Intel® Intelligent Storage Acceleration Library Crypto (Intel® ISA-L Crypto) Open Source Version

API Reference Manual - Version 2.19.0

July 27, 2017

Contents

CHAPTER 1
STORAGE LIBRARY

1.1 About This Document

This document describes the software programming interface and operation of functions in the library. Sections in this document are grouped by the functions found in individual header files that define the function prototypes. Subsections include function parameters, description and type.

This document refers to the open release crypto version of the library. A separate, general release called the Intel® Intelligent Storage Acceleration Library (Intel® ISA-L) is also available and contains an extended set of functions.

1.2 Overview

The Intel® Intelligent Storage Acceleration Library Crypto (Intel® ISA-L Crypto) Open Source Version is a collection of functions used in storage applications optimized for Intel architecture Intel® 64. In some cases, multiple versions of the same function are available that are optimized for a particular Intel architecture and instruction set. This software takes advantage of new instructions and users should ensure that the chosen function is compatible with hardware it will run on.

Multibinary support has been added for many units in ISA-L. With multibinary support functions, an appropriate version is selected at first run and can be called instead of the architecture-specific versions. This allows users to deploy a single binary with multiple function versions and choose at run time based on platform features. Users can still call the architecture-specific versions directly to reduce code size. There are also base functions, written in C, which the multibinary function will call if none of the required instruction sets are enabled.

1.3 Multi-buffer Hashing Functions

Functions in the Multi-buffer MD5, SHA1, SHA256 and SHA512 sections are used to increase the performance of the secure hash algorithms on a single processor core by operating on multiple jobs at once. By buffering jobs, the algorithm can exploit the instruction-level parallelism inherent in modern IA cores to an extent not possible in a serial implementation. The multi-buffer API is similar to that used in the whitepaper Fast Multi-buffer IPsec Implementations on Intel Architecture Processors.

It uses the HASH_CTX_MGR context. Benefits to using the API include:

- Multibinary functionality. Call one function and the appropriate architecture-specific version is fixed up at runtime.
- No restriction on update length. Submitting an update block no longer has to have length a multiple of the fundamental block size.

As noted in the above document, the scheduler routines do not enforce atomic access to the context structure. If a single scheduler state structure is being used by multiple threads, then the application must take care that calls are not made from different threads at the same time, i.e. thread-safety should be implemented at a level higher than these routines. This could be implemented by employing a separate context structure for each worker thread.

1.4 Alignment for Input Parameters

The alignment required for the input parameters of each of the Intel® ISA-L functions is documented in the relevant sections of this API manual. The table below outlines these requirements.

Function	Alignment Required
AES-XTS 128	No
AES-XTS 256	No
MB Hashing - new API	No (Members of CTX structures defined with required alignment, already aligned once initialised. On FreeBSD, or when using Linux/icc, the CTX_MGR structure may need to be aligned to 16B.)

1.5 System Requirements

Individual functions may have various run-time requirements such as the minimum version of SSE as described in Instruction Set Requirements. General requirements are listed below.

Recommended Hardware:

- em64t: A system based on the Intel® Xeon® processor with Intel® 64 architecture.
- IA32: When available for 32-bit functions; A system based on the Intel® Xeon® processor or subsequent IA-32 architecture based processor.

Software Requirements:

Most functions in the library use the 64-bit embedded and Unix standard for calling convention http://refspecs.linuxfoundation.org/elf/x86_64-abi-0.95.pdf. When available, 32-bit versions use cdecl. Individual functions are written to be statically linked with an application.

Building Library Functions:

- Yasm Assembler: version at least v1.2.0. (v1.3.0 would be better)
- or Nasm Assembler: version at least v2.10.

Building Examples and Tests:

Examples and test source follow simple command line POSIX standards and should be portable to any mostly POSIX-compliant OS.

Note

Please note that the library assumes 1MB = 1,000,000 bytes in reported performance figures.

CHAPTER 2
FUNCTION VERSION NUMBERS

2.1 Function Version Numbers

Individual functions are given version numbers with the format mm-vv-ssss.
- mm = Two hex digits indicating the processor a function was optimized for.

  - 00 = Nehalem/Jasper Forest/Multibinary
  - 01 = Westmere
  - 02 = Sandybridge
  - 03 = Ivy Bridge
  - 04 = Haswell
  - 05 = Silvermont
  - 06 = Skylake
  - 07 = Goldmont
  - 08 = Cannonlake

