Parameters:

in The list to free

Definition at line 24 of file der_sequence_free.c.

References XFREE.

Referenced by der_decode_sequence_flexi().

```
{
   ltc_asn1_list *l;

   /* walk to the start of the chain */
   while (in->prev != NULL || in->parent != NULL) {
      if (in->parent != NULL) {
          in = in->parent;
      } else {
          in = in->prev;
      }
   }

   /* now walk the list and free stuff */
   while (in != NULL) {
      /* is there a child? */
      if (in->child) {
         /* disconnect */
         in->child->parent = NULL;
         der_sequence_free(in->child);
      }

      switch (in->type) {
         case LTC_ASN1_SET:
         case LTC_ASN1_SETOF:
         case LTC_ASN1_SEQUENCE: break;
         case LTC_ASN1_INTEGER : if (in->data != NULL) { mp_clear(in->data); } break;
```

```
         default               : if (in->data != NULL) { XFREE(in->data);    }
      }

      /* move to next and free current */
      l = in->next;
      free(in);
      in = l;
   }
}
```

5.235 pk/asn1/der/set/der_encode_set.c File Reference

5.235.1 Detailed Description

ASN.1 DER, Encode a SET, Tom St Denis.

Definition in file der_encode_set.c.

```
#include "tomcrypt.h"
```

Include dependency graph for der_encode_set.c:

Functions

- static int ltc_to_asn1 (int v)
- static int qsort_helper (const void ∗a, const void ∗b)
- int der_encode_set (ltc_asn1_list ∗list, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

5.235.2 Function Documentation

5.235.2.1 int der_encode_set (ltc_asn1_list ∗ *list*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Definition at line 66 of file der_encode_set.c.

References CRYPT_MEM, and XCALLOC.

```
{
   ltc_asn1_list  *copy;
   unsigned long   x;
   int             err;

   /* make copy of list */
   copy = XCALLOC(inlen, sizeof(*copy));
   if (copy == NULL) {
      return CRYPT_MEM;
   }

   /* fill in used member with index so we can fully sort it */
   for (x = 0; x < inlen; x++) {
       copy[x]       = list[x];
       copy[x].used = x;
   }

   /* sort it by the "type" field */
   XQSORT(copy, inlen, sizeof(*copy), &qsort_helper);

   /* call der_encode_sequence_ex() */
   err = der_encode_sequence_ex(copy, inlen, out, outlen, LTC_ASN1_SET);

   /* free list */
   XFREE(copy);

   return err;
}
```

5.235.2.2 static int ltc_to_asn1 (int *v*) [static]

Definition at line 21 of file der_encode_set.c.

Referenced by qsort_helper().

```
{
   switch (v) {
      case LTC_ASN1_BOOLEAN:                 return 0x01;
      case LTC_ASN1_INTEGER:
      case LTC_ASN1_SHORT_INTEGER:           return 0x02;
      case LTC_ASN1_BIT_STRING:              return 0x03;
      case LTC_ASN1_OCTET_STRING:            return 0x04;
      case LTC_ASN1_NULL:                    return 0x05;
      case LTC_ASN1_OBJECT_IDENTIFIER:       return 0x06;
      case LTC_ASN1_PRINTABLE_STRING:        return 0x13;
      case LTC_ASN1_IA5_STRING:              return 0x16;
      case LTC_ASN1_UTCTIME:                 return 0x17;
      case LTC_ASN1_SEQUENCE:                return 0x30;
      case LTC_ASN1_SET:
      case LTC_ASN1_SETOF:                   return 0x31;
      default: return -1;
   }
}
```

5.235.2.3 static int qsort_helper (const void ∗ *a*, const void ∗ *b*) [static]

Definition at line 42 of file der_encode_set.c.

References B, and ltc_to_asn1().

```
{
   ltc_asn1_list *A = (ltc_asn1_list *)a, *B = (ltc_asn1_list *)b;
   int            r;

   r = ltc_to_asn1(A->type) - ltc_to_asn1(B->type);

   /* for QSORT the order is UNDEFINED if they are "equal" which means it is NOT DETERMINISTIC.  So we
   if (r == 0) {
      /* their order in the original list now determines the position */
      return A->used - B->used;
   } else {
      return r;
   }
}
```

Here is the call graph for this function:

5.236 pk/asn1/der/set/der_encode_setof.c File Reference

5.236.1 Detailed Description

ASN.1 DER, Encode SET OF, Tom St Denis.

Definition in file der_encode_setof.c.

`#include "tomcrypt.h"`

Include dependency graph for der_encode_setof.c:

Data Structures

- struct edge

Functions

- static int qsort_helper (const void ∗a, const void ∗b)
- int der_encode_setof (ltc_asn1_list ∗list, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen)

  Encode a SETOF stucture.

5.236.2 Function Documentation

5.236.2.1 int der_encode_setof (ltc_asn1_list ∗ *list*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Encode a SETOF stucture.

Parameters:

list The list of items to encode

inlen The number of items in the list

out [out] The destination

outlen [in/out] The size of the output

Returns:

CRYPT_OK on success

Definition at line 61 of file der_encode_setof.c.

References CRYPT_INVALID_ARG.

```
{
   unsigned long  x, y, z, hdrlen;
   int            err;
   struct edge   *edges;
   unsigned char *ptr, *buf;

   /* check that they're all the same type */
   for (x = 1; x < inlen; x++) {
      if (list[x].type != list[x-1].type) {
```

```
         return CRYPT_INVALID_ARG;
      }
   }

   /* alloc buffer to store copy of output */
   buf = XCALLOC(1, *outlen);
   if (buf == NULL) {
      return CRYPT_MEM;
   }

   /* encode list */
   if ((err = der_encode_sequence_ex(list, inlen, buf, outlen, LTC_ASN1_SETOF)) != CRYPT_OK) {
       XFREE(buf);
       return err;
   }

   /* allocate edges */
   edges = XCALLOC(inlen, sizeof(*edges));
   if (edges == NULL) {
      XFREE(buf);
      return CRYPT_MEM;
   }

   /* skip header */
      ptr = buf + 1;

      /* now skip length data */
      x = *ptr++;
       if (x >= 0x80) {
          ptr += (x & 0x7F);
       }

       /* get the size of the static header */
       hdrlen = ((unsigned long)ptr) - ((unsigned long)buf);


    /* scan for edges */
    x = 0;
    while (ptr < (buf + *outlen)) {
       /* store start */
       edges[x].start = ptr;

       /* skip type */
       z = 1;

       /* parse length */
       y = ptr[z++];
       if (y < 128) {
          edges[x].size = y;
       } else {
          y &= 0x7F;
          edges[x].size = 0;
          while (y--) {
             edges[x].size = (edges[x].size << 8) | ((unsigned long)ptr[z++]);
          }
       }

       /* skip content */
       edges[x].size += z;
       ptr           += edges[x].size;
       ++x;
    }

    /* sort based on contents (using edges) */
    XQSORT(edges, inlen, sizeof(*edges), &qsort_helper);

    /* copy static header */
```

```
   XMEMCPY(out, buf, hdrlen);

   /* copy+sort using edges+indecies to output from buffer */
   for (y = hdrlen, x = 0; x < inlen; x++) {
      XMEMCPY(out+y, edges[x].start, edges[x].size);
      y += edges[x].size;
   }

#ifdef LTC_CLEAN_STACK
   zeromem(buf, *outlen);
#endif

   /* free buffers */
   XFREE(edges);
   XFREE(buf);

   return CRYPT_OK;
}
```

5.236.2.2 static int qsort_helper (const void ∗ *a*, const void ∗ *b*) [static]

Definition at line 25 of file der_encode_setof.c.

References B, MIN, edge::size, edge::start, and XMEMCMP.

```
{
   struct edge   *A = (struct edge *)a, *B = (struct edge *)b;
   int            r;
   unsigned long  x;

   /* compare min length */
   r = XMEMCMP(A->start, B->start, MIN(A->size, B->size));

   if (r == 0 && A->size != B->size) {
      if (A->size > B->size) {
         for (x = B->size; x < A->size; x++) {
            if (A->start[x]) {
               return 1;
            }
         }
      } else {
         for (x = A->size; x < B->size; x++) {
            if (B->start[x]) {
               return -1;
            }
         }
      }
   }

   return r;
}
```

5.237 pk/asn1/der/short_integer/der_decode_short_integer.c File Reference

5.237.1 Detailed Description

ASN.1 DER, decode an integer, Tom St Denis.

Definition in file der_decode_short_integer.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_short_integer.c:

Functions

- int der_decode_short_integer (const unsigned char ∗in, unsigned long inlen, unsigned long ∗num)

  Read a short integer.

5.237.2 Function Documentation

5.237.2.1 int der_decode_short_integer (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned long ∗ *num*)

Read a short integer.

Parameters:

in The DER encoded data

inlen Size of data

num [out] The integer to decode

Returns:

CRYPT_OK if successful

Definition at line 28 of file der_decode_short_integer.c.

References CRYPT_INVALID_PACKET, CRYPT_OK, len, and LTC_ARGCHK.

```
{
   unsigned long len, x, y;

   LTC_ARGCHK(num    != NULL);
   LTC_ARGCHK(in     != NULL);

   /* check length */
   if (inlen < 2) {
      return CRYPT_INVALID_PACKET;
   }

   /* check header */
   x = 0;
   if ((in[x++] & 0x1F) != 0x02) {
      return CRYPT_INVALID_PACKET;
   }

```

```
   /* get the packet len */
   len = in[x++];

   if (x + len > inlen) {
      return CRYPT_INVALID_PACKET;
   }

   /* read number */
   y = 0;
   while (len--) {
      y = (y<<8) | (unsigned long)in[x++];
   }
   *num = y;

   return CRYPT_OK;

}
```

5.238 pk/asn1/der/short_integer/der_encode_short_integer.c File Reference

5.238.1 Detailed Description

ASN.1 DER, encode an integer, Tom St Denis.

Definition in file der_encode_short_integer.c.

`#include "tomcrypt.h"`

Include dependency graph for der_encode_short_integer.c:

Functions

- int der_encode_short_integer (unsigned long num, unsigned char ∗out, unsigned long ∗outlen)
  Store a short integer in the range (0,2^32-1).

5.238.2 Function Documentation

5.238.2.1 int der_encode_short_integer (unsigned long *num*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Store a short integer in the range (0,2^32-1).

Parameters:

num The integer to encode

out [out] The destination for the DER encoded integers

outlen [in/out] The max size and resulting size of the DER encoded integers

Returns:

CRYPT_OK if successful

Definition at line 28 of file der_encode_short_integer.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, der_length_short_integer(), len, and LTC_-ARGCHK.

```
{
   unsigned long len, x, y, z;
   int           err;

   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* force to 32 bits */
   num &= 0xFFFFFFFFUL;

   /* find out how big this will be */
   if ((err = der_length_short_integer(num, &len)) != CRYPT_OK) {
      return err;
   }

   if (*outlen < len) {
```

```
      *outlen = len;
      return CRYPT_BUFFER_OVERFLOW;
   }

   /* get len of output */
   z = 0;
   y = num;
   while (y) {
     ++z;
     y >>= 8;
   }

   /* handle zero */
   if (z == 0) {
      z = 1;
   }

   /* see if msb is set */
   z += (num&(1UL<<((z<<3) - 1))) ? 1 : 0;

   /* adjust the number so the msB is non-zero */
   for (x = 0; (z <= 4) && (x < (4 - z)); x++) {
      num <<= 8;
   }

   /* store header */
   x = 0;
   out[x++] = 0x02;
   out[x++] = z;

   /* if 31st bit is set output a leading zero and decrement count */
   if (z == 5) {
      out[x++] = 0;
      --z;
   }

   /* store values */
   for (y = 0; y < z; y++) {
      out[x++] = (num >> 24) & 0xFF;
      num    <<= 8;
   }

   /* we good */
   *outlen = x;

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.239 pk/asn1/der/short_integer/der_length_short_integer.c File Reference

5.239.1 Detailed Description

ASN.1 DER, get length of encoding, Tom St Denis.

Definition in file der_length_short_integer.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_short_integer.c:

Functions

- int der_length_short_integer (unsigned long num, unsigned long ∗outlen)

  Gets length of DER encoding of num.

5.239.2 Function Documentation

5.239.2.1 int der_length_short_integer (unsigned long *num*, unsigned long ∗ *outlen*)

Gets length of DER encoding of num.

Parameters:

num The integer to get the size of

outlen [out] The length of the DER encoding for the given integer

Returns:

CRYPT_OK if successful

Definition at line 26 of file der_length_short_integer.c.

References CRYPT_OK, len, and LTC_ARGCHK.

Referenced by der_encode_short_integer().

```
{
  unsigned long z, y, len;

  LTC_ARGCHK(outlen  != NULL);

  /* force to 32 bits */
  num &= 0xFFFFFFFFUL;

  /* get the number of bytes */
  z = 0;
  y = num;
  while (y) {
    ++z;
    y >>= 8;
  }

  /* handle zero */
  if (z == 0) {
```

```
      z = 1;
   }

   /* we need a 0x02 to indicate it's INTEGER */
   len = 1;

   /* length byte */
   ++len;

   /* bytes in value */
   len += z;

   /* see if msb is set */
   len += (num&(1UL<<((z<<3) - 1))) ? 1 : 0;

   /* return length */
   *outlen = len;

   return CRYPT_OK;
}
```

5.240 pk/asn1/der/utctime/der_decode_utctime.c File Reference

5.240.1 Detailed Description

ASN.1 DER, decode a UTCTIME, Tom St Denis.

Definition in file der_decode_utctime.c.

`#include "tomcrypt.h"`

Include dependency graph for der_decode_utctime.c:

Defines

- #define DECODE_V(y, max)

Functions

- static int char_to_int (unsigned char x)
- int der_decode_utctime (const unsigned char ∗in, unsigned long ∗inlen, ltc_utctime ∗out)

  Decodes a UTC time structure in DER format (reads all 6 valid encoding formats).

5.240.2 Define Documentation

5.240.2.1 #define DECODE_V(y, max)

Value:

```
y  = char_to_int(buf[x])*10 + char_to_int(buf[x+1]); \
   if (y >= max) return CRYPT_INVALID_PACKET;           \
   x += 2;
```

Definition at line 37 of file der_decode_utctime.c.

5.240.3 Function Documentation

5.240.3.1 static int char_to_int (unsigned char *x*) [static]

Definition at line 20 of file der_decode_utctime.c.

```
21 {
22    switch (x)  {
23       case '0': return 0;
24       case '1': return 1;
25       case '2': return 2;
26       case '3': return 3;
27       case '4': return 4;
28       case '5': return 5;
29       case '6': return 6;
30       case '7': return 7;
31       case '8': return 8;
32       case '9': return 9;
33    }
```

```
   return 100;
}
```

5.240.3.2 int der_decode_utctime (const unsigned char ∗ *in*, unsigned long ∗ *inlen*, ltc_utctime ∗ *out*)

Decodes a UTC time structure in DER format (reads all 6 valid encoding formats).

Parameters:

in Input buffer

inlen Length of input buffer in octets

out [out] Destination of UTC time structure

Returns:

CRYPT_OK if successful

Definition at line 49 of file der_decode_utctime.c.

References CRYPT_INVALID_PACKET, der_ia5_value_decode(), and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi().

```
{
   unsigned char buf[32];
   unsigned long x;
   int           y;

   LTC_ARGCHK(in    != NULL);
   LTC_ARGCHK(inlen != NULL);
   LTC_ARGCHK(out   != NULL);

   /* check header */
   if (*inlen < 2UL || (in[1] >= sizeof(buf)) || ((in[1] + 2UL) > *inlen)) {
      return CRYPT_INVALID_PACKET;
   }

   /* decode the string */
   for (x = 0; x < in[1]; x++) {
       y = der_ia5_value_decode(in[x+2]);
       if (y == -1) {
          return CRYPT_INVALID_PACKET;
       }
       buf[x] = y;
   }
   *inlen = 2 + x;


   /* possible encodings are
YYMMDDhhmmZ
YYMMDDhhmm+hh'mm'
YYMMDDhhmm-hh'mm'
YYMMDDhhmmssZ
YYMMDDhhmmss+hh'mm'
YYMMDDhhmmss-hh'mm'

    So let's do a trivial decode upto [including] mm
   */

    x = 0;
    DECODE_V(out->YY, 100);
```

```
   DECODE_V(out->MM, 13);
   DECODE_V(out->DD, 32);
   DECODE_V(out->hh, 24);
   DECODE_V(out->mm, 60);

   /* clear timezone and seconds info */
   out->off_dir = out->off_hh = out->off_mm = out->ss = 0;

   /* now is it Z, +, - or 0-9 */
   if (buf[x] == 'Z') {
      return CRYPT_OK;
   } else if (buf[x] == '+' || buf[x] == '-') {
      out->off_dir = (buf[x++] == '+') ? 0 : 1;
      DECODE_V(out->off_hh, 24);
      DECODE_V(out->off_mm, 60);
      return CRYPT_OK;
   }

   /* decode seconds */
   DECODE_V(out->ss, 60);

   /* now is it Z, +, - */
   if (buf[x] == 'Z') {
      return CRYPT_OK;
   } else if (buf[x] == '+' || buf[x] == '-') {
      out->off_dir = (buf[x++] == '+') ? 0 : 1;
      DECODE_V(out->off_hh, 24);
      DECODE_V(out->off_mm, 60);
      return CRYPT_OK;
   } else {
      return CRYPT_INVALID_PACKET;
   }
}
```

Here is the call graph for this function:

5.241 pk/asn1/der/utctime/der_encode_utctime.c File Reference

5.241.1 Detailed Description

ASN.1 DER, encode a UTCTIME, Tom St Denis.

Definition in file der_encode_utctime.c.

`#include "tomcrypt.h"`

Include dependency graph for der_encode_utctime.c:

Defines

- #define STORE_V(y)

Functions

- int der_encode_utctime (ltc_utctime ∗utctime, unsigned char ∗out, unsigned long ∗outlen)
    Encodes a UTC time structure in DER format.

Variables

- static const char ∗ baseten = "0123456789"

5.241.2 Define Documentation

5.241.2.1 #define STORE_V(y)

Value:

```
out[x++] = der_ia5_char_encode(baseten[(y/10) % 10]); \
    out[x++] = der_ia5_char_encode(baseten[y % 10]);
```

Definition at line 22 of file der_encode_utctime.c.

Referenced by der_encode_utctime().

5.241.3 Function Documentation

5.241.3.1 int der_encode_utctime (ltc_utctime ∗ *utctime*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Encodes a UTC time structure in DER format.

Parameters:

utctime The UTC time structure to encode

out The destination of the DER encoding of the UTC time structure

outlen [in/out] The length of the DER encoding

Returns:

CRYPT_OK if successful

Definition at line 33 of file der_encode_utctime.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, der_ia5_char_encode(), der_length_utctime(), LTC_ARGCHK, and STORE_V.

```
{
    unsigned long x, tmplen;
    int           err;

    LTC_ARGCHK(utctime != NULL);
    LTC_ARGCHK(out     != NULL);
    LTC_ARGCHK(outlen  != NULL);

    if ((err = der_length_utctime(utctime, &tmplen)) != CRYPT_OK) {
       return err;
    }
    if (tmplen > *outlen) {
        *outlen = tmplen;
        return CRYPT_BUFFER_OVERFLOW;
    }

    /* store header */
    out[0] = 0x17;

    /* store values */
    x = 2;
    STORE_V(utctime->YY);
    STORE_V(utctime->MM);
    STORE_V(utctime->DD);
    STORE_V(utctime->hh);
    STORE_V(utctime->mm);
    STORE_V(utctime->ss);

    if (utctime->off_mm || utctime->off_hh) {
       out[x++] = der_ia5_char_encode(utctime->off_dir ? '-' : '+');
       STORE_V(utctime->off_hh);
       STORE_V(utctime->off_mm);
    } else {
       out[x++] = der_ia5_char_encode('Z');
    }

    /* store length */
    out[1] = x - 2;

    /* all good let's return */
    *outlen = x;
    return CRYPT_OK;
}
```

Here is the call graph for this function:

5.241.4 Variable Documentation

5.241.4.1 const char* baseten = "0123456789" [static]

Definition at line 20 of file der_encode_utctime.c.

5.242 pk/asn1/der/utctime/der_length_utctime.c File Reference

5.242.1 Detailed Description

ASN.1 DER, get length of UTCTIME, Tom St Denis.

Definition in file der_length_utctime.c.

`#include "tomcrypt.h"`

Include dependency graph for der_length_utctime.c:

Functions

- int der_length_utctime (ltc_utctime ∗utctime, unsigned long ∗outlen)

  Gets length of DER encoding of UTCTIME.

5.242.2 Function Documentation

5.242.2.1 int der_length_utctime (ltc_utctime ∗ *utctime*, unsigned long ∗ *outlen*)

Gets length of DER encoding of UTCTIME.

Parameters:

utctime The UTC time structure to get the size of

outlen [out] The length of the DER encoding

Returns:

CRYPT_OK if successful

Definition at line 26 of file der_length_utctime.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by der_decode_sequence_flexi(), and der_encode_utctime().

```
{
   LTC_ARGCHK(outlen  != NULL);
   LTC_ARGCHK(utctime != NULL);

   if (utctime->off_hh == 0 && utctime->off_mm == 0) {
      /* we encode as YYMMDDhhmmssZ */
      *outlen = 2 + 13;
   } else {
      /* we encode as YYMMDDhhmmss{+|-}hh'mm' */
      *outlen = 2 + 17;
   }

   return CRYPT_OK;
}
```

5.243 pk/dsa/dsa_decrypt_key.c File Reference

5.243.1 Detailed Description

DSA Crypto, Tom St Denis.

Definition in file dsa_decrypt_key.c.

`#include "tomcrypt.h"`

Include dependency graph for dsa_decrypt_key.c:

Functions

- int dsa_decrypt_key (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, dsa_key ∗key)

  Decrypt an DSA encrypted key.

5.243.2 Function Documentation

5.243.2.1 int dsa_decrypt_key (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, dsa_key ∗ *key*)

Decrypt an DSA encrypted key.

Parameters:

in The ciphertext

inlen The length of the ciphertext (octets)

out [out] The plaintext

outlen [in/out] The max size and resulting size of the plaintext

key The corresponding private DSA key

Returns:

CRYPT_OK if successful

Definition at line 29 of file dsa_decrypt_key.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, CRYPT_PK_NOT_PRIVATE, dsa_shared_secret(), find_hash_oid(), hash_is_valid(), hash_memory(), LTC_ARGCHK, MAXBLOCKSIZE, MIN, PK_PRIVATE, edge::size, XFREE, and XMALLOC.

```
{
   unsigned char  *skey, *expt;
   void           *g_pub;
   unsigned long  x, y, hashOID[32];
   int            hash, err;
   ltc_asn1_list  decode[3];

   LTC_ARGCHK(in      != NULL);
   LTC_ARGCHK(out     != NULL);
   LTC_ARGCHK(outlen  != NULL);
   LTC_ARGCHK(key     != NULL);
```

```

   /* right key type? */
   if (key->type != PK_PRIVATE) {
      return CRYPT_PK_NOT_PRIVATE;
   }

   /* decode to find out hash */
   LTC_SET_ASN1(decode, 0, LTC_ASN1_OBJECT_IDENTIFIER, hashOID, sizeof(hashOID)/sizeof(hashOID[0]));

   if ((err = der_decode_sequence(in, inlen, decode, 1)) != CRYPT_OK) {
      return err;
   }

   hash = find_hash_oid(hashOID, decode[0].size);
   if (hash_is_valid(hash) != CRYPT_OK) {
      return CRYPT_INVALID_PACKET;
   }

   /* we now have the hash! */

   if ((err = mp_init(&g_pub)) != CRYPT_OK) {
      return err;
   }

   /* allocate memory */
   expt   = XMALLOC(mp_unsigned_bin_size(key->p) + 1);
   skey   = XMALLOC(MAXBLOCKSIZE);
   if (expt == NULL || skey == NULL) {
      if (expt != NULL) {
         XFREE(expt);
      }
      if (skey != NULL) {
         XFREE(skey);
      }
      mp_clear(g_pub);
      return CRYPT_MEM;
   }

   LTC_SET_ASN1(decode, 1, LTC_ASN1_INTEGER,          g_pub,      1UL);
   LTC_SET_ASN1(decode, 2, LTC_ASN1_OCTET_STRING,      skey,      MAXBLOCKSIZE);

   /* read the structure in now */
   if ((err = der_decode_sequence(in, inlen, decode, 3)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* make shared key */
   x = mp_unsigned_bin_size(key->p) + 1;
   if ((err = dsa_shared_secret(key->x, g_pub, key, expt, &x)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   y = MIN(mp_unsigned_bin_size(key->p) + 1, MAXBLOCKSIZE);
   if ((err = hash_memory(hash, expt, x, expt, &y)) != CRYPT_OK) {
      goto LBL_ERR;
   }

    /* ensure the hash of the shared secret is at least as big as the encrypt itself */
    if (decode[2].size > y) {
       err = CRYPT_INVALID_PACKET;
       goto LBL_ERR;
    }

    /* avoid buffer overflow */
    if (*outlen < decode[2].size) {
       *outlen = decode[2].size;
       err = CRYPT_BUFFER_OVERFLOW;
```

```
      goto LBL_ERR;
   }

   /* Decrypt the key */
   for (x = 0; x < decode[2].size; x++) {
     out[x] = expt[x] ^ skey[x];
   }
   *outlen = x;

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(expt,   mp_unsigned_bin_size(key->p) + 1);
   zeromem(skey,   MAXBLOCKSIZE);
#endif

   XFREE(expt);
   XFREE(skey);

   mp_clear(g_pub);

   return err;
}
```

Here is the call graph for this function:

5.244 pk/dsa/dsa_encrypt_key.c File Reference

5.244.1 Detailed Description

DSA Crypto, Tom St Denis.

Definition in file dsa_encrypt_key.c.

`#include "tomcrypt.h"`

Include dependency graph for dsa_encrypt_key.c:

Functions

- int dsa_encrypt_key (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, prng_state ∗prng, int wprng, int hash, dsa_key ∗key)

  Encrypt a symmetric key with DSA.

5.244.2 Function Documentation

5.244.2.1 int dsa_encrypt_key (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, prng_state ∗ *prng*, int *wprng*, int *hash*, dsa_key ∗ *key*)

Encrypt a symmetric key with DSA.

Parameters:

in The symmetric key you want to encrypt

inlen The length of the key to encrypt (octets)

out [out] The destination for the ciphertext

outlen [in/out] The max size and resulting size of the ciphertext

prng An active PRNG state

wprng The index of the PRNG you wish to use

hash The index of the hash you want to use

key The DSA key you want to encrypt to

Returns:

CRYPT_OK if successful

Definition at line 32 of file dsa_encrypt_key.c.

References CRYPT_ERROR_READPRNG, CRYPT_INVALID_HASH, CRYPT_MEM, CRYPT_OK, dsa_shared_secret(), hash_descriptor, hash_is_valid(), hash_memory(), LTC_ARGCHK, MAXBLOCK-SIZE, prng_descriptor, prng_is_valid(), XFREE, and XMALLOC.

```
{
    unsigned char *expt, *skey;
    void          *g_pub, *g_priv;
    unsigned long  x, y;
    int            err;

    LTC_ARGCHK(in      != NULL);
```

```
   LTC_ARGCHK(out      != NULL);
   LTC_ARGCHK(outlen   != NULL);
   LTC_ARGCHK(key      != NULL);

   /* check that wprng/cipher/hash are not invalid */
   if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
      return err;
   }

   if ((err = hash_is_valid(hash)) != CRYPT_OK) {
      return err;
   }

   if (inlen > hash_descriptor[hash].hashsize) {
      return CRYPT_INVALID_HASH;
   }

   /* make a random key and export the public copy */
   if ((err = mp_init_multi(&g_pub, &g_priv, NULL)) != CRYPT_OK) {
      return err;
   }

   expt       = XMALLOC(mp_unsigned_bin_size(key->p) + 1);
   skey       = XMALLOC(MAXBLOCKSIZE);
   if (expt == NULL  || skey == NULL) {
      if (expt != NULL) {
         XFREE(expt);
      }
      if (skey != NULL) {
         XFREE(skey);
      }
      mp_clear_multi(g_pub, g_priv, NULL);
      return CRYPT_MEM;
   }

   /* make a random x, g^x pair */
   x = mp_unsigned_bin_size(key->q);
   if (prng_descriptor[wprng].read(expt, x, prng) != x) {
      err = CRYPT_ERROR_READPRNG;
      goto LBL_ERR;
   }

   /* load x */
   if ((err = mp_read_unsigned_bin(g_priv, expt, x)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* compute y */
   if ((err = mp_exptmod(key->g, g_priv, key->p, g_pub)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* make random key */
   x        = mp_unsigned_bin_size(key->p) + 1;
   if ((err = dsa_shared_secret(g_priv, key->y, key, expt, &x)) != CRYPT_OK) {
      goto LBL_ERR;
   }

    y = MAXBLOCKSIZE;
    if ((err = hash_memory(hash, expt, x, skey, &y)) != CRYPT_OK) {
       goto LBL_ERR;
    }

    /* Encrypt key */
    for (x = 0; x < inlen; x++) {
      skey[x] ^= in[x];
    }
```

```

    err = der_encode_sequence_multi(out, outlen,
                                    LTC_ASN1_OBJECT_IDENTIFIER,  hash_descriptor[hash].OIDlen,   hash_
                                    LTC_ASN1_INTEGER,            1UL,                            g_pub
                                    LTC_ASN1_OCTET_STRING,       inlen,                          skey,
                                    LTC_ASN1_EOL,                0UL,                            NULL)

LBL_ERR:
#ifdef LTC_CLEAN_STACK
    /* clean up */
    zeromem(expt,   mp_unsigned_bin_size(key->p) + 1);
    zeromem(skey,   MAXBLOCKSIZE);
#endif

    XFREE(skey);
    XFREE(expt);

    mp_clear_multi(g_pub, g_priv, NULL);

    return err;
}
```

Here is the call graph for this function:

5.245 pk/dsa/dsa_export.c File Reference

5.245.1 Detailed Description

DSA implementation, export key, Tom St Denis.

Definition in file dsa_export.c.

`#include "tomcrypt.h"`

Include dependency graph for dsa_export.c:

Functions

- int dsa_export (unsigned char ∗out, unsigned long ∗outlen, int type, dsa_key ∗key)

  Export a DSA key to a binary packet.

5.245.2 Function Documentation

5.245.2.1 int dsa_export (unsigned char ∗ *out*, unsigned long ∗ *outlen*, int *type*, dsa_key ∗ *key*)

Export a DSA key to a binary packet.

Parameters:

out [out] Where to store the packet

outlen [in/out] The max size and resulting size of the packet

type The type of key to export (PK_PRIVATE or PK_PUBLIC)

key The key to export

Returns:

CRYPT_OK if successful

Definition at line 28 of file dsa_export.c.

References CRYPT_INVALID_ARG, CRYPT_PK_TYPE_MISMATCH, der_encode_sequence_multi(), LTC_ARGCHK, PK_PRIVATE, and PK_PUBLIC.

```
29 {
30    unsigned char flags[1];
31 
32    LTC_ARGCHK(out     != NULL);
33    LTC_ARGCHK(outlen  != NULL);
34    LTC_ARGCHK(key     != NULL);
35 
36    /* can we store the static header?  */
37    if (type == PK_PRIVATE && key->type != PK_PRIVATE) {
38       return CRYPT_PK_TYPE_MISMATCH;
39    }
40 
41    if (type != PK_PUBLIC && type != PK_PRIVATE) {
42       return CRYPT_INVALID_ARG;
43    }
44 
45    flags[0] = (type != PK_PUBLIC) ? 1 : 0;
```

```

   if (type == PK_PRIVATE) {
      return der_encode_sequence_multi(out, outlen,
                                 LTC_ASN1_BIT_STRING,   1UL, flags,
                                 LTC_ASN1_INTEGER,      1UL, key->g,
                                 LTC_ASN1_INTEGER,      1UL, key->p,
                                 LTC_ASN1_INTEGER,      1UL, key->q,
                                 LTC_ASN1_INTEGER,      1UL, key->y,
                                 LTC_ASN1_INTEGER,      1UL, key->x,
                                 LTC_ASN1_EOL,          0UL, NULL);
   } else {
      return der_encode_sequence_multi(out, outlen,
                                 LTC_ASN1_BIT_STRING,   1UL, flags,
                                 LTC_ASN1_INTEGER,      1UL, key->g,
                                 LTC_ASN1_INTEGER,      1UL, key->p,
                                 LTC_ASN1_INTEGER,      1UL, key->q,
                                 LTC_ASN1_INTEGER,      1UL, key->y,
                                 LTC_ASN1_EOL,          0UL, NULL);
   }
}
```

Here is the call graph for this function:

5.246 pk/dsa/dsa_free.c File Reference

5.246.1 Detailed Description

DSA implementation, free a DSA key, Tom St Denis.

Definition in file dsa_free.c.

`#include "tomcrypt.h"`

Include dependency graph for dsa_free.c:

Functions

- void dsa_free (dsa_key ∗key)

  Free a DSA key.

5.246.2 Function Documentation

5.246.2.1 void dsa_free (dsa_key ∗ *key*)

Free a DSA key.

Parameters:

key The key to free from memory

Definition at line 24 of file dsa_free.c.

References LTC_ARGCHKVD.

```
{
   LTC_ARGCHKVD(key != NULL);
   mp_clear_multi(key->g, key->q, key->p, key->x, key->y, NULL);
}
```

5.247 pk/dsa/dsa_import.c File Reference

5.247.1 Detailed Description

DSA implementation, import a DSA key, Tom St Denis.

Definition in file dsa_import.c.

`#include "tomcrypt.h"`

Include dependency graph for dsa_import.c:

Functions

- int dsa_import (const unsigned char ∗in, unsigned long inlen, dsa_key ∗key)

  Import a DSA key.

5.247.2 Function Documentation

5.247.2.1 int dsa_import (const unsigned char ∗ *in*, unsigned long *inlen*, dsa_key ∗ *key*)

Import a DSA key.

Parameters:

in The binary packet to import from

inlen The length of the binary packet

key [out] Where to store the imported key

Returns:

CRYPT_OK if successful, upon error this function will free all allocated memory

Definition at line 27 of file dsa_import.c.

References CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, der_decode_sequence_multi(), LTC_ARGCHK, ltc_mp, ltc_math_descriptor::name, PK_PRIVATE, and PK_PUBLIC.