- vv = function version number
- ssss = function serial number

2.2 Function Version Numbers Tables

Function	Version
sha1_update	00-02-0050
sha1_opt	00-02-0051
XTS_AES_128_enc	01-04-0071
XTS_AES_128_enc_expanded_key	01-04-0072
XTS_AES_128_dec	01-04-0073
XTS_AES_128_dec_expanded_key	01-04-0074
XTS_AES_256_enc	01-04-0076
XTS_AES_256_enc_expanded_key	01-04-0077
XTS_AES_256_dec	01-04-0078
XTS_AES_256_dec_expanded_key	01-04-0079
sha1_ctx_mgr_init_sse	00-02-0139
sha1_ctx_mgr_submit_sse	00-02-0140
sha1_ctx_mgr_flush_sse	00-02-0141
sha1_ctx_mgr_init_avx	02-02-0142
sha1_ctx_mgr_submit_avx	02-02-0143
sha1_ctx_mgr_flush_avx	02-02-0144
sha1_ctx_mgr_init_avx2	04-02-0145
sha1_ctx_mgr_submit_avx2	04-02-0146
sha1_ctx_mgr_flush_avx2	04-02-0147
sha1_ctx_mgr_init	00-03-0148
sha1_ctx_mgr_submit	00-03-0149
sha1_ctx_mgr_init_avx512	06-00-014a
sha1_ctx_mgr_submit_avx512	06-00-014b
sha1_ctx_mgr_flush_avx512	06-00-014c
sha1_ctx_mgr_flush	00-03-0150
sha256_ctx_mgr_init_sse	00-02-0151
sha256_ctx_mgr_submit_sse	00-02-0152
sha256_ctx_mgr_flush_sse	00-02-0153
sha256_ctx_mgr_init_avx	02-02-0154
sha256_ctx_mgr_submit_avx	02-02-0155
sha256_ctx_mgr_flush_avx	02-02-0156
sha256_ctx_mgr_init_avx2	04-02-0157
sha256_ctx_mgr_submit_avx2	04-02-0158
sha256_ctx_mgr_flush_avx2	04-02-0159
sha256_ctx_mgr_init_avx512	06-00-015a
sha256_ctx_mgr_submit_avx512	06-00-015b
sha256_ctx_mgr_flush_avx512	06-00-015c
sha256_ctx_mgr_init	00-03-0160
sha256_ctx_mgr_submit	00-03-0161
sha256_ctx_mgr_flush	00-03-0162
sha512_ctx_mgr_init_sse	00-02-0163

Function	Version
sha512_ctx_mgr_submit_sse	00-02-0164
sha512_ctx_mgr_flush_sse	00-02-0165
sha512_ctx_mgr_init_avx	02-02-0166
sha512_ctx_mgr_submit_avx	02-02-0167
sha512_ctx_mgr_flush_avx	02-02-0168
sha512_ctx_mgr_init_avx2	04-02-0169
sha512_ctx_mgr_init_avx512	06-00-016a
sha512_ctx_mgr_submit_avx512	06-00-016b
sha512_ctx_mgr_flush_avx512	06-00-016c
sha512_ctx_mgr_submit_avx2	04-02-0170
sha512_ctx_mgr_flush_avx2	04-02-0171
sha512_ctx_mgr_init_sb_sse4	05-02-0172
sha512_ctx_mgr_submit_sb_sse4	05-02-0173
sha512_ctx_mgr_flush_sb_sse4	05-02-0174
sha512_ctx_mgr_init	00-03-0175
sha512_ctx_mgr_submit	00-03-0176
sha512_ctx_mgr_flush	00-03-0177
md5_ctx_mgr_init_sse	00-02-0180
md5_ctx_mgr_submit_sse	00-02-0181
md5_ctx_mgr_flush_sse	00-02-0182
md5_ctx_mgr_init_avx	02-02-0183
md5_ctx_mgr_submit_avx	02-02-0184
md5_ctx_mgr_flush_avx	02-02-0185
md5_ctx_mgr_init_avx2	04-02-0186
md5_ctx_mgr_submit_avx2	04-02-0187
md5_ctx_mgr_flush_avx2	04-02-0188
md5_ctx_mgr_init	00-03-0189
md5_ctx_mgr_submit	00-03-018a
md5_ctx_mgr_flush	00-03-018b
md5_ctx_mgr_init_avx512	06-00-018c
md5_ctx_mgr_submit_avx512	06-00-018d
md5_ctx_mgr_flush_avx512	06-00-018e
mh_sha1_murmur3_x64_128_init	00-00-0251
mh_sha1_murmur3_x64_128_update	00-02-0252
mh_sha1_murmur3_x64_128_finalize	00-02-0253
mh_sha1_murmur3_x64_128_update_sse	00-00-0254
mh_sha1_murmur3_x64_128_finalize_sse	00-00-0255
mh_sha1_murmur3_x64_128_update_avx	02-00-0256
mh_sha1_murmur3_x64_128_finalize_avx	02-00-0257
mh_sha1_murmur3_x64_128_update_avx2	04-00-0258

Function	Version
mh_sha1_murmur3_x64_128_finalize_avx2	04-00-0259
mh_sha1_murmur3_x64_128_update_base	00-00-025a
mh_sha1_murmur3_x64_128_finalize_base	00-00-025b
mh_sha1_murmur3_x64_128_update_avx512	06-00-025c
mh_sha1_murmur3_x64_128_finalize_avx512	06-00-025d
rolling_hashx_mask_gen	00-00-0260
rolling_hash1_init	00-00-0261
rolling_hash1_reset	00-00-0262
rolling_hash1_run	00-00-0263
rolling_hash2_init	00-00-0264
rolling_hash2_reset	00-00-0265
rolling_hash2_run	00-00-0266
mh_sha1_init	00-00-0271
mh_sha1_update	00-02-0272
mh_sha1_finalize	00-02-0273
mh_sha1_update_sse	00-00-0274
mh_sha1_finalize_sse	00-00-0275
mh_sha1_update_avx	02-00-0276
mh_sha1_finalize_avx	02-00-0277
mh_sha1_update_avx2	04-00-0278
mh_sha1_finalize_avx2	04-00-0279
mh_sha1_update_base	00-00-027a
mh_sha1_finalize_base	00-00-027b
mh_sha1_update_avx512	06-00-027c
mh_sha1_finalize_avx512	06-00-027d
aesni_gcm128_enc	00-00-0280
aesni_gcm128_dec	00-00-0281
aesni_gcm128_init	00-00-0282
aesni_gcm128_enc_update	00-00-0283
aesni_gcm128_dec_update	00-00-0284
aesni_gcm128_enc_finalize	00-00-0285
aesni_gcm128_dec_finalize	00-00-0286
aesni_gcm128_pre	00-00-0287
aesni_gcm256_enc	00-00-0288
aesni_gcm256_dec	00-00-0289
aesni_gcm256_init	00-00-028a
aesni_gcm256_enc_update	00-00-028b
aesni_gcm256_dec_update	00-00-028c
aesni_gcm256_enc_finalize	00-00-028d
aesni_gcm256_dec_finalize	00-00-028e