```
28 {
29    unsigned char flags[1];
30    int           err;
31
32    LTC_ARGCHK(in  != NULL);
33    LTC_ARGCHK(key != NULL);
34    LTC_ARGCHK(ltc_mp.name != NULL);
35
36    /* init key */
37    if (mp_init_multi(&key->p, &key->g, &key->q, &key->x, &key->y, NULL) != CRYPT_OK) {
38       return CRYPT_MEM;
39    }
40
41    /* get key type */
42    if ((err = der_decode_sequence_multi(in, inlen,
43                                   LTC_ASN1_BIT_STRING, 1UL, flags,
44                                   LTC_ASN1_EOL, 0UL, NULL)) != CRYPT_OK) {
45       goto error;
46    }
```

```

   if (flags[0] == 1) {
      if ((err = der_decode_sequence_multi(in, inlen,
                                 LTC_ASN1_BIT_STRING,   1UL, flags,
                                 LTC_ASN1_INTEGER,      1UL, key->g,
                                 LTC_ASN1_INTEGER,      1UL, key->p,
                                 LTC_ASN1_INTEGER,      1UL, key->q,
                                 LTC_ASN1_INTEGER,      1UL, key->y,
                                 LTC_ASN1_INTEGER,      1UL, key->x,
                                 LTC_ASN1_EOL,          0UL, NULL)) != CRYPT_OK) {
         goto error;
      }
      key->type = PK_PRIVATE;
   } else {
      if ((err = der_decode_sequence_multi(in, inlen,
                                 LTC_ASN1_BIT_STRING,   1UL, flags,
                                 LTC_ASN1_INTEGER,      1UL, key->g,
                                 LTC_ASN1_INTEGER,      1UL, key->p,
                                 LTC_ASN1_INTEGER,      1UL, key->q,
                                 LTC_ASN1_INTEGER,      1UL, key->y,
                                 LTC_ASN1_EOL,          0UL, NULL)) != CRYPT_OK) {
         goto error;
      }
      key->type = PK_PUBLIC;
  }
  key->qord = mp_unsigned_bin_size(key->q);

  if (key->qord >= MDSA_MAX_GROUP || key->qord <= 15 ||
      (unsigned long)key->qord >= mp_unsigned_bin_size(key->p) || (mp_unsigned_bin_size(key->p) - key->
      err = CRYPT_INVALID_PACKET;
      goto error;
   }

  return CRYPT_OK;
error:
   mp_clear_multi(key->p, key->g, key->q, key->x, key->y, NULL);
   return err;
}
```

Here is the call graph for this function:

5.248 pk/dsa/dsa_make_key.c File Reference

5.248.1 Detailed Description

DSA implementation, generate a DSA key, Tom St Denis.

Definition in file dsa_make_key.c.

```
#include "tomcrypt.h"
```

Include dependency graph for dsa_make_key.c:

Functions

- int dsa_make_key (prng_state ∗prng, int wprng, int group_size, int modulus_size, dsa_key ∗key)
  Create a DSA key.

5.248.2 Function Documentation

5.248.2.1 int dsa_make_key (prng_state ∗ *prng*, int *wprng*, int *group_size*, int *modulus_size*, dsa_key ∗ *key*)

Create a DSA key.

Parameters:

prng An active PRNG state

wprng The index of the PRNG desired

group_size Size of the multiplicative group (octets)

modulus_size Size of the modulus (octets)

key [out] Where to store the created key

Returns:

CRYPT_OK if successful, upon error this function will free all allocated memory

Definition at line 29 of file dsa_make_key.c.

References CRYPT_ERROR_READPRNG, CRYPT_INVALID_ARG, CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, ltc_mp, ltc_math_descriptor::name, prng_descriptor, prng_is_valid(), rand_prime(), and XMALLOC.

```
30 {
31    void           *tmp, *tmp2;
32    int            err, res;
33    unsigned char *buf;
34
35    LTC_ARGCHK(key  != NULL);
36    LTC_ARGCHK(ltc_mp.name != NULL);
37
38    /* check prng */
39    if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
40       return err;
41    }
42
```

```
   /* check size */
   if (group_size >= MDSA_MAX_GROUP || group_size <= 15 ||
       group_size >= modulus_size || (modulus_size - group_size) >= MDSA_DELTA) {
      return CRYPT_INVALID_ARG;
   }

   /* allocate ram */
   buf = XMALLOC(MDSA_DELTA);
   if (buf == NULL) {
      return CRYPT_MEM;
   }

   /* init mp_ints  */
   if ((err = mp_init_multi(&tmp, &tmp2, &key->g, &key->q, &key->p, &key->x, &key->y, NULL)) != CRYPT_C
      goto LBL_ERR;
   }

   /* make our prime q */
   if ((err = rand_prime(key->q, group_size, prng, wprng)) != CRYPT_OK)             { goto LBL_ERR; }

   /* double q  */
   if ((err = mp_add(key->q, key->q, tmp)) != CRYPT_OK)                                { goto error; }

   /* now make a random string and multply it against q */
   if (prng_descriptor[wprng].read(buf+1, modulus_size - group_size, prng) != (unsigned long)(modulus_s
      err = CRYPT_ERROR_READPRNG;
      goto LBL_ERR;
   }

   /* force magnitude */
   buf[0] |= 0xC0;

   /* force even */
   buf[modulus_size - group_size - 1] &= ~1;

   if ((err = mp_read_unsigned_bin(tmp2, buf, modulus_size - group_size)) != CRYPT_OK) { goto error; }
   if ((err = mp_mul(key->q, tmp2, key->p)) != CRYPT_OK)                               { goto error; }
   if ((err = mp_add_d(key->p, 1, key->p)) != CRYPT_OK)                                { goto error; }

   /* now loop until p is prime */
   for (;;) {
       if ((err = mp_prime_is_prime(key->p, 8, &res)) != CRYPT_OK)                     { goto LBL_ERR;
       if (res == LTC_MP_YES) break;

       /* add 2q to p and 2 to tmp2 */
       if ((err = mp_add(tmp, key->p, key->p)) != CRYPT_OK)                            { goto error; }
       if ((err = mp_add_d(tmp2, 2, tmp2)) != CRYPT_OK)                                { goto error; }
   }

   /* now p = (q * tmp2) + 1 is prime, find a value g for which g^tmp2 != 1 */
   mp_set(key->g, 1);

   do {
      if ((err = mp_add_d(key->g, 1, key->g)) != CRYPT_OK)                             { goto error; }
      if ((err = mp_exptmod(key->g, tmp2, key->p, tmp)) != CRYPT_OK)                   { goto error; }
   } while (mp_cmp_d(tmp, 1) == LTC_MP_EQ);

   /* at this point tmp generates a group of order q mod p */
   mp_exch(tmp, key->g);

   /* so now we have our DH structure, generator g, order q, modulus p
      Now we need a random exponent [mod q] and it's power g^x mod p
    */
   do {
      if (prng_descriptor[wprng].read(buf, group_size, prng) != (unsigned long)group_size) {
         err = CRYPT_ERROR_READPRNG;
         goto LBL_ERR;
```

```
      }
      if ((err = mp_read_unsigned_bin(key->x, buf, group_size)) != CRYPT_OK)           { goto error; }
   } while (mp_cmp_d(key->x, 1) != LTC_MP_GT);
   if ((err = mp_exptmod(key->g, key->x, key->p, key->y)) != CRYPT_OK)                  { goto error; }

   key->type = PK_PRIVATE;
   key->qord = group_size;

#ifdef LTC_CLEAN_STACK
   zeromem(buf, MDSA_DELTA);
#endif

   err = CRYPT_OK;
   goto done;
error:
LBL_ERR:
    mp_clear_multi(key->g, key->q, key->p, key->x, key->y, NULL);
done:
    mp_clear_multi(tmp, tmp2, NULL);

    XFREE(buf);
    return err;
}
```

Here is the call graph for this function:

5.249 pk/dsa/dsa_shared_secret.c File Reference

5.249.1 Detailed Description

DSA Crypto, Tom St Denis.

Definition in file dsa_shared_secret.c.

`#include "tomcrypt.h"`

Include dependency graph for dsa_shared_secret.c:

Functions

- int dsa_shared_secret (void ∗private_key, void ∗base, dsa_key ∗public_key, unsigned char ∗out, unsigned long ∗outlen)

    Create a DSA shared secret between two keys.

5.249.2 Function Documentation

5.249.2.1 int dsa_shared_secret (void ∗ *private_key*, void ∗ *base*, dsa_key ∗ *public_key*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Create a DSA shared secret between two keys.

Parameters:

private_key The private DSA key (the exponent)

base The base of the exponentiation (allows this to be used for both encrypt and decrypt)

public_key The public key

out [out] Destination of the shared secret

outlen [in/out] The max size and resulting size of the shared secret

Returns:

CRYPT_OK if successful

Definition at line 29 of file dsa_shared_secret.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, LTC_ARGCHK, and zeromem().

Referenced by dsa_decrypt_key(), and dsa_encrypt_key().

```
{
   unsigned long x;
   void *res;
   int err;

   LTC_ARGCHK(private_key != NULL);
   LTC_ARGCHK(public_key  != NULL);
   LTC_ARGCHK(out         != NULL);
   LTC_ARGCHK(outlen      != NULL);

   /* make new point */
   if ((err = mp_init(&res)) != CRYPT_OK) {
      return err;
```

```
   }

   if ((err = mp_exptmod(base, private_key, public_key->p, res)) != CRYPT_OK) {
      mp_clear(res);
      return err;
   }

   x = (unsigned long)mp_unsigned_bin_size(res);
   if (*outlen < x) {
      *outlen = x;
      err = CRYPT_BUFFER_OVERFLOW;
      goto done;
   }
   zeromem(out, x);
   if ((err = mp_to_unsigned_bin(res, out + (x - mp_unsigned_bin_size(res))))   != CRYPT_OK)          {

   err     = CRYPT_OK;
   *outlen = x;
done:
   mp_clear(res);
   return err;
}
```

Here is the call graph for this function:

5.250 pk/dsa/dsa_sign_hash.c File Reference

5.250.1 Detailed Description

DSA implementation, sign a hash, Tom St Denis.

Definition in file dsa_sign_hash.c.

`#include "tomcrypt.h"`

Include dependency graph for dsa_sign_hash.c:

Functions

- int dsa_sign_hash_raw (const unsigned char ∗in, unsigned long inlen, void ∗r, void ∗s, prng_state ∗prng, int wprng, dsa_key ∗key)

  Sign a hash with DSA.

- int dsa_sign_hash (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, prng_state ∗prng, int wprng, dsa_key ∗key)

  Sign a hash with DSA.

5.250.2 Function Documentation

5.250.2.1 int dsa_sign_hash (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, prng_state ∗ *prng*, int *wprng*, dsa_key ∗ *key*)

Sign a hash with DSA.

Parameters:

in The hash to sign

inlen The length of the hash to sign

out [out] Where to store the signature

outlen [in/out] The max size and resulting size of the signature

prng An active PRNG state

wprng The index of the PRNG desired

key A private DSA key

Returns:

CRYPT_OK if successful

Definition at line 124 of file dsa_sign_hash.c.

References CRYPT_MEM, CRYPT_OK, der_encode_sequence_multi(), dsa_sign_hash_raw(), and LTC_-ARGCHK.

```
{
   void          *r, *s;
   int            err;

   LTC_ARGCHK(in       != NULL);
```

```
   LTC_ARGCHK(out      != NULL);
   LTC_ARGCHK(outlen   != NULL);
   LTC_ARGCHK(key      != NULL);

   if (mp_init_multi(&r, &s, NULL) != CRYPT_OK) {
      return CRYPT_MEM;
   }

   if ((err = dsa_sign_hash_raw(in, inlen, r, s, prng, wprng, key)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   err = der_encode_sequence_multi(out, outlen,
                             LTC_ASN1_INTEGER, 1UL, r,
                             LTC_ASN1_INTEGER, 1UL, s,
                             LTC_ASN1_EOL,     0UL, NULL);

LBL_ERR:
   mp_clear_multi(r, s, NULL);
   return err;
}
```

Here is the call graph for this function:

5.250.2.2 int dsa_sign_hash_raw (const unsigned char ∗ *in*, unsigned long *inlen*, void ∗ *r*, void ∗ *s*, prng_state ∗ *prng*, int *wprng*, dsa_key ∗ *key*)

Sign a hash with DSA.

Parameters:

in The hash to sign

inlen The length of the hash to sign

r The "r" integer of the signature (caller must initialize with mp_init() first)

s The "s" integer of the signature (caller must initialize with mp_init() first)

prng An active PRNG state

wprng The index of the PRNG desired

key A private DSA key

Returns:

CRYPT_OK if successful

Definition at line 31 of file dsa_sign_hash.c.

References CRYPT_ERROR_READPRNG, CRYPT_INVALID_ARG, CRYPT_MEM, CRYPT_OK, CRYPT_PK_NOT_PRIVATE, LTC_ARGCHK, LTC_MP_EQ, LTC_MP_GT, LTC_MP_YES, PK_-PRIVATE, prng_descriptor, prng_is_valid(), XFREE, XMALLOC, and zeromem().

Referenced by dsa_sign_hash().

```
{
   void          *k, *kinv, *tmp;
   unsigned char *buf;
   int            err;

   LTC_ARGCHK(in  != NULL);
   LTC_ARGCHK(r   != NULL);
   LTC_ARGCHK(s   != NULL);
```

```
   LTC_ARGCHK(key != NULL);

   if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
      return err;
   }
   if (key->type != PK_PRIVATE) {
      return CRYPT_PK_NOT_PRIVATE;
   }

   /* check group order size  */
   if (key->qord >= MDSA_MAX_GROUP) {
      return CRYPT_INVALID_ARG;
   }

   buf = XMALLOC(MDSA_MAX_GROUP);
   if (buf == NULL) {
      return CRYPT_MEM;
   }

   /* Init our temps */
   if ((err = mp_init_multi(&k, &kinv, &tmp, NULL)) != CRYPT_OK)                       { goto error; }

retry:

   do {
      /* gen random k */
      if (prng_descriptor[wprng].read(buf, key->qord, prng) != (unsigned long)key->qord) {
         err = CRYPT_ERROR_READPRNG;
         goto LBL_ERR;
      }

      /* read k */
      if ((err = mp_read_unsigned_bin(k, buf, key->qord)) != CRYPT_OK)                 { goto error; }

      /* k > 1 ? */
      if (mp_cmp_d(k, 1) != LTC_MP_GT)                                                 { goto retry; }

      /* test gcd */
      if ((err = mp_gcd(k, key->q, tmp)) != CRYPT_OK)                                  { goto error; }
   } while (mp_cmp_d(tmp, 1) != LTC_MP_EQ);

   /* now find 1/k mod q */
   if ((err = mp_invmod(k, key->q, kinv)) != CRYPT_OK)                                 { goto error; }

   /* now find r = g^k mod p mod q */
   if ((err = mp_exptmod(key->g, k, key->p, r)) != CRYPT_OK)                           { goto error; }
   if ((err = mp_mod(r, key->q, r)) != CRYPT_OK)                                       { goto error; }

   if (mp_iszero(r) == LTC_MP_YES)                                                     { goto retry; }

   /* now find s = (in + xr)/k mod q */
   if ((err = mp_read_unsigned_bin(tmp, (unsigned char *)in, inlen)) != CRYPT_OK)      { goto error; }
   if ((err = mp_mul(key->x, r, s)) != CRYPT_OK)                                       { goto error; }
   if ((err = mp_add(s, tmp, s)) != CRYPT_OK)                                          { goto error; }
   if ((err = mp_mulmod(s, kinv, key->q, s)) != CRYPT_OK)                              { goto error; }

   if (mp_iszero(s) == LTC_MP_YES)                                                     { goto retry; }

   err = CRYPT_OK;
   goto LBL_ERR;

error:
LBL_ERR:
   mp_clear_multi(k, kinv, tmp, NULL);
#ifdef LTC_CLEAN_STACK
   zeromem(buf, MDSA_MAX_GROUP);
#endif
```

```
   XFREE(buf);
   return err;
}
```

Here is the call graph for this function:

5.251 pk/dsa/dsa_verify_hash.c File Reference

5.251.1 Detailed Description

DSA implementation, verify a signature, Tom St Denis.

Definition in file dsa_verify_hash.c.

`#include "tomcrypt.h"`

Include dependency graph for dsa_verify_hash.c:

Functions

- int dsa_verify_hash_raw (void ∗r, void ∗s, const unsigned char ∗hash, unsigned long hashlen, int ∗stat, dsa_key ∗key)

  Verify a DSA signature.

- int dsa_verify_hash (const unsigned char ∗sig, unsigned long siglen, const unsigned char ∗hash, unsigned long hashlen, int ∗stat, dsa_key ∗key)

  Verify a DSA signature.

5.251.2 Function Documentation

5.251.2.1 int dsa_verify_hash (const unsigned char ∗ *sig*, unsigned long *siglen*, const unsigned char ∗ *hash*, unsigned long *hashlen*, int ∗ *stat*, dsa_key ∗ *key*)

Verify a DSA signature.

Parameters:

sig The signature

siglen The length of the signature (octets)

hash The hash that was signed

hashlen The length of the hash that was signed

stat [out] The result of the signature verification, 1==valid, 0==invalid

key The corresponding public DH key

Returns:

CRYPT_OK if successful (even if the signature is invalid)

Definition at line 96 of file dsa_verify_hash.c.

References CRYPT_MEM, CRYPT_OK, der_decode_sequence_multi(), and dsa_verify_hash_raw().

```
99 {
100    int     err;
101    void    *r, *s;
102
103    if ((err = mp_init_multi(&r, &s, NULL)) != CRYPT_OK) {
104       return CRYPT_MEM;
105    }
106
```

```
   /* decode the sequence */
   if ((err = der_decode_sequence_multi(sig, siglen,
                                  LTC_ASN1_INTEGER, 1UL, r,
                                  LTC_ASN1_INTEGER, 1UL, s,
                                  LTC_ASN1_EOL,     0UL, NULL)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* do the op */
   err = dsa_verify_hash_raw(r, s, hash, hashlen, stat, key);

LBL_ERR:
   mp_clear_multi(r, s, NULL);
   return err;
}
```

Here is the call graph for this function:

5.251.2.2 int dsa_verify_hash_raw (void ∗ *r*, void ∗ *s*, const unsigned char ∗ *hash*, unsigned long *hashlen*, int ∗ *stat*, dsa_key ∗ *key*)

Verify a DSA signature.

Parameters:

r DSA "r" parameter

s DSA "s" parameter

hash The hash that was signed

hashlen The length of the hash that was signed

stat [out] The result of the signature verification, 1==valid, 0==invalid

key The corresponding public DH key

Returns:

CRYPT_OK if successful (even if the signature is invalid)

Definition at line 31 of file dsa_verify_hash.c.

References CRYPT_INVALID_PACKET, CRYPT_OK, LTC_ARGCHK, LTC_MP_EQ, LTC_MP_LT, and LTC_MP_YES.

Referenced by dsa_verify_hash().

```
{
  void          *w, *v, *u1, *u2;
  int           err;

  LTC_ARGCHK(r    != NULL);
  LTC_ARGCHK(s    != NULL);
  LTC_ARGCHK(stat != NULL);
  LTC_ARGCHK(key  != NULL);

  /* default to invalid signature */
  *stat = 0;

  /* init our variables */
  if ((err = mp_init_multi(&w, &v, &u1, &u2, NULL)) != CRYPT_OK) {
     return err;
  }

```

```
   /* neither r or s can be null or >q*/
   if (mp_iszero(r) == LTC_MP_YES || mp_iszero(s) == LTC_MP_YES || mp_cmp(r, key->q) != LTC_MP_LT || mp
      err = CRYPT_INVALID_PACKET;
      goto done;
   }

   /* w = 1/s mod q */
   if ((err = mp_invmod(s, key->q, w)) != CRYPT_OK)                                       { goto error;

   /* u1 = m * w mod q */
   if ((err = mp_read_unsigned_bin(u1, (unsigned char *)hash, hashlen)) != CRYPT_OK)      { goto error;
   if ((err = mp_mulmod(u1, w, key->q, u1)) != CRYPT_OK)                                  { goto error;

   /* u2 = r*w mod q */
   if ((err = mp_mulmod(r, w, key->q, u2)) != CRYPT_OK)                                   { goto error;

   /* v = g^u1 * y^u2 mod p mod q */
   if ((err = mp_exptmod(key->g, u1, key->p, u1)) != CRYPT_OK)                            { goto error;
   if ((err = mp_exptmod(key->y, u2, key->p, u2)) != CRYPT_OK)                            { goto error;
   if ((err = mp_mulmod(u1, u2, key->p, v)) != CRYPT_OK)                                  { goto error;
   if ((err = mp_mod(v, key->q, v)) != CRYPT_OK)                                          { goto error;

   /* if r = v then we're set */
   if (mp_cmp(r, v) == LTC_MP_EQ) {
      *stat = 1;
   }

   err = CRYPT_OK;
   goto done;

error :
done  : mp_clear_multi(w, v, u1, u2, NULL);
   return err;
}
```

5.252 pk/dsa/dsa_verify_key.c File Reference

5.252.1 Detailed Description

DSA implementation, verify a key, Tom St Denis.

Definition in file dsa_verify_key.c.

```
#include "tomcrypt.h"
```

Include dependency graph for dsa_verify_key.c:

Functions

- int dsa_verify_key (dsa_key ∗key, int ∗stat)

  Verify a DSA key for validity.

5.252.2 Function Documentation

5.252.2.1 int dsa_verify_key (dsa_key ∗ *key*, int ∗ *stat*)

Verify a DSA key for validity.

Parameters:

key The key to verify

stat [out] Result of test, 1==valid, 0==invalid

Returns:

CRYPT_OK if successful

Definition at line 26 of file dsa_verify_key.c.

References CRYPT_OK, LTC_ARGCHK, LTC_MP_EQ, LTC_MP_GT, LTC_MP_LT, and LTC_MP_YES.

```
{
   void   *tmp, *tmp2;
   int    res, err;

   LTC_ARGCHK(key  != NULL);
   LTC_ARGCHK(stat != NULL);

   /* default to an invalid key */
   *stat = 0;

   /* first make sure key->q and key->p are prime */
   if ((err = mp_prime_is_prime(key->q, 8, &res)) != CRYPT_OK) {
      return err;
   }
   if (res == 0) {
      return CRYPT_OK;
   }

   if ((err = mp_prime_is_prime(key->p, 8, &res)) != CRYPT_OK) {
      return err;
   }
```

```
   if (res == 0) {
      return CRYPT_OK;
   }

   /* now make sure that g is not -1, 0 or 1 and <p */
   if (mp_cmp_d(key->g, 0) == LTC_MP_EQ || mp_cmp_d(key->g, 1) == LTC_MP_EQ) {
      return CRYPT_OK;
   }
   if ((err = mp_init_multi(&tmp, &tmp2, NULL)) != CRYPT_OK)               { goto error; }
   if ((err = mp_sub_d(key->p, 1, tmp)) != CRYPT_OK)                       { goto error; }
   if (mp_cmp(tmp, key->g) == LTC_MP_EQ || mp_cmp(key->g, key->p) != LTC_MP_LT) {
      err = CRYPT_OK;
      goto done;
   }

   /* 1 < y < p-1 */
   if (!(mp_cmp_d(key->y, 1) == LTC_MP_GT && mp_cmp(key->y, tmp) == LTC_MP_LT)) {
      err = CRYPT_OK;
      goto done;
   }

   /* now we have to make sure that g^q = 1, and that p-1/q gives 0 remainder */
   if ((err = mp_div(tmp, key->q, tmp, tmp2)) != CRYPT_OK)             { goto error; }
   if (mp_iszero(tmp2) != LTC_MP_YES) {
      err = CRYPT_OK;
      goto done;
   }

   if ((err = mp_exptmod(key->g, key->q, key->p, tmp)) != CRYPT_OK)    { goto error; }
   if (mp_cmp_d(tmp, 1) != LTC_MP_EQ) {
      err = CRYPT_OK;
      goto done;
   }

   /* now we have to make sure that y^q = 1, this makes sure y \in g^x mod p */
   if ((err = mp_exptmod(key->y, key->q, key->p, tmp)) != CRYPT_OK)       { goto error; }
   if (mp_cmp_d(tmp, 1) != LTC_MP_EQ) {
      err = CRYPT_OK;
      goto done;
   }

   /* at this point we are out of tests ;-( */
   err   = CRYPT_OK;
   *stat = 1;
   goto done;
error:
done : mp_clear_multi(tmp, tmp2, NULL);
   return err;
}
```

5.253 pk/ecc/ecc.c File Reference

5.253.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ecc.c:

Variables

- const ltc_ecc_set_type ltc_ecc_sets [ ]

5.253.2 Variable Documentation

5.253.2.1 const ltc_ecc_set_type ltc_ecc_sets[ ]

Definition at line 27 of file ecc.c.

Referenced by ecc_ansi_x963_export(), ecc_ansi_x963_import(), ecc_export(), ecc_get_size(), ecc_import(), ecc_make_key(), ecc_shared_secret(), ecc_sign_hash(), ecc_sizes(), ecc_test(), ecc_verify_hash(), is_point(), and ltc_ecc_is_valid_idx().

5.254 pk/ecc/ecc_ansi_x963_export.c File Reference

5.254.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_ansi_x963_export.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_ansi_x963_export.c:

Functions

- int ecc_ansi_x963_export (ecc_key ∗key, unsigned char ∗out, unsigned long ∗outlen)

  ECC X9.63 (Sec.

5.254.2 Function Documentation

5.254.2.1 int ecc_ansi_x963_export (ecc_key ∗ *key*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

ECC X9.63 (Sec.

4.3.6) uncompressed export

Parameters:

key Key to export

out [out] destination of export

outlen [in/out] Length of destination and final output size Return CRYPT_OK on success

Definition at line 32 of file ecc_ansi_x963_export.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, ltc_ecc_is_valid_idx(), ltc_ecc_sets, XMEMCPY, and zeromem().

```
33 {
34    unsigned char buf[128];
35    unsigned long numlen;
36
37    LTC_ARGCHK(key    != NULL);
38    LTC_ARGCHK(out    != NULL);
39    LTC_ARGCHK(outlen != NULL);
40
41    if (ltc_ecc_is_valid_idx(key->idx) == 0) {
42       return CRYPT_INVALID_ARG;
43    }
44    numlen = ltc_ecc_sets[key->idx].size;
45
46    if (*outlen < (1 + 2*numlen)) {
47       *outlen = 1 + 2*numlen;
48       return CRYPT_BUFFER_OVERFLOW;
49    }
50
51    /* store byte 0x04 */
52    out[0] = 0x04;
53
54    /* pad and store x */
```

```
   zeromem(buf, sizeof(buf));
   mp_to_unsigned_bin(key->pubkey.x, buf + (numlen - mp_unsigned_bin_size(key->pubkey.x)));
   XMEMCPY(out+1, buf, numlen);

   /* pad and store y */
   zeromem(buf, sizeof(buf));
   mp_to_unsigned_bin(key->pubkey.y, buf + (numlen - mp_unsigned_bin_size(key->pubkey.y)));
   XMEMCPY(out+1+numlen, buf, numlen);

   *outlen = 1 + 2*numlen;
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.255 pk/ecc/ecc_ansi_x963_import.c File Reference

5.255.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_ansi_x963_import.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_ansi_x963_import.c:

Functions

- int ecc_ansi_x963_import (const unsigned char ∗in, unsigned long inlen, ecc_key ∗key)

  Import an ANSI X9.63 format public key.

5.255.2 Function Documentation

5.255.2.1 int ecc_ansi_x963_import (const unsigned char ∗ *in*, unsigned long *inlen*, ecc_key ∗ *key*)

Import an ANSI X9.63 format public key.

Parameters:

in The input data to read

inlen The length of the input data

key [out] destination to store imported key \

Definition at line 31 of file ecc_ansi_x963_import.c.

References CRYPT_INVALID_ARG, CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, LTC_-ARGCHK, ltc_ecc_sets, and edge::size.

```
32 {
33    int x, err;
34
35    LTC_ARGCHK(in != NULL);
36    LTC_ARGCHK(key != NULL);
37
38    /* must be odd */
39    if ((inlen & 1) == 0) {
40       return CRYPT_INVALID_ARG;
41    }
42
43    /* init key */
44    if (mp_init_multi(&key->pubkey.x, &key->pubkey.y, &key->pubkey.z, &key->k, NULL) != CRYPT_OK) {
45       return CRYPT_MEM;
46    }
47
48    /* check for 4, 6 or 7 */
49    if (in[0] != 4 && in[0] != 6 && in[0] != 7) {
50       err = CRYPT_INVALID_PACKET;
51       goto error;
52    }
53
54    /* read data */
```

```
   if ((err = mp_read_unsigned_bin(key->pubkey.x, (unsigned char *)in+1, (inlen-1)>>1)) != CRYPT_OK) {
      goto error;
   }

   if ((err = mp_read_unsigned_bin(key->pubkey.y, (unsigned char *)in+1+((inlen-1)>>1), (inlen-1)>>1))
      goto error;
   }
   mp_set(key->pubkey.z, 1);

   /* determine the idx */
   for (x = 0; ltc_ecc_sets[x].size != 0; x++) {
      if ((unsigned)ltc_ecc_sets[x].size >= ((inlen-1)>>1)) {
         break;
      }
   }
   if (ltc_ecc_sets[x].size == 0) {
      err = CRYPT_INVALID_PACKET;
      goto error;
   }

   /* set the idx */
   key->idx  = x;
   key->type = PK_PUBLIC;

   /* we're done */
   return CRYPT_OK;
error:
   mp_clear_multi(key->pubkey.x, key->pubkey.y, key->pubkey.z, key->k, NULL);
   return err;
}
```

5.256 pk/ecc/ecc_decrypt_key.c File Reference

5.256.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_decrypt_key.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_decrypt_key.c:

Functions

- int ecc_decrypt_key (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, ecc_key ∗key)

    Decrypt an ECC encrypted key.

5.256.2 Function Documentation

5.256.2.1 int ecc_decrypt_key (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, ecc_key ∗ *key*)

Decrypt an ECC encrypted key.

Parameters:

in The ciphertext

inlen The length of the ciphertext (octets)

out [out] The plaintext

outlen [in/out] The max size and resulting size of the plaintext

key The corresponding private ECC key

Returns:

CRYPT_OK if successful

Definition at line 35 of file ecc_decrypt_key.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, CRYPT_PK_NOT_PRIVATE, ecc_free(), ecc_import(), ecc_shared_secret(), find_hash_oid(), hash_is_valid(), hash_memory(), LTC_ARGCHK, MAXBLOCKSIZE, MIN, PK_PRIVATE, edge::size, XFREE, and XMALLOC.

```
{
   unsigned char *ecc_shared, *skey, *pub_expt;
   unsigned long  x, y, hashOID[32];
   int            hash, err;
   ecc_key        pubkey;
   ltc_asn1_list  decode[3];

   LTC_ARGCHK(in      != NULL);
   LTC_ARGCHK(out     != NULL);
   LTC_ARGCHK(outlen  != NULL);
   LTC_ARGCHK(key     != NULL);
```

```

   /* right key type? */
   if (key->type != PK_PRIVATE) {
      return CRYPT_PK_NOT_PRIVATE;
   }

   /* decode to find out hash */
   LTC_SET_ASN1(decode, 0, LTC_ASN1_OBJECT_IDENTIFIER, hashOID, sizeof(hashOID)/sizeof(hashOID[0]));

   if ((err = der_decode_sequence(in, inlen, decode, 1)) != CRYPT_OK) {
      return err;
   }

   hash = find_hash_oid(hashOID, decode[0].size);
   if (hash_is_valid(hash) != CRYPT_OK) {
      return CRYPT_INVALID_PACKET;
   }

   /* we now have the hash! */

   /* allocate memory */
   pub_expt   = XMALLOC(ECC_BUF_SIZE);
   ecc_shared = XMALLOC(ECC_BUF_SIZE);
   skey       = XMALLOC(MAXBLOCKSIZE);
   if (pub_expt == NULL || ecc_shared == NULL || skey == NULL) {
      if (pub_expt != NULL) {
         XFREE(pub_expt);
      }
      if (ecc_shared != NULL) {
         XFREE(ecc_shared);
      }
      if (skey != NULL) {
         XFREE(skey);
      }
      return CRYPT_MEM;
   }
   LTC_SET_ASN1(decode, 1, LTC_ASN1_OCTET_STRING,      pub_expt,  ECC_BUF_SIZE);
   LTC_SET_ASN1(decode, 2, LTC_ASN1_OCTET_STRING,      skey,      MAXBLOCKSIZE);

   /* read the structure in now */
   if ((err = der_decode_sequence(in, inlen, decode, 3)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* import ECC key from packet */
   if ((err = ecc_import(decode[1].data, decode[1].size, &pubkey)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* make shared key */
   x = ECC_BUF_SIZE;
   if ((err = ecc_shared_secret(key, &pubkey, ecc_shared, &x)) != CRYPT_OK) {
      ecc_free(&pubkey);
      goto LBL_ERR;
   }
   ecc_free(&pubkey);

   y = MIN(ECC_BUF_SIZE, MAXBLOCKSIZE);
   if ((err = hash_memory(hash, ecc_shared, x, ecc_shared, &y)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* ensure the hash of the shared secret is at least as big as the encrypt itself */
   if (decode[2].size > y) {
      err = CRYPT_INVALID_PACKET;
      goto LBL_ERR;
   }
```

```

   /* avoid buffer overflow */
   if (*outlen < decode[2].size) {
      *outlen = decode[2].size;
      err = CRYPT_BUFFER_OVERFLOW;
      goto LBL_ERR;
   }

   /* Decrypt the key */
   for (x = 0; x < decode[2].size; x++) {
     out[x] = skey[x] ^ ecc_shared[x];
   }
   *outlen = x;

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(pub_expt,   ECC_BUF_SIZE);
   zeromem(ecc_shared, ECC_BUF_SIZE);
   zeromem(skey,       MAXBLOCKSIZE);
#endif

   XFREE(pub_expt);
   XFREE(ecc_shared);
   XFREE(skey);

   return err;
}
```

Here is the call graph for this function:

5.257 pk/ecc/ecc_encrypt_key.c File Reference

5.257.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_encrypt_key.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_encrypt_key.c:

Functions

- int ecc_encrypt_key (const unsigned char *in, unsigned long inlen, unsigned char *out, unsigned long *outlen, prng_state *prng, int wprng, int hash, ecc_key *key)

  Encrypt a symmetric key with ECC.