Function	Version
aesni_gcm256_pre	00-00-028f
aes_cbc_enc_128	00-00-0291
aes_cbc_dec_128	00-00-0292
aes_cbc_enc_192	00-00-0293
aes_cbc_dec_192	00-00-0294
aes_cbc_enc_256	00-00-0295
aes_cbc_dec_256	00-00-0296
aes_cbc_precomp	00-00-0297
aes_keyexp_128	00-01-02a1
aes_keyexp_192	00-01-02a2
aes_keyexp_256	00-01-02a3
mh_sha256_init	00-00-02b1
mh_sha256_update	00-00-02b2
mh_sha256_finalize	00-00-02b3
mh_sha256_update_sse	00-00-02b4
mh_sha256_finalize_sse	00-00-02b5
mh_sha256_update_avx	02-00-02b6
mh_sha256_finalize_avx	02-00-02b7
mh_sha256_update_avx2	04-00-02b8
mh_sha256_finalize_avx2	04-00-02b9
mh_sha256_update_base	00-00-02ba
mh_sha256_finalize_base	00-00-02bb
mh_sha256_update_avx512	06-00-02bc
mh_sha256_finalize_avx512	06-00-02bd
sha1_ctx_mgr_init_sse_ni	07-00-02c1
sha1_ctx_mgr_submit_sse_ni	07-00-02c2
sha1_ctx_mgr_flush_sse_ni	07-00-02c3
sha1_ctx_mgr_init_avx512_ni	08-00-02c4
sha1_ctx_mgr_submit_avx512_ni	08-00-02c5
sha1_ctx_mgr_flush_avx512_ni	08-00-02c6
sha256_ctx_mgr_init_sse_ni	07-00-02c7
sha256_ctx_mgr_submit_sse_ni	07-00-02c8
sha256_ctx_mgr_flush_sse_ni	07-00-02c9
sha256_ctx_mgr_init_avx512_ni	08-00-02ca
sha256_ctx_mgr_submit_avx512_ni	08-00-02cb
sha256_ctx_mgr_flush_avx512_ni	08-00-02cc

CHAPTER 3
INSTRUCTION SET REQUIREMENTS

aes_cbc_dec_128 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)
SSE4.1 and AESNI

aes_cbc_dec_192 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)
SSE4.1 and AESNI

aes_cbc_dec_256 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)
SSE4.1 and AESNI

aes_cbc_enc_128 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)
SSE4.1 and AESNI

aes_cbc_enc_192 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)
SSE4.1 and AESNI

aes_cbc_enc_256 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)
SSE4.1 and AESNI

aes_cbc_precomp (uint8_t *key, int key_size, struct cbc_key_data *keys_blk)
SSE4.1 and AESNI

aes_gcm_dec_128 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, uint8_t const *in, uint64_t len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aes_gcm_dec_128_finalize (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aes_gcm_dec_128_update (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, const uint8_t *in, uint64_t len)
SSE4.1 and AESNI

aes_gcm_dec_256 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, uint8_t const *in, uint64_t len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aes_gcm_dec_256_finalize (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aes_gcm_dec_256_update (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, const uint8_t *in, uint64_t len)
SSE4.1 and AESNI

aes_gcm_enc_128 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, uint8_t const *in, uint64_t len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aes_gcm_enc_128_finalize (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aes_gcm_enc_128_update (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, const uint8_t *in, uint64_t len)
SSE4.1 and AESNI

aes_gcm_enc_256 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, uint8_t const *in, uint64_t len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aes_gcm_enc_256_finalize (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aes_gcm_enc_256_update (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, const uint8_t *in, uint64_t len)
SSE4.1 and AESNI

aes_gcm_init_128 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *iv, uint8_t const *aad, uint64_t aad_len)
SSE4.1 and AESNI

aes_gcm_init_256 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *iv, uint8_t const *aad, uint64_t aad_len)
SSE4.1 and AESNI

aes_gcm_pre_128 (const void *key, struct gcm_key_data *key_data)
SSE4.1 and AESNI

aes_gcm_pre_256 (const void *key, struct gcm_key_data *key_data)
SSE4.1 and AESNI

aes_keyexp_128 (uint8_t *key, uint8_t *exp_key_enc, uint8_t *exp_key_dec)
SSE4.1

aes_keyexp_192 (uint8_t *key, uint8_t *exp_key_enc, uint8_t *exp_key_dec)
SSE4.1

aes_keyexp_256 (uint8_t *key, uint8_t *exp_key_enc, uint8_t *exp_key_dec)
SSE4.1

aesni_gcm128_dec (struct gcm_data *my_ctx_data, uint8_t *out, uint8_t const *in, uint64_t plaintext_len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aesni_gcm128_dec_finalize (struct gcm_data *my_ctx_data, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aesni_gcm128_dec_update (struct gcm_data *my_ctx_data, uint8_t *out, const uint8_t *in, uint64_t plaintext_len)
SSE4.1 and AESNI