5.257.2 Function Documentation

5.257.2.1 int ecc_encrypt_key (const unsigned char * *in*, unsigned long *inlen*, unsigned char * *out*, unsigned long * *outlen*, prng_state * *prng*, int *wprng*, int *hash*, ecc_key * *key*)

Encrypt a symmetric key with ECC.

Parameters:

in The symmetric key you want to encrypt

inlen The length of the key to encrypt (octets)

out [out] The destination for the ciphertext

outlen [in/out] The max size and resulting size of the ciphertext

prng An active PRNG state

wprng The index of the PRNG you wish to use

hash The index of the hash you want to use

key The ECC key you want to encrypt to

Returns:

CRYPT_OK if successful

Definition at line 38 of file ecc_encrypt_key.c.

References CRYPT_INVALID_HASH, CRYPT_MEM, CRYPT_OK, ecc_export(), ecc_free(), ecc_get_size(), ecc_make_key(), ecc_shared_secret(), hash_descriptor, hash_is_valid(), hash_memory(), LTC_ARGCHK, MAXBLOCKSIZE, PK_PUBLIC, prng_is_valid(), XFREE, and XMALLOC.

```
{
   unsigned char *pub_expt, *ecc_shared, *skey;
   ecc_key        pubkey;
   unsigned long  x, y, pubkeysize;
   int            err;

   LTC_ARGCHK(in       != NULL);
```

```
   LTC_ARGCHK(out     != NULL);
   LTC_ARGCHK(outlen  != NULL);
   LTC_ARGCHK(key     != NULL);

   /* check that wprng/cipher/hash are not invalid */
   if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
      return err;
   }

   if ((err = hash_is_valid(hash)) != CRYPT_OK) {
      return err;
   }

   if (inlen > hash_descriptor[hash].hashsize) {
      return CRYPT_INVALID_HASH;
   }

   /* make a random key and export the public copy */
   if ((err = ecc_make_key(prng, wprng, ecc_get_size(key), &pubkey)) != CRYPT_OK) {
      return err;
   }

   pub_expt   = XMALLOC(ECC_BUF_SIZE);
   ecc_shared = XMALLOC(ECC_BUF_SIZE);
   skey       = XMALLOC(MAXBLOCKSIZE);
   if (pub_expt == NULL || ecc_shared == NULL || skey == NULL) {
      if (pub_expt != NULL) {
         XFREE(pub_expt);
      }
      if (ecc_shared != NULL) {
         XFREE(ecc_shared);
      }
      if (skey != NULL) {
         XFREE(skey);
      }
      ecc_free(&pubkey);
      return CRYPT_MEM;
   }

   pubkeysize = ECC_BUF_SIZE;
   if ((err = ecc_export(pub_expt, &pubkeysize, PK_PUBLIC, &pubkey)) != CRYPT_OK) {
      ecc_free(&pubkey);
      goto LBL_ERR;
   }

   /* make random key */
   x        = ECC_BUF_SIZE;
   if ((err = ecc_shared_secret(&pubkey, key, ecc_shared, &x)) != CRYPT_OK) {
      ecc_free(&pubkey);
      goto LBL_ERR;
   }
   ecc_free(&pubkey);
   y = MAXBLOCKSIZE;
   if ((err = hash_memory(hash, ecc_shared, x, skey, &y)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* Encrypt key */
   for (x = 0; x < inlen; x++) {
     skey[x] ^= in[x];
   }

   err = der_encode_sequence_multi(out, outlen,
                                   LTC_ASN1_OBJECT_IDENTIFIER,  hash_descriptor[hash].OIDlen,   hash_
                                   LTC_ASN1_OCTET_STRING,       pubkeysize,                     pub_e
                                   LTC_ASN1_OCTET_STRING,       inlen,                          skey,
                                   LTC_ASN1_EOL,                0UL,                            NULL)
```

```

LBL_ERR:
#ifdef LTC_CLEAN_STACK
    /* clean up */
    zeromem(pub_expt,   ECC_BUF_SIZE);
    zeromem(ecc_shared, ECC_BUF_SIZE);
    zeromem(skey,       MAXBLOCKSIZE);
#endif

    XFREE(skey);
    XFREE(ecc_shared);
    XFREE(pub_expt);

    return err;
}
```

Here is the call graph for this function:

5.258 pk/ecc/ecc_export.c File Reference

5.258.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_export.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_export.c:

Functions

- int ecc_export (unsigned char ∗out, unsigned long ∗outlen, int type, ecc_key ∗key)
  Export an ECC key as a binary packet.

5.258.2 Function Documentation

5.258.2.1 int ecc_export (unsigned char ∗ *out*, unsigned long ∗ *outlen*, int *type*, ecc_key ∗ *key*)

Export an ECC key as a binary packet.

Parameters:

out [out] Destination for the key

outlen [in/out] Max size and resulting size of the exported key

type The type of key you want to export (PK_PRIVATE or PK_PUBLIC)

key The key to export

Returns:

CRYPT_OK if successful

Definition at line 34 of file ecc_export.c.

References CRYPT_INVALID_ARG, CRYPT_PK_TYPE_MISMATCH, der_encode_sequence_multi(), LTC_ARGCHK, ltc_ecc_is_valid_idx(), ltc_ecc_sets, and PK_PRIVATE.

Referenced by ecc_encrypt_key().

```
35 {
36    int           err;
37    unsigned char flags[1];
38    unsigned long key_size;
39
40    LTC_ARGCHK(out    != NULL);
41    LTC_ARGCHK(outlen != NULL);
42    LTC_ARGCHK(key    != NULL);
43
44    /* type valid? */
45    if (key->type != PK_PRIVATE && type == PK_PRIVATE) {
46       return CRYPT_PK_TYPE_MISMATCH;
47    }
48
49    if (ltc_ecc_is_valid_idx(key->idx) == 0) {
50       return CRYPT_INVALID_ARG;
```

```
    }

    /* we store the NIST byte size */
    key_size = ltc_ecc_sets[key->idx].size;

    if (type == PK_PRIVATE) {
        flags[0] = 1;
        err = der_encode_sequence_multi(out, outlen,
                                  LTC_ASN1_BIT_STRING,      1UL, flags,
                                  LTC_ASN1_SHORT_INTEGER,   1UL, &key_size,
                                  LTC_ASN1_INTEGER,         1UL, key->pubkey.x,
                                  LTC_ASN1_INTEGER,         1UL, key->pubkey.y,
                                  LTC_ASN1_INTEGER,         1UL, key->k,
                                  LTC_ASN1_EOL,             0UL, NULL);
    } else {
        flags[0] = 0;
        err = der_encode_sequence_multi(out, outlen,
                                  LTC_ASN1_BIT_STRING,      1UL, flags,
                                  LTC_ASN1_SHORT_INTEGER,   1UL, &key_size,
                                  LTC_ASN1_INTEGER,         1UL, key->pubkey.x,
                                  LTC_ASN1_INTEGER,         1UL, key->pubkey.y,
                                  LTC_ASN1_EOL,             0UL, NULL);
    }

    return err;
}
```

Here is the call graph for this function:

5.259 pk/ecc/ecc_free.c File Reference

5.259.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_free.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_free.c:

Functions

- void ecc_free (ecc_key ∗key)

  Free an ECC key from memory.

5.259.2 Function Documentation

5.259.2.1 void ecc_free (ecc_key ∗ *key*)

Free an ECC key from memory.

Parameters:

key The key you wish to free

Definition at line 30 of file ecc_free.c.

References LTC_ARGCHKVD.

Referenced by ecc_decrypt_key(), ecc_encrypt_key(), and ecc_sign_hash().

```
{
   LTC_ARGCHKVD(key != NULL);
   mp_clear_multi(key->pubkey.x, key->pubkey.y, key->pubkey.z, key->k, NULL);
}
```

5.260 pk/ecc/ecc_get_size.c File Reference

5.260.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_get_size.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_get_size.c:

Functions

- int ecc_get_size (ecc_key ∗key)

  Get the size of an ECC key.

5.260.2 Function Documentation

5.260.2.1 int ecc_get_size (ecc_key ∗ *key*)

Get the size of an ECC key.

Parameters:

key The key to get the size of

Returns:

The size (octets) of the key or INT_MAX on error

Definition at line 31 of file ecc_get_size.c.

References LTC_ARGCHK, ltc_ecc_is_valid_idx(), and ltc_ecc_sets.

Referenced by ecc_encrypt_key(), and ecc_sign_hash().

```
{
   LTC_ARGCHK(key != NULL);
   if (ltc_ecc_is_valid_idx(key->idx))
      return ltc_ecc_sets[key->idx].size;
   else
      return INT_MAX; /* large value known to cause it to fail when passed to ecc_make_key() */
}
```

Here is the call graph for this function:

5.261 pk/ecc/ecc_import.c File Reference

5.261.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_import.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_import.c:

Functions

- static int is_point (ecc_key ∗key)
- int ecc_import (const unsigned char ∗in, unsigned long inlen, ecc_key ∗key)

    Import an ECC key from a binary packet.

5.261.2 Function Documentation

5.261.2.1 int ecc_import (const unsigned char ∗ *in*, unsigned long *inlen*, ecc_key ∗ *key*)

Import an ECC key from a binary packet.

Parameters:

in The packet to import

inlen The length of the packet

key [out] The destination of the import

Returns:

CRYPT_OK if successful, upon error all allocated memory will be freed

Definition at line 81 of file ecc_import.c.

References CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, der_decode_sequence_multi(), is_point(), LTC_ARGCHK, ltc_ecc_sets, ltc_mp, ltc_math_descriptor::name, PK_PRIVATE, and PK_-PUBLIC.

Referenced by ecc_decrypt_key().

```
{
   unsigned long key_size;
   unsigned char flags[1];
   int           err;

   LTC_ARGCHK(in  != NULL);
   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(ltc_mp.name != NULL);

   /* init key */
   if (mp_init_multi(&key->pubkey.x, &key->pubkey.y, &key->pubkey.z, &key->k, NULL) != CRYPT_OK) {
      return CRYPT_MEM;
   }

   /* find out what type of key it is */
```

```
   if ((err = der_decode_sequence_multi(in, inlen,
                                  LTC_ASN1_BIT_STRING, 1UL, &flags,
                                  LTC_ASN1_EOL,        0UL, NULL)) != CRYPT_OK) {
      goto done;
   }


   if (flags[0] == 1) {
      /* private key */
      key->type = PK_PRIVATE;
      if ((err = der_decode_sequence_multi(in, inlen,
                                     LTC_ASN1_BIT_STRING,      1UL, flags,
                                     LTC_ASN1_SHORT_INTEGER,   1UL, &key_size,
                                     LTC_ASN1_INTEGER,         1UL, key->pubkey.x,
                                     LTC_ASN1_INTEGER,         1UL, key->pubkey.y,
                                     LTC_ASN1_INTEGER,         1UL, key->k,
                                     LTC_ASN1_EOL,             0UL, NULL)) != CRYPT_OK) {
         goto done;
      }
   } else {
      /* public key */
      key->type = PK_PUBLIC;
      if ((err = der_decode_sequence_multi(in, inlen,
                                     LTC_ASN1_BIT_STRING,      1UL, flags,
                                     LTC_ASN1_SHORT_INTEGER,   1UL, &key_size,
                                     LTC_ASN1_INTEGER,         1UL, key->pubkey.x,
                                     LTC_ASN1_INTEGER,         1UL, key->pubkey.y,
                                     LTC_ASN1_EOL,             0UL, NULL)) != CRYPT_OK) {
         goto done;
      }
   }

   /* find the idx */
   for (key->idx = 0; ltc_ecc_sets[key->idx].size && (unsigned long)ltc_ecc_sets[key->idx].size != key
   if (ltc_ecc_sets[key->idx].size == 0) {
      err = CRYPT_INVALID_PACKET;
      goto done;
   }

   /* set z */
   mp_set(key->pubkey.z, 1);

   /* is it a point on the curve?  */
   if ((err = is_point(key)) != CRYPT_OK) {
      goto done;
   }

   /* we're good */
   return CRYPT_OK;
done:
   mp_clear_multi(key->pubkey.x, key->pubkey.y, key->pubkey.z, key->k, NULL);
   return err;
}
```

Here is the call graph for this function:

5.261.2.2 static int is_point (ecc_key ∗ *key*) [static]

Definition at line 26 of file ecc_import.c.

References CRYPT_INVALID_PACKET, CRYPT_OK, ltc_ecc_sets, LTC_MP_EQ, LTC_MP_LT, t1, and t2.

Referenced by ecc_import().

```
{
   void *prime, *b, *t1, *t2;
   int err;

   if ((err = mp_init_multi(&prime, &b, &t1, &t2, NULL)) != CRYPT_OK) {
      return err;
   }

   /* load prime and b */
   if ((err = mp_read_radix(prime, ltc_ecc_sets[key->idx].prime, 16)) != CRYPT_OK)            { goto er
   if ((err = mp_read_radix(b, ltc_ecc_sets[key->idx].B, 16)) != CRYPT_OK)                    { goto er

   /* compute y^2 */
   if ((err = mp_sqr(key->pubkey.y, t1)) != CRYPT_OK)                                         { goto er

   /* compute x^3 */
   if ((err = mp_sqr(key->pubkey.x, t2)) != CRYPT_OK)                                         { goto er
   if ((err = mp_mod(t2, prime, t2)) != CRYPT_OK)                                             { goto er
   if ((err = mp_mul(key->pubkey.x, t2, t2)) != CRYPT_OK)                                     { goto er

   /* compute y^2 - x^3 */
   if ((err = mp_sub(t1, t2, t1)) != CRYPT_OK)                                                { goto er

   /* compute y^2 - x^3 + 3x */
   if ((err = mp_add(t1, key->pubkey.x, t1)) != CRYPT_OK)                                     { goto er
   if ((err = mp_add(t1, key->pubkey.x, t1)) != CRYPT_OK)                                     { goto er
   if ((err = mp_add(t1, key->pubkey.x, t1)) != CRYPT_OK)                                     { goto er
   if ((err = mp_mod(t1, prime, t1)) != CRYPT_OK)                                             { goto er
   while (mp_cmp_d(t1, 0) == LTC_MP_LT) {
      if ((err = mp_add(t1, prime, t1)) != CRYPT_OK)                                          { goto er
   }
   while (mp_cmp(t1, prime) != LTC_MP_LT) {
      if ((err = mp_sub(t1, prime, t1)) != CRYPT_OK)                                          { goto er
   }

   /* compare to b */
   if (mp_cmp(t1, b) != LTC_MP_EQ) {
      err = CRYPT_INVALID_PACKET;
   } else {
      err = CRYPT_OK;
   }

error:
   mp_clear_multi(prime, b, t1, t2, NULL);
   return err;
}
```

5.262 pk/ecc/ecc_make_key.c File Reference

5.262.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_make_key.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_make_key.c:

Functions

- int ecc_make_key (prng_state ∗prng, int wprng, int keysize, ecc_key ∗key)

  Make a new ECC key.

5.262.2 Function Documentation

5.262.2.1 int ecc_make_key (prng_state ∗ *prng*, int *wprng*, int *keysize*, ecc_key ∗ *key*)

Make a new ECC key.

Parameters:

prng An active PRNG state

wprng The index of the PRNG you wish to use

keysize The keysize for the new key (in octets from 20 to 65 bytes)

key [out] Destination of the newly created key

Returns:

CRYPT_OK if successful, upon error all allocated memory will be freed

Definition at line 34 of file ecc_make_key.c.

References CRYPT_INVALID_KEYSIZE, CRYPT_OK, LTC_ARGCHK, ltc_ecc_sets, ltc_mp, ltc_math_descriptor::name, prng_is_valid(), and edge::size.

Referenced by ecc_encrypt_key(), and ecc_sign_hash().

```
35 {
36    int            x, err;
37    ecc_point     *base;
38    void          *prime;
39    unsigned char *buf;
40
41    LTC_ARGCHK(key != NULL);
42    LTC_ARGCHK(ltc_mp.name != NULL);
43
44    /* good prng? */
45    if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
46       return err;
47    }
48
49    /* find key size */
50    for (x = 0; (keysize > ltc_ecc_sets[x].size) && (ltc_ecc_sets[x].size != 0); x++);
```

```
   keysize = ltc_ecc_sets[x].size;

   if (keysize > ECC_MAXSIZE || ltc_ecc_sets[x].size == 0) {
      return CRYPT_INVALID_KEYSIZE;
   }
   key->idx = x;

   /* allocate ram */
   base = NULL;
   buf  = XMALLOC(ECC_MAXSIZE);
   if (buf == NULL) {
      return CRYPT_MEM;
   }

   /* make up random string */
   if (prng_descriptor[wprng].read(buf, (unsigned long)keysize, prng) != (unsigned long)keysize) {
      err = CRYPT_ERROR_READPRNG;
      goto LBL_ERR2;
   }

   /* setup the key variables */
   if ((err = mp_init_multi(&key->pubkey.x, &key->pubkey.y, &key->pubkey.z, &key->k, &prime, NULL)) !=
      goto done;
   }
   base = ltc_ecc_new_point();
   if (base == NULL) {
      mp_clear_multi(key->pubkey.x, key->pubkey.y, key->pubkey.z, key->k, prime, NULL);
      err = CRYPT_MEM;
      goto done;
   }

   /* read in the specs for this key */
   if ((err = mp_read_radix(prime, (char *)ltc_ecc_sets[key->idx].prime, 16)) != CRYPT_OK)      { goto
   if ((err = mp_read_radix(base->x, (char *)ltc_ecc_sets[key->idx].Gx, 16)) != CRYPT_OK)       { goto
   if ((err = mp_read_radix(base->y, (char *)ltc_ecc_sets[key->idx].Gy, 16)) != CRYPT_OK)       { goto
   mp_set(base->z, 1);
   if ((err = mp_read_unsigned_bin(key->k, (unsigned char *)buf, keysize)) != CRYPT_OK)         { goto

   /* make the public key */
   if ((err = ltc_mp.ecc_ptmul(key->k, base, &key->pubkey, prime, 1)) != CRYPT_OK)              { goto
   key->type = PK_PRIVATE;

   /* free up ram */
   err = CRYPT_OK;
done:
   ltc_ecc_del_point(base);
   mp_clear(prime);
LBL_ERR2:
#ifdef LTC_CLEAN_STACK
   zeromem(buf, ECC_MAXSIZE);
#endif

   XFREE(buf);

   return err;
}
```

Here is the call graph for this function:

5.263 pk/ecc/ecc_shared_secret.c File Reference

5.263.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_shared_secret.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_shared_secret.c:

Functions

- int ecc_shared_secret (ecc_key ∗private_key, ecc_key ∗public_key, unsigned char ∗out, unsigned long ∗outlen)

  Create an ECC shared secret between two keys.

5.263.2 Function Documentation

5.263.2.1 int ecc_shared_secret (ecc_key ∗ *private_key*, ecc_key ∗ *public_key*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

Create an ECC shared secret between two keys.

Parameters:

private_key The private ECC key

public_key The public key

out [out] Destination of the shared secret (Conforms to EC-DH from ANSI X9.63)

outlen [in/out] The max size and resulting size of the shared secret

Returns:

CRYPT_OK if successful

Definition at line 34 of file ecc_shared_secret.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_MEM, CRYPT_OK, CRYPT_PK_NOT_PRIVATE, CRYPT_PK_TYPE_MISMATCH, ltc_math_descriptor::ecc_ptmul, LTC_ARGCHK, ltc_ecc_del_point(), ltc_ecc_is_valid_idx(), ltc_ecc_new_point(), ltc_ecc_sets, ltc_mp, PK_PRIVATE, and zeromem().

Referenced by ecc_decrypt_key(), and ecc_encrypt_key().

```
{
   unsigned long x;
   ecc_point *result;
   void *prime;
   int err;

   LTC_ARGCHK(private_key != NULL);
   LTC_ARGCHK(public_key  != NULL);
   LTC_ARGCHK(out         != NULL);
   LTC_ARGCHK(outlen      != NULL);

```

```
   /* type valid? */
   if (private_key->type != PK_PRIVATE) {
      return CRYPT_PK_NOT_PRIVATE;
   }

   if (ltc_ecc_is_valid_idx(private_key->idx) == 0) {
      return CRYPT_INVALID_ARG;
   }

   if (private_key->idx != public_key->idx) {
      return CRYPT_PK_TYPE_MISMATCH;
   }

   /* make new point */
   result = ltc_ecc_new_point();
   if (result == NULL) {
      return CRYPT_MEM;
   }

   if ((err = mp_init(&prime)) != CRYPT_OK) {
      ltc_ecc_del_point(result);
      return err;
   }

   if ((err = mp_read_radix(prime, (char *)ltc_ecc_sets[private_key->idx].prime, 16)) != CRYPT_OK)
   if ((err = ltc_mp.ecc_ptmul(private_key->k, &public_key->pubkey, result, prime, 1)) != CRYPT_OK)

   x = (unsigned long)mp_unsigned_bin_size(prime);
   if (*outlen < x) {
      *outlen = x;
      err = CRYPT_BUFFER_OVERFLOW;
      goto done;
   }
   zeromem(out, x);
   if ((err = mp_to_unsigned_bin(result->x, out + (x - mp_unsigned_bin_size(result->x))))   != CRYPT_OK

   err     = CRYPT_OK;
   *outlen = x;
done:
   mp_clear(prime);
   ltc_ecc_del_point(result);
   return err;
}
```

Here is the call graph for this function:

5.264 pk/ecc/ecc_sign_hash.c File Reference

5.264.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_sign_hash.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_sign_hash.c:

Functions

- int ecc_sign_hash (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, prng_state ∗prng, int wprng, ecc_key ∗key)

  Sign a message digest.

5.264.2 Function Documentation

5.264.2.1 int ecc_sign_hash (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, prng_state ∗ *prng*, int *wprng*, ecc_key ∗ *key*)

Sign a message digest.

Parameters:

in The message digest to sign

inlen The length of the digest

out [out] The destination for the signature

outlen [in/out] The max size and resulting size of the signature

prng An active PRNG state

wprng The index of the PRNG you wish to use

key A private ECC key

Returns:

CRYPT_OK if successful

Definition at line 37 of file ecc_sign_hash.c.

References CRYPT_OK, CRYPT_PK_INVALID_TYPE, CRYPT_PK_NOT_PRIVATE, ecc_free(), ecc_-get_size(), ecc_make_key(), LTC_ARGCHK, ltc_ecc_is_valid_idx(), ltc_ecc_sets, PK_PRIVATE, and prng_is_valid().

```
{
   ecc_key       pubkey;
   void          *r, *s, *e, *p;
   int           err;

   LTC_ARGCHK(in      != NULL);
   LTC_ARGCHK(out     != NULL);
   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(key     != NULL);
```

```

   /* is this a private key? */
   if (key->type != PK_PRIVATE) {
      return CRYPT_PK_NOT_PRIVATE;
   }

   /* is the IDX valid ?  */
   if (ltc_ecc_is_valid_idx(key->idx) != 1) {
      return CRYPT_PK_INVALID_TYPE;
   }

   if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
      return err;
   }

   /* get the hash and load it as a bignum into 'e' */
   /* init the bignums */
   if ((err = mp_init_multi(&r, &s, &p, &e, NULL)) != CRYPT_OK) {
      ecc_free(&pubkey);
      goto LBL_ERR;
   }
   if ((err = mp_read_radix(p, (char *)ltc_ecc_sets[key->idx].order, 16)) != CRYPT_OK)        { goto er
   if ((err = mp_read_unsigned_bin(e, (unsigned char *)in, (int)inlen)) != CRYPT_OK)          { goto er

   /* make up a key and export the public copy */
   for (;;) {
      if ((err = ecc_make_key(prng, wprng, ecc_get_size(key), &pubkey)) != CRYPT_OK) {
         return err;
      }

      /* find r = x1 mod n */
      if ((err = mp_mod(pubkey.pubkey.x, p, r)) != CRYPT_OK)                 { goto error; }

      if (mp_iszero(r)) {
         ecc_free(&pubkey);
      } else {
        /* find s = (e + xr)/k */
        if ((err = mp_invmod(pubkey.k, p, pubkey.k)) != CRYPT_OK)            { goto error; } /* k = 1/k
        if ((err = mp_mulmod(key->k, r, p, s)) != CRYPT_OK)                  { goto error; } /* s = xr
        if ((err = mp_add(e, s, s)) != CRYPT_OK)                             { goto error; } /* s = e +
        if ((err = mp_mod(s, p, s)) != CRYPT_OK)                             { goto error; } /* s = e +
        if ((err = mp_mulmod(s, pubkey.k, p, s)) != CRYPT_OK)                { goto error; } /* s = (e

        if (mp_iszero(s)) {
           ecc_free(&pubkey);
        } else {
           break;
        }
      }
   }

   /* store as SEQUENCE { r, s -- integer } */
   err = der_encode_sequence_multi(out, outlen,
                             LTC_ASN1_INTEGER, 1UL, r,
                             LTC_ASN1_INTEGER, 1UL, s,
                             LTC_ASN1_EOL, 0UL, NULL);
   goto LBL_ERR;
error:
LBL_ERR:
   mp_clear_multi(r, s, p, e, NULL);
   ecc_free(&pubkey);

   return err;
}
```

Here is the call graph for this function:

5.265 pk/ecc/ecc_sizes.c File Reference

5.265.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_sizes.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ecc_sizes.c:

Functions

- void ecc_sizes (int ∗low, int ∗high)

5.265.2 Function Documentation

5.265.2.1 void ecc_sizes (int ∗ *low*, int ∗ *high*)

Definition at line 26 of file ecc_sizes.c.

References LTC_ARGCHKVD, ltc_ecc_sets, and edge::size.

```
27 {
28  int i;
29  LTC_ARGCHKVD(low  != NULL);
30  LTC_ARGCHKVD(high != NULL);
31 
32  *low = INT_MAX;
33  *high = 0;
34  for (i = 0; ltc_ecc_sets[i].size != 0; i++) {
35      if (ltc_ecc_sets[i].size < *low)  {
36         *low  = ltc_ecc_sets[i].size;
37      }
38      if (ltc_ecc_sets[i].size > *high) {
39         *high = ltc_ecc_sets[i].size;
40      }
41  }
42 }
```

5.266 pk/ecc/ecc_test.c File Reference

5.266.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_test.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_test.c:

Functions

- int ecc_test (void)

  Perform on the ECC system.

5.266.2 Function Documentation

5.266.2.1 int ecc_test (void)

Perform on the ECC system.

Returns:

CRYPT_OK if successful

Definition at line 30 of file ecc_test.c.

References CRYPT_MEM, CRYPT_OK, G, GG, ltc_ecc_del_point(), ltc_ecc_new_point(), ltc_ecc_sets, and edge::size.

```
{
   void       *modulus, *order;
   ecc_point  *G, *GG;
   int i, err, primality;

   if ((err = mp_init_multi(&modulus, &order, NULL)) != CRYPT_OK) {
      return err;
   }

   G   = ltc_ecc_new_point();
   GG  = ltc_ecc_new_point();
   if (G == NULL || GG == NULL) {
      mp_clear_multi(modulus, order, NULL);
      ltc_ecc_del_point(G);
      ltc_ecc_del_point(GG);
      return CRYPT_MEM;
   }

   for (i = 0; ltc_ecc_sets[i].size; i++) {
       #if 0
          printf("Testing %d\n", ltc_ecc_sets[i].size);
       #endif
       if ((err = mp_read_radix(modulus, (char *)ltc_ecc_sets[i].prime, 16)) != CRYPT_OK)   { goto done
       if ((err = mp_read_radix(order, (char *)ltc_ecc_sets[i].order, 16)) != CRYPT_OK)     { goto done

       /* is prime actually prime? */
```

```
        if ((err = mp_prime_is_prime(modulus, 8, &primality)) != CRYPT_OK)                   { goto done
        if (primality == 0) {
           err = CRYPT_FAIL_TESTVECTOR;
           goto done;
        }

        /* is order prime ? */
        if ((err = mp_prime_is_prime(order, 8, &primality)) != CRYPT_OK)                     { goto done
        if (primality == 0) {
           err = CRYPT_FAIL_TESTVECTOR;
           goto done;
        }

        if ((err = mp_read_radix(G->x, (char *)ltc_ecc_sets[i].Gx, 16)) != CRYPT_OK)         { goto done
        if ((err = mp_read_radix(G->y, (char *)ltc_ecc_sets[i].Gy, 16)) != CRYPT_OK)         { goto done
        mp_set(G->z, 1);

        /* then we should have G == (order + 1)G */
        if ((err = mp_add_d(order, 1, order)) != CRYPT_OK)                                   { goto done
        if ((err = ltc_mp.ecc_ptmul(order, G, GG, modulus, 1)) != CRYPT_OK)                  { goto done
        if (mp_cmp(G->x, GG->x) != LTC_MP_EQ || mp_cmp(G->y, GG->y) != LTC_MP_EQ) {
           err = CRYPT_FAIL_TESTVECTOR;
           goto done;
        }
    }
    err = CRYPT_OK;
    goto done;
done:
    ltc_ecc_del_point(GG);
    ltc_ecc_del_point(G);
    mp_clear_multi(order, modulus, NULL);
    return err;
}
```

Here is the call graph for this function:

5.267 pk/ecc/ecc_verify_hash.c File Reference

5.267.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ecc_verify_hash.c.

`#include "tomcrypt.h"`

Include dependency graph for ecc_verify_hash.c:

Functions

- int ecc_verify_hash (const unsigned char ∗sig, unsigned long siglen, const unsigned char ∗hash, unsigned long hashlen, int ∗stat, ecc_key ∗key)

  Verify an ECC signature.

5.267.2 Function Documentation

5.267.2.1 int ecc_verify_hash (const unsigned char ∗ *sig*, unsigned long *siglen*, const unsigned char ∗ *hash*, unsigned long *hashlen*, int ∗ *stat*, ecc_key ∗ *key*)

Verify an ECC signature.

Parameters:

sig The signature to verify

siglen The length of the signature (octets)

hash The hash (message digest) that was signed

hashlen The length of the hash (octets)

stat Result of signature, 1==valid, 0==invalid

key The corresponding public ECC key

Returns:

CRYPT_OK if successful (even if the signature is not valid)

Definition at line 46 of file ecc_verify_hash.c.

References CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, CRYPT_PK_INVALID_TYPE, der_decode_sequence_multi(), ltc_math_descriptor::ecc_ptmul, LTC_ARGCHK, ltc_ecc_is_valid_idx(), ltc_ecc_new_point(), ltc_ecc_sets, ltc_mp, and LTC_MP_LT.

```
{
   ecc_point    *mG, *mQ;
   void          *r, *s, *v, *w, *u1, *u2, *e, *p, *m;
   void          *mp;
   int           err;

   LTC_ARGCHK(sig  != NULL);
   LTC_ARGCHK(hash != NULL);
   LTC_ARGCHK(stat != NULL);
   LTC_ARGCHK(key  != NULL);
```

```

   /* default to invalid signature */
   *stat = 0;
   mp    = NULL;

   /* is the IDX valid ?  */
   if (ltc_ecc_is_valid_idx(key->idx) != 1) {
      return CRYPT_PK_INVALID_TYPE;
   }

   /* allocate ints */
   if ((err = mp_init_multi(&r, &s, &v, &w, &u1, &u2, &p, &e, &m, NULL)) != CRYPT_OK) {
      return CRYPT_MEM;
   }

   /* allocate points */
   mG = ltc_ecc_new_point();
   mQ = ltc_ecc_new_point();
   if (mQ  == NULL || mG == NULL) {
      err = CRYPT_MEM;
      goto done;
   }

   /* parse header */
   if ((err = der_decode_sequence_multi(sig, siglen,
                                  LTC_ASN1_INTEGER, 1UL, r,
                                  LTC_ASN1_INTEGER, 1UL, s,
                                  LTC_ASN1_EOL, 0UL, NULL)) != CRYPT_OK) {
      goto done;
   }

   /* get the order */
   if ((err = mp_read_radix(p, (char *)ltc_ecc_sets[key->idx].order, 16)) != CRYPT_OK)

   /* get the modulus */
   if ((err = mp_read_radix(m, (char *)ltc_ecc_sets[key->idx].prime, 16)) != CRYPT_OK)

   /* check for zero */
   if (mp_iszero(r) || mp_iszero(s) || mp_cmp(r, p) != LTC_MP_LT || mp_cmp(s, p) != LTC_MP_LT) {
      err = CRYPT_INVALID_PACKET;
      goto done;
   }

   /* read hash */
   if ((err = mp_read_unsigned_bin(e, (unsigned char *)hash, (int)hashlen)) != CRYPT_OK)

   /*  w  = s^-1 mod n */
   if ((err = mp_invmod(s, p, w)) != CRYPT_OK)

   /* u1 = ew */
   if ((err = mp_mulmod(e, w, p, u1)) != CRYPT_OK)

   /* u2 = rw */
   if ((err = mp_mulmod(r, w, p, u2)) != CRYPT_OK)

   /* find mG = u1*G */
   if ((err = mp_read_radix(mG->x, (char *)ltc_ecc_sets[key->idx].Gx, 16)) != CRYPT_OK)
   if ((err = mp_read_radix(mG->y, (char *)ltc_ecc_sets[key->idx].Gy, 16)) != CRYPT_OK)
   mp_set(mG->z, 1);
   if ((err = ltc_mp.ecc_ptmul(u1, mG, mG, m, 0)) != CRYPT_OK)

   /* find mQ = u2*Q */
   if ((err = mp_copy(key->pubkey.x, mQ->x)) != CRYPT_OK)
   if ((err = mp_copy(key->pubkey.y, mQ->y)) != CRYPT_OK)
   if ((err = mp_copy(key->pubkey.z, mQ->z)) != CRYPT_OK)
   if ((err = ltc_mp.ecc_ptmul(u2, mQ, mQ, m, 0)) != CRYPT_OK)

```

```
   /* find the montgomery mp */
   if ((err = mp_montgomery_setup(m, &mp)) != CRYPT_OK)
   /* add them */
   if ((err = ltc_mp.ecc_ptadd(mQ, mG, mG, m, mp)) != CRYPT_OK)

   /* reduce */
   if ((err = ltc_mp.ecc_map(mG, m, mp)) != CRYPT_OK)

   /* v = X_x1 mod n */
   if ((err = mp_mod(mG->x, p, v)) != CRYPT_OK)

   /* does v == r */
   if (mp_cmp(v, r) == LTC_MP_EQ) {
      *stat = 1;
   }

   /* clear up and return */
   err = CRYPT_OK;
   goto done;
error:
done:
   ltc_ecc_del_point(mG);
   ltc_ecc_del_point(mQ);
   mp_clear_multi(r, s, v, w, u1, u2, p, e, m, NULL);
   if (mp != NULL) {
      mp_montgomery_free(mp);
   }
   return err;
}
```

Here is the call graph for this function:

5.268 pk/ecc/ltc_ecc_is_valid_idx.c File Reference

5.268.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ltc_ecc_is_valid_idx.c.

`#include "tomcrypt.h"`

Include dependency graph for ltc_ecc_is_valid_idx.c:

Functions

- int ltc_ecc_is_valid_idx (int n)

  Returns whether an ECC idx is valid or not.

5.268.2 Function Documentation

5.268.2.1 int ltc_ecc_is_valid_idx (int *n*)

Returns whether an ECC idx is valid or not.

Parameters:

n The idx number to check

Returns:

1 if valid, 0 if not

Definition at line 30 of file ltc_ecc_is_valid_idx.c.

References ltc_ecc_sets.

Referenced by ecc_ansi_x963_export(), ecc_export(), ecc_get_size(), ecc_shared_secret(), ecc_sign_-hash(), and ecc_verify_hash().

```
{
   int x;

   for (x = 0; ltc_ecc_sets[x].size != 0; x++);
   if ((n < 0) || (n >= x)) {
      return 0;
   }
   return 1;
}
```

5.269 pk/ecc/ltc_ecc_map.c File Reference

5.269.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ltc_ecc_map.c.

`#include "tomcrypt.h"`

Include dependency graph for ltc_ecc_map.c:

Functions

- int ltc_ecc_map (ecc_point *P, void *modulus, void *mp)

  Map a projective jacbobian point back to affine space.

5.269.2 Function Documentation

5.269.2.1 int ltc_ecc_map (ecc_point * *P*, void * *modulus*, void * *mp*)

Map a projective jacbobian point back to affine space.

Parameters:

P [in/out] The point to map

modulus The modulus of the field the ECC curve is in

mp The "b" value from montgomery_setup()

Returns:

CRYPT_OK on success

Definition at line 33 of file ltc_ecc_map.c.

References CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, t1, and t2.

```
{
   void *t1, *t2;
   int err;

   LTC_ARGCHK(P       != NULL);
   LTC_ARGCHK(modulus != NULL);
   LTC_ARGCHK(mp      != NULL);

   if ((err = mp_init_multi(&t1, &t2, NULL)) != CRYPT_OK) {
      return CRYPT_MEM;
   }

   /* first map z back to normal */
   if ((err = mp_montgomery_reduce(P->z, modulus, mp)) != CRYPT_OK)           { goto done; }

   /* get 1/z */
   if ((err = mp_invmod(P->z, modulus, t1)) != CRYPT_OK)                      { goto done; }

   /* get 1/z^2 and 1/z^3 */
   if ((err = mp_sqr(t1, t2)) != CRYPT_OK)                                    { goto done; }
```

```
   if ((err = mp_mod(t2, modulus, t2)) != CRYPT_OK)                              { goto done; }
   if ((err = mp_mul(t1, t2, t1)) != CRYPT_OK)                                   { goto done; }
   if ((err = mp_mod(t1, modulus, t1)) != CRYPT_OK)                              { goto done; }

   /* multiply against x/y */
   if ((err = mp_mul(P->x, t2, P->x)) != CRYPT_OK)                               { goto done; }
   if ((err = mp_montgomery_reduce(P->x, modulus, mp)) != CRYPT_OK)              { goto done; }
   if ((err = mp_mul(P->y, t1, P->y)) != CRYPT_OK)                               { goto done; }
   if ((err = mp_montgomery_reduce(P->y, modulus, mp)) != CRYPT_OK)              { goto done; }
   mp_set(P->z, 1);

   err = CRYPT_OK;
   goto done;
done:
   mp_clear_multi(t1, t2, NULL);
   return err;
}
```

5.270 pk/ecc/ltc_ecc_mulmod.c File Reference

5.270.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ltc_ecc_mulmod.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ltc_ecc_mulmod.c:

Defines

- #define WINSIZE 4

Functions

- int ltc_ecc_mulmod (void ∗k, ecc_point ∗G, ecc_point ∗R, void ∗modulus, int map)

  Perform a point multiplication.

5.270.2 Define Documentation

5.270.2.1 #define WINSIZE 4

Definition at line 28 of file ltc_ecc_mulmod.c.

5.270.3 Function Documentation

5.270.3.1 int ltc_ecc_mulmod (void ∗ *k*, ecc_point ∗ *G*, ecc_point ∗ *R*, void ∗ *modulus*, int *map*)

Perform a point multiplication.

Parameters:

k The scalar to multiply by

G The base point

R [out] Destination for kG

modulus The modulus of the field the ECC curve is in

map Boolean whether to map back to affine or not (1==map, 0 == leave in projective)

Returns:

CRYPT_OK on success

Definition at line 39 of file ltc_ecc_mulmod.c.

References CRYPT_OK, LTC_ARGCHK, ltc_ecc_del_point(), and ltc_ecc_new_point().

```
{
   ecc_point *tG, *M[8];
   int        i, j, err;
   void       *mu, *mp;
   unsigned long buf;
   int        first, bitbuf, bitcpy, bitcnt, mode, digidx;

   LTC_ARGCHK(k       != NULL);
   LTC_ARGCHK(G       != NULL);
   LTC_ARGCHK(R       != NULL);
   LTC_ARGCHK(modulus != NULL);

   /* init montgomery reduction */
   if ((err = mp_montgomery_setup(modulus, &mp)) != CRYPT_OK) {
      return err;
   }
   if ((err = mp_init(&mu)) != CRYPT_OK) {
      return err;
   }
   if ((err = mp_montgomery_normalization(mu, modulus)) != CRYPT_OK) {
      mp_montgomery_free(mp);
      mp_clear(mu);
      return err;
   }

  /* alloc ram for window temps */
  for (i = 0; i < 8; i++) {
      M[i] = ltc_ecc_new_point();
      if (M[i] == NULL) {
         for (j = 0; j < i; j++) {
             ltc_ecc_del_point(M[j]);
         }
         mp_montgomery_free(mp);
         mp_clear(mu);
         return CRYPT_MEM;
      }
  }

   /* make a copy of G incase R==G */
   tG = ltc_ecc_new_point();
   if (tG == NULL)                                                                 { err = CRYPT_MEM;

   /* tG = G  and convert to montgomery */
   if (mp_cmp_d(mu, 1) == LTC_MP_EQ) {
      if ((err = mp_copy(G->x, tG->x)) != CRYPT_OK)                                { goto done; }
      if ((err = mp_copy(G->y, tG->y)) != CRYPT_OK)                                { goto done; }
      if ((err = mp_copy(G->z, tG->z)) != CRYPT_OK)                                { goto done; }
   } else {
      if ((err = mp_mulmod(G->x, mu, modulus, tG->x)) != CRYPT_OK)                 { goto done; }
      if ((err = mp_mulmod(G->y, mu, modulus, tG->y)) != CRYPT_OK)                 { goto done; }
      if ((err = mp_mulmod(G->z, mu, modulus, tG->z)) != CRYPT_OK)                 { goto done; }
   }
   mp_clear(mu);

   /* calc the M tab, which holds kG for k==8..15 */
   /* M[0] == 8G */
   if ((err = ltc_mp.ecc_ptdbl(tG, M[0], modulus, mp)) != CRYPT_OK)               { goto done; }
   if ((err = ltc_mp.ecc_ptdbl(M[0], M[0], modulus, mp)) != CRYPT_OK)             { goto done; }
   if ((err = ltc_mp.ecc_ptdbl(M[0], M[0], modulus, mp)) != CRYPT_OK)             { goto done; }

   /* now find (8+k)G for k=1..7 */
   for (j = 9; j < 16; j++) {
       if ((err = ltc_mp.ecc_ptadd(M[j-9], tG, M[j-8], modulus, mp)) != CRYPT_OK)   { goto done; }
   }

   /* setup sliding window */
   mode   = 0;
```

```
   bitcnt = 1;
   buf    = 0;
   digidx = mp_get_digit_count(k) - 1;
   bitcpy = bitbuf = 0;
   first  = 1;

   /* perform ops */
   for (;;) {
     /* grab next digit as required */
     if (--bitcnt == 0) {
       if (digidx == -1) {
          break;
       }
       buf    = mp_get_digit(k, digidx);
       bitcnt = (int) ltc_mp.bits_per_digit;
       --digidx;
     }

     /* grab the next msb from the ltiplicand */
     i = (buf >> (ltc_mp.bits_per_digit - 1)) & 1;
     buf <<= 1;

     /* skip leading zero bits */
     if (mode == 0 && i == 0) {
        continue;
     }

     /* if the bit is zero and mode == 1 then we double */
     if (mode == 1 && i == 0) {
        if ((err = ltc_mp.ecc_ptdbl(R, R, modulus, mp)) != CRYPT_OK)                 { goto done; }
        continue;
     }

     /* else we add it to the window */
     bitbuf |= (i << (WINSIZE - ++bitcpy));
     mode = 2;

     if (bitcpy == WINSIZE) {
       /* if this is the first window we do a simple copy */
       if (first == 1) {
          /* R = kG [k = first window] */
          if ((err = mp_copy(M[bitbuf-8]->x, R->x)) != CRYPT_OK)                     { goto done; }
          if ((err = mp_copy(M[bitbuf-8]->y, R->y)) != CRYPT_OK)                     { goto done; }
          if ((err = mp_copy(M[bitbuf-8]->z, R->z)) != CRYPT_OK)                     { goto done; }
          first = 0;
       } else {
         /* normal window */
         /* ok window is filled so double as required and add  */
         /* double first */
         for (j = 0; j < WINSIZE; j++) {
           if ((err = ltc_mp.ecc_ptdbl(R, R, modulus, mp)) != CRYPT_OK)             { goto done; }
         }

         /* then add, bitbuf will be 8..15 [8..2^WINSIZE] guaranteed */
         if ((err = ltc_mp.ecc_ptadd(R, M[bitbuf-8], R, modulus, mp)) != CRYPT_OK)  { goto done; }
       }
       /* empty window and reset */
       bitcpy = bitbuf = 0;
       mode = 1;
    }
  }

   /* if bits remain then double/add */
   if (mode == 2 && bitcpy > 0) {
     /* double then add */
     for (j = 0; j < bitcpy; j++) {
       /* only double if we have had at least one add first */
```

```
       if (first == 0) {
          if ((err = ltc_mp.ecc_ptdbl(R, R, modulus, mp)) != CRYPT_OK)             { goto done; }
       }

       bitbuf <<= 1;
       if ((bitbuf & (1 << WINSIZE)) != 0) {
         if (first == 1){
            /* first add, so copy */
            if ((err = mp_copy(tG->x, R->x)) != CRYPT_OK)                          { goto done; }
            if ((err = mp_copy(tG->y, R->y)) != CRYPT_OK)                          { goto done; }
            if ((err = mp_copy(tG->z, R->z)) != CRYPT_OK)                          { goto done; }
            first = 0;
         } else {
            /* then add */
            if ((err = ltc_mp.ecc_ptadd(R, tG, R, modulus, mp)) != CRYPT_OK)       { goto done; }
         }
       }
     }
   }

   /* map R back from projective space */
   if (map) {
      err = ltc_ecc_map(R, modulus, mp);
   } else {
      err = CRYPT_OK;
   }
done:
   mp_montgomery_free(mp);
   ltc_ecc_del_point(tG);
   for (i = 0; i < 8; i++) {
       ltc_ecc_del_point(M[i]);
   }
   return err;
}
```

Here is the call graph for this function:

5.271 pk/ecc/ltc_ecc_mulmod_timing.c File Reference

5.271.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ltc_ecc_mulmod_timing.c.

```
#include "tomcrypt.h"
```

Include dependency graph for ltc_ecc_mulmod_timing.c:

5.272 pk/ecc/ltc_ecc_points.c File Reference

5.272.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ltc_ecc_points.c.

`#include "tomcrypt.h"`

Include dependency graph for ltc_ecc_points.c:

Functions

- ecc_point ∗ ltc_ecc_new_point (void)
  Allocate a new ECC point.
- void ltc_ecc_del_point (ecc_point ∗p)
  Free an ECC point from memory.

5.272.2 Function Documentation

5.272.2.1 void ltc_ecc_del_point (ecc_point ∗ *p*)

Free an ECC point from memory.

Parameters:

p The point to free

Definition at line 47 of file ltc_ecc_points.c.

References XFREE.

Referenced by ecc_shared_secret(), ecc_test(), and ltc_ecc_mulmod().

```
{
   /* prevents free'ing null arguments */
   if (p != NULL) {
      mp_clear_multi(p->x, p->y, p->z, NULL); /* note: p->z may be NULL but that's ok with this functio
      XFREE(p);
   }
}
```

5.272.2.2 ecc_point∗ ltc_ecc_new_point (void)

Allocate a new ECC point.

Returns:

A newly allocated point or NULL on error

Definition at line 30 of file ltc_ecc_points.c.

References CRYPT_OK, XFREE, and XMALLOC.

Referenced by ecc_shared_secret(), ecc_test(), ecc_verify_hash(), and ltc_ecc_mulmod().

```
{
   ecc_point *p;
   p = XMALLOC(sizeof(*p));
   if (p == NULL) {
      return NULL;
   }
   if (mp_init_multi(&p->x, &p->y, &p->z, NULL) != CRYPT_OK) {
      XFREE(p);
      return NULL;
   }
   return p;
}
```

5.273 pk/ecc/ltc_ecc_projective_add_point.c File Reference

5.273.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ltc_ecc_projective_add_point.c.

`#include "tomcrypt.h"`

Include dependency graph for ltc_ecc_projective_add_point.c:

Functions

- int ltc_ecc_projective_add_point (ecc_point ∗P, ecc_point ∗Q, ecc_point ∗R, void ∗modulus, void ∗mp)

  Add two ECC points.

5.273.2 Function Documentation

5.273.2.1 int ltc_ecc_projective_add_point (ecc_point ∗ *P*, ecc_point ∗ *Q*, ecc_point ∗ *R*, void ∗ *modulus*, void ∗ *mp*)

Add two ECC points.

Parameters:

P The point to add

Q The point to add

R [out] The destination of the double

modulus The modulus of the field the ECC curve is in

mp The "b" value from montgomery_setup()

Returns:

CRYPT_OK on success

Definition at line 35 of file ltc_ecc_projective_add_point.c.

References CRYPT_OK, LTC_ARGCHK, ltc_ecc_projective_dbl_point(), LTC_MP_EQ, LTC_MP_LT, t1, and t2.

```
{
   void  *t1, *t2, *x, *y, *z;
   int err;

   LTC_ARGCHK(P       != NULL);
   LTC_ARGCHK(Q       != NULL);
   LTC_ARGCHK(R       != NULL);
   LTC_ARGCHK(modulus != NULL);
   LTC_ARGCHK(mp      != NULL);

   if ((err = mp_init_multi(&t1, &t2, &x, &y, &z, NULL)) != CRYPT_OK) {
      return err;
   }
```

```

   /* should we dbl instead? */
   if ((err = mp_sub(modulus, Q->y, t1)) != CRYPT_OK)                           { goto done; }

   if ( (mp_cmp(P->x, Q->x) == LTC_MP_EQ) &&
        (Q->z != NULL && mp_cmp(P->z, Q->z) == LTC_MP_EQ) &&
        (mp_cmp(P->y, Q->y) == LTC_MP_EQ || mp_cmp(P->y, t1) == LTC_MP_EQ)) {
        mp_clear_multi(t1, t2, x, y, z, NULL);
        return ltc_ecc_projective_dbl_point(P, R, modulus, mp);
   }

   if ((err = mp_copy(P->x, x)) != CRYPT_OK)                                    { goto done; }
   if ((err = mp_copy(P->y, y)) != CRYPT_OK)                                    { goto done; }
   if ((err = mp_copy(P->z, z)) != CRYPT_OK)                                    { goto done; }

   /* if Z is one then these are no-operations */
   if (Q->z != NULL) {
      /* T1 = Z' * Z' */
      if ((err = mp_sqr(Q->z, t1)) != CRYPT_OK)                                 { goto done; }
      if ((err = mp_montgomery_reduce(t1, modulus, mp)) != CRYPT_OK)            { goto done; }
      /* X = X * T1 */
      if ((err = mp_mul(t1, x, x)) != CRYPT_OK)                                 { goto done; }
      if ((err = mp_montgomery_reduce(x, modulus, mp)) != CRYPT_OK)             { goto done; }
      /* T1 = Z' * T1 */
      if ((err = mp_mul(Q->z, t1, t1)) != CRYPT_OK)                             { goto done; }
      if ((err = mp_montgomery_reduce(t1, modulus, mp)) != CRYPT_OK)            { goto done; }
      /* Y = Y * T1 */
      if ((err = mp_mul(t1, y, y)) != CRYPT_OK)                                 { goto done; }
      if ((err = mp_montgomery_reduce(y, modulus, mp)) != CRYPT_OK)             { goto done; }
   }

   /* T1 = Z*Z */
   if ((err = mp_sqr(z, t1)) != CRYPT_OK)                                       { goto done; }
   if ((err = mp_montgomery_reduce(t1, modulus, mp)) != CRYPT_OK)               { goto done; }
   /* T2 = X' * T1 */
   if ((err = mp_mul(Q->x, t1, t2)) != CRYPT_OK)                                { goto done; }
   if ((err = mp_montgomery_reduce(t2, modulus, mp)) != CRYPT_OK)               { goto done; }
   /* T1 = Z * T1 */
   if ((err = mp_mul(z, t1, t1)) != CRYPT_OK)                                   { goto done; }
   if ((err = mp_montgomery_reduce(t1, modulus, mp)) != CRYPT_OK)               { goto done; }
   /* T1 = Y' * T1 */
   if ((err = mp_mul(Q->y, t1, t1)) != CRYPT_OK)                                { goto done; }
   if ((err = mp_montgomery_reduce(t1, modulus, mp)) != CRYPT_OK)               { goto done; }

   /* Y = Y - T1 */
   if ((err = mp_sub(y, t1, y)) != CRYPT_OK)                                    { goto done; }
   if (mp_cmp_d(y, 0) == LTC_MP_LT) {
      if ((err = mp_add(y, modulus, y)) != CRYPT_OK)                            { goto done; }
   }
   /* T1 = 2T1 */
   if ((err = mp_add(t1, t1, t1)) != CRYPT_OK)                                  { goto done; }
   if (mp_cmp(t1, modulus) != LTC_MP_LT) {
      if ((err = mp_sub(t1, modulus, t1)) != CRYPT_OK)                          { goto done; }
   }
   /* T1 = Y + T1 */
   if ((err = mp_add(t1, y, t1)) != CRYPT_OK)                                   { goto done; }
   if (mp_cmp(t1, modulus) != LTC_MP_LT) {
      if ((err = mp_sub(t1, modulus, t1)) != CRYPT_OK)                          { goto done; }
   }
   /* X = X - T2 */
   if ((err = mp_sub(x, t2, x)) != CRYPT_OK)                                    { goto done; }
   if (mp_cmp_d(x, 0) == LTC_MP_LT) {
      if ((err = mp_add(x, modulus, x)) != CRYPT_OK)                            { goto done; }
   }
   /* T2 = 2T2 */
   if ((err = mp_add(t2, t2, t2)) != CRYPT_OK)                                  { goto done; }
   if (mp_cmp(t2, modulus) != LTC_MP_LT) {
```

```
      if ((err = mp_sub(t2, modulus, t2)) != CRYPT_OK)                         { goto done; }
   }
   /* T2 = X + T2 */
   if ((err = mp_add(t2, x, t2)) != CRYPT_OK)                                  { goto done; }
   if (mp_cmp(t2, modulus) != LTC_MP_LT) {
      if ((err = mp_sub(t2, modulus, t2)) != CRYPT_OK)                         { goto done; }
   }

   /* if Z' != 1 */
   if (Q->z != NULL) {
      /* Z = Z * Z' */
      if ((err = mp_mul(z, Q->z, z)) != CRYPT_OK)                              { goto done; }
      if ((err = mp_montgomery_reduce(z, modulus, mp)) != CRYPT_OK)            { goto done; }
   }

   /* Z = Z * X */
   if ((err = mp_mul(z, x, z)) != CRYPT_OK)                                    { goto done; }
   if ((err = mp_montgomery_reduce(z, modulus, mp)) != CRYPT_OK)               { goto done; }

   /* T1 = T1 * X  */
   if ((err = mp_mul(t1, x, t1)) != CRYPT_OK)                                  { goto done; }
   if ((err = mp_montgomery_reduce(t1, modulus, mp)) != CRYPT_OK)              { goto done; }
   /* X = X * X */
   if ((err = mp_sqr(x, x)) != CRYPT_OK)                                       { goto done; }
   if ((err = mp_montgomery_reduce(x, modulus, mp)) != CRYPT_OK)               { goto done; }
   /* T2 = T2 * x */
   if ((err = mp_mul(t2, x, t2)) != CRYPT_OK)                                  { goto done; }
   if ((err = mp_montgomery_reduce(t2, modulus, mp)) != CRYPT_OK)              { goto done; }
   /* T1 = T1 * X  */
   if ((err = mp_mul(t1, x, t1)) != CRYPT_OK)                                  { goto done; }
   if ((err = mp_montgomery_reduce(t1, modulus, mp)) != CRYPT_OK)              { goto done; }

   /* X = Y*Y */
   if ((err = mp_sqr(y, x)) != CRYPT_OK)                                       { goto done; }
   if ((err = mp_montgomery_reduce(x, modulus, mp)) != CRYPT_OK)               { goto done; }
   /* X = X - T2 */
   if ((err = mp_sub(x, t2, x)) != CRYPT_OK)                                   { goto done; }
   if (mp_cmp_d(x, 0) == LTC_MP_LT) {
      if ((err = mp_add(x, modulus, x)) != CRYPT_OK)                           { goto done; }
   }

   /* T2 = T2 - X */
   if ((err = mp_sub(t2, x, t2)) != CRYPT_OK)                                  { goto done; }
   if (mp_cmp_d(t2, 0) == LTC_MP_LT) {
      if ((err = mp_add(t2, modulus, t2)) != CRYPT_OK)                         { goto done; }
   }
   /* T2 = T2 - X */
   if ((err = mp_sub(t2, x, t2)) != CRYPT_OK)                                  { goto done; }
   if (mp_cmp_d(t2, 0) == LTC_MP_LT) {
      if ((err = mp_add(t2, modulus, t2)) != CRYPT_OK)                         { goto done; }
   }
   /* T2 = T2 * Y */
   if ((err = mp_mul(t2, y, t2)) != CRYPT_OK)                                  { goto done; }
   if ((err = mp_montgomery_reduce(t2, modulus, mp)) != CRYPT_OK)              { goto done; }
   /* Y = T2 - T1 */
   if ((err = mp_sub(t2, t1, y)) != CRYPT_OK)                                  { goto done; }
   if (mp_cmp_d(y, 0) == LTC_MP_LT) {
      if ((err = mp_add(y, modulus, y)) != CRYPT_OK)                           { goto done; }
   }
   /* Y = Y/2 */
   if (mp_isodd(y)) {
      if ((err = mp_add(y, modulus, y)) != CRYPT_OK)                           { goto done; }
   }
   if ((err = mp_div_2(y, y)) != CRYPT_OK)                                     { goto done; }

   if ((err = mp_copy(x, R->x)) != CRYPT_OK)                                   { goto done; }
   if ((err = mp_copy(y, R->y)) != CRYPT_OK)                                   { goto done; }
```

```
   if ((err = mp_copy(z, R->z)) != CRYPT_OK)                                      { goto done; }

   err = CRYPT_OK;
   goto done;
done:
   mp_clear_multi(t1, t2, x, y, z, NULL);
   return err;
}
```

Here is the call graph for this function:

5.274 pk/ecc/ltc_ecc_projective_dbl_point.c File Reference

5.274.1 Detailed Description

ECC Crypto, Tom St Denis.

Definition in file ltc_ecc_projective_dbl_point.c.

`#include "tomcrypt.h"`

Include dependency graph for ltc_ecc_projective_dbl_point.c:

Functions

- int ltc_ecc_projective_dbl_point (ecc_point ∗P, ecc_point ∗R, void ∗modulus, void ∗mp)

  Double an ECC point.

5.274.2 Function Documentation

5.274.2.1 int ltc_ecc_projective_dbl_point (ecc_point ∗ *P*, ecc_point ∗ *R*, void ∗ *modulus*, void ∗ *mp*)

Double an ECC point.

Parameters:

P The point to double

R [out] The destination of the double

modulus The modulus of the field the ECC curve is in

mp The "b" value from montgomery_setup()

Returns:

CRYPT_OK on success

Definition at line 34 of file ltc_ecc_projective_dbl_point.c.

References CRYPT_OK, LTC_ARGCHK, LTC_MP_LT, t1, and t2.

Referenced by ltc_ecc_projective_add_point().

```
{
   void *t1, *t2;
   int err;

   LTC_ARGCHK(P       != NULL);
   LTC_ARGCHK(R       != NULL);
   LTC_ARGCHK(modulus != NULL);
   LTC_ARGCHK(mp      != NULL);

   if ((err = mp_init_multi(&t1, &t2, NULL)) != CRYPT_OK) {
      return err;
   }

   if (P != R) {
      if ((err = mp_copy(P->x, R->x)) != CRYPT_OK)                                { goto done; }
      if ((err = mp_copy(P->y, R->y)) != CRYPT_OK)                                { goto done; }
      if ((err = mp_copy(P->z, R->z)) != CRYPT_OK)                                { goto done; }
```

```
   }

   /* t1 = Z * Z */
   if ((err = mp_sqr(R->z, t1)) != CRYPT_OK)                                  { goto done; }
   if ((err = mp_montgomery_reduce(t1, modulus, mp)) != CRYPT_OK)             { goto done; }
   /* Z = Y * Z */
   if ((err = mp_mul(R->z, R->y, R->z)) != CRYPT_OK)                          { goto done; }
   if ((err = mp_montgomery_reduce(R->z, modulus, mp)) != CRYPT_OK)           { goto done; }
   /* Z = 2Z */
   if ((err = mp_add(R->z, R->z, R->z)) != CRYPT_OK)                          { goto done; }
   if (mp_cmp(R->z, modulus) != LTC_MP_LT) {
      if ((err = mp_sub(R->z, modulus, R->z)) != CRYPT_OK)                    { goto done; }
   }

   /* T2 = X - T1 */
   if ((err = mp_sub(R->x, t1, t2)) != CRYPT_OK)                              { goto done; }
   if (mp_cmp_d(t2, 0) == LTC_MP_LT) {
      if ((err = mp_add(t2, modulus, t2)) != CRYPT_OK)                        { goto done; }
   }
   /* T1 = X + T1 */
   if ((err = mp_add(t1, R->x, t1)) != CRYPT_OK)                              { goto done; }
   if (mp_cmp(t1, modulus) != LTC_MP_LT) {
      if ((err = mp_sub(t1, modulus, t1)) != CRYPT_OK)                        { goto done; }
   }
   /* T2 = T1 * T2 */
   if ((err = mp_mul(t1, t2, t2)) != CRYPT_OK)                                { goto done; }
   if ((err = mp_montgomery_reduce(t2, modulus, mp)) != CRYPT_OK)             { goto done; }
   /* T1 = 2T2 */
   if ((err = mp_add(t2, t2, t1)) != CRYPT_OK)                                { goto done; }
   if (mp_cmp(t1, modulus) != LTC_MP_LT) {
      if ((err = mp_sub(t1, modulus, t1)) != CRYPT_OK)                        { goto done; }
   }
   /* T1 = T1 + T2 */
   if ((err = mp_add(t1, t2, t1)) != CRYPT_OK)                                { goto done; }
   if (mp_cmp(t1, modulus) != LTC_MP_LT) {
      if ((err = mp_sub(t1, modulus, t1)) != CRYPT_OK)                        { goto done; }
   }

   /* Y = 2Y */
   if ((err = mp_add(R->y, R->y, R->y)) != CRYPT_OK)                          { goto done; }
   if (mp_cmp(R->y, modulus) != LTC_MP_LT) {
      if ((err = mp_sub(R->y, modulus, R->y)) != CRYPT_OK)                    { goto done; }
   }
   /* Y = Y * Y */
   if ((err = mp_sqr(R->y, R->y)) != CRYPT_OK)                                { goto done; }
   if ((err = mp_montgomery_reduce(R->y, modulus, mp)) != CRYPT_OK)           { goto done; }
   /* T2 = Y * Y */
   if ((err = mp_sqr(R->y, t2)) != CRYPT_OK)                                  { goto done; }
   if ((err = mp_montgomery_reduce(t2, modulus, mp)) != CRYPT_OK)             { goto done; }
   /* T2 = T2/2 */
   if (mp_isodd(t2)) {
      if ((err = mp_add(t2, modulus, t2)) != CRYPT_OK)                        { goto done; }
   }
   if ((err = mp_div_2(t2, t2)) != CRYPT_OK)                                  { goto done; }
   /* Y = Y * X */
   if ((err = mp_mul(R->y, R->x, R->y)) != CRYPT_OK)                          { goto done; }
   if ((err = mp_montgomery_reduce(R->y, modulus, mp)) != CRYPT_OK)           { goto done; }

   /* X  = T1 * T1 */
   if ((err = mp_sqr(t1, R->x)) != CRYPT_OK)                                  { goto done; }
   if ((err = mp_montgomery_reduce(R->x, modulus, mp)) != CRYPT_OK)           { goto done; }
   /* X = X - Y */
   if ((err = mp_sub(R->x, R->y, R->x)) != CRYPT_OK)                          { goto done; }
   if (mp_cmp_d(R->x, 0) == LTC_MP_LT) {
      if ((err = mp_add(R->x, modulus, R->x)) != CRYPT_OK)                    { goto done; }
   }
   /* X = X - Y */
```

```
   if ((err = mp_sub(R->x, R->y, R->x)) != CRYPT_OK)                              { goto done; }
   if (mp_cmp_d(R->x, 0) == LTC_MP_LT) {
      if ((err = mp_add(R->x, modulus, R->x)) != CRYPT_OK)                        { goto done; }
   }

   /* Y = Y - X */
   if ((err = mp_sub(R->y, R->x, R->y)) != CRYPT_OK)                              { goto done; }
   if (mp_cmp_d(R->y, 0) == LTC_MP_LT) {
      if ((err = mp_add(R->y, modulus, R->y)) != CRYPT_OK)                        { goto done; }
   }
   /* Y = Y * T1 */
   if ((err = mp_mul(R->y, t1, R->y)) != CRYPT_OK)                                { goto done; }
   if ((err = mp_montgomery_reduce(R->y, modulus, mp)) != CRYPT_OK)               { goto done; }
   /* Y = Y - T2 */
   if ((err = mp_sub(R->y, t2, R->y)) != CRYPT_OK)                                { goto done; }
   if (mp_cmp_d(R->y, 0) == LTC_MP_LT) {
      if ((err = mp_add(R->y, modulus, R->y)) != CRYPT_OK)                        { goto done; }
   }

   err = CRYPT_OK;
   goto done;
done:
   mp_clear_multi(t1, t2, NULL);
   return err;
}
```

5.275 pk/katja/katja_decrypt_key.c File Reference

5.275.1 Detailed Description

Katja PKCS #1 OAEP Decryption, Tom St Denis.

Definition in file katja_decrypt_key.c.

```
#include "tomcrypt.h"
```

Include dependency graph for katja_decrypt_key.c:

5.276 pk/katja/katja_encrypt_key.c File Reference

5.276.1 Detailed Description

Katja PKCS-style OAEP encryption, Tom St Denis.

Definition in file katja_encrypt_key.c.

```
#include "tomcrypt.h"
```

Include dependency graph for katja_encrypt_key.c:

5.277 pk/katja/katja_export.c File Reference

5.277.1 Detailed Description

Export Katja PKCS-style keys, Tom St Denis.

Definition in file katja_export.c.

```
#include "tomcrypt.h"
```

Include dependency graph for katja_export.c:

5.278 pk/katja/katja_exptmod.c File Reference

5.278.1 Detailed Description

Katja PKCS-style exptmod, Tom St Denis.

Definition in file katja_exptmod.c.

```
#include "tomcrypt.h"
```

Include dependency graph for katja_exptmod.c:

5.279 pk/katja/katja_free.c File Reference

5.279.1 Detailed Description

Free an Katja key, Tom St Denis.

Definition in file katja_free.c.

`#include "tomcrypt.h"`

Include dependency graph for katja_free.c:

5.280 pk/katja/katja_import.c File Reference

5.280.1 Detailed Description

Import a PKCS-style Katja key, Tom St Denis.

Definition in file katja_import.c.

`#include "tomcrypt.h"`

Include dependency graph for katja_import.c:

5.281 pk/katja/katja_make_key.c File Reference

5.281.1 Detailed Description

Katja key generation, Tom St Denis.

Definition in file katja_make_key.c.

```
#include "tomcrypt.h"
```

Include dependency graph for katja_make_key.c:

5.282 pk/pkcs1/pkcs_1_i2osp.c File Reference

5.282.1 Detailed Description

Integer to Octet I2OSP, Tom St Denis.

Definition in file pkcs_1_i2osp.c.

`#include "tomcrypt.h"`

Include dependency graph for pkcs_1_i2osp.c:

Functions

- int pkcs_1_i2osp (void ∗n, unsigned long modulus_len, unsigned char ∗out)
  PKCS #1 Integer to binary.

5.282.2 Function Documentation

5.282.2.1 int pkcs_1_i2osp (void ∗ *n*, unsigned long *modulus_len*, unsigned char ∗ *out*)

PKCS #1 Integer to binary.

Parameters:

n The integer to store

modulus_len The length of the RSA modulus

out [out] The destination for the integer

Returns:

CRYPT_OK if successful

Definition at line 31 of file pkcs_1_i2osp.c.

References CRYPT_BUFFER_OVERFLOW, edge::size, and zeromem().

```
32 {
33    unsigned long size;
34 
35    size = mp_unsigned_bin_size(n);
36 
37    if (size > modulus_len) {
38       return CRYPT_BUFFER_OVERFLOW;
39    }
40 
41    /* store it */
42    zeromem(out, modulus_len);
43    return mp_to_unsigned_bin(n, out+(modulus_len-size));
44 }
```

Here is the call graph for this function:

5.283 pk/pkcs1/pkcs_1_mgf1.c File Reference

5.283.1 Detailed Description

The Mask Generation Function (MGF1) for PKCS #1, Tom St Denis.