aesni_gcm128_enc (struct gcm_data *my_ctx_data, uint8_t *out, uint8_t const *in, uint64_t plaintext_len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aesni_gcm128_enc_finalize (struct gcm_data *my_ctx_data, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aesni_gcm128_enc_update (struct gcm_data *my_ctx_data, uint8_t *out, const uint8_t *in, uint64_t plaintext_len)
SSE4.1 and AESNI

aesni_gcm128_init (struct gcm_data *my_ctx_data, uint8_t *iv, uint8_t const *aad, uint64_t aad_len)
SSE4.1 and AESNI

aesni_gcm256_dec (struct gcm_data *my_ctx_data, uint8_t *out, uint8_t const *in, uint64_t plaintext_len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aesni_gcm256_dec_finalize (struct gcm_data *my_ctx_data, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aesni_gcm256_dec_update (struct gcm_data *my_ctx_data, uint8_t *out, const uint8_t *in, uint64_t plaintext_len)
SSE4.1 and AESNI

aesni_gcm256_enc (struct gcm_data *my_ctx_data, uint8_t *out, uint8_t const *in, uint64_t plaintext_len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aesni_gcm256_enc_finalize (struct gcm_data *my_ctx_data, uint8_t *auth_tag, uint64_t auth_tag_len)
SSE4.1 and AESNI

aesni_gcm256_enc_update (struct gcm_data *my_ctx_data, uint8_t *out, const uint8_t *in, uint64_t plaintext_len)
SSE4.1 and AESNI

aesni_gcm256_init (struct gcm_data *my_ctx_data, uint8_t *iv, uint8_t const *aad, uint64_t aad_len)
SSE4.1 and AESNI

md5_ctx_mgr_flush (MD5_HASH_CTX_MGR *mgr)
SSE4.1 or AVX or AVX2 or AVX512

md5_ctx_mgr_flush_avx (MD5_HASH_CTX_MGR *mgr)
AVX

md5_ctx_mgr_flush_avx2 (MD5_HASH_CTX_MGR *mgr)
AVX2

md5_ctx_mgr_flush_avx512 (MD5_HASH_CTX_MGR *mgr)
AVX512

md5_ctx_mgr_flush_sse (MD5_HASH_CTX_MGR *mgr)
SSE4.1

md5_ctx_mgr_init (MD5_HASH_CTX_MGR *mgr)
SSE4.1 or AVX or AVX2 or AVX512

md5_ctx_mgr_init_avx (MD5_HASH_CTX_MGR *mgr)
AVX

md5_ctx_mgr_init_avx2 (MD5_HASH_CTX_MGR *mgr)
AVX2

md5_ctx_mgr_init_avx512 (MD5_HASH_CTX_MGR *mgr)
AVX512

md5_ctx_mgr_init_sse (MD5_HASH_CTX_MGR *mgr)
SSE4.1

md5_ctx_mgr_submit (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1 or AVX or AVX2 or AVX512

md5_ctx_mgr_submit_avx (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX

md5_ctx_mgr_submit_avx2 (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX2

md5_ctx_mgr_submit_avx512 (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX512

md5_ctx_mgr_submit_sse (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1

mh_sha1_finalize_avx (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)
AVX

mh_sha1_finalize_avx2 (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)
AVX2

mh_sha1_finalize_avx512 (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)
AVX512

mh_sha1_finalize_sse (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)
SSE

mh_sha1_murmur3_x64_128_finalize_avx (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)
AVX

mh_sha1_murmur3_x64_128_finalize_avx2 (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)
AVX2

mh_sha1_murmur3_x64_128_finalize_avx512 (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)
AVX512

mh_sha1_murmur3_x64_128_finalize_sse (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)
SSE

mh_sha1_murmur3_x64_128_update_avx (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)
AVX

mh_sha1_murmur3_x64_128_update_avx2 (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)
AVX2

mh_sha1_murmur3_x64_128_update_avx512 (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)
AVX512

mh_sha1_murmur3_x64_128_update_sse (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)
SSE

mh_sha1_update_avx (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)
AVX

mh_sha1_update_avx2 (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)
AVX2

mh_sha1_update_avx512 (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)
AVX512

mh_sha1_update_sse (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)
SSE

mh_sha256_finalize_avx (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)
AVX

mh_sha256_finalize_avx2 (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)
AVX2

mh_sha256_finalize_avx512 (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)
AVX512

mh_sha256_finalize_sse (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)
SSE

mh_sha256_update_avx (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)
AVX

mh_sha256_update_avx2 (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)
AVX2

mh_sha256_update_avx512 (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)
AVX512

mh_sha256_update_sse (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)
SSE

rolling_hash1_run (struct rh_state1 *state, uint8_t *buffer, uint32_t buffer_length, uint32_t mask, uint32_t trigger, uint32_t *offset)
SSE4.2

sha1_ctx_mgr_flush (SHA1_HASH_CTX_MGR *mgr)
SSE4.1 or AVX or AVX2 or AVX512

sha1_ctx_mgr_flush_avx (SHA1_HASH_CTX_MGR *mgr)
AVX

sha1_ctx_mgr_flush_avx2 (SHA1_HASH_CTX_MGR *mgr)
AVX2

sha1_ctx_mgr_flush_avx512 (SHA1_HASH_CTX_MGR *mgr)
AVX512

sha1_ctx_mgr_flush_avx512_ni (SHA1_HASH_CTX_MGR *mgr)
AVX512 and SHANI

sha1_ctx_mgr_flush_sse (SHA1_HASH_CTX_MGR *mgr)
SSE4.1

sha1_ctx_mgr_flush_sse_ni (SHA1_HASH_CTX_MGR *mgr)
SSE4.1 and SHANI

sha1_ctx_mgr_init (SHA1_HASH_CTX_MGR *mgr)
SSE4.1 or AVX or AVX2 or AVX512

sha1_ctx_mgr_init_avx (SHA1_HASH_CTX_MGR *mgr)
AVX

sha1_ctx_mgr_init_avx2 (SHA1_HASH_CTX_MGR *mgr)
AVX2

sha1_ctx_mgr_init_avx512 (SHA1_HASH_CTX_MGR *mgr)
AVX512

sha1_ctx_mgr_init_avx512_ni (SHA1_HASH_CTX_MGR *mgr)
AVX512 and SHANI

sha1_ctx_mgr_init_sse (SHA1_HASH_CTX_MGR *mgr)
SSE4.1

sha1_ctx_mgr_init_sse_ni (SHA1_HASH_CTX_MGR *mgr)
SSE4.1 and SHANI

sha1_ctx_mgr_submit (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1 or AVX or AVX2 or AVX512

sha1_ctx_mgr_submit_avx (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX

sha1_ctx_mgr_submit_avx2 (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX2

sha1_ctx_mgr_submit_avx512 (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX512

sha1_ctx_mgr_submit_avx512_ni (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX512 and SHANI

sha1_ctx_mgr_submit_sse (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1

sha1_ctx_mgr_submit_sse_ni (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1 and SHANI

sha1_opt (unsigned char *input, unsigned int *digest, int len)
SSE3

sha1_update (unsigned int *digest, unsigned char *input, size_t num_blocks)
SSE3

sha256_ctx_mgr_flush (SHA256_HASH_CTX_MGR *mgr)
SSE4.1 or AVX or AVX2

sha256_ctx_mgr_flush_avx (SHA256_HASH_CTX_MGR *mgr)
AVX

sha256_ctx_mgr_flush_avx2 (SHA256_HASH_CTX_MGR *mgr)
AVX2

sha256_ctx_mgr_flush_avx512 (SHA256_HASH_CTX_MGR *mgr)
AVX512

sha256_ctx_mgr_flush_avx512_ni (SHA256_HASH_CTX_MGR *mgr)
AVX512 and SHANI

sha256_ctx_mgr_flush_sse (SHA256_HASH_CTX_MGR *mgr)
SSE4.1

sha256_ctx_mgr_flush_sse_ni (SHA256_HASH_CTX_MGR *mgr)
SSE4.1 and SHANI

sha256_ctx_mgr_init (SHA256_HASH_CTX_MGR *mgr)
SSE4.1 or AVX or AVX2

sha256_ctx_mgr_init_avx (SHA256_HASH_CTX_MGR *mgr)
AVX

sha256_ctx_mgr_init_avx2 (SHA256_HASH_CTX_MGR *mgr)
AVX2

sha256_ctx_mgr_init_avx512 (SHA256_HASH_CTX_MGR *mgr)
AVX512

sha256_ctx_mgr_init_avx512_ni (SHA256_HASH_CTX_MGR *mgr)
AVX512 and SHANI

sha256_ctx_mgr_init_sse (SHA256_HASH_CTX_MGR *mgr)
SSE4.1

sha256_ctx_mgr_init_sse_ni (SHA256_HASH_CTX_MGR *mgr)
SSE4.1 and SHANI

sha256_ctx_mgr_submit (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1 or AVX or AVX2

sha256_ctx_mgr_submit_avx (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX

sha256_ctx_mgr_submit_avx2 (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX2

sha256_ctx_mgr_submit_avx512 (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX512

sha256_ctx_mgr_submit_avx512_ni (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX512 and SHANI

sha256_ctx_mgr_submit_sse (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1

sha256_ctx_mgr_submit_sse_ni (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1 and SHANI

sha512_ctx_mgr_flush (SHA512_HASH_CTX_MGR *mgr)
SSE4.1 or AVX or AVX2 or AVX512

sha512_ctx_mgr_flush_avx (SHA512_HASH_CTX_MGR *mgr)
AVX

sha512_ctx_mgr_flush_avx2 (SHA512_HASH_CTX_MGR *mgr)
AVX2

sha512_ctx_mgr_flush_avx512 (SHA512_HASH_CTX_MGR *mgr)
AVX512

sha512_ctx_mgr_flush_sb_sse4 (SHA512_HASH_CTX_MGR *mgr)
SSE4

sha512_ctx_mgr_flush_sse (SHA512_HASH_CTX_MGR *mgr)
SSE4.1

sha512_ctx_mgr_init (SHA512_HASH_CTX_MGR *mgr)
SSE4.1 or AVX or AVX2 or AVX512

sha512_ctx_mgr_init_avx (SHA512_HASH_CTX_MGR *mgr)
AVX

sha512_ctx_mgr_init_avx2 (SHA512_HASH_CTX_MGR *mgr)
AVX2

sha512_ctx_mgr_init_avx512 (SHA512_HASH_CTX_MGR *mgr)
AVX512

sha512_ctx_mgr_init_sb_sse4 (SHA512_HASH_CTX_MGR *mgr)
SSE4

sha512_ctx_mgr_init_sse (SHA512_HASH_CTX_MGR *mgr)
SSE4.1

sha512_ctx_mgr_submit (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1 or AVX or AVX2 or AVX512