Definition in file pkcs_1_mgf1.c.

`#include "tomcrypt.h"`

Include dependency graph for pkcs_1_mgf1.c:

Functions

- int pkcs_1_mgf1 (int hash_idx, const unsigned char ∗seed, unsigned long seedlen, unsigned char ∗mask, unsigned long masklen)

    Perform PKCS #1 MGF1 (internal).

5.283.2 Function Documentation

5.283.2.1 int pkcs_1_mgf1 (int *hash_idx*, const unsigned char ∗ *seed*, unsigned long *seedlen*, unsigned char ∗ *mask*, unsigned long *masklen*)

Perform PKCS #1 MGF1 (internal).

Parameters:

seed The seed for MGF1

seedlen The length of the seed

hash_idx The index of the hash desired

mask [out] The destination

masklen The length of the mask desired

Returns:

CRYPT_OK if successful

Definition at line 29 of file pkcs_1_mgf1.c.

References CRYPT_MEM, CRYPT_OK, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, XFREE, and XMALLOC.

Referenced by pkcs_1_oaep_decode(), pkcs_1_oaep_encode(), pkcs_1_pss_decode(), and pkcs_1_pss_-encode().

```
{
   unsigned long hLen, x;
   ulong32       counter;
   int           err;
   hash_state    *md;
   unsigned char *buf;

   LTC_ARGCHK(seed != NULL);
   LTC_ARGCHK(mask != NULL);

```

```
   /* ensure valid hash */
   if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
      return err;
   }

   /* get hash output size */
   hLen = hash_descriptor[hash_idx].hashsize;

   /* allocate memory */
   md  = XMALLOC(sizeof(hash_state));
   buf = XMALLOC(hLen);
   if (md == NULL || buf == NULL) {
      if (md != NULL) {
         XFREE(md);
      }
      if (buf != NULL) {
         XFREE(buf);
      }
      return CRYPT_MEM;
   }

   /* start counter */
   counter = 0;

   while (masklen > 0) {
       /* handle counter */
       STORE32H(counter, buf);
       ++counter;

       /* get hash of seed || counter */
       if ((err = hash_descriptor[hash_idx].init(md)) != CRYPT_OK) {
          goto LBL_ERR;
       }
       if ((err = hash_descriptor[hash_idx].process(md, seed, seedlen)) != CRYPT_OK) {
          goto LBL_ERR;
       }
       if ((err = hash_descriptor[hash_idx].process(md, buf, 4)) != CRYPT_OK) {
          goto LBL_ERR;
       }
       if ((err = hash_descriptor[hash_idx].done(md, buf)) != CRYPT_OK) {
          goto LBL_ERR;
       }

       /* store it */
       for (x = 0; x < hLen && masklen > 0; x++, masklen--) {
          *mask++ = buf[x];
       }
   }

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(buf, hLen);
   zeromem(md,  sizeof(hash_state));
#endif

   XFREE(buf);
   XFREE(md);

   return err;
}
```

Here is the call graph for this function:

5.284 pk/pkcs1/pkcs_1_oaep_decode.c File Reference

5.284.1 Detailed Description

OAEP Padding for PKCS #1, Tom St Denis.

Definition in file pkcs_1_oaep_decode.c.

```
#include "tomcrypt.h"
```

Include dependency graph for pkcs_1_oaep_decode.c:

Functions

- int pkcs_1_oaep_decode (const unsigned char ∗msg, unsigned long msglen, const unsigned char ∗lparam, unsigned long lparamlen, unsigned long modulus_bitlen, int hash_idx, unsigned char ∗out, unsigned long ∗outlen, int ∗res)

  PKCS #1 v2.00 OAEP decode.

5.284.2 Function Documentation

5.284.2.1 int pkcs_1_oaep_decode (const unsigned char ∗ *msg*, unsigned long *msglen*, const unsigned char ∗ *lparam*, unsigned long *lparamlen*, unsigned long *modulus_bitlen*, int *hash_idx*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, int ∗ *res*)

PKCS #1 v2.00 OAEP decode.

Parameters:

msg The encoded data to decode

msglen The length of the encoded data (octets)

lparam The session or system data (can be NULL)

lparamlen The length of the lparam

modulus_bitlen The bit length of the RSA modulus

hash_idx The index of the hash desired

out [out] Destination of decoding

outlen [in/out] The max size and resulting size of the decoding

res [out] Result of decoding, 1==valid, 0==invalid

Returns:

CRYPT_OK if successful (even if invalid)

Definition at line 33 of file pkcs_1_oaep_decode.c.

References CRYPT_MEM, CRYPT_OK, CRYPT_PK_INVALID_SIZE, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, mask, pkcs_1_mgf1(), XFREE, XMALLOC, and XMEMCPY.

Referenced by rsa_decrypt_key_ex().

```
{
   unsigned char *DB, *seed, *mask;
   unsigned long hLen, x, y, modulus_len;
   int           err;

   LTC_ARGCHK(msg    != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(res    != NULL);

   /* default to invalid packet */
   *res = 0;

   /* test valid hash */
   if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
      return err;
   }
   hLen        = hash_descriptor[hash_idx].hashsize;
   modulus_len = (modulus_bitlen >> 3) + (modulus_bitlen & 7 ? 1 : 0);

   /* test hash/message size */
   if ((2*hLen >= (modulus_len - 2)) || (msglen != modulus_len)) {
      return CRYPT_PK_INVALID_SIZE;
   }

   /* allocate ram for DB/mask/salt of size modulus_len */
   DB   = XMALLOC(modulus_len);
   mask = XMALLOC(modulus_len);
   seed = XMALLOC(hLen);
   if (DB == NULL || mask == NULL || seed == NULL) {
      if (DB != NULL) {
         XFREE(DB);
      }
      if (mask != NULL) {
         XFREE(mask);
      }
      if (seed != NULL) {
         XFREE(seed);
      }
      return CRYPT_MEM;
   }

   /* ok so it's now in the form

      0x00  || maskedseed || maskedDB

       1    ||   hLen     ||  modulus_len - hLen - 1

    */

   /* must have leading 0x00 byte */
   if (msg[0] != 0x00) {
      err = CRYPT_OK;
      goto LBL_ERR;
   }

   /* now read the masked seed */
   x = 1;
   XMEMCPY(seed, msg + x, hLen);
   x += hLen;

   /* now read the masked DB */
   XMEMCPY(DB, msg + x, modulus_len - hLen - 1);
   x += modulus_len - hLen - 1;

   /* compute MGF1 of maskedDB (hLen) */
   if ((err = pkcs_1_mgf1(hash_idx, DB, modulus_len - hLen - 1, mask, hLen)) != CRYPT_OK) {
```

```
      goto LBL_ERR;
   }

   /* XOR against seed */
   for (y = 0; y < hLen; y++) {
      seed[y] ^= mask[y];
   }

   /* compute MGF1 of seed (k - hlen - 1) */
   if ((err = pkcs_1_mgf1(hash_idx, seed, hLen, mask, modulus_len - hLen - 1)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* xor against DB */
   for (y = 0; y < (modulus_len - hLen - 1); y++) {
       DB[y] ^= mask[y];
   }

   /* now DB == lhash || PS || 0x01 || M, PS == k - mlen - 2hlen - 2 zeroes */

   /* compute lhash and store it in seed [reuse temps!] */
   x = modulus_len;
   if (lparam != NULL) {
      if ((err = hash_memory(hash_idx, lparam, lparamlen, seed, &x)) != CRYPT_OK) {
         goto LBL_ERR;
      }
   } else {
      /* can't pass hash_memory a NULL so use DB with zero length */
      if ((err = hash_memory(hash_idx, DB, 0, seed, &x)) != CRYPT_OK) {
         goto LBL_ERR;
      }
   }

   /* compare the lhash'es */
   if (XMEMCMP(seed, DB, hLen) != 0) {
      err = CRYPT_OK;
      goto LBL_ERR;
   }

   /* now zeroes before a 0x01 */
   for (x = hLen; x < (modulus_len - hLen - 1) && DB[x] == 0x00; x++) {
      /* step... */
   }

   /* error out if wasn't 0x01 */
   if (x == (modulus_len - hLen - 1) || DB[x] != 0x01) {
      err = CRYPT_INVALID_PACKET;
      goto LBL_ERR;
   }

   /* rest is the message (and skip 0x01) */
   if ((modulus_len - hLen - 1 - ++x) > *outlen) {
      *outlen = modulus_len - hLen - 1 - x;
      err = CRYPT_BUFFER_OVERFLOW;
      goto LBL_ERR;
   }

   /* copy message */
   *outlen = modulus_len - hLen - 1 - x;
   XMEMCPY(out, DB + x, modulus_len - hLen - 1 - x);
   x += modulus_len - hLen - 1;

   /* valid packet */
   *res = 1;

   err = CRYPT_OK;
LBL_ERR:
```

```
#ifdef LTC_CLEAN_STACK
   zeromem(DB,   modulus_len);
   zeromem(seed, hLen);
   zeromem(mask, modulus_len);
#endif

   XFREE(seed);
   XFREE(mask);
   XFREE(DB);

   return err;
}
```

Here is the call graph for this function:

5.285 pk/pkcs1/pkcs_1_oaep_encode.c File Reference

5.285.1 Detailed Description

OAEP Padding for PKCS #1, Tom St Denis.

Definition in file pkcs_1_oaep_encode.c.

`#include "tomcrypt.h"`

Include dependency graph for pkcs_1_oaep_encode.c:

Functions

- int pkcs_1_oaep_encode (const unsigned char ∗msg, unsigned long msglen, const unsigned char ∗lparam, unsigned long lparamlen, unsigned long modulus_bitlen, prng_state ∗prng, int prng_idx, int hash_idx, unsigned char ∗out, unsigned long ∗outlen)

  PKCS #1 v2.00 OAEP encode.

5.285.2 Function Documentation

5.285.2.1 int pkcs_1_oaep_encode (const unsigned char ∗ *msg*, unsigned long *msglen*, const unsigned char ∗ *lparam*, unsigned long *lparamlen*, unsigned long *modulus_bitlen*, prng_state ∗ *prng*, int *prng_idx*, int *hash_idx*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

PKCS #1 v2.00 OAEP encode.

Parameters:

msg The data to encode

msglen The length of the data to encode (octets)

lparam A session or system parameter (can be NULL)

lparamlen The length of the lparam data

modulus_bitlen The bit length of the RSA modulus

prng An active PRNG state

prng_idx The index of the PRNG desired

hash_idx The index of the hash desired

out [out] The destination for the encoded data

outlen [in/out] The max size and resulting size of the encoded data

Returns:

CRYPT_OK if successful

Definition at line 34 of file pkcs_1_oaep_encode.c.

References CRYPT_ERROR_READPRNG, CRYPT_MEM, CRYPT_OK, CRYPT_PK_INVALID_SIZE, hash_descriptor, hash_is_valid(), hash_memory(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, mask, pkcs_1_mgf1(), prng_descriptor, prng_is_valid(), XFREE, XMALLOC, XMEMCPY, and XMEMSET.

Referenced by rsa_encrypt_key_ex().

```
{
   unsigned char *DB, *seed, *mask;
   unsigned long hLen, x, y, modulus_len;
   int           err;

   LTC_ARGCHK(msg    != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);

   /* test valid hash */
   if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
      return err;
   }

   /* valid prng */
   if ((err = prng_is_valid(prng_idx)) != CRYPT_OK) {
      return err;
   }

   hLen        = hash_descriptor[hash_idx].hashsize;
   modulus_len = (modulus_bitlen >> 3) + (modulus_bitlen & 7 ? 1 : 0);

   /* test message size */
   if ((2*hLen >= (modulus_len - 2)) || (msglen > (modulus_len - 2*hLen - 2))) {
      return CRYPT_PK_INVALID_SIZE;
   }

   /* allocate ram for DB/mask/salt of size modulus_len */
   DB   = XMALLOC(modulus_len);
   mask = XMALLOC(modulus_len);
   seed = XMALLOC(hLen);
   if (DB == NULL || mask == NULL || seed == NULL) {
      if (DB != NULL) {
         XFREE(DB);
      }
      if (mask != NULL) {
         XFREE(mask);
      }
      if (seed != NULL) {
         XFREE(seed);
      }
      return CRYPT_MEM;
   }

   /* get lhash */
   /* DB == lhash || PS || 0x01 || M, PS == k - mlen - 2hlen - 2 zeroes */
   x = modulus_len;
   if (lparam != NULL) {
      if ((err = hash_memory(hash_idx, lparam, lparamlen, DB, &x)) != CRYPT_OK) {
         goto LBL_ERR;
      }
   } else {
      /* can't pass hash_memory a NULL so use DB with zero length */
      if ((err = hash_memory(hash_idx, DB, 0, DB, &x)) != CRYPT_OK) {
         goto LBL_ERR;
      }
   }

   /* append PS then 0x01 (to lhash)  */
   x = hLen;
   y = modulus_len - msglen - 2*hLen - 2;
   XMEMSET(DB+x, 0, y);
   x += y;

   /* 0x01 byte */
   DB[x++] = 0x01;

```

```
   /* message (length = msglen) */
   XMEMCPY(DB+x, msg, msglen);
   x += msglen;

   /* now choose a random seed */
   if (prng_descriptor[prng_idx].read(seed, hLen, prng) != hLen) {
      err = CRYPT_ERROR_READPRNG;
      goto LBL_ERR;
   }

   /* compute MGF1 of seed (k - hlen - 1) */
   if ((err = pkcs_1_mgf1(hash_idx, seed, hLen, mask, modulus_len - hLen - 1)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* xor against DB */
   for (y = 0; y < (modulus_len - hLen - 1); y++) {
       DB[y] ^= mask[y];
   }

   /* compute MGF1 of maskedDB (hLen) */
   if ((err = pkcs_1_mgf1(hash_idx, DB, modulus_len - hLen - 1, mask, hLen)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* XOR against seed */
   for (y = 0; y < hLen; y++) {
      seed[y] ^= mask[y];
   }

   /* create string of length modulus_len */
   if (*outlen < modulus_len) {
      *outlen = modulus_len;
      err = CRYPT_BUFFER_OVERFLOW;
      goto LBL_ERR;
   }

   /* start output which is 0x00 || maskedSeed || maskedDB */
   x = 0;
   out[x++] = 0x00;
   XMEMCPY(out+x, seed, hLen);
   x += hLen;
   XMEMCPY(out+x, DB, modulus_len - hLen - 1);
   x += modulus_len - hLen - 1;

   *outlen = x;

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(DB,   modulus_len);
   zeromem(seed, hLen);
   zeromem(mask, modulus_len);
#endif

   XFREE(seed);
   XFREE(mask);
   XFREE(DB);

   return err;
}
```

Here is the call graph for this function:

5.286 pk/pkcs1/pkcs_1_os2ip.c File Reference

5.286.1 Detailed Description

Octet to Integer OS2IP, Tom St Denis.

Definition in file pkcs_1_os2ip.c.

```
#include "tomcrypt.h"
```

Include dependency graph for pkcs_1_os2ip.c:

Functions

- int pkcs_1_os2ip (void *n, unsigned char *in, unsigned long inlen)

  Read a binary string into an mp_int.

5.286.2 Function Documentation

5.286.2.1 int pkcs_1_os2ip (void * *n*, unsigned char * *in*, unsigned long *inlen*)

Read a binary string into an mp_int.

Parameters:

n [out] The mp_int destination

in The binary string to read

inlen The length of the binary string

Returns:

CRYPT_OK if successful

Definition at line 26 of file pkcs_1_os2ip.c.

```
{
   return mp_read_unsigned_bin(n, in, inlen);
}
```

5.287 pk/pkcs1/pkcs_1_pss_decode.c File Reference

5.287.1 Detailed Description

PKCS #1 PSS Signature Padding, Tom St Denis.

Definition in file pkcs_1_pss_decode.c.

`#include "tomcrypt.h"`

Include dependency graph for pkcs_1_pss_decode.c:

Functions

- int pkcs_1_pss_decode (const unsigned char ∗msghash, unsigned long msghashlen, const unsigned char ∗sig, unsigned long siglen, unsigned long saltlen, int hash_idx, unsigned long modulus_bitlen, int ∗res)

  PKCS #1 v2.00 PSS decode.

5.287.2 Function Documentation

5.287.2.1 int pkcs_1_pss_decode (const unsigned char ∗ *msghash*, unsigned long *msghashlen*, const unsigned char ∗ *sig*, unsigned long *siglen*, unsigned long *saltlen*, int *hash_idx*, unsigned long *modulus_bitlen*, int ∗ *res*)

PKCS #1 v2.00 PSS decode.

Parameters:

msghash The hash to verify

msghashlen The length of the hash (octets)

sig The signature data (encoded data)

siglen The length of the signature data (octets)

saltlen The length of the salt used (octets)

hash_idx The index of the hash desired

modulus_bitlen The bit length of the RSA modulus

res [out] The result of the comparison, 1==valid, 0==invalid

Returns:

CRYPT_OK if successful (even if the comparison failed)

Definition at line 32 of file pkcs_1_pss_decode.c.

References CRYPT_MEM, CRYPT_OK, CRYPT_PK_INVALID_SIZE, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, mask, pkcs_1_mgf1(), XFREE, XMALLOC, and XMEMCPY.

Referenced by rsa_verify_hash_ex().

```
{
   unsigned char *DB, *mask, *salt, *hash;
   unsigned long x, y, hLen, modulus_len;
```

```
   int           err;
   hash_state    md;

   LTC_ARGCHK(msghash != NULL);
   LTC_ARGCHK(res     != NULL);

   /* default to invalid */
   *res = 0;

   /* ensure hash is valid */
   if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
      return err;
   }

   hLen        = hash_descriptor[hash_idx].hashsize;
   modulus_len = (modulus_bitlen>>3) + (modulus_bitlen & 7 ? 1 : 0);

   /* check sizes */
   if ((saltlen > modulus_len) ||
       (modulus_len < hLen + saltlen + 2) || (siglen != modulus_len)) {
      return CRYPT_PK_INVALID_SIZE;
   }

   /* allocate ram for DB/mask/salt/hash of size modulus_len */
   DB   = XMALLOC(modulus_len);
   mask = XMALLOC(modulus_len);
   salt = XMALLOC(modulus_len);
   hash = XMALLOC(modulus_len);
   if (DB == NULL || mask == NULL || salt == NULL || hash == NULL) {
      if (DB != NULL) {
         XFREE(DB);
      }
      if (mask != NULL) {
         XFREE(mask);
      }
      if (salt != NULL) {
         XFREE(salt);
      }
      if (hash != NULL) {
         XFREE(hash);
      }
      return CRYPT_MEM;
   }

   /* ensure the 0xBC byte */
   if (sig[siglen-1] != 0xBC) {
      err = CRYPT_OK;
      goto LBL_ERR;
   }

   /* copy out the DB */
   x = 0;
   XMEMCPY(DB, sig + x, modulus_len - hLen - 1);
   x += modulus_len - hLen - 1;

   /* copy out the hash */
   XMEMCPY(hash, sig + x, hLen);
   x += hLen;

   /* check the MSB */
   if ((sig[0] & ~(0xFF >> ((modulus_len<<3) - (modulus_bitlen-1)))) != 0) {
      err = CRYPT_OK;
      goto LBL_ERR;
   }

   /* generate mask of length modulus_len - hLen - 1 from hash */
   if ((err = pkcs_1_mgf1(hash_idx, hash, hLen, mask, modulus_len - hLen - 1)) != CRYPT_OK) {
```

```
      goto LBL_ERR;
   }

   /* xor against DB */
   for (y = 0; y < (modulus_len - hLen - 1); y++) {
      DB[y] ^= mask[y];
   }

   /* now clear the first byte [make sure smaller than modulus] */
   DB[0] &= 0xFF >> ((modulus_len<<3) - (modulus_bitlen-1));

   /* DB = PS || 0x01 || salt, PS == modulus_len - saltlen - hLen - 2 zero bytes */

   /* check for zeroes and 0x01 */
   for (x = 0; x < modulus_len - saltlen - hLen - 2; x++) {
       if (DB[x] != 0x00) {
          err = CRYPT_OK;
          goto LBL_ERR;
       }
   }

   /* check for the 0x01 */
   if (DB[x++] != 0x01) {
      err = CRYPT_OK;
      goto LBL_ERR;
   }

   /* M = (eight) 0x00 || msghash || salt, mask = H(M) */
   if ((err = hash_descriptor[hash_idx].init(&md)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   zeromem(mask, 8);
   if ((err = hash_descriptor[hash_idx].process(&md, mask, 8)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   if ((err = hash_descriptor[hash_idx].process(&md, msghash, msghashlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   if ((err = hash_descriptor[hash_idx].process(&md, DB+x, saltlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   if ((err = hash_descriptor[hash_idx].done(&md, mask)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* mask == hash means valid signature */
   if (XMEMCMP(mask, hash, hLen) == 0) {
      *res = 1;
   }

   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(DB,   modulus_len);
   zeromem(mask, modulus_len);
   zeromem(salt, modulus_len);
   zeromem(hash, modulus_len);
#endif

   XFREE(hash);
   XFREE(salt);
   XFREE(mask);
   XFREE(DB);

   return err;
}
```

Here is the call graph for this function:

5.288 pk/pkcs1/pkcs_1_pss_encode.c File Reference

5.288.1 Detailed Description

PKCS #1 PSS Signature Padding, Tom St Denis.

Definition in file pkcs_1_pss_encode.c.

`#include "tomcrypt.h"`

Include dependency graph for pkcs_1_pss_encode.c:

Functions

- int pkcs_1_pss_encode (const unsigned char ∗msghash, unsigned long msghashlen, unsigned long saltlen, prng_state ∗prng, int prng_idx, int hash_idx, unsigned long modulus_bitlen, unsigned char ∗out, unsigned long ∗outlen)

  PKCS #1 v2.00 Signature Encoding.

5.288.2 Function Documentation

5.288.2.1 int pkcs_1_pss_encode (const unsigned char ∗ *msghash*, unsigned long *msghashlen*, unsigned long *saltlen*, prng_state ∗ *prng*, int *prng_idx*, int *hash_idx*, unsigned long *modulus_bitlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*)

PKCS #1 v2.00 Signature Encoding.

Parameters:

msghash The hash to encode

msghashlen The length of the hash (octets)

saltlen The length of the salt desired (octets)

prng An active PRNG context

prng_idx The index of the PRNG desired

hash_idx The index of the hash desired

modulus_bitlen The bit length of the RSA modulus

out [out] The destination of the encoding

outlen [in/out] The max size and resulting size of the encoded data

Returns:

CRYPT_OK if successful

Definition at line 33 of file pkcs_1_pss_encode.c.

References CRYPT_ERROR_READPRNG, CRYPT_MEM, CRYPT_OK, CRYPT_PK_INVALID_SIZE, hash_descriptor, hash_is_valid(), ltc_hash_descriptor::hashsize, LTC_ARGCHK, mask, pkcs_1_mgf1(), prng_descriptor, prng_is_valid(), XFREE, XMALLOC, XMEMCPY, XMEMSET, and zeromem().

Referenced by rsa_sign_hash_ex().

```
{
   unsigned char *DB, *mask, *salt, *hash;
   unsigned long x, y, hLen, modulus_len;
   int           err;
   hash_state    md;

   LTC_ARGCHK(msghash != NULL);
   LTC_ARGCHK(out     != NULL);
   LTC_ARGCHK(outlen  != NULL);

   /* ensure hash and PRNG are valid */
   if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
      return err;
   }
   if ((err = prng_is_valid(prng_idx)) != CRYPT_OK) {
      return err;
   }

   hLen        = hash_descriptor[hash_idx].hashsize;
   modulus_len = (modulus_bitlen>>3) + (modulus_bitlen & 7 ? 1 : 0);

   /* check sizes */
   if ((saltlen > modulus_len) || (modulus_len < hLen + saltlen + 2)) {
      return CRYPT_PK_INVALID_SIZE;
   }

   /* allocate ram for DB/mask/salt/hash of size modulus_len */
   DB   = XMALLOC(modulus_len);
   mask = XMALLOC(modulus_len);
   salt = XMALLOC(modulus_len);
   hash = XMALLOC(modulus_len);
   if (DB == NULL || mask == NULL || salt == NULL || hash == NULL) {
      if (DB != NULL) {
         XFREE(DB);
      }
      if (mask != NULL) {
         XFREE(mask);
      }
      if (salt != NULL) {
         XFREE(salt);
      }
      if (hash != NULL) {
         XFREE(hash);
      }
      return CRYPT_MEM;
   }


   /* generate random salt */
   if (saltlen > 0) {
      if (prng_descriptor[prng_idx].read(salt, saltlen, prng) != saltlen) {
         err = CRYPT_ERROR_READPRNG;
         goto LBL_ERR;
      }
   }

   /* M = (eight) 0x00 || msghash || salt, hash = H(M) */
   if ((err = hash_descriptor[hash_idx].init(&md)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   zeromem(DB, 8);
   if ((err = hash_descriptor[hash_idx].process(&md, DB, 8)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   if ((err = hash_descriptor[hash_idx].process(&md, msghash, msghashlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
```

```
   if ((err = hash_descriptor[hash_idx].process(&md, salt, saltlen)) != CRYPT_OK) {
      goto LBL_ERR;
   }
   if ((err = hash_descriptor[hash_idx].done(&md, hash)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* generate DB = PS || 0x01 || salt, PS == modulus_len - saltlen - hLen - 2 zero bytes */
   x = 0;
   XMEMSET(DB + x, 0, modulus_len - saltlen - hLen - 2);
   x += modulus_len - saltlen - hLen - 2;
   DB[x++] = 0x01;
   XMEMCPY(DB + x, salt, saltlen);
   x += saltlen;

   /* generate mask of length modulus_len - hLen - 1 from hash */
   if ((err = pkcs_1_mgf1(hash_idx, hash, hLen, mask, modulus_len - hLen - 1)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   /* xor against DB */
   for (y = 0; y < (modulus_len - hLen - 1); y++) {
      DB[y] ^= mask[y];
   }

   /* output is DB || hash || 0xBC */
   if (*outlen < modulus_len) {
      *outlen = modulus_len;
      err = CRYPT_BUFFER_OVERFLOW;
      goto LBL_ERR;
   }

   /* DB len = modulus_len - hLen - 1 */
   y = 0;
   XMEMCPY(out + y, DB, modulus_len - hLen - 1);
   y += modulus_len - hLen - 1;

   /* hash */
   XMEMCPY(out + y, hash, hLen);
   y += hLen;

   /* 0xBC */
   out[y] = 0xBC;

   /* now clear the 8*modulus_len - modulus_bitlen most significant bits */
   out[0] &= 0xFF >> ((modulus_len<<3) - (modulus_bitlen-1));

   /* store output size */
   *outlen = modulus_len;
   err = CRYPT_OK;
LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(DB,   modulus_len);
   zeromem(mask, modulus_len);
   zeromem(salt, modulus_len);
   zeromem(hash, modulus_len);
#endif

   XFREE(hash);
   XFREE(salt);
   XFREE(mask);
   XFREE(DB);

   return err;
}
```

Here is the call graph for this function:

5.289 pk/pkcs1/pkcs_1_v1_5_decode.c File Reference

5.289.1 Detailed Description

PKCS #1 v1.5 Padding. (Andreas Lange)

Definition in file pkcs_1_v1_5_decode.c.

`#include "tomcrypt.h"`

Include dependency graph for pkcs_1_v1_5_decode.c:

Functions

- int pkcs_1_v1_5_decode (const unsigned char ∗msg, unsigned long msglen, int block_type, unsigned long modulus_bitlen, unsigned char ∗out, unsigned long ∗outlen, int ∗is_valid)

    PKCS #1 v1.5 decode.

5.289.2 Function Documentation

5.289.2.1 int pkcs_1_v1_5_decode (const unsigned char ∗ *msg*, unsigned long *msglen*, int *block_type*, unsigned long *modulus_bitlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, int ∗ *is_valid*)

PKCS #1 v1.5 decode.

Parameters:

msg The encoded data to decode

msglen The length of the encoded data (octets)

block_type Block type to use in padding (

See also:

ltc_pkcs_1_v1_5_blocks)

Parameters:

modulus_bitlen The bit length of the RSA modulus

out [out] Destination of decoding

outlen [in/out] The max size and resulting size of the decoding

Returns:

CRYPT_OK if successful (even if invalid)

Definition at line 31 of file pkcs_1_v1_5_decode.c.

References CRYPT_INVALID_PACKET, and CRYPT_PK_INVALID_SIZE.

Referenced by rsa_decrypt_key_ex(), and rsa_verify_hash_ex().

```
{
  unsigned long modulus_len, ps_len, i;
  int result;

```

```
  /* default to invalid packet */
  *is_valid = 0;

  modulus_len = (modulus_bitlen >> 3) + (modulus_bitlen & 7 ? 1 : 0);

  /* test message size */

  if ((msglen > modulus_len) || (modulus_len < 11)) {
    return CRYPT_PK_INVALID_SIZE;
  }

  /* separate encoded message */

  if ((msg[0] != 0x00) || (msg[1] != (unsigned char)block_type)) {
    result = CRYPT_INVALID_PACKET;
    goto bail;
  }

  if (block_type == LTC_PKCS_1_EME) {
    for (i = 2; i < modulus_len; i++) {
      /* separator */
      if (msg[i] == 0x00) { break; }
    }
    ps_len = i++ - 2;

    if ((i >= modulus_len) || (ps_len < 8)) {
      /* There was no octet with hexadecimal value 0x00 to separate ps from m,
       * or the length of ps is less than 8 octets.
       */
      result = CRYPT_INVALID_PACKET;
      goto bail;
    }
  } else {
    for (i = 2; i < modulus_len - 1; i++) {
       if (msg[i] != 0xFF) { break; }
    }

    /* separator check */
    if (msg[i] != 0) {
      /* There was no octet with hexadecimal value 0x00 to separate ps from m. */
      result = CRYPT_INVALID_PACKET;
      goto bail;
    }

    ps_len = i - 2;
  }

  if (*outlen < (msglen - (2 + ps_len + 1))) {
    *outlen = msglen - (2 + ps_len + 1);
    result = CRYPT_BUFFER_OVERFLOW;
    goto bail;
  }

  *outlen = (msglen - (2 + ps_len + 1));
  XMEMCPY(out, &msg[2 + ps_len + 1], *outlen);

  /* valid packet */
  *is_valid = 1;
  result    = CRYPT_OK;
bail:
  return result;
} /* pkcs_1_v1_5_decode */
```

5.290 pk/pkcs1/pkcs_1_v1_5_encode.c File Reference

5.290.1 Detailed Description

PKCS #1 v1.5 Padding (Andreas Lange)

Definition in file pkcs_1_v1_5_encode.c.

`#include "tomcrypt.h"`

Include dependency graph for pkcs_1_v1_5_encode.c:

Functions

- int pkcs_1_v1_5_encode (const unsigned char *msg, unsigned long msglen, int block_type, unsigned long modulus_bitlen, prng_state *prng, int prng_idx, unsigned char *out, unsigned long *outlen)

  PKCS #1 v1.5 encode.

5.290.2 Function Documentation

5.290.2.1 int pkcs_1_v1_5_encode (const unsigned char * *msg*, unsigned long *msglen*, int *block_type*, unsigned long *modulus_bitlen*, prng_state * *prng*, int *prng_idx*, unsigned char * *out*, unsigned long * *outlen*)

PKCS #1 v1.5 encode.

Parameters:

msg The data to encode

msglen The length of the data to encode (octets)

block_type Block type to use in padding (

See also:

ltc_pkcs_1_v1_5_blocks)

Parameters:

modulus_bitlen The bit length of the RSA modulus

prng An active PRNG state (only for LTC_PKCS_1_EME)

prng_idx The index of the PRNG desired (only for LTC_PKCS_1_EME)

out [out] The destination for the encoded data

outlen [in/out] The max size and resulting size of the encoded data

Returns:

CRYPT_OK if successful

Definition at line 33 of file pkcs_1_v1_5_encode.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_ERROR_READPRNG, CRYPT_OK, CRYPT_-PK_INVALID_PADDING, CRYPT_PK_INVALID_SIZE, prng_descriptor, and prng_is_valid().

Referenced by rsa_encrypt_key_ex(), and rsa_sign_hash_ex().

```
{
  unsigned long modulus_len, ps_len, i;
  unsigned char *ps;
  int result;

  /* valid block_type? */
  if ((block_type != LTC_PKCS_1_EMSA) &&
      (block_type != LTC_PKCS_1_EME)) {
     return CRYPT_PK_INVALID_PADDING;
  }

  if (block_type == LTC_PKCS_1_EME) {    /* encryption padding, we need a valid PRNG */
    if ((result = prng_is_valid(prng_idx)) != CRYPT_OK) {
       return result;
    }
  }

  modulus_len = (modulus_bitlen >> 3) + (modulus_bitlen & 7 ? 1 : 0);

  /* test message size */
  if ((msglen + 11) > modulus_len) {
    return CRYPT_PK_INVALID_SIZE;
  }

  if (*outlen < modulus_len) {
    *outlen = modulus_len;
    result = CRYPT_BUFFER_OVERFLOW;
    goto bail;
  }

  /* generate an octets string PS */
  ps = &out[2];
  ps_len = modulus_len - msglen - 3;

  if (block_type == LTC_PKCS_1_EME) {
    /* now choose a random ps */
    if (prng_descriptor[prng_idx].read(ps, ps_len, prng) != ps_len) {
      result = CRYPT_ERROR_READPRNG;
      goto bail;
    }

    /* transform zero bytes (if any) to non-zero random bytes */
    for (i = 0; i < ps_len; i++) {
      while (ps[i] == 0) {
        if (prng_descriptor[prng_idx].read(&ps[i], 1, prng) != 1) {
          result = CRYPT_ERROR_READPRNG;
          goto bail;
        }
      }
    }
  } else {
    XMEMSET(ps, 0xFF, ps_len);
  }

  /* create string of length modulus_len */
  out[0]          = 0x00;
  out[1]          = (unsigned char)block_type;  /* block_type 1 or 2 */
  out[2 + ps_len] = 0x00;
  XMEMCPY(&out[2 + ps_len + 1], msg, msglen);
  *outlen = modulus_len;

  result  = CRYPT_OK;
bail:
  return result;
} /* pkcs_1_v1_5_encode */
```

Here is the call graph for this function:

5.291 pk/rsa/rsa_decrypt_key.c File Reference

5.291.1 Detailed Description

RSA PKCS #1 Decryption, Tom St Denis and Andreas Lange.

Definition in file rsa_decrypt_key.c.

```
#include "tomcrypt.h"
```

Include dependency graph for rsa_decrypt_key.c:

Functions

- int rsa_decrypt_key_ex (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, const unsigned char ∗lparam, unsigned long lparamlen, int hash_idx, int padding, int ∗stat, rsa_key ∗key)

  PKCS #1 decrypt then v1.5 or OAEP depad.

5.291.2 Function Documentation

5.291.2.1 int rsa_decrypt_key_ex (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, const unsigned char ∗ *lparam*, unsigned long *lparamlen*, int *hash_idx*, int *padding*, int ∗ *stat*, rsa_key ∗ *key*)

PKCS #1 decrypt then v1.5 or OAEP depad.

Parameters:

in The ciphertext

inlen The length of the ciphertext (octets)

out [out] The plaintext

outlen [in/out] The max size and resulting size of the plaintext (octets)

lparam The system "lparam" value

lparamlen The length of the lparam value (octets)

hash_idx The index of the hash desired

padding Type of padding (LTC_PKCS_1_OAEP or LTC_PKCS_1_V1_5)

stat [out] Result of the decryption, 1==valid, 0==invalid

key The corresponding private RSA key

Returns:

CRYPT_OK if succcessul (even if invalid)

Definition at line 34 of file rsa_decrypt_key.c.

References CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, CRYPT_PK_INVALID_-PADDING, hash_is_valid(), LTC_ARGCHK, ltc_mp, PK_PRIVATE, pkcs_1_oaep_decode(), pkcs_1_-v1_5_decode(), ltc_math_descriptor::rsa_me, XFREE, and XMALLOC.

```
{
  unsigned long modulus_bitlen, modulus_bytelen, x;
  int           err;
  unsigned char *tmp;

  LTC_ARGCHK(out    != NULL);
  LTC_ARGCHK(outlen != NULL);
  LTC_ARGCHK(key    != NULL);
  LTC_ARGCHK(stat   != NULL);

  /* default to invalid */
  *stat = 0;

  /* valid padding? */

  if ((padding != LTC_PKCS_1_V1_5) &&
      (padding != LTC_PKCS_1_OAEP)) {
    return CRYPT_PK_INVALID_PADDING;
  }

  if (padding == LTC_PKCS_1_OAEP) {
    /* valid hash ? */
    if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
       return err;
    }
  }

  /* get modulus len in bits */
  modulus_bitlen = mp_count_bits( (key->N));

  /* outlen must be at least the size of the modulus */
  modulus_bytelen = mp_unsigned_bin_size( (key->N));
  if (modulus_bytelen != inlen) {
     return CRYPT_INVALID_PACKET;
  }

  /* allocate ram */
  tmp = XMALLOC(inlen);
  if (tmp == NULL) {
     return CRYPT_MEM;
  }

  /* rsa decode the packet */
  x = inlen;
  if ((err = ltc_mp.rsa_me(in, inlen, tmp, &x, PK_PRIVATE, key)) != CRYPT_OK) {
     XFREE(tmp);
     return err;
  }

  if (padding == LTC_PKCS_1_OAEP) {
    /* now OAEP decode the packet */
    err = pkcs_1_oaep_decode(tmp, x, lparam, lparamlen, modulus_bitlen, hash_idx,
                             out, outlen, stat);
  } else {
    /* now PKCS #1 v1.5 depad the packet */
    err = pkcs_1_v1_5_decode(tmp, x, LTC_PKCS_1_EME, modulus_bitlen, out, outlen, stat);
  }

  XFREE(tmp);
  return err;
}
```

Here is the call graph for this function:

5.292 pk/rsa/rsa_encrypt_key.c File Reference

5.292.1 Detailed Description

RSA PKCS #1 encryption, Tom St Denis and Andreas Lange.

Definition in file rsa_encrypt_key.c.

`#include "tomcrypt.h"`

Include dependency graph for rsa_encrypt_key.c:

Functions

- int rsa_encrypt_key_ex (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, const unsigned char ∗lparam, unsigned long lparamlen, prng_state ∗prng, int prng_idx, int hash_idx, int padding, rsa_key ∗key)

  (PKCS #1 v2.0) OAEP pad then encrypt

5.292.2 Function Documentation

5.292.2.1 int rsa_encrypt_key_ex (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, const unsigned char ∗ *lparam*, unsigned long *lparamlen*, prng_state ∗ *prng*, int *prng_idx*, int *hash_idx*, int *padding*, rsa_key ∗ *key*)

(PKCS #1 v2.0) OAEP pad then encrypt

Parameters:

in The plaintext

inlen The length of the plaintext (octets)

out [out] The ciphertext

outlen [in/out] The max size and resulting size of the ciphertext

lparam The system "lparam" for the encryption

lparamlen The length of lparam (octets)

prng An active PRNG

prng_idx The index of the desired prng

hash_idx The index of the desired hash

padding Type of padding (LTC_PKCS_1_OAEP or LTC_PKCS_1_V1_5)

key The RSA key to encrypt to

Returns:

CRYPT_OK if successful

Definition at line 35 of file rsa_encrypt_key.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_OK, CRYPT_PK_INVALID_PADDING, hash_is_valid(), LTC_ARGCHK, ltc_mp, PK_PUBLIC, pkcs_1_oaep_encode(), pkcs_1_v1_5_encode(), prng_is_valid(), and ltc_math_descriptor::rsa_me.

```
{
  unsigned long modulus_bitlen, modulus_bytelen, x;
  int           err;

  LTC_ARGCHK(in     != NULL);
  LTC_ARGCHK(out    != NULL);
  LTC_ARGCHK(outlen != NULL);
  LTC_ARGCHK(key    != NULL);

  /* valid padding? */
  if ((padding != LTC_PKCS_1_V1_5) &&
      (padding != LTC_PKCS_1_OAEP)) {
    return CRYPT_PK_INVALID_PADDING;
  }

  /* valid prng? */
  if ((err = prng_is_valid(prng_idx)) != CRYPT_OK) {
     return err;
  }

  if (padding == LTC_PKCS_1_OAEP) {
    /* valid hash? */
    if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
       return err;
    }
  }

  /* get modulus len in bits */
  modulus_bitlen = mp_count_bits( (key->N));

  /* outlen must be at least the size of the modulus */
  modulus_bytelen = mp_unsigned_bin_size( (key->N));
  if (modulus_bytelen > *outlen) {
     *outlen = modulus_bytelen;
     return CRYPT_BUFFER_OVERFLOW;
  }

  if (padding == LTC_PKCS_1_OAEP) {
    /* OAEP pad the key */
    x = *outlen;
    if ((err = pkcs_1_oaep_encode(in, inlen, lparam,
                                  lparamlen, modulus_bitlen, prng, prng_idx, hash_idx,
                                  out, &x)) != CRYPT_OK) {
       return err;
    }
  } else {
    /* PKCS #1 v1.5 pad the key */
    x = *outlen;
    if ((err = pkcs_1_v1_5_encode(in, inlen, LTC_PKCS_1_EME,
                                  modulus_bitlen, prng, prng_idx,
                                  out, &x)) != CRYPT_OK) {
      return err;
    }
  }

  /* rsa exptmod the OAEP or PKCS #1 v1.5 pad */
  return ltc_mp.rsa_me(out, x, out, outlen, PK_PUBLIC, key);
}
```

Here is the call graph for this function:

5.293 pk/rsa/rsa_export.c File Reference

5.293.1 Detailed Description

Export RSA PKCS keys, Tom St Denis.

Definition in file rsa_export.c.

```
#include "tomcrypt.h"
```

Include dependency graph for rsa_export.c:

Functions

- int rsa_export (unsigned char ∗out, unsigned long ∗outlen, int type, rsa_key ∗key)
  This will export either an RSAPublicKey or RSAPrivateKey [defined in PKCS #1 v2.1].

5.293.2 Function Documentation

5.293.2.1 int rsa_export (unsigned char ∗ *out*, unsigned long ∗ *outlen*, int *type*, rsa_key ∗ *key*)

This will export either an RSAPublicKey or RSAPrivateKey [defined in PKCS #1 v2.1].

Parameters:

out [out] Destination of the packet

outlen [in/out] The max size and resulting size of the packet

type The type of exported key (PK_PRIVATE or PK_PUBLIC)

key The RSA key to export

Returns:

CRYPT_OK if successful

Definition at line 28 of file rsa_export.c.

References CRYPT_PK_INVALID_TYPE, der_encode_sequence_multi(), LTC_ARGCHK, and PK_-PRIVATE.

```
29 {
30    unsigned long zero=0;
31    LTC_ARGCHK(out    != NULL);
32    LTC_ARGCHK(outlen != NULL);
33    LTC_ARGCHK(key    != NULL);
34
35    /* type valid? */
36    if (!(key->type == PK_PRIVATE) && (type == PK_PRIVATE)) {
37       return CRYPT_PK_INVALID_TYPE;
38    }
39
40    if (type == PK_PRIVATE) {
41       /* private key */
42       /* output is
43             Version, n, e, d, p, q, d mod (p-1), d mod (q - 1), 1/q mod p
44        */
45       return der_encode_sequence_multi(out, outlen,
```

```
                          LTC_ASN1_SHORT_INTEGER, 1UL, &zero,
                          LTC_ASN1_INTEGER, 1UL,  key->N,
                          LTC_ASN1_INTEGER, 1UL,  key->e,
                          LTC_ASN1_INTEGER, 1UL,  key->d,
                          LTC_ASN1_INTEGER, 1UL,  key->p,
                          LTC_ASN1_INTEGER, 1UL,  key->q,
                          LTC_ASN1_INTEGER, 1UL,  key->dP,
                          LTC_ASN1_INTEGER, 1UL,  key->dQ,
                          LTC_ASN1_INTEGER, 1UL,  key->qP,
                          LTC_ASN1_EOL,     0UL, NULL);
   } else {
      /* public key */
      return der_encode_sequence_multi(out, outlen,
                                 LTC_ASN1_INTEGER, 1UL,  key->N,
                                 LTC_ASN1_INTEGER, 1UL,  key->e,
                                 LTC_ASN1_EOL,     0UL, NULL);
   }
}
```

Here is the call graph for this function:

5.294 pk/rsa/rsa_exptmod.c File Reference

5.294.1 Detailed Description

RSA PKCS exptmod, Tom St Denis.

Definition in file rsa_exptmod.c.

`#include "tomcrypt.h"`

Include dependency graph for rsa_exptmod.c:

Functions

- int rsa_exptmod (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, int which, rsa_key ∗key)

  Compute an RSA modular exponentiation.

5.294.2 Function Documentation

5.294.2.1 int rsa_exptmod (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, int *which*, rsa_key ∗ *key*)

Compute an RSA modular exponentiation.

Parameters:

in The input data to send into RSA

inlen The length of the input (octets)

out [out] The destination

outlen [in/out] The max size and resulting size of the output

which Which exponent to use, e.g. PK_PRIVATE or PK_PUBLIC

key The RSA key to use

Returns:

CRYPT_OK if successful

Definition at line 30 of file rsa_exptmod.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_ERROR, CRYPT_OK, CRYPT_PK_INVALID_-SIZE, CRYPT_PK_INVALID_TYPE, CRYPT_PK_NOT_PRIVATE, LTC_ARGCHK, LTC_MP_LT, PK_PRIVATE, PK_PUBLIC, and zeromem().

```
{
   void          *tmp, *tmpa, *tmpb;
   unsigned long x;
   int            err;

   LTC_ARGCHK(in      != NULL);
   LTC_ARGCHK(out     != NULL);
   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(key     != NULL);

```

```
   /* is the key of the right type for the operation? */
   if (which == PK_PRIVATE && (key->type != PK_PRIVATE)) {
      return CRYPT_PK_NOT_PRIVATE;
   }

   /* must be a private or public operation */
   if (which != PK_PRIVATE && which != PK_PUBLIC) {
      return CRYPT_PK_INVALID_TYPE;
   }

   /* init and copy into tmp */
   if ((err = mp_init_multi(&tmp, &tmpa, &tmpb, NULL)) != CRYPT_OK)
   if ((err = mp_read_unsigned_bin(tmp, (unsigned char *)in, (int)inlen)) != CRYPT_OK)

   /* sanity check on the input */
   if (mp_cmp(key->N, tmp) == LTC_MP_LT) {
      err = CRYPT_PK_INVALID_SIZE;
      goto done;
   }

   /* are we using the private exponent and is the key optimized? */
   if (which == PK_PRIVATE) {
      /* tmpa = tmp^dP mod p */
      if ((err = mp_exptmod(tmp, key->dP, key->p, tmpa)) != CRYPT_OK)                               { g

      /* tmpb = tmp^dQ mod q */
      if ((err = mp_exptmod(tmp, key->dQ, key->q, tmpb)) != CRYPT_OK)                               { g

      /* tmp = (tmpa - tmpb) * qInv (mod p) */
      if ((err = mp_sub(tmpa, tmpb, tmp)) != CRYPT_OK)                                              { g
      if ((err = mp_mulmod(tmp, key->qP, key->p, tmp)) != CRYPT_OK)                                { go

      /* tmp = tmpb + q * tmp */
      if ((err = mp_mul(tmp, key->q, tmp)) != CRYPT_OK)                                             { g
      if ((err = mp_add(tmp, tmpb, tmp)) != CRYPT_OK)                                               { g
   } else {
      /* exptmod it */
      if ((err = mp_exptmod(tmp, key->e, key->N, tmp)) != CRYPT_OK)                                { go
   }

   /* read it back */
   x = (unsigned long)mp_unsigned_bin_size(key->N);
   if (x > *outlen) {
      *outlen = x;
      err = CRYPT_BUFFER_OVERFLOW;
      goto done;
   }

   /* this should never happen ... */
   if (mp_unsigned_bin_size(tmp) > mp_unsigned_bin_size(key->N)) {
      err = CRYPT_ERROR;
      goto done;
   }
   *outlen = x;

   /* convert it */
   zeromem(out, x);
   if ((err = mp_to_unsigned_bin(tmp, out+(x-mp_unsigned_bin_size(tmp)))) != CRYPT_OK)          {

   /* clean up and return */
   err = CRYPT_OK;
   goto done;
error:
done:
   mp_clear_multi(tmp, tmpa, tmpb, NULL);
   return err;
}
```

Here is the call graph for this function:

5.295 pk/rsa/rsa_free.c File Reference

5.295.1 Detailed Description

Free an RSA key, Tom St Denis.

Definition in file rsa_free.c.

`#include "tomcrypt.h"`

Include dependency graph for rsa_free.c:

Functions

- void rsa_free (rsa_key ∗key)

  Free an RSA key from memory.

5.295.2 Function Documentation

5.295.2.1 void rsa_free (rsa_key ∗ *key*)

Free an RSA key from memory.

Parameters:

key The RSA key to free

Definition at line 24 of file rsa_free.c.

References LTC_ARGCHKVD.

```
{
   LTC_ARGCHKVD(key != NULL);
   mp_clear_multi( key->e,  key->d,  key->N,  key->dQ,  key->dP,
                   key->qP,  key->p,  key->q, NULL);
}
```

5.296 pk/rsa/rsa_import.c File Reference

5.296.1 Detailed Description

Import a PKCS RSA key, Tom St Denis.

Definition in file rsa_import.c.

`#include "tomcrypt.h"`

Include dependency graph for rsa_import.c:

Functions

- int rsa_import (const unsigned char ∗in, unsigned long inlen, rsa_key ∗key)

  Import an RSAPublicKey or RSAPrivateKey [two-prime only, only support >= 1024-bit keys, defined in PKCS #1 v2.1].

5.296.2 Function Documentation

5.296.2.1 int rsa_import (const unsigned char ∗ *in*, unsigned long *inlen*, rsa_key ∗ *key*)

Import an RSAPublicKey or RSAPrivateKey [two-prime only, only support >= 1024-bit keys, defined in PKCS #1 v2.1].

Parameters:

in The packet to import from

inlen It's length (octets)

key [out] Destination for newly imported key

Returns:

CRYPT_OK if successful, upon error allocated memory is freed

Definition at line 27 of file rsa_import.c.

References CRYPT_MEM, CRYPT_OK, LTC_ARGCHK, ltc_mp, ltc_math_descriptor::name, and XCALLOC.

```
{
   int           err;
   void         *zero;
   unsigned char *tmpbuf;
   unsigned long  t, x, y, z, tmpoid[16];
   ltc_asn1_list ssl_pubkey_hashoid[2];
   ltc_asn1_list ssl_pubkey[2];

   LTC_ARGCHK(in  != NULL);
   LTC_ARGCHK(key != NULL);
   LTC_ARGCHK(ltc_mp.name != NULL);

   /* init key */
   if ((err = mp_init_multi(&key->e, &key->d, &key->N, &key->dQ,
                            &key->dP, &key->qP, &key->p, &key->q, NULL)) != CRYPT_OK) {
      return err;
   }
```

```

   /* see if the OpenSSL DER format RSA public key will work */
   tmpbuf = XCALLOC(1, MAX_RSA_SIZE*8);
   if (tmpbuf == NULL) {
       err = CRYPT_MEM;
       goto LBL_ERR;
   }

   /* this includes the internal hash ID and optional params (NULL in this case) */
   LTC_SET_ASN1(ssl_pubkey_hashoid, 0, LTC_ASN1_OBJECT_IDENTIFIER, tmpoid,                sizeof(tmpoic
   LTC_SET_ASN1(ssl_pubkey_hashoid, 1, LTC_ASN1_NULL,              NULL,                  0);

   /* the actual format of the SSL DER key is odd, it stores a RSAPublicKey in a **BIT** string ... so
      then proceed to convert bit to octet
    */
   LTC_SET_ASN1(ssl_pubkey, 0,         LTC_ASN1_SEQUENCE,          &ssl_pubkey_hashoid,   2);
   LTC_SET_ASN1(ssl_pubkey, 1,         LTC_ASN1_BIT_STRING,        tmpbuf,                MAX_RSA_SIZE*

   if (der_decode_sequence(in, inlen,
                           ssl_pubkey, 2UL) == CRYPT_OK) {

      /* ok now we have to reassemble the BIT STRING to an OCTET STRING.  Thanks OpenSSL... */
      for (t = y = z = x = 0; x < ssl_pubkey[1].size; x++) {
          y = (y << 1) | tmpbuf[x];
          if (++z == 8) {
             tmpbuf[t++] = y;
             y           = 0;
             z           = 0;
          }
      }

      /* now it should be SEQUENCE { INTEGER, INTEGER } */
      if ((err = der_decode_sequence_multi(tmpbuf, t,
                                           LTC_ASN1_INTEGER, 1UL, key->N,
                                           LTC_ASN1_INTEGER, 1UL, key->e,
                                           LTC_ASN1_EOL,     0UL, NULL)) != CRYPT_OK) {
         XFREE(tmpbuf);
         goto LBL_ERR;
      }
      XFREE(tmpbuf);
      key->type = PK_PUBLIC;
      return CRYPT_OK;
   }
   XFREE(tmpbuf);

   /* not SSL public key, try to match against PKCS #1 standards */
   if ((err = der_decode_sequence_multi(in, inlen,
                                  LTC_ASN1_INTEGER, 1UL, key->N,
                                  LTC_ASN1_EOL,     0UL, NULL)) != CRYPT_OK) {
      goto LBL_ERR;
   }

   if (mp_cmp_d(key->N, 0) == LTC_MP_EQ) {
      if ((err = mp_init(&zero)) != CRYPT_OK) {
         goto LBL_ERR;
       }
       /* it's a private key */
       if ((err = der_decode_sequence_multi(in, inlen,
                           LTC_ASN1_INTEGER, 1UL, zero,
                           LTC_ASN1_INTEGER, 1UL, key->N,
                           LTC_ASN1_INTEGER, 1UL, key->e,
                           LTC_ASN1_INTEGER, 1UL, key->d,
                           LTC_ASN1_INTEGER, 1UL, key->p,
                           LTC_ASN1_INTEGER, 1UL, key->q,
                           LTC_ASN1_INTEGER, 1UL, key->dP,
                           LTC_ASN1_INTEGER, 1UL, key->dQ,
                           LTC_ASN1_INTEGER, 1UL, key->qP,
```

```
                          LTC_ASN1_EOL,     0UL, NULL)) != CRYPT_OK) {
         mp_clear(zero);
         goto LBL_ERR;
      }
      mp_clear(zero);
      key->type = PK_PRIVATE;
   } else if (mp_cmp_d(key->N, 1) == LTC_MP_EQ) {
      /* we don't support multi-prime RSA */
      err = CRYPT_PK_INVALID_TYPE;
      goto LBL_ERR;
   } else {
      /* it's a public key and we lack e */
      if ((err = der_decode_sequence_multi(in, inlen,
                                     LTC_ASN1_INTEGER, 1UL, key->N,
                                     LTC_ASN1_INTEGER, 1UL, key->e,
                                     LTC_ASN1_EOL,     0UL, NULL)) != CRYPT_OK) {
         goto LBL_ERR;
      }
      key->type = PK_PUBLIC;
   }
   return CRYPT_OK;
LBL_ERR:
   mp_clear_multi(key->d,  key->e, key->N, key->dQ, key->dP,
                  key->qP, key->p, key->q, NULL);
   return err;
}
```

5.297 pk/rsa/rsa_make_key.c File Reference

5.297.1 Detailed Description

RSA key generation, Tom St Denis.

Definition in file rsa_make_key.c.

`#include "tomcrypt.h"`

Include dependency graph for rsa_make_key.c:

Functions

- int rsa_make_key (prng_state ∗prng, int wprng, int size, long e, rsa_key ∗key)

  Create an RSA key.

5.297.2 Function Documentation

5.297.2.1 int rsa_make_key (prng_state ∗ *prng*, int *wprng*, int *size*, long *e*, rsa_key ∗ *key*)

Create an RSA key.

Parameters:

prng An active PRNG state

wprng The index of the PRNG desired

size The size of the modulus (key size) desired (octets)

e The "e" value (public key). e==65537 is a good choice

key [out] Destination of a newly created private key pair

Returns:

CRYPT_OK if successful, upon error all allocated ram is freed

Definition at line 29 of file rsa_make_key.c.

References CRYPT_INVALID_ARG, CRYPT_INVALID_KEYSIZE, CRYPT_OK, LTC_ARGCHK, ltc_mp, ltc_math_descriptor::name, PK_PRIVATE, prng_is_valid(), and rand_prime().

```
{
   void *p, *q, *tmp1, *tmp2, *tmp3;
   int    err;

   LTC_ARGCHK(ltc_mp.name != NULL);
   LTC_ARGCHK(key != NULL);

   if ((size < (MIN_RSA_SIZE/8)) || (size > (MAX_RSA_SIZE/8))) {
      return CRYPT_INVALID_KEYSIZE;
   }

   if ((e < 3) || ((e & 1) == 0)) {
      return CRYPT_INVALID_ARG;
   }

   if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
```

```
      return err;
   }

   if ((err = mp_init_multi(&p, &q, &tmp1, &tmp2, &tmp3, NULL)) != CRYPT_OK) {
      return err;
   }

   /* make primes p and q (optimization provided by Wayne Scott) */
   if ((err = mp_set_int(tmp3, e)) != CRYPT_OK) { goto error; }            /* tmp3 = e */

   /* make prime "p" */
   do {
       if ((err = rand_prime( p, size/2, prng, wprng)) != CRYPT_OK)  { goto done; }
       if ((err = mp_sub_d( p, 1,  tmp1)) != CRYPT_OK)               { goto error; }  /* tmp1 = p-1 */
       if ((err = mp_gcd( tmp1,  tmp3,  tmp2)) != CRYPT_OK)          { goto error; }  /* tmp2 = gcd(p-1
   } while (mp_cmp_d( tmp2, 1) != 0);                                                  /* while e divides

   /* make prime "q" */
   do {
       if ((err = rand_prime( q, size/2, prng, wprng)) != CRYPT_OK) { goto done; }
       if ((err = mp_sub_d( q, 1,  tmp1)) != CRYPT_OK)              { goto error; } /* tmp1 = q-1 */
       if ((err = mp_gcd( tmp1,  tmp3,  tmp2)) != CRYPT_OK)         { goto error; } /* tmp2 = gcd(q-1,
   } while (mp_cmp_d( tmp2, 1) != 0);                                                /* while e divides

   /* tmp1 = lcm(p-1, q-1) */
   if ((err = mp_sub_d( p, 1,  tmp2)) != CRYPT_OK)                  { goto error; } /* tmp2 = p-1 */
                                                                   /* tmp1 = q-1 (previous do/while loo
   if ((err = mp_lcm( tmp1,  tmp2,  tmp1)) != CRYPT_OK)             { goto error; } /* tmp1 = lcm(p-1,

   /* make key */
   if ((err = mp_init_multi(&key->e, &key->d, &key->N, &key->dQ, &key->dP,
                     &key->qP, &key->p, &key->q, NULL)) != CRYPT_OK) {
      goto error;
   }

   if ((err = mp_set_int( key->e, e)) != CRYPT_OK)                     { goto error2; } /* key->e =  e
   if ((err = mp_invmod( key->e,  tmp1,  key->d)) != CRYPT_OK)         { goto error2; } /* key->d = 1/e
   if ((err = mp_mul( p,  q,  key->N)) != CRYPT_OK)                    { goto error2; } /* key->N = pq

   /* optimize for CRT now */
   /* find d mod q-1 and d mod p-1 */
   if ((err = mp_sub_d( p, 1,  tmp1)) != CRYPT_OK)                     { goto error2; } /* tmp1 = q-1 *
   if ((err = mp_sub_d( q, 1,  tmp2)) != CRYPT_OK)                     { goto error2; } /* tmp2 = p-1 *
   if ((err = mp_mod( key->d,  tmp1,  key->dP)) != CRYPT_OK)           { goto error2; } /* dP = d mod p
   if ((err = mp_mod( key->d,  tmp2,  key->dQ)) != CRYPT_OK)           { goto error2; } /* dQ = d mod q
   if ((err = mp_invmod( q,  p,  key->qP)) != CRYPT_OK)                { goto error2; } /* qP = 1/q mod

   if ((err = mp_copy( p,  key->p)) != CRYPT_OK)                       { goto error2; }
   if ((err = mp_copy( q,  key->q)) != CRYPT_OK)                       { goto error2; }

   /* set key type (in this case it's CRT optimized) */
   key->type = PK_PRIVATE;

   /* return ok and free temps */
   err       = CRYPT_OK;
   goto done;
error2:
   mp_clear_multi( key->d,  key->e,  key->N,  key->dQ,  key->dP,
                  key->qP,  key->p,  key->q, NULL);
error:
done:
   mp_clear_multi( tmp3,  tmp2,  tmp1,  p,  q, NULL);
   return err;
}
```

Here is the call graph for this function:

5.298 pk/rsa/rsa_sign_hash.c File Reference

5.298.1 Detailed Description

RSA PKCS #1 v1.5 and v2 PSS sign hash, Tom St Denis and Andreas Lange.

Definition in file rsa_sign_hash.c.

`#include "tomcrypt.h"`

Include dependency graph for rsa_sign_hash.c:

Functions

- int rsa_sign_hash_ex (const unsigned char ∗in, unsigned long inlen, unsigned char ∗out, unsigned long ∗outlen, int padding, prng_state ∗prng, int prng_idx, int hash_idx, unsigned long saltlen, rsa_key ∗key)

  PKCS #1 pad then sign.

5.298.2 Function Documentation

5.298.2.1 int rsa_sign_hash_ex (const unsigned char ∗ *in*, unsigned long *inlen*, unsigned char ∗ *out*, unsigned long ∗ *outlen*, int *padding*, prng_state ∗ *prng*, int *prng_idx*, int *hash_idx*, unsigned long *saltlen*, rsa_key ∗ *key*)

PKCS #1 pad then sign.

Parameters:

in The hash to sign

inlen The length of the hash to sign (octets)

out [out] The signature

outlen [in/out] The max size and resulting size of the signature

padding Type of padding (LTC_PKCS_1_PSS or LTC_PKCS_1_V1_5)

prng An active PRNG state

prng_idx The index of the PRNG desired

hash_idx The index of the hash desired

saltlen The length of the salt desired (octets)

key The private RSA key to use

Returns:

CRYPT_OK if successful

Definition at line 34 of file rsa_sign_hash.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_INVALID_ARG, CRYPT_MEM, CRYPT_OK, CRYPT_PK_INVALID_PADDING, hash_descriptor, hash_is_valid(), LTC_ARGCHK, ltc_mp, PK_-PRIVATE, pkcs_1_pss_encode(), pkcs_1_v1_5_encode(), prng_is_valid(), ltc_math_descriptor::rsa_me, XFREE, and XMALLOC.

```
{
   unsigned long modulus_bitlen, modulus_bytelen, x, y;
   int           err;

   LTC_ARGCHK(in       != NULL);
   LTC_ARGCHK(out      != NULL);
   LTC_ARGCHK(outlen   != NULL);
   LTC_ARGCHK(key      != NULL);

   /* valid padding? */
   if ((padding != LTC_PKCS_1_V1_5) && (padding != LTC_PKCS_1_PSS)) {
     return CRYPT_PK_INVALID_PADDING;
   }

   if (padding == LTC_PKCS_1_PSS) {
     /* valid prng and hash ? */
     if ((err = prng_is_valid(prng_idx)) != CRYPT_OK) {
        return err;
     }
     if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
        return err;
     }
   }

   /* get modulus len in bits */
   modulus_bitlen = mp_count_bits((key->N));

  /* outlen must be at least the size of the modulus */
  modulus_bytelen = mp_unsigned_bin_size((key->N));
  if (modulus_bytelen > *outlen) {
     *outlen = modulus_bytelen;
     return CRYPT_BUFFER_OVERFLOW;
  }

  if (padding == LTC_PKCS_1_PSS) {
    /* PSS pad the key */
    x = *outlen;
    if ((err = pkcs_1_pss_encode(in, inlen, saltlen, prng, prng_idx,
                                 hash_idx, modulus_bitlen, out, &x)) != CRYPT_OK) {
       return err;
    }
  } else {
    /* PKCS #1 v1.5 pad the hash */
    unsigned char *tmpin;
    ltc_asn1_list digestinfo[2], siginfo[2];

    /* not all hashes have OIDs... so sad */
    if (hash_descriptor[hash_idx].OIDlen == 0) {
       return CRYPT_INVALID_ARG;
    }

    /* construct the SEQUENCE
      SEQUENCE {
         SEQUENCE {hashoid OID
                   blah    NULL
         }
         hash    OCTET STRING
      }
   */
    LTC_SET_ASN1(digestinfo, 0, LTC_ASN1_OBJECT_IDENTIFIER, hash_descriptor[hash_idx].OID, hash_descrip
    LTC_SET_ASN1(digestinfo, 1, LTC_ASN1_NULL,              NULL,                          0);
    LTC_SET_ASN1(siginfo,    0, LTC_ASN1_SEQUENCE,          digestinfo,                    2);
    LTC_SET_ASN1(siginfo,    1, LTC_ASN1_OCTET_STRING,      in,                            inlen);

    /* allocate memory for the encoding */
    y = mp_unsigned_bin_size(key->N);
    tmpin = XMALLOC(y);
```

```
    if (tmpin == NULL) {
       return CRYPT_MEM;
    }

    if ((err = der_encode_sequence(siginfo, 2, tmpin, &y)) != CRYPT_OK) {
       XFREE(tmpin);
       return err;
    }

    x = *outlen;
    if ((err = pkcs_1_v1_5_encode(tmpin, y, LTC_PKCS_1_EMSA,
                                  modulus_bitlen, NULL, 0,
                                  out, &x)) != CRYPT_OK) {
      XFREE(tmpin);
      return err;
    }
    XFREE(tmpin);
  }

  /* RSA encode it */
  return ltc_mp.rsa_me(out, x, out, outlen, PK_PRIVATE, key);
}
```

Here is the call graph for this function:

5.299 pk/rsa/rsa_verify_hash.c File Reference

5.299.1 Detailed Description

RSA PKCS #1 v1.5 or v2 PSS signature verification, Tom St Denis and Andreas Lange.

Definition in file rsa_verify_hash.c.

`#include "tomcrypt.h"`

Include dependency graph for rsa_verify_hash.c:

Functions

- int rsa_verify_hash_ex (const unsigned char *sig, unsigned long siglen, const unsigned char *hash, unsigned long hashlen, int padding, int hash_idx, unsigned long saltlen, int *stat, rsa_key *key)

  PKCS #1 de-sign then v1.5 or PSS depad.

5.299.2 Function Documentation

5.299.2.1 int rsa_verify_hash_ex (const unsigned char * *sig*, unsigned long *siglen*, const unsigned char * *hash*, unsigned long *hashlen*, int *padding*, int *hash_idx*, unsigned long *saltlen*, int * *stat*, rsa_key * *key*)

PKCS #1 de-sign then v1.5 or PSS depad.

Parameters:

sig The signature data

siglen The length of the signature data (octets)

hash The hash of the message that was signed

hashlen The length of the hash of the message that was signed (octets)

padding Type of padding (LTC_PKCS_1_PSS or LTC_PKCS_1_V1_5)

hash_idx The index of the desired hash

saltlen The length of the salt used during signature

stat [out] The result of the signature comparison, 1==valid, 0==invalid

key The public RSA key corresponding to the key that performed the signature

Returns:

CRYPT_OK on success (even if the signature is invalid)

Definition at line 33 of file rsa_verify_hash.c.