sha512_ctx_mgr_submit_avx (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX

sha512_ctx_mgr_submit_avx2 (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX2

sha512_ctx_mgr_submit_avx512 (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
AVX512

sha512_ctx_mgr_submit_sb_sse4 (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4

sha512_ctx_mgr_submit_sse (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)
SSE4.1

XTS_AES_128_dec (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *ct, uint8_t *pt)
AES-NI

XTS_AES_128_dec_expanded_key (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *ct, uint8_t *pt)
AES-NI

XTS_AES_128_enc (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *pt, uint8_t *ct)
AES-NI

XTS_AES_128_enc_expanded_key (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *pt, uint8_t *ct)
AES-NI

XTS_AES_256_dec (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *ct, uint8_t *pt)
AES-NI

XTS_AES_256_dec_expanded_key (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *ct, uint8_t *pt)
AES-NI

XTS_AES_256_enc (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *pt, uint8_t *ct)
AES-NI

XTS_AES_256_enc_expanded_key (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *pt, uint8_t *ct)
AES-NI

CHAPTER 4
DATA STRUCTURE INDEX

4.1 Data Structures

Here are the data structures with brief descriptions:

CHAPTER 5
FILE INDEX

5.1 File List

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

CHAPTER 6
DATA STRUCTURE DOCUMENTATION

6.1 cbc_key_data Struct Reference

holds intermediate key data used in encryption/decryption

`#include <aes_cbc.h>`

6.1.1 Detailed Description

holds intermediate key data used in encryption/decryption

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

- aes_cbc.h

6.2 gcm_context_data Struct Reference

holds GCM operation context

`#include <aes_gcm.h>`

6.2.1 Detailed Description

holds GCM operation context

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

- aes_gcm.h

6.3 gcm_data Struct Reference

holds intermediate key data needed to improve performance

`#include <aes_gcm.h>`

6.3.1 Detailed Description

holds intermediate key data needed to improve performance

gcm_data hold internal key information used by gcm128 and gcm256.

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

- aes_gcm.h

6.4 gcm_key_data Struct Reference

holds intermediate key data needed to improve performance

`#include <aes_gcm.h>`

6.4.1 Detailed Description

holds intermediate key data needed to improve performance

gcm_key_data hold internal key information used by gcm128, gcm192 and gcm256.

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

- aes_gcm.h

6.5 MD5_HASH_CTX Struct Reference

Context layer - Holds info describing a single MD5 job for the multi-buffer CTX manager.

`#include <md5_mb.h>`

Data Fields

- HASH_CTX_STS status
  Context status flag.
- HASH_CTX_ERROR error
  Context error flag.
- uint32_t total_length
  Running counter of length processed for this CTX's job.
- const void * incoming_buffer
  pointer to data input buffer for this CTX's job
- uint32_t incoming_buffer_length
  length of buffer for this job in bytes.
- uint8_t partial_block_buffer [MD5_BLOCK_SIZE *2]
  CTX partial blocks.

- void * user_data

  pointer for user to keep any job-related data

6.5.1 Detailed Description

Context layer - Holds info describing a single MD5 job for the multi-buffer CTX manager.

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

- md5_mb.h

6.6 MD5_HASH_CTX_MGR Struct Reference

Context layer - Holds state for multi-buffer MD5 jobs.

`#include <md5_mb.h>`

6.6.1 Detailed Description

Context layer - Holds state for multi-buffer MD5 jobs.

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

- md5_mb.h

6.7 MD5_JOB Struct Reference

Scheduler layer - Holds info describing a single MD5 job for the multi-buffer manager.

`#include <md5_mb.h>`

Data Fields

- uint8_t * buffer

  pointer to data buffer for this job
- uint32_t len

  length of buffer for this job in blocks.
- JOB_STS status

  output job status
- void * user_data

  pointer for user's job-related data

6.7.1 Detailed Description

Scheduler layer - Holds info describing a single MD5 job for the multi-buffer manager.

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

- md5_mb.h

6.8 MD5_LANE_DATA Struct Reference

Scheduler layer - Lane data.

`#include <md5_mb.h>`

6.8.1 Detailed Description

Scheduler layer - Lane data.

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

- md5_mb.h

6.9 MD5_MB_ARGS_X32 Struct Reference

Scheduler layer - Holds arguments for submitted MD5 job.

`#include <md5_mb.h>`

6.9.1 Detailed Description

Scheduler layer - Holds arguments for submitted MD5 job.

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

- md5_mb.h

6.10 MD5_MB_JOB_MGR Struct Reference

Scheduler layer - Holds state for multi-buffer MD5 jobs.

`#include <md5_mb.h>`

Data Fields

- uint64_t unused_lanes [4]

  each byte or nibble is index (0...31 or 15) of unused lanes.