References CRYPT_INVALID_ARG, CRYPT_INVALID_PACKET, CRYPT_MEM, CRYPT_OK, CRYPT_PK_INVALID_PADDING, hash_descriptor, hash_is_valid(), LTC_ARGCHK, ltc_mp, PK_-PUBLIC, pkcs_1_pss_decode(), pkcs_1_v1_5_decode(), ltc_math_descriptor::rsa_me, edge::size, XFREE, XMALLOC, XMEMCMP, and zeromem().

```
{
  unsigned long modulus_bitlen, modulus_bytelen, x;
  int           err;
```

```
unsigned char *tmpbuf;

LTC_ARGCHK(hash  != NULL);
LTC_ARGCHK(sig   != NULL);
LTC_ARGCHK(stat  != NULL);
LTC_ARGCHK(key   != NULL);

/* default to invalid */
*stat = 0;

/* valid padding? */

if ((padding != LTC_PKCS_1_V1_5) &&
    (padding != LTC_PKCS_1_PSS)) {
  return CRYPT_PK_INVALID_PADDING;
}

if (padding == LTC_PKCS_1_PSS) {
  /* valid hash ? */
  if ((err = hash_is_valid(hash_idx)) != CRYPT_OK) {
     return err;
  }
}

/* get modulus len in bits */
modulus_bitlen = mp_count_bits( (key->N));

/* outlen must be at least the size of the modulus */
modulus_bytelen = mp_unsigned_bin_size( (key->N));
if (modulus_bytelen != siglen) {
   return CRYPT_INVALID_PACKET;
}

/* allocate temp buffer for decoded sig */
tmpbuf = XMALLOC(siglen);
if (tmpbuf == NULL) {
   return CRYPT_MEM;
}

/* RSA decode it  */
x = siglen;
if ((err = ltc_mp.rsa_me(sig, siglen, tmpbuf, &x, PK_PUBLIC, key)) != CRYPT_OK) {
   XFREE(tmpbuf);
   return err;
}

/* make sure the output is the right size */
if (x != siglen) {
   return CRYPT_INVALID_PACKET;
}

if (padding == LTC_PKCS_1_PSS) {
  /* PSS decode and verify it */
  err = pkcs_1_pss_decode(hash, hashlen, tmpbuf, x, saltlen, hash_idx, modulus_bitlen, stat);
} else {
  /* PKCS #1 v1.5 decode it */
  unsigned char *out;
  unsigned long outlen, loid[16];
  int           decoded;
  ltc_asn1_list digestinfo[2], siginfo[2];

  /* not all hashes have OIDs... so sad */
  if (hash_descriptor[hash_idx].OIDlen == 0) {
     err = CRYPT_INVALID_ARG;
     goto bail_2;
  }

```

```
    /* allocate temp buffer for decoded hash */
    outlen = ((modulus_bitlen >> 3) + (modulus_bitlen & 7 ? 1 : 0)) - 3;
    out    = XMALLOC(outlen);
    if (out == NULL) {
      err = CRYPT_MEM;
      goto bail_2;
    }

    if ((err = pkcs_1_v1_5_decode(tmpbuf, x, LTC_PKCS_1_EMSA, modulus_bitlen, out, &outlen, &decoded))
      XFREE(out);
      goto bail_2;
    }

    /* now we must decode out[0...outlen-1] using ASN.1, test the OID and then test the hash */
    /* construct the SEQUENCE
      SEQUENCE {
         SEQUENCE {hashoid OID
                   blah    NULL
         }
         hash    OCTET STRING
      }
   */
    LTC_SET_ASN1(digestinfo, 0, LTC_ASN1_OBJECT_IDENTIFIER, loid, sizeof(loid)/sizeof(loid[0]));
    LTC_SET_ASN1(digestinfo, 1, LTC_ASN1_NULL,              NULL,                          0);
    LTC_SET_ASN1(siginfo,    0, LTC_ASN1_SEQUENCE,          digestinfo,                    2);
    LTC_SET_ASN1(siginfo,    1, LTC_ASN1_OCTET_STRING,      tmpbuf,                        siglen);

    if ((err = der_decode_sequence(out, outlen, siginfo, 2)) != CRYPT_OK) {
       XFREE(out);
       goto bail_2;
    }

    /* test OID */
    if ((digestinfo[0].size == hash_descriptor[hash_idx].OIDlen) &&
        (XMEMCMP(digestinfo[0].data, hash_descriptor[hash_idx].OID, sizeof(unsigned long) * hash_descr
        (siginfo[1].size == hashlen) &&
        (XMEMCMP(siginfo[1].data, hash, hashlen) == 0)) {
       *stat = 1;
    }

#ifdef LTC_CLEAN_STACK
    zeromem(out, outlen);
#endif
    XFREE(out);
  }

bail_2:
#ifdef LTC_CLEAN_STACK
  zeromem(tmpbuf, siglen);
#endif
  XFREE(tmpbuf);
  return err;
}
```

Here is the call graph for this function:

5.300 prngs/fortuna.c File Reference

5.300.1 Detailed Description

Fortuna PRNG, Tom St Denis.

Definition in file fortuna.c.

```
#include "tomcrypt.h"
```

Include dependency graph for fortuna.c:

Functions

- static void fortuna_update_iv (prng_state *prng)
- static int fortuna_reseed (prng_state *prng)
- int fortuna_start (prng_state *prng)

  Start the PRNG.
- int fortuna_add_entropy (const unsigned char *in, unsigned long inlen, prng_state *prng)

  Add entropy to the PRNG state.
- int fortuna_ready (prng_state *prng)

  Make the PRNG ready to read from.
- unsigned long fortuna_read (unsigned char *out, unsigned long outlen, prng_state *prng)

  Read from the PRNG.
- int fortuna_done (prng_state *prng)

  Terminate the PRNG.
- int fortuna_export (unsigned char *out, unsigned long *outlen, prng_state *prng)

  Export the PRNG state.
- int fortuna_import (const unsigned char *in, unsigned long inlen, prng_state *prng)

  Import a PRNG state.
- int fortuna_test (void)

  PRNG self-test.

Variables

- const struct ltc_prng_descriptor fortuna_desc

5.300.2 Function Documentation

5.300.2.1 int fortuna_add_entropy (const unsigned char * *in*, unsigned long *inlen*, prng_state * *prng*)

Add entropy to the PRNG state.

Parameters:

in The data to add

inlen Length of the data to add

prng PRNG state to update

Returns:

CRYPT_OK if successful

Definition at line 171 of file fortuna.c.

References CRYPT_INVALID_ARG, FORTUNA_POOLS, LTC_ARGCHK, LTC_MUTEX_LOCK, and LTC_MUTEX_UNLOCK.

Referenced by fortuna_import().

```
{
   unsigned char tmp[2];
   int           err;

   LTC_ARGCHK(in  != NULL);
   LTC_ARGCHK(prng != NULL);

   LTC_MUTEX_LOCK(&prng->fortuna.prng_lock);

   /* ensure inlen <= 32 */
   if (inlen > 32) {
      LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
      return CRYPT_INVALID_ARG;
   }

   /* add s || length(in) || in to pool[pool_idx] */
   tmp[0] = 0;
   tmp[1] = inlen;
   if ((err = sha256_process(&prng->fortuna.pool[prng->fortuna.pool_idx], tmp, 2)) != CRYPT_OK) {
      LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
      return err;
   }
   if ((err = sha256_process(&prng->fortuna.pool[prng->fortuna.pool_idx], in, inlen)) != CRYPT_OK) {
      LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
      return err;
   }
   if (prng->fortuna.pool_idx == 0) {
      prng->fortuna.pool0_len += inlen;
   }
   if (++(prng->fortuna.pool_idx) == FORTUNA_POOLS) {
      prng->fortuna.pool_idx = 0;
   }

   LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
   return CRYPT_OK;
}
```

5.300.2.2 int fortuna_done (prng_state * *prng*)

Terminate the PRNG.

Parameters:

prng The PRNG to terminate

Returns:

CRYPT_OK if successful

Definition at line 284 of file fortuna.c.

References CRYPT_OK, FORTUNA_POOLS, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_-UNLOCK, and sha256_done().

```
{
   int           err, x;
   unsigned char tmp[32];

   LTC_ARGCHK(prng != NULL);
   LTC_MUTEX_LOCK(&prng->fortuna.prng_lock);

   /* terminate all the hashes */
   for (x = 0; x < FORTUNA_POOLS; x++) {
       if ((err = sha256_done(&(prng->fortuna.pool[x]), tmp)) != CRYPT_OK) {
          LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
          return err;
       }
   }
   /* call cipher done when we invent one ;-) */

#ifdef LTC_CLEAN_STACK
   zeromem(tmp, sizeof(tmp));
#endif

   LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.300.2.3 int fortuna_export (unsigned char ∗ *out*, unsigned long ∗ *outlen*, prng_state ∗ *prng*)

Export the PRNG state.

Parameters:

out [out] Destination

outlen [in/out] Max size and resulting size of the state

prng The PRNG to export

Returns:

CRYPT_OK if successful

Definition at line 316 of file fortuna.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_MEM, CRYPT_OK, FORTUNA_POOLS, LTC_-ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, sha256_done(), XMALLOC, and XMEM-CPY.

```
{
   int         x, err;
   hash_state *md;

   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(prng   != NULL);

   LTC_MUTEX_LOCK(&prng->fortuna.prng_lock);

```

```
   /* we'll write bytes for s&g's */
   if (*outlen < 32*FORTUNA_POOLS) {
      LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
      *outlen = 32*FORTUNA_POOLS;
      return CRYPT_BUFFER_OVERFLOW;
   }

   md = XMALLOC(sizeof(hash_state));
   if (md == NULL) {
      LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
      return CRYPT_MEM;
   }

   /* to emit the state we copy each pool, terminate it then hash it again so
    * an attacker who sees the state can't determine the current state of the PRNG
    */
   for (x = 0; x < FORTUNA_POOLS; x++) {
      /* copy the PRNG */
      XMEMCPY(md, &(prng->fortuna.pool[x]), sizeof(*md));

      /* terminate it */
      if ((err = sha256_done(md, out+x*32)) != CRYPT_OK) {
         goto LBL_ERR;
      }

      /* now hash it */
      if ((err = sha256_init(md)) != CRYPT_OK) {
         goto LBL_ERR;
      }
      if ((err = sha256_process(md, out+x*32, 32)) != CRYPT_OK) {
         goto LBL_ERR;
      }
      if ((err = sha256_done(md, out+x*32)) != CRYPT_OK) {
         goto LBL_ERR;
      }
   }
   *outlen = 32*FORTUNA_POOLS;
   err = CRYPT_OK;

LBL_ERR:
#ifdef LTC_CLEAN_STACK
   zeromem(md, sizeof(*md));
#endif
   XFREE(md);
   LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
   return err;
}
```

Here is the call graph for this function:

5.300.2.4 int fortuna_import (const unsigned char ∗ *in*, unsigned long *inlen*, prng_state ∗ *prng*)

Import a PRNG state.

Parameters:

in The PRNG state

inlen Size of the state

prng The PRNG to import

Returns:

CRYPT_OK if successful

Definition at line 382 of file fortuna.c.

References CRYPT_INVALID_ARG, CRYPT_OK, fortuna_add_entropy(), FORTUNA_POOLS, fortuna_start(), and LTC_ARGCHK.

```
{
   int err, x;

   LTC_ARGCHK(in   != NULL);
   LTC_ARGCHK(prng != NULL);

   if (inlen != 32*FORTUNA_POOLS) {
      return CRYPT_INVALID_ARG;
   }

   if ((err = fortuna_start(prng)) != CRYPT_OK) {
      return err;
   }
   for (x = 0; x < FORTUNA_POOLS; x++) {
      if ((err = fortuna_add_entropy(in+x*32, 32, prng)) != CRYPT_OK) {
         return err;
      }
   }
   return err;
}
```

Here is the call graph for this function:

5.300.2.5 unsigned long fortuna_read (unsigned char * *out*, unsigned long *outlen*, prng_state * *prng*)

Read from the PRNG.

Parameters:

out Destination

outlen Length of output

prng The active PRNG to read from

Returns:

Number of octets read

Definition at line 226 of file fortuna.c.

References CRYPT_OK, fortuna_reseed(), fortuna_update_iv(), FORTUNA_WD, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, XMEMCPY, and zeromem().

```
{
   unsigned char tmp[16];
   int           err;
   unsigned long tlen;

   LTC_ARGCHK(out  != NULL);
   LTC_ARGCHK(prng != NULL);

   LTC_MUTEX_LOCK(&prng->fortuna.prng_lock);

   /* do we have to reseed? */
   if (++prng->fortuna.wd == FORTUNA_WD || prng->fortuna.pool0_len >= 64) {
      if ((err = fortuna_reseed(prng)) != CRYPT_OK) {
```

```
         LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
         return 0;
      }
   }

   /* now generate the blocks required */
   tlen = outlen;

   /* handle whole blocks without the extra XMEMCPY */
   while (outlen >= 16) {
      /* encrypt the IV and store it */
      rijndael_ecb_encrypt(prng->fortuna.IV, out, &prng->fortuna.skey);
      out += 16;
      outlen -= 16;
      fortuna_update_iv(prng);
   }

   /* left over bytes? */
   if (outlen > 0) {
      rijndael_ecb_encrypt(prng->fortuna.IV, tmp, &prng->fortuna.skey);
      XMEMCPY(out, tmp, outlen);
      fortuna_update_iv(prng);
   }

   /* generate new key */
   rijndael_ecb_encrypt(prng->fortuna.IV, prng->fortuna.K   , &prng->fortuna.skey); fortuna_update_iv(
   rijndael_ecb_encrypt(prng->fortuna.IV, prng->fortuna.K+16, &prng->fortuna.skey); fortuna_update_iv(
   if ((err = rijndael_setup(prng->fortuna.K, 32, 0, &prng->fortuna.skey)) != CRYPT_OK) {
      LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
      return 0;
   }

#ifdef LTC_CLEAN_STACK
   zeromem(tmp, sizeof(tmp));
#endif
   LTC_MUTEX_UNLOCK(&prng->fortuna.prng_lock);
   return tlen;
}
```

Here is the call graph for this function:

5.300.2.6 int fortuna_ready (prng_state * *prng*)

Make the PRNG ready to read from.

Parameters:

prng The PRNG to make active

Returns:

CRYPT_OK if successful

Definition at line 214 of file fortuna.c.

References fortuna_reseed().

```
{
   return fortuna_reseed(prng);
}
```

Here is the call graph for this function:

5.300.2.7 static int fortuna_reseed (prng_state * *prng*) [static]

Definition at line 66 of file fortuna.c.

References CRYPT_OK, FORTUNA_POOLS, MAXBLOCKSIZE, sha256_done(), and sha256_init().

Referenced by fortuna_read(), and fortuna_ready().

```
{
   unsigned char tmp[MAXBLOCKSIZE];
   hash_state    md;
   int           err, x;

   ++prng->fortuna.reset_cnt;

   /* new K == SHA256(K || s) where s == SHA256(P0) || SHA256(P1) ... */
   sha256_init(&md);
   if ((err = sha256_process(&md, prng->fortuna.K, 32)) != CRYPT_OK) {
      sha256_done(&md, tmp);
      return err;
   }

   for (x = 0; x < FORTUNA_POOLS; x++) {
       if (x == 0 || ((prng->fortuna.reset_cnt >> (x-1)) & 1) == 0) {
          /* terminate this hash */
          if ((err = sha256_done(&prng->fortuna.pool[x], tmp)) != CRYPT_OK) {
             sha256_done(&md, tmp);
             return err;
          }
          /* add it to the string */
          if ((err = sha256_process(&md, tmp, 32)) != CRYPT_OK) {
             sha256_done(&md, tmp);
             return err;
          }
          /* reset this pool */
          if ((err = sha256_init(&prng->fortuna.pool[x])) != CRYPT_OK) {
             sha256_done(&md, tmp);
             return err;
          }
       } else {
          break;
       }
   }

   /* finish key */
   if ((err = sha256_done(&md, prng->fortuna.K)) != CRYPT_OK) {
      return err;
   }
   if ((err = rijndael_setup(prng->fortuna.K, 32, 0, &prng->fortuna.skey)) != CRYPT_OK) {
      return err;
   }
   fortuna_update_iv(prng);

   /* reset pool len */
   prng->fortuna.pool0_len = 0;
   prng->fortuna.wd        = 0;


#ifdef LTC_CLEAN_STACK
   zeromem(&md, sizeof(md));
   zeromem(tmp, sizeof(tmp));
#endif

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.300.2.8 int fortuna_start (prng_state * *prng*)

Start the PRNG.

Parameters:

prng [out] The PRNG state to initialize

Returns:

CRYPT_OK if successful

Definition at line 130 of file fortuna.c.

References CRYPT_OK, FORTUNA_POOLS, LTC_ARGCHK, MAXBLOCKSIZE, sha256_done(), and sha256_init().

Referenced by fortuna_import().

```
{
   int err, x, y;
   unsigned char tmp[MAXBLOCKSIZE];

   LTC_ARGCHK(prng != NULL);

   /* initialize the pools */
   for (x = 0; x < FORTUNA_POOLS; x++) {
       if ((err = sha256_init(&prng->fortuna.pool[x])) != CRYPT_OK) {
          for (y = 0; y < x; y++) {
              sha256_done(&prng->fortuna.pool[y], tmp);
          }
          return err;
       }
   }
   prng->fortuna.pool_idx = prng->fortuna.pool0_len = prng->fortuna.reset_cnt =
   prng->fortuna.wd = 0;

   /* reset bufs */
   zeromem(prng->fortuna.K, 32);
   if ((err = rijndael_setup(prng->fortuna.K, 32, 0, &prng->fortuna.skey)) != CRYPT_OK) {
      for (x = 0; x < FORTUNA_POOLS; x++) {
          sha256_done(&prng->fortuna.pool[x], tmp);
      }
      return err;
   }
   zeromem(prng->fortuna.IV, 16);

   LTC_MUTEX_INIT(&prng->fortuna.prng_lock)

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.300.2.9 int fortuna_test (void)

PRNG self-test.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-testing has been disabled

Definition at line 408 of file fortuna.c.

References CRYPT_NOP, CRYPT_OK, and sha256_test().

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   int err;

   if ((err = sha256_test()) != CRYPT_OK) {
      return err;
   }
   return rijndael_test();
#endif
}
```

Here is the call graph for this function:

5.300.2.10 static void fortuna_update_iv (prng_state ∗ *prng*) [static]

Definition at line 53 of file fortuna.c.

Referenced by fortuna_read().

```
{
   int            x;
   unsigned char *IV;
   /* update IV */
   IV = prng->fortuna.IV;
   for (x = 0; x < 16; x++) {
      IV[x] = (IV[x] + 1) & 255;
      if (IV[x] != 0) break;
   }
}
```

5.300.3 Variable Documentation

5.300.3.1 const struct ltc_prng_descriptor fortuna_desc

Initial value:

```
 {
    "fortuna", 1024,
    &fortuna_start,
    &fortuna_add_entropy,
    &fortuna_ready,
    &fortuna_read,
    &fortuna_done,
    &fortuna_export,
    &fortuna_import,
    &fortuna_test
}
```

Definition at line 40 of file fortuna.c.

5.301 prngs/rc4.c File Reference

5.301.1 Detailed Description

RC4 PRNG, Tom St Denis.

Definition in file rc4.c.

```
#include "tomcrypt.h"
```

Include dependency graph for rc4.c:

Functions

- int rc4_start (prng_state *prng)
  Start the PRNG.
- int rc4_add_entropy (const unsigned char *in, unsigned long inlen, prng_state *prng)
  Add entropy to the PRNG state.
- int rc4_ready (prng_state *prng)
  Make the PRNG ready to read from.
- unsigned long rc4_read (unsigned char *out, unsigned long outlen, prng_state *prng)
  Read from the PRNG.
- int rc4_done (prng_state *prng)
  Terminate the PRNG.
- int rc4_export (unsigned char *out, unsigned long *outlen, prng_state *prng)
  Export the PRNG state.
- int rc4_import (const unsigned char *in, unsigned long inlen, prng_state *prng)
  Import a PRNG state.
- int rc4_test (void)
  PRNG self-test.

Variables

- const struct ltc_prng_descriptor rc4_desc

5.301.2 Function Documentation

5.301.2.1 int rc4_add_entropy (const unsigned char * *in*, unsigned long *inlen*, prng_state * *prng*)

Add entropy to the PRNG state.

Parameters:

in The data to add

inlen Length of the data to add

prng PRNG state to update

Returns:

CRYPT_OK if successful

Definition at line 55 of file rc4.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by rc4_import().

```
{
   LTC_ARGCHK(in  != NULL);
   LTC_ARGCHK(prng != NULL);

   /* trim as required */
   if (prng->rc4.x + inlen > 256) {
      if (prng->rc4.x == 256) {
         /* I can't possibly accept another byte, ok maybe a mint wafer... */
         return CRYPT_OK;
      } else {
         /* only accept part of it */
         inlen = 256 - prng->rc4.x;
      }
   }

   while (inlen--) {
      prng->rc4.buf[prng->rc4.x++] = *in++;
   }

   return CRYPT_OK;

}
```

5.301.2.2 int rc4_done (prng_state * *prng*)

Terminate the PRNG.

Parameters:

prng The PRNG to terminate

Returns:

CRYPT_OK if successful

Definition at line 158 of file rc4.c.

References CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(prng != NULL);
   return CRYPT_OK;
}
```

5.301.2.3 int rc4_export (unsigned char ∗ *out*, unsigned long ∗ *outlen*, prng_state ∗ *prng*)

Export the PRNG state.

Parameters:

out [out] Destination

outlen [in/out] Max size and resulting size of the state

prng The PRNG to export

Returns:

CRYPT_OK if successful

Definition at line 171 of file rc4.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_ERROR_READPRNG, CRYPT_OK, LTC_-ARGCHK, and rc4_read().

```
{
   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(prng   != NULL);

   if (*outlen < 32) {
      *outlen = 32;
      return CRYPT_BUFFER_OVERFLOW;
   }

   if (rc4_read(out, 32, prng) != 32) {
      return CRYPT_ERROR_READPRNG;
   }
   *outlen = 32;

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.301.2.4 int rc4_import (const unsigned char ∗ *in*, unsigned long *inlen*, prng_state ∗ *prng*)

Import a PRNG state.

Parameters:

in The PRNG state

inlen Size of the state

prng The PRNG to import

Returns:

CRYPT_OK if successful

Definition at line 197 of file rc4.c.

References CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, rc4_add_entropy(), and rc4_start().

```
{
   int err;
```

```
   LTC_ARGCHK(in   != NULL);
   LTC_ARGCHK(prng != NULL);

   if (inlen != 32) {
      return CRYPT_INVALID_ARG;
   }

   if ((err = rc4_start(prng)) != CRYPT_OK) {
      return err;
   }
   return rc4_add_entropy(in, 32, prng);
}
```

Here is the call graph for this function:

5.301.2.5 unsigned long rc4_read (unsigned char ∗ *out*, unsigned long *outlen*, prng_state ∗ *prng*)

Read from the PRNG.

Parameters:

out Destination

outlen Length of output

prng The active PRNG to read from

Returns:

Number of octets read

Definition at line 125 of file rc4.c.

References LTC_ARGCHK, and zeromem().

Referenced by rc4_export().

```
{
   unsigned char x, y, *s, tmp;
   unsigned long n;

   LTC_ARGCHK(out != NULL);
   LTC_ARGCHK(prng != NULL);

#ifdef LTC_VALGRIND
   zeromem(out, outlen);
#endif

   n = outlen;
   x = prng->rc4.x;
   y = prng->rc4.y;
   s = prng->rc4.buf;
   while (outlen--) {
      x = (x + 1) & 255;
      y = (y + s[x]) & 255;
      tmp = s[x]; s[x] = s[y]; s[y] = tmp;
      tmp = (s[x] + s[y]) & 255;
      *out++ ^= s[tmp];
   }
   prng->rc4.x = x;
   prng->rc4.y = y;
   return n;
}
```

Here is the call graph for this function:

5.301.2.6 int rc4_ready (prng_state * *prng*)

Make the PRNG ready to read from.

Parameters:

prng The PRNG to make active

Returns:

CRYPT_OK if successful

Definition at line 84 of file rc4.c.

References LTC_ARGCHK, and XMEMCPY.

```
{
    unsigned char key[256], tmp, *s;
    int keylen, x, y, j;

    LTC_ARGCHK(prng != NULL);

    /* extract the key */
    s = prng->rc4.buf;
    XMEMCPY(key, s, 256);
    keylen = prng->rc4.x;

    /* make RC4 perm and shuffle */
    for (x = 0; x < 256; x++) {
        s[x] = x;
    }

    for (j = x = y = 0; x < 256; x++) {
        y = (y + prng->rc4.buf[x] + key[j++]) & 255;
        if (j == keylen) {
           j = 0;
        }
        tmp = s[x]; s[x] = s[y]; s[y] = tmp;
    }
    prng->rc4.x = 0;
    prng->rc4.y = 0;

#ifdef LTC_CLEAN_STACK
    zeromem(key, sizeof(key));
#endif

    return CRYPT_OK;
}
```

5.301.2.7 int rc4_start (prng_state * *prng*)

Start the PRNG.

Parameters:

prng [out] The PRNG state to initialize

Returns:

CRYPT_OK if successful

Definition at line 38 of file rc4.c.

References CRYPT_OK, and LTC_ARGCHK.

Referenced by rc4_import(), and rc4_test().

```
{
   LTC_ARGCHK(prng != NULL);

   /* set keysize to zero */
   prng->rc4.x = 0;

   return CRYPT_OK;
}
```

5.301.2.8 int rc4_test (void)

PRNG self-test.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-testing has been disabled

Definition at line 217 of file rc4.c.

References CRYPT_NOP, CRYPT_OK, and rc4_start().

```
{
#if !defined(LTC_TEST) || defined(LTC_VALGRIND)
   return CRYPT_NOP;
#else
   static const struct {
      unsigned char key[8], pt[8], ct[8];
   } tests[] = {
{
   { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef },
   { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef },
   { 0x75, 0xb7, 0x87, 0x80, 0x99, 0xe0, 0xc5, 0x96 }
}
};
   prng_state prng;
   unsigned char dst[8];
   int err, x;

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       if ((err = rc4_start(&prng)) != CRYPT_OK) {
          return err;
       }
       if ((err = rc4_add_entropy(tests[x].key, 8, &prng)) != CRYPT_OK) {
          return err;
       }
       if ((err = rc4_ready(&prng)) != CRYPT_OK) {
          return err;
       }
       XMEMCPY(dst, tests[x].pt, 8);
       if (rc4_read(dst, 8, &prng) != 8) {
          return CRYPT_ERROR_READPRNG;
       }
       rc4_done(&prng);
       if (XMEMCMP(dst, tests[x].ct, 8)) {
#if 0
          int y;
          printf("\n\nRC4 failed, I got:\n");
          for (y = 0; y < 8; y++) printf("%02x ", dst[y]);
          printf("\n");
#endif
          return CRYPT_FAIL_TESTVECTOR;
       }
   }
   return CRYPT_OK;
```

```
#endif
}
```

Here is the call graph for this function:

5.301.3 Variable Documentation

5.301.3.1 const struct ltc_prng_descriptor rc4_desc

Initial value:

```
{
   "rc4", 32,
    &rc4_start,
    &rc4_add_entropy,
    &rc4_ready,
    &rc4_read,
    &rc4_done,
    &rc4_export,
    &rc4_import,
    &rc4_test
}
```

Definition at line 20 of file rc4.c.

5.302 prngs/rng_get_bytes.c File Reference

5.302.1 Detailed Description

portable way to get secure random bits to feed a PRNG (Tom St Denis)

Definition in file rng_get_bytes.c.

`#include "tomcrypt.h"`

Include dependency graph for rng_get_bytes.c:

Functions

- static unsigned long rng_nix (unsigned char ∗buf, unsigned long len, void(∗callback)(void))
- unsigned long rng_get_bytes (unsigned char ∗out, unsigned long outlen, void(∗callback)(void))

  Read the system RNG.

5.302.2 Function Documentation

5.302.2.1 unsigned long rng_get_bytes (unsigned char ∗ *out*, unsigned long *outlen*, void(∗)(void) *callback*)

Read the system RNG.

Parameters:

out Destination

outlen Length desired (octets)

callback Pointer to void function to act as "callback" when RNG is slow. This can be NULL

Returns:

Number of octets read

Definition at line 123 of file rng_get_bytes.c.

References LTC_ARGCHK, and rng_nix().

Referenced by rng_make_prng(), and sprng_read().

```
{
   unsigned long x;

   LTC_ARGCHK(out != NULL);

#if defined(DEVRANDOM)
   x = rng_nix(out, outlen, callback);   if (x != 0) { return x; }
#endif
#ifdef WIN32
   x = rng_win32(out, outlen, callback); if (x != 0) { return x; }
#endif
#ifdef ANSI_RNG
   x = rng_ansic(out, outlen, callback); if (x != 0) { return x; }
#endif
   return 0;
}
```

Here is the call graph for this function:

5.302.2.2 static unsigned long rng_nix (unsigned char * *buf*, unsigned long *len*, void(*)(void) *callback*) [static]

Definition at line 20 of file rng_get_bytes.c.

Referenced by rng_get_bytes().

```
{
#ifdef LTC_NO_FILE
    return 0;
#else
    FILE *f;
    unsigned long x;
#ifdef TRY_URANDOM_FIRST
    f = fopen("/dev/urandom", "rb");
    if (f == NULL)
#endif /* TRY_URANDOM_FIRST */
       f = fopen("/dev/random", "rb");

    if (f == NULL) {
       return 0;
    }

    /* disable buffering */
    if (setvbuf(f, NULL, _IONBF, 0) != 0) {
       fclose(f);
       return 0;
    }

    x = (unsigned long)fread(buf, 1, (size_t)len, f);
    fclose(f);
    return x;
#endif /* LTC_NO_FILE */
}
```

5.303 prngs/rng_make_prng.c File Reference

5.303.1 Detailed Description

portable way to get secure random bits to feed a PRNG (Tom St Denis)

Definition in file rng_make_prng.c.

`#include "tomcrypt.h"`

Include dependency graph for rng_make_prng.c:

Functions

- int rng_make_prng (int bits, int wprng, prng_state ∗prng, void(∗callback)(void))

  Create a PRNG from a RNG.

5.303.2 Function Documentation

5.303.2.1 int rng_make_prng (int *bits*, int *wprng*, prng_state ∗ *prng*, void(∗)(void) *callback*)

Create a PRNG from a RNG.

Parameters:

bits Number of bits of entropy desired (64 ... 1024)

wprng Index of which PRNG to setup

prng [out] PRNG state to initialize

callback A pointer to a void function for when the RNG is slow, this can be NULL

Returns:

CRYPT_OK if successful

Definition at line 26 of file rng_make_prng.c.

References CRYPT_ERROR_READPRNG, CRYPT_INVALID_PRNGSIZE, CRYPT_OK, LTC_ARGCHK, prng_descriptor, prng_is_valid(), rng_get_bytes(), and zeromem().

```
28 {
29    unsigned char buf[256];
30    int err;
31
32    LTC_ARGCHK(prng != NULL);
33
34    /* check parameter */
35    if ((err = prng_is_valid(wprng)) != CRYPT_OK) {
36       return err;
37    }
38
39    if (bits < 64 || bits > 1024) {
40       return CRYPT_INVALID_PRNGSIZE;
41    }
42
43    if ((err = prng_descriptor[wprng].start(prng)) != CRYPT_OK) {
44       return err;
45    }
```

```

   bits = ((bits/8)+((bits&7)!=0?1:0)) * 2;
   if (rng_get_bytes(buf, (unsigned long)bits, callback) != (unsigned long)bits) {
      return CRYPT_ERROR_READPRNG;
   }

   if ((err = prng_descriptor[wprng].add_entropy(buf, (unsigned long)bits, prng)) != CRYPT_OK) {
      return err;
   }

   if ((err = prng_descriptor[wprng].ready(prng)) != CRYPT_OK) {
      return err;
   }

   #ifdef LTC_CLEAN_STACK
      zeromem(buf, sizeof(buf));
   #endif
   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.304 prngs/sober128.c File Reference

5.304.1 Detailed Description

Implementation of SOBER-128 by Tom St Denis.

Based on s128fast.c reference code supplied by Greg Rose of QUALCOMM.

Definition in file sober128.c.

```
#include "tomcrypt.h"
#include "sober128tab.c"
```

Include dependency graph for sober128.c:

Defines

- #define N 17
- #define FOLD N
- #define INITKONST 0x6996c53a
- #define KEYP 15
- #define FOLDP 4
- #define B(x, i) ((unsigned char)(((x) >> (8∗i)) & 0xFF))
- #define WORD2BYTE(w, b) STORE32L(b, w)
- #define OFF(zero, i) (((zero)+(i)) % N)
- #define STEP(R, z) R[OFF(z,0)] = R[OFF(z,15)] ^ R[OFF(z,4)] ^ (R[OFF(z,0)] << 8) ^ Multab[(R[OFF(z,0)] >> 24) & 0xFF];
- #define NLFUNC(c, z)
- #define ADDKEY(k) c → R[KEYP] += (k);
- #define XORNL(nl) c → R[FOLDP] ^= (nl);
- #define DROUND(z) STEP(c → R,z); NLFUNC(c,(z+1)); c → R[OFF((z+1),FOLDP)] ^= t;
- #define SROUND(z) STEP(c → R,z); NLFUNC(c,(z+1)); XORWORD(t, out+(z∗4));

Functions

- static ulong32 BYTE2WORD (unsigned char ∗b)
- static void XORWORD (ulong32 w, unsigned char ∗b)
- static void cycle (ulong32 ∗R)
- static ulong32 nltap (struct sober128_prng ∗c)
- int sober128_start (prng_state ∗prng)

    Start the PRNG.

- static void s128_savestate (struct sober128_prng ∗c)
- static void s128_reloadstate (struct sober128_prng ∗c)
- static void s128_genkonst (struct sober128_prng ∗c)
- static void s128_diffuse (struct sober128_prng ∗c)
- int sober128_add_entropy (const unsigned char ∗in, unsigned long inlen, prng_state ∗prng)

    Add entropy to the PRNG state.

- int sober128_ready (prng_state ∗prng)

    Make the PRNG ready to read from.

- unsigned long sober128_read (unsigned char ∗out, unsigned long outlen, prng_state ∗prng)

  Read from the PRNG.

- int sober128_done (prng_state ∗prng)

  Terminate the PRNG.

- int sober128_export (unsigned char ∗out, unsigned long ∗outlen, prng_state ∗prng)

  Export the PRNG state.

- int sober128_import (const unsigned char ∗in, unsigned long inlen, prng_state ∗prng)

  Import a PRNG state.

- int sober128_test (void)

  PRNG self-test.

Variables

- const struct ltc_prng_descriptor sober128_desc

5.304.2 Define Documentation

5.304.2.1 #define ADDKEY(k) c → R[KEYP] += (k);

Definition at line 169 of file sober128.c.

Referenced by sober128_add_entropy().

5.304.2.2 #define B(x, i) ((unsigned char)(((x) >> (8∗i)) & 0xFF))

Definition at line 43 of file sober128.c.

Referenced by blowfish_setup(), gcm_init(), gf_mult(), lrw_start(), qsort_helper(), rc5_ecb_decrypt(), rc5_ecb_encrypt(), rc5_setup(), rc6_setup(), and twofish_setup().

5.304.2.3 #define DROUND(z) STEP(c → R,z); NLFUNC(c,(z+1)); c → R[OFF((z+1),FOLDP)] ^= t;

Definition at line 176 of file sober128.c.

Referenced by s128_diffuse().

5.304.2.4 #define FOLD N

Definition at line 38 of file sober128.c.

5.304.2.5 #define FOLDP 4

Definition at line 41 of file sober128.c.

5.304.2.6 #define INITKONST 0x6996c53a

Definition at line 39 of file sober128.c.

5.304.2.7 #define KEYP 15

Definition at line 40 of file sober128.c.

5.304.2.8 #define N 17

Definition at line 37 of file sober128.c.

Referenced by anubis_setup(), cycle(), s128_reloadstate(), s128_savestate(), sober128_read(), and sober128_start().

5.304.2.9 #define NLFUNC(c, z)

Value:

```
{ \
    t = c->R[OFF(z,0)] + c->R[OFF(z,16)]; \
    t ^= Sbox[(t >> 24) & 0xFF]; \
    t = RORc(t, 8); \
    t = ((t + c->R[OFF(z,1)]) ^ c->konst) + c->R[OFF(z,6)]; \
    t ^= Sbox[(t >> 24) & 0xFF]; \
    t = t + c->R[OFF(z,13)]; \
}
```

Definition at line 87 of file sober128.c.

Referenced by nltap().

5.304.2.10 #define OFF(zero, i) (((zero)+(i)) % N)

Definition at line 65 of file sober128.c.

5.304.2.11 #define SROUND(z) STEP(c → R,z); NLFUNC(c,(z+1)); XORWORD(t, out+(z∗4));

Definition at line 280 of file sober128.c.

Referenced by sober128_read().

5.304.2.12 #define STEP(R, z) R[OFF(z,0)] = R[OFF(z,15)] ^ R[OFF(z,4)] ^ (R[OFF(z,0)] << 8) ^ Multab[(R[OFF(z,0)] >> 24) & 0xFF];

Definition at line 69 of file sober128.c.

Referenced by cycle().

5.304.2.13 #define WORD2BYTE(w, b) STORE32L(b, w)

Definition at line 52 of file sober128.c.

5.304.2.14 #define XORNL(nl) c → R[FOLDP] ^= (nl);

Definition at line 172 of file sober128.c.

Referenced by sober128_add_entropy().

5.304.3 Function Documentation

5.304.3.1 static ulong32 BYTE2WORD (unsigned char * *b*) [static]

Definition at line 45 of file sober128.c.

Referenced by sober128_add_entropy().