6.10.1 Detailed Description

Scheduler layer - Holds state for multi-buffer MD5 jobs.

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

- md5_mb.h

6.11 mh_sha1_ctx Struct Reference

Holds info describing a single mh_sha1.

`#include <mh_sha1.h>`

Data Fields

- uint32_t mh_sha1_digest [SHA1_DIGEST_WORDS]

  the digest of multi-hash SHA1
- uint64_t total_length

  Parameters for update feature, describe the lengths of input buffers in bytes.
- uint8_t partial_block_buffer [MH_SHA1_BLOCK_SIZE ∗2]

  Padding the tail of input data for SHA1.
- uint8_t mh_sha1_interim_digests [sizeof(uint32_t)∗SHA1_DIGEST_WORDS ∗HASH_SEGS]

  Storing the SHA1 interim digests of all 16 segments. Each time, it will be copied to stack for 64-byte alignment purpose.
- uint8_t frame_buffer [MH_SHA1_BLOCK_SIZE+AVX512_ALIGNED]

  Re-structure sha1 block data from different segments to fit big endian. Use AVX512_ALIGNED for 64-byte alignment purpose.

6.11.1 Detailed Description

Holds info describing a single mh_sha1.

It is better to use heap to allocate this data structure to avoid stack overflow.

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

- mh_sha1.h

6.12 mh_sha1_murmur3_x64_128_ctx Struct Reference

Holds info describing a single mh_sha1_murmur3_x64_128.

`#include <mh_sha1_murmur3_x64_128.h>`

Data Fields

- uint32_t mh_sha1_digest [SHA1_DIGEST_WORDS]

  the digest of multi-hash SHA1
- uint32_t murmur3_x64_128_digest [MURMUR3_x64_128_DIGEST_WORDS]

  the digest of murmur3_x64_128
- uint64_t total_length

  Parameters for update feature, describe the lengths of input buffers in bytes.
- uint8_t partial_block_buffer [MH_SHA1_BLOCK_SIZE ∗2]

  Padding the tail of input data for SHA1.
- uint8_t mh_sha1_interim_digests [sizeof(uint32_t)∗SHA1_DIGEST_WORDS ∗HASH_SEGS]

  Storing the SHA1 interim digests of all 16 segments. Each time, it will be copied to stack for 64-byte alignment purpose.
- uint8_t frame_buffer [MH_SHA1_BLOCK_SIZE+AVX512_ALIGNED]

  Re-structure sha1 block data from different segments to fit big endian. Use AVX512_ALIGNED for 64-byte alignment purpose.

6.12.1 Detailed Description

Holds info describing a single mh_sha1_murmur3_x64_128.

It is better to use heap to allocate this data structure to avoid stack overflow.

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

- mh_sha1_murmur3_x64_128.h

6.13 mh_sha256_ctx Struct Reference

Holds info describing a single mh_sha256.

`#include <mh_sha256.h>`

Data Fields

- uint32_t mh_sha256_digest [SHA256_DIGEST_WORDS]

  the digest of multi-hash SHA256

- uint64_t total_length

  Parameters for update feature, describe the lengths of input buffers in bytes.
- uint8_t partial_block_buffer [MH_SHA256_BLOCK_SIZE ∗2]

  Padding the tail of input data for SHA256.
- uint8_t mh_sha256_interim_digests [sizeof(uint32_t)∗SHA256_DIGEST_WORDS ∗HASH_SEGS]

  Storing the SHA256 interim digests of all 16 segments. Each time, it will be copied to stack for 64-byte alignment purpose.
- uint8_t frame_buffer [MH_SHA256_BLOCK_SIZE+AVX512_ALIGNED]

  Re-structure sha256 block data from different segments to fit big endian. Use AVX512_ALIGNED for 64-byte alignment purpose.

6.13.1 Detailed Description

Holds info describing a single mh_sha256.

It is better to use heap to allocate this data structure to avoid stack overflow.

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

- mh_sha256.h

6.14 rh_state1 Struct Reference

Context for rolling_hash1 functions.

```
#include <rolling_hashx.h>
```

6.14.1 Detailed Description

Context for rolling_hash1 functions.

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

- rolling_hashx.h

6.15 rh_state2 Struct Reference

Context for rolling_hash2 functions.

```
#include <rolling_hashx.h>
```

6.15.1 Detailed Description

Context for rolling_hash2 functions.

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

- rolling_hashx.h

6.16 SHA1_HASH_CTX Struct Reference

Context layer - Holds info describing a single SHA1 job for the multi-buffer CTX manager.

`#include <sha1_mb.h>`

Data Fields

- HASH_CTX_STS status
  Context status flag.
- HASH_CTX_ERROR error
  Context error flag.
- uint32_t total_length
  Running counter of length processed for this CTX's job.
- const void * incoming_buffer
  pointer to data input buffer for this CTX's job
- uint32_t incoming_buffer_length
  length of buffer for this job in bytes.
- uint8_t partial_block_buffer [SHA1_BLOCK_SIZE *2]
  CTX partial blocks.
- void * user_data
  pointer for user to keep any job-related data

6.16.1 Detailed Description

Context layer - Holds info describing a single SHA1 job for the multi-buffer CTX manager.

Examples:

sha1_multi_buffer_example.c.

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

- sha1_mb.h

6.17 SHA1_HASH_CTX_MGR Struct Reference

Context layer - Holds state for multi-buffer SHA1 jobs.