```
{
   ulong32 t;
   LOAD32L(t, b);
   return t;
}
```

5.304.3.2 static void cycle (ulong32 * *R*) [static]

Definition at line 72 of file sober128.c.

References N, and STEP.

Referenced by s128_genkonst(), sober128_add_entropy(), and sober128_read().

```
{
    ulong32 t;
    int     i;

    STEP(R,0);
    t = R[0];
    for (i = 1; i < N; ++i) {
        R[i-1] = R[i];
    }
    R[N-1] = t;
}
```

5.304.3.3 static ulong32 nltap (struct sober128_prng * *c*) [static]

Definition at line 97 of file sober128.c.

References NLFUNC.

Referenced by s128_genkonst(), sober128_add_entropy(), and sober128_read().

```
{
    ulong32 t;
    NLFUNC(c, 0);
    return t;
}
```

5.304.3.4 static void s128_diffuse (struct sober128_prng * *c*) [static]

Definition at line 177 of file sober128.c.

References DROUND.

Referenced by sober128_add_entropy().

```
{
    ulong32 t;
    /* relies on FOLD == N == 17! */
    DROUND(0);
    DROUND(1);
    DROUND(2);
    DROUND(3);
    DROUND(4);
    DROUND(5);
    DROUND(6);
    DROUND(7);
    DROUND(8);
    DROUND(9);
    DROUND(10);
    DROUND(11);
    DROUND(12);
    DROUND(13);
    DROUND(14);
    DROUND(15);
    DROUND(16);
}
```

5.304.3.5 static void s128_genkonst (struct sober128_prng * *c*) [static]

Definition at line 156 of file sober128.c.

References cycle(), and nltap().

Referenced by sober128_add_entropy().

```
{
    ulong32 newkonst;

    do {
       cycle(c->R);
       newkonst = nltap(c);
    } while ((newkonst & 0xFF000000) == 0);
    c->konst = newkonst;
}
```

Here is the call graph for this function:

5.304.3.6 static void s128_reloadstate (struct sober128_prng * *c*) [static]

Definition at line 145 of file sober128.c.

References N.

```
{
    int i;

    for (i = 0; i < N; ++i) {
```

```
        c->R[i] = c->initR[i];
    }
}
```

5.304.3.7 static void s128_savestate (struct sober128_prng * *c*) [static]

Definition at line 135 of file sober128.c.

References N.

Referenced by sober128_add_entropy().

```
{
    int i;
    for (i = 0; i < N; ++i) {
        c->initR[i] = c->R[i];
    }
}
```

5.304.3.8 int sober128_add_entropy (const unsigned char * *in*, unsigned long *inlen*, prng_state * *prng*)

Add entropy to the PRNG state.

Parameters:

in The data to add

inlen Length of the data to add

prng PRNG state to update

Returns:

CRYPT_OK if successful

Definition at line 207 of file sober128.c.

References ADDKEY, BYTE2WORD(), c, CRYPT_INVALID_KEYSIZE, cycle(), LTC_ARGCHK, nltap(), s128_diffuse(), s128_genkonst(), s128_savestate(), and XORNL.

Referenced by sober128_import().

```
{
    struct sober128_prng *c;
    ulong32              i, k;

    LTC_ARGCHK(in != NULL);
    LTC_ARGCHK(prng != NULL);
    c = &(prng->sober128);

    if (c->flag == 1) {
       /* this is the first call to the add_entropy so this input is the key */
       /* inlen must be multiple of 4 bytes */
       if ((inlen & 3) != 0) {
          return CRYPT_INVALID_KEYSIZE;
       }

       for (i = 0; i < inlen; i += 4) {
           k = BYTE2WORD((unsigned char *)&in[i]);
          ADDKEY(k);
```

```
            cycle(c->R);
            XORNL(nltap(c));
         }

         /* also fold in the length of the key */
         ADDKEY(inlen);

         /* now diffuse */
         s128_diffuse(c);

         s128_genkonst(c);
         s128_savestate(c);
         c->nbuf = 0;
         c->flag = 0;
         c->set  = 1;
      } else {
         /* ok we are adding an IV then... */
         s128_reloadstate(c);

         /* inlen must be multiple of 4 bytes */
         if ((inlen & 3) != 0) {
            return CRYPT_INVALID_KEYSIZE;
         }

         for (i = 0; i < inlen; i += 4) {
             k = BYTE2WORD((unsigned char *)&in[i]);
            ADDKEY(k);
            cycle(c->R);
            XORNL(nltap(c));
         }

         /* also fold in the length of the key */
         ADDKEY(inlen);

         /* now diffuse */
         s128_diffuse(c);
         c->nbuf = 0;
      }

      return CRYPT_OK;
}
```

Here is the call graph for this function:

5.304.3.9 int sober128_done (prng_state * *prng*)

Terminate the PRNG.

Parameters:
prng The PRNG to terminate

Returns:
CRYPT_OK if successful

Definition at line 368 of file sober128.c.

References CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(prng != NULL);
   return CRYPT_OK;
}
```

5.304.3.10 int sober128_export (unsigned char * *out*, unsigned long * *outlen*, prng_state * *prng*)

Export the PRNG state.

Parameters:

out [out] Destination

outlen [in/out] Max size and resulting size of the state

prng The PRNG to export

Returns:

CRYPT_OK if successful

Definition at line 381 of file sober128.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_ERROR_READPRNG, CRYPT_OK, LTC_ARGCHK, and sober128_read().

```
{
   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(prng   != NULL);

   if (*outlen < 64) {
      *outlen = 64;
      return CRYPT_BUFFER_OVERFLOW;
   }

   if (sober128_read(out, 64, prng) != 64) {
      return CRYPT_ERROR_READPRNG;
   }
   *outlen = 64;

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.304.3.11 int sober128_import (const unsigned char * *in*, unsigned long *inlen*, prng_state * *prng*)

Import a PRNG state.

Parameters:

in The PRNG state

inlen Size of the state

prng The PRNG to import

Returns:

CRYPT_OK if successful

Definition at line 407 of file sober128.c.

References CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, sober128_add_entropy(), sober128_ready(), and sober128_start().

```
{
   int err;
   LTC_ARGCHK(in   != NULL);
   LTC_ARGCHK(prng != NULL);

   if (inlen != 64) {
      return CRYPT_INVALID_ARG;
   }

   if ((err = sober128_start(prng)) != CRYPT_OK) {
      return err;
   }
   if ((err = sober128_add_entropy(in, 64, prng)) != CRYPT_OK) {
      return err;
   }
   return sober128_ready(prng);
}
```

Here is the call graph for this function:

5.304.3.12 unsigned long sober128_read (unsigned char * *out*, unsigned long *outlen*, prng_state * *prng*)

Read from the PRNG.

Parameters:

out Destination

outlen Length of output

prng The active PRNG to read from

Returns:

Number of octets read

Definition at line 289 of file sober128.c.

References c, cycle(), LTC_ARGCHK, N, nltap(), SROUND, XORWORD(), and zeromem().

Referenced by sober128_export().

```
{
   struct sober128_prng *c;
   ulong32              t, tlen;

   LTC_ARGCHK(out  != NULL);
   LTC_ARGCHK(prng != NULL);

#ifdef LTC_VALGRIND
   zeromem(out, outlen);
#endif

   c = &(prng->sober128);
   t = 0;
   tlen = outlen;

   /* handle any previously buffered bytes */
   while (c->nbuf != 0 && outlen != 0) {
      *out++ ^= c->sbuf & 0xFF;
       c->sbuf >>= 8;
       c->nbuf -= 8;
       --outlen;
```

```
   }

#ifndef LTC_SMALL_CODE
    /* do lots at a time, if there's enough to do */
    while (outlen >= N*4) {
      SROUND(0);
      SROUND(1);
      SROUND(2);
      SROUND(3);
      SROUND(4);
      SROUND(5);
      SROUND(6);
      SROUND(7);
      SROUND(8);
      SROUND(9);
      SROUND(10);
      SROUND(11);
      SROUND(12);
      SROUND(13);
      SROUND(14);
      SROUND(15);
      SROUND(16);
      out    += 4*N;
      outlen -= 4*N;
    }
#endif

    /* do small or odd size buffers the slow way */
    while (4 <= outlen) {
      cycle(c->R);
      t = nltap(c);
      XORWORD(t, out);
      out    += 4;
      outlen -= 4;
    }

    /* handle any trailing bytes */
    if (outlen != 0) {
      cycle(c->R);
      c->sbuf = nltap(c);
      c->nbuf = 32;
      while (c->nbuf != 0 && outlen != 0) {
          *out++ ^= c->sbuf & 0xFF;
          c->sbuf >>= 8;
          c->nbuf -= 8;
          --outlen;
      }
    }

    return tlen;
}
```

Here is the call graph for this function:

5.304.3.13 int sober128_ready (prng_state * *prng*)

Make the PRNG ready to read from.

Parameters:

prng The PRNG to make active

Returns:

CRYPT_OK if successful

Definition at line 273 of file sober128.c.

References CRYPT_ERROR, and CRYPT_OK.

Referenced by sober128_import().

```
{
   return prng->sober128.set == 1 ? CRYPT_OK : CRYPT_ERROR;
}
```

5.304.3.14 int sober128_start (prng_state * *prng*)

Start the PRNG.

Parameters:

prng [out] The PRNG state to initialize

Returns:

CRYPT_OK if successful

Definition at line 109 of file sober128.c.

References c, LTC_ARGCHK, and N.

Referenced by sober128_import(), and sober128_test().

```
{
    int                   i;
    struct sober128_prng *c;

    LTC_ARGCHK(prng != NULL);

    c = &(prng->sober128);

    /* Register initialised to Fibonacci numbers */
    c->R[0] = 1;
    c->R[1] = 1;
    for (i = 2; i < N; ++i) {
       c->R[i] = c->R[i-1] + c->R[i-2];
    }
    c->konst = INITKONST;

    /* next add_entropy will be the key */
    c->flag  = 1;
    c->set   = 0;

    return CRYPT_OK;
}
```

5.304.3.15 int sober128_test (void)

PRNG self-test.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-testing has been disabled

Definition at line 430 of file sober128.c.

References CRYPT_NOP, CRYPT_OK, len, and sober128_start().

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   static const struct {
     int keylen, ivlen, len;
     unsigned char key[16], iv[4], out[20];
   } tests[] = {

{
   16, 4, 20,

   /* key */
   { 0x74, 0x65, 0x73, 0x74, 0x20, 0x6b, 0x65, 0x79,
     0x20, 0x31, 0x32, 0x38, 0x62, 0x69, 0x74, 0x73 },

   /* IV */
   { 0x00, 0x00, 0x00, 0x00 },

   /* expected output */
   { 0x43, 0x50, 0x0c, 0xcf, 0x89, 0x91, 0x9f, 0x1d,
     0xaa, 0x37, 0x74, 0x95, 0xf4, 0xb4, 0x58, 0xc2,
     0x40, 0x37, 0x8b, 0xbb }
}

};
   prng_state    prng;
   unsigned char dst[20];
   int           err, x;

   for (x = 0; x < (int)(sizeof(tests)/sizeof(tests[0])); x++) {
       if ((err = sober128_start(&prng)) != CRYPT_OK) {
          return err;
       }
       if ((err = sober128_add_entropy(tests[x].key, tests[x].keylen, &prng)) != CRYPT_OK) {
          return err;
       }
       /* add IV */
       if ((err = sober128_add_entropy(tests[x].iv, tests[x].ivlen, &prng)) != CRYPT_OK) {
          return err;
       }

       /* ready up */
       if ((err = sober128_ready(&prng)) != CRYPT_OK) {
          return err;
       }
       XMEMSET(dst, 0, tests[x].len);
       if (sober128_read(dst, tests[x].len, &prng) != (unsigned long)tests[x].len) {
          return CRYPT_ERROR_READPRNG;
       }
       sober128_done(&prng);
       if (XMEMCMP(dst, tests[x].out, tests[x].len)) {
#if 0
          printf("\n\nSOBER128 failed, I got:\n");
          for (y = 0; y < tests[x].len; y++) printf("%02x ", dst[y]);
          printf("\n");
#endif
          return CRYPT_FAIL_TESTVECTOR;
       }
   }
   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.304.3.16 static void XORWORD (ulong32 *w*, unsigned char ∗ *b*) [static]

Definition at line 54 of file sober128.c.

Referenced by sober128_read().

```
{
   ulong32 t;
   LOAD32L(t, b);
   t ^= w;
   STORE32L(t, b);
}
```

5.304.4 Variable Documentation

5.304.4.1 const struct ltc_prng_descriptor sober128_desc

Initial value:

```
{
   "sober128", 64,
    &sober128_start,
    &sober128_add_entropy,
    &sober128_ready,
    &sober128_read,
    &sober128_done,
    &sober128_export,
    &sober128_import,
    &sober128_test
}
```

Definition at line 23 of file sober128.c.

5.305 prngs/sober128tab.c File Reference

5.305.1 Detailed Description

SOBER-128 Tables.

Definition in file sober128tab.c.

This graph shows which files directly or indirectly include this file:

Variables

- static const ulong32 Multab [256]
- static const ulong32 Sbox [256]

5.305.2 Variable Documentation

5.305.2.1 const ulong32 Multab[256] `[static]`

Definition at line 8 of file sober128tab.c.

5.305.2.2 const ulong32 Sbox[256] `[static]`

Definition at line 93 of file sober128tab.c.

5.306 prngs/sprng.c File Reference

5.306.1 Detailed Description

Secure PRNG, Tom St Denis.

Definition in file sprng.c.

`#include "tomcrypt.h"`

Include dependency graph for sprng.c:

Functions

- int sprng_start (prng_state *prng)
  Start the PRNG.
- int sprng_add_entropy (const unsigned char *in, unsigned long inlen, prng_state *prng)
  Add entropy to the PRNG state.
- int sprng_ready (prng_state *prng)
  Make the PRNG ready to read from.
- unsigned long sprng_read (unsigned char *out, unsigned long outlen, prng_state *prng)
  Read from the PRNG.
- int sprng_done (prng_state *prng)
  Terminate the PRNG.
- int sprng_export (unsigned char *out, unsigned long *outlen, prng_state *prng)
  Export the PRNG state.
- int sprng_import (const unsigned char *in, unsigned long inlen, prng_state *prng)
  Import a PRNG state.
- int sprng_test (void)
  PRNG self-test.

Variables

- const struct ltc_prng_descriptor sprng_desc

5.306.2 Function Documentation

5.306.2.1 int sprng_add_entropy (const unsigned char * *in*, unsigned long *inlen*, prng_state * *prng*)

Add entropy to the PRNG state.

Parameters:

in The data to add

inlen Length of the data to add

prng PRNG state to update

Returns:

CRYPT_OK if successful

Definition at line 55 of file sprng.c.

References CRYPT_OK.

```
{
   return CRYPT_OK;
}
```

5.306.2.2 int sprng_done (prng_state * *prng*)

Terminate the PRNG.

Parameters:

prng The PRNG to terminate

Returns:

CRYPT_OK if successful

Definition at line 88 of file sprng.c.

References CRYPT_OK.

```
{
   return CRYPT_OK;
}
```

5.306.2.3 int sprng_export (unsigned char * *out*, unsigned long * *outlen*, prng_state * *prng*)

Export the PRNG state.

Parameters:

out [out] Destination

outlen [in/out] Max size and resulting size of the state

prng The PRNG to export

Returns:

CRYPT_OK if successful

Definition at line 100 of file sprng.c.

References CRYPT_OK, and LTC_ARGCHK.

```
{
   LTC_ARGCHK(outlen != NULL);

   *outlen = 0;
   return CRYPT_OK;
}
```

5.306.2.4 int sprng_import (const unsigned char * *in*, unsigned long *inlen*, prng_state * *prng*)

Import a PRNG state.

Parameters:

- ***in*** The PRNG state
- ***inlen*** Size of the state
- ***prng*** The PRNG to import

Returns:

CRYPT_OK if successful

Definition at line 115 of file sprng.c.

References CRYPT_OK.

```
{
   return CRYPT_OK;
}
```

5.306.2.5 unsigned long sprng_read (unsigned char * *out*, unsigned long *outlen*, prng_state * *prng*)

Read from the PRNG.

Parameters:

- ***out*** Destination
- ***outlen*** Length of output
- ***prng*** The active PRNG to read from

Returns:

Number of octets read

Definition at line 77 of file sprng.c.

References LTC_ARGCHK, and rng_get_bytes().

```
{
   LTC_ARGCHK(out != NULL);
   return rng_get_bytes(out, outlen, NULL);
}
```

Here is the call graph for this function:

5.306.2.6 int sprng_ready (prng_state * *prng*)

Make the PRNG ready to read from.

Parameters:

- ***prng*** The PRNG to make active

Returns:

CRYPT_OK if successful

Definition at line 65 of file sprng.c.

References CRYPT_OK.

```
{
   return CRYPT_OK;
}
```

5.306.2.7 int sprng_start (prng_state * *prng*)

Start the PRNG.

Parameters:

prng [out] The PRNG state to initialize

Returns:

CRYPT_OK if successful

Definition at line 43 of file sprng.c.

References CRYPT_OK.

```
{
   return CRYPT_OK;
}
```

5.306.2.8 int sprng_test (void)

PRNG self-test.

Returns:

CRYPT_OK if successful, CRYPT_NOP if self-testing has been disabled

Definition at line 124 of file sprng.c.

References CRYPT_OK.

```
{
   return CRYPT_OK;
}
```

5.306.3 Variable Documentation

5.306.3.1 const struct ltc_prng_descriptor sprng_desc

Initial value:

```
{
    "sprng", 0,
    &sprng_start,
    &sprng_add_entropy,
    &sprng_ready,
    &sprng_read,
```

```
    &sprng_done,
    &sprng_export,
    &sprng_import,
    &sprng_test
}
```

Definition at line 25 of file sprng.c.

5.307 prngs/yarrow.c File Reference

5.307.1 Detailed Description

Yarrow PRNG, Tom St Denis.

Definition in file yarrow.c.

`#include "tomcrypt.h"`

Include dependency graph for yarrow.c:

Functions

- int yarrow_start (prng_state *prng)
  Start the PRNG.
- int yarrow_add_entropy (const unsigned char *in, unsigned long inlen, prng_state *prng)
  Add entropy to the PRNG state.
- int yarrow_ready (prng_state *prng)
  Make the PRNG ready to read from.
- unsigned long yarrow_read (unsigned char *out, unsigned long outlen, prng_state *prng)
  Read from the PRNG.
- int yarrow_done (prng_state *prng)
  Terminate the PRNG.
- int yarrow_export (unsigned char *out, unsigned long *outlen, prng_state *prng)
  Export the PRNG state.
- int yarrow_import (const unsigned char *in, unsigned long inlen, prng_state *prng)
  Import a PRNG state.
- int yarrow_test (void)
  PRNG self-test.

Variables

- const struct ltc_prng_descriptor yarrow_desc

5.307.2 Function Documentation

5.307.2.1 int yarrow_add_entropy (const unsigned char * *in*, unsigned long *inlen*, prng_state * *prng*)

Add entropy to the PRNG state.

Parameters:

in The data to add

inlen Length of the data to add

prng PRNG state to update

Returns:

CRYPT_OK if successful

Definition at line 135 of file yarrow.c.

References CRYPT_OK, hash_descriptor, hash_is_valid(), LTC_ARGCHK, LTC_MUTEX_LOCK, and LTC_MUTEX_UNLOCK.

Referenced by yarrow_import().

```
136 {
137    hash_state md;
138    int err;
139
140    LTC_ARGCHK(in  != NULL);
141    LTC_ARGCHK(prng != NULL);
142
143    LTC_MUTEX_LOCK(&prng->yarrow.prng_lock);
144
145    if ((err = hash_is_valid(prng->yarrow.hash)) != CRYPT_OK) {
146       LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
147       return err;
148    }
149
150    /* start the hash */
151    if ((err = hash_descriptor[prng->yarrow.hash].init(&md)) != CRYPT_OK) {
152       LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
153       return err;
154    }
155
156    /* hash the current pool */
157    if ((err = hash_descriptor[prng->yarrow.hash].process(&md, prng->yarrow.pool,
158                                                         hash_descriptor[prng->yarrow.hash].hashsize))
159       LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
160       return err;
161    }
162
163    /* add the new entropy */
164    if ((err = hash_descriptor[prng->yarrow.hash].process(&md, in, inlen)) != CRYPT_OK) {
165       LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
166       return err;
167    }
168
169    /* store result */
170    if ((err = hash_descriptor[prng->yarrow.hash].done(&md, prng->yarrow.pool)) != CRYPT_OK) {
171       LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
172       return err;
173    }
174
175    LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
176    return CRYPT_OK;
177 }
```

Here is the call graph for this function:

5.307.2.2 int yarrow_done (prng_state ∗ *prng*)

Terminate the PRNG.

Parameters:

prng The PRNG to terminate

Returns:

CRYPT_OK if successful

Definition at line 252 of file yarrow.c.

References ctr_done(), LTC_ARGCHK, LTC_MUTEX_LOCK, and LTC_MUTEX_UNLOCK.

```
{
   int err;
   LTC_ARGCHK(prng != NULL);

   LTC_MUTEX_LOCK(&prng->yarrow.prng_lock);

   /* call cipher done when we invent one ;-) */

   /* we invented one */
   err = ctr_done(&prng->yarrow.ctr);

   LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
   return err;
}
```

Here is the call graph for this function:

5.307.2.3 int yarrow_export (unsigned char ∗ *out*, unsigned long ∗ *outlen*, prng_state ∗ *prng*)

Export the PRNG state.

Parameters:

out [out] Destination

outlen [in/out] Max size and resulting size of the state

prng The PRNG to export

Returns:

CRYPT_OK if successful

Definition at line 275 of file yarrow.c.

References CRYPT_BUFFER_OVERFLOW, CRYPT_ERROR_READPRNG, CRYPT_OK, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_UNLOCK, and yarrow_read().

```
{
   LTC_ARGCHK(out    != NULL);
   LTC_ARGCHK(outlen != NULL);
   LTC_ARGCHK(prng   != NULL);

   LTC_MUTEX_LOCK(&prng->yarrow.prng_lock);

   /* we'll write 64 bytes for s&g's */
   if (*outlen < 64) {
      LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
      *outlen = 64;
      return CRYPT_BUFFER_OVERFLOW;
   }

```

```
   if (yarrow_read(out, 64, prng) != 64) {
      LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
      return CRYPT_ERROR_READPRNG;
   }
   *outlen = 64;

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.307.2.4 int yarrow_import (const unsigned char * *in*, unsigned long *inlen*, prng_state * *prng*)

Import a PRNG state.

Parameters:

in The PRNG state

inlen Size of the state

prng The PRNG to import

Returns:

CRYPT_OK if successful

Definition at line 306 of file yarrow.c.

References CRYPT_INVALID_ARG, CRYPT_OK, LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_-MUTEX_UNLOCK, yarrow_add_entropy(), and yarrow_start().

```
{
   int err;

   LTC_ARGCHK(in   != NULL);
   LTC_ARGCHK(prng != NULL);

   LTC_MUTEX_LOCK(&prng->yarrow.prng_lock);

   if (inlen != 64) {
      LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
      return CRYPT_INVALID_ARG;
   }

   if ((err = yarrow_start(prng)) != CRYPT_OK) {
      LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
      return err;
   }
   err = yarrow_add_entropy(in, 64, prng);
   LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
   return err;
}
```

Here is the call graph for this function:

5.307.2.5 unsigned long yarrow_read (unsigned char * *out*, unsigned long *outlen*, prng_state * *prng*)

Read from the PRNG.

Parameters:

out Destination

outlen Length of output

prng The active PRNG to read from

Returns:

Number of octets read

Definition at line 228 of file yarrow.c.

References CRYPT_OK, ctr_encrypt(), LTC_ARGCHK, LTC_MUTEX_LOCK, LTC_MUTEX_-UNLOCK, and zeromem().

Referenced by yarrow_export().

```
{
   LTC_ARGCHK(out  != NULL);
   LTC_ARGCHK(prng != NULL);

   LTC_MUTEX_LOCK(&prng->yarrow.prng_lock);

   /* put out in predictable state first */
   zeromem(out, outlen);

   /* now randomize it */
   if (ctr_encrypt(out, out, outlen, &prng->yarrow.ctr) != CRYPT_OK) {
      LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
      return 0;
   }
   LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
   return outlen;
}
```

Here is the call graph for this function:

5.307.2.6 int yarrow_ready (prng_state * *prng*)

Make the PRNG ready to read from.

Parameters:

prng The PRNG to make active

Returns:

CRYPT_OK if successful

Definition at line 184 of file yarrow.c.

References cipher_descriptor, cipher_is_valid(), CRYPT_OK, ctr_start(), hash_descriptor, hash_is_valid(), LTC_ARGCHK, LTC_MUTEX_LOCK, and LTC_MUTEX_UNLOCK.

```
{
   int ks, err;

   LTC_ARGCHK(prng != NULL);
   LTC_MUTEX_LOCK(&prng->yarrow.prng_lock);

   if ((err = hash_is_valid(prng->yarrow.hash)) != CRYPT_OK) {
      LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
```

```
         return err;
      }

      if ((err = cipher_is_valid(prng->yarrow.cipher)) != CRYPT_OK) {
         LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
         return err;
      }

      /* setup CTR mode using the "pool" as the key */
      ks = (int)hash_descriptor[prng->yarrow.hash].hashsize;
      if ((err = cipher_descriptor[prng->yarrow.cipher].keysize(&ks)) != CRYPT_OK) {
         LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
         return err;
      }

      if ((err = ctr_start(prng->yarrow.cipher,     /* what cipher to use */
                           prng->yarrow.pool,       /* IV */
                           prng->yarrow.pool, ks,   /* KEY and key size */
                           0,                       /* number of rounds */
                           CTR_COUNTER_LITTLE_ENDIAN, /* little endian counter */
                           &prng->yarrow.ctr)) != CRYPT_OK) {
         LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
         return err;
      }
      LTC_MUTEX_UNLOCK(&prng->yarrow.prng_lock);
      return CRYPT_OK;
   }
```

Here is the call graph for this function:

5.307.2.7 int yarrow_start (prng_state * *prng*)

Start the PRNG.

Parameters:

prng [out] The PRNG state to initialize

Returns:

CRYPT_OK if successful

Definition at line 38 of file yarrow.c.

References aes_desc, anubis_desc, blowfish_desc, cast5_desc, cipher_is_valid(), CRYPT_OK, des3_desc, hash_is_valid(), khazad_desc, kseed_desc, LTC_ARGCHK, LTC_MUTEX_INIT, md2_desc, md4_desc, md5_desc, noekeon_desc, rc2_desc, rc5_desc, rc6_desc, register_cipher(), register_hash(), rijndael_desc, rmd128_desc, rmd160_desc, rmd256_desc, rmd320_desc, safer_sk128_desc, saferp_desc, sha1_desc, sha256_desc, sha512_desc, tiger_desc, twofish_desc, whirlpool_desc, xtea_desc, and zeromem().

Referenced by yarrow_import(), and yarrow_test().

```
{
   int err;

   LTC_ARGCHK(prng != NULL);

   /* these are the default hash/cipher combo used */
#ifdef RIJNDAEL
#if    YARROW_AES==0
   prng->yarrow.cipher = register_cipher(&rijndael_enc_desc);
#elif  YARROW_AES==1
   prng->yarrow.cipher = register_cipher(&aes_enc_desc);
```

```
#elif  YARROW_AES==2
   prng->yarrow.cipher = register_cipher(&rijndael_desc);
#elif  YARROW_AES==3
   prng->yarrow.cipher = register_cipher(&aes_desc);
#endif
#elif defined(BLOWFISH)
   prng->yarrow.cipher = register_cipher(&blowfish_desc);
#elif defined(TWOFISH)
   prng->yarrow.cipher = register_cipher(&twofish_desc);
#elif defined(RC6)
   prng->yarrow.cipher = register_cipher(&rc6_desc);
#elif defined(RC5)
   prng->yarrow.cipher = register_cipher(&rc5_desc);
#elif defined(SAFERP)
   prng->yarrow.cipher = register_cipher(&saferp_desc);
#elif defined(RC2)
   prng->yarrow.cipher = register_cipher(&rc2_desc);
#elif defined(NOEKEON)
   prng->yarrow.cipher = register_cipher(&noekeon_desc);
#elif defined(ANUBIS)
   prng->yarrow.cipher = register_cipher(&anubis_desc);
#elif defined(KSEED)
   prng->yarrow.cipher = register_cipher(&kseed_desc);
#elif defined(KHAZAD)
   prng->yarrow.cipher = register_cipher(&khazad_desc);
#elif defined(CAST5)
   prng->yarrow.cipher = register_cipher(&cast5_desc);
#elif defined(XTEA)
   prng->yarrow.cipher = register_cipher(&xtea_desc);
#elif defined(SAFER)
   prng->yarrow.cipher = register_cipher(&safer_sk128_desc);
#elif defined(DES)
   prng->yarrow.cipher = register_cipher(&des3_desc);
#else
   #error YARROW needs at least one CIPHER
#endif
   if ((err = cipher_is_valid(prng->yarrow.cipher)) != CRYPT_OK) {
      return err;
   }

#ifdef SHA256
   prng->yarrow.hash   = register_hash(&sha256_desc);
#elif defined(SHA512)
   prng->yarrow.hash   = register_hash(&sha512_desc);
#elif defined(TIGER)
   prng->yarrow.hash   = register_hash(&tiger_desc);
#elif defined(SHA1)
   prng->yarrow.hash   = register_hash(&sha1_desc);
#elif defined(RIPEMD320)
   prng->yarrow.hash   = register_hash(&rmd320_desc);
#elif defined(RIPEMD256)
   prng->yarrow.hash   = register_hash(&rmd256_desc);
#elif defined(RIPEMD160)
   prng->yarrow.hash   = register_hash(&rmd160_desc);
#elif defined(RIPEMD128)
   prng->yarrow.hash   = register_hash(&rmd128_desc);
#elif defined(MD5)
   prng->yarrow.hash   = register_hash(&md5_desc);
#elif defined(MD4)
   prng->yarrow.hash   = register_hash(&md4_desc);
#elif defined(MD2)
   prng->yarrow.hash   = register_hash(&md2_desc);
#elif defined(WHIRLPOOL)
   prng->yarrow.hash   = register_hash(&whirlpool_desc);
#else
   #error YARROW needs at least one HASH
#endif
```

```
   if ((err = hash_is_valid(prng->yarrow.hash)) != CRYPT_OK) {
      return err;
   }

   /* zero the memory used */
   zeromem(prng->yarrow.pool, sizeof(prng->yarrow.pool));
   LTC_MUTEX_INIT(&prng->yarrow.prng_lock)

   return CRYPT_OK;
}
```

Here is the call graph for this function:

5.307.2.8 int yarrow_test (void)

PRNG self-test.

Returns:
CRYPT_OK if successful, CRYPT_NOP if self-testing has been disabled

Definition at line 333 of file yarrow.c.

References cipher_descriptor, CRYPT_NOP, CRYPT_OK, hash_descriptor, and yarrow_start().

```
{
#ifndef LTC_TEST
   return CRYPT_NOP;
#else
   int err;
   prng_state prng;

   if ((err = yarrow_start(&prng)) != CRYPT_OK) {
      return err;
   }

   /* now let's test the hash/cipher that was chosen */
   if ((err = cipher_descriptor[prng.yarrow.cipher].test()) != CRYPT_OK) {
      return err;
   }
   if ((err = hash_descriptor[prng.yarrow.hash].test()) != CRYPT_OK) {
      return err;
   }

   return CRYPT_OK;
#endif
}
```

Here is the call graph for this function:

5.307.3 Variable Documentation

5.307.3.1 const struct ltc_prng_descriptor yarrow_desc

Initial value:

```
{
    "yarrow", 64,
    &yarrow_start,
    &yarrow_add_entropy,
```

```
    &yarrow_ready,
    &yarrow_read,
    &yarrow_done,
    &yarrow_export,
    &yarrow_import,
    &yarrow_test
}
```

Definition at line 20 of file yarrow.c.

Index