`#include <sha1_mb.h>`

6.17.1 Detailed Description

Context layer - Holds state for multi-buffer SHA1 jobs.

Examples:

sha1_multi_buffer_example.c.

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

- sha1_mb.h

6.18 SHA1_JOB Struct Reference

Scheduler layer - Holds info describing a single SHA1 job for the multi-buffer manager.

`#include <sha1_mb.h>`

Data Fields

- uint8_t * buffer
  pointer to data buffer for this job
- uint32_t len
  length of buffer for this job in blocks.
- JOB_STS status
  output job status
- void * user_data
  pointer for user's job-related data

6.18.1 Detailed Description

Scheduler layer - Holds info describing a single SHA1 job for the multi-buffer manager.

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

- sha1_mb.h

6.19 SHA1_LANE_DATA Struct Reference

Scheduler layer - Lane data.

`#include <sha1_mb.h>`

6.19.1 Detailed Description

Scheduler layer - Lane data.

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

- sha1_mb.h

6.20 SHA1_MB_ARGS_X16 Struct Reference

Scheduler layer - Holds arguments for submitted SHA1 job.

`#include <sha1_mb.h>`

6.20.1 Detailed Description

Scheduler layer - Holds arguments for submitted SHA1 job.

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

- sha1_mb.h

6.21 SHA1_MB_JOB_MGR Struct Reference

Scheduler layer - Holds state for multi-buffer SHA1 jobs.

`#include <sha1_mb.h>`

Data Fields

- uint64_t unused_lanes

  each nibble is index (0...3 or 0...7 or 0...15) of unused lanes, nibble 4 or 8 is set to F as a flag

6.21.1 Detailed Description

Scheduler layer - Holds state for multi-buffer SHA1 jobs.

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

- sha1_mb.h

6.22 SHA256_HASH_CTX Struct Reference

Context layer - Holds info describing a single SHA256 job for the multi-buffer CTX manager.

`#include <sha256_mb.h>`

Data Fields

- HASH_CTX_STS status
  Context status flag.
- HASH_CTX_ERROR error
  Context error flag.
- uint32_t total_length
  Running counter of length processed for this CTX's job.
- const void * incoming_buffer
  pointer to data input buffer for this CTX's job
- uint32_t incoming_buffer_length
  length of buffer for this job in bytes.
- uint8_t partial_block_buffer [SHA256_BLOCK_SIZE *2]
  CTX partial blocks.
- void * user_data
  pointer for user to keep any job-related data

6.22.1 Detailed Description

Context layer - Holds info describing a single SHA256 job for the multi-buffer CTX manager.

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

- sha256_mb.h

6.23 SHA256_HASH_CTX_MGR Struct Reference

Context layer - Holds state for multi-buffer SHA256 jobs.

`#include <sha256_mb.h>`

6.23.1 Detailed Description

Context layer - Holds state for multi-buffer SHA256 jobs.

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

- sha256_mb.h

6.24 SHA256_JOB Struct Reference

Scheduler layer - Holds info describing a single SHA256 job for the multi-buffer manager.

`#include <sha256_mb.h>`

Data Fields

- uint8_t * buffer
  pointer to data buffer for this job
- uint64_t len
  length of buffer for this job in blocks.
- JOB_STS status
  output job status
- void * user_data
  pointer for user's job-related data

6.24.1 Detailed Description

Scheduler layer - Holds info describing a single SHA256 job for the multi-buffer manager.

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

- sha256_mb.h

6.25 SHA256_LANE_DATA Struct Reference

Scheduler layer - Lane data.

`#include <sha256_mb.h>`

6.25.1 Detailed Description

Scheduler layer - Lane data.

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

- sha256_mb.h

6.26 SHA256_MB_ARGS_X16 Struct Reference

Scheduler layer - Holds arguments for submitted SHA256 job.

`#include <sha256_mb.h>`

6.26.1 Detailed Description

Scheduler layer - Holds arguments for submitted SHA256 job.

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

- sha256_mb.h

6.27 SHA256_MB_JOB_MGR Struct Reference

Scheduler layer - Holds state for multi-buffer SHA256 jobs.

`#include <sha256_mb.h>`

Data Fields

- uint64_t unused_lanes

  each nibble is index (0...3 or 0...7) of unused lanes, nibble 4 or 8 is set to F as a flag

6.27.1 Detailed Description

Scheduler layer - Holds state for multi-buffer SHA256 jobs.

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

- sha256_mb.h

6.28 SHA512_HASH_CTX Struct Reference

Context layer - Holds info describing a single SHA512 job for the multi-buffer CTX manager.

`#include <sha512_mb.h>`

Data Fields

- HASH_CTX_STS status
  Context status flag.
- HASH_CTX_ERROR error
  Context error flag.
- uint32_t total_length
  Running counter of length processed for this CTX's job.
- const void * incoming_buffer
  pointer to data input buffer for this CTX's job
- uint32_t incoming_buffer_length
  length of buffer for this job in bytes.
- uint8_t partial_block_buffer [SHA512_BLOCK_SIZE *2]
  CTX partial blocks.
- void * user_data
  pointer for user to keep any job-related data

6.28.1 Detailed Description

Context layer - Holds info describing a single SHA512 job for the multi-buffer CTX manager.

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

- sha512_mb.h

6.29 SHA512_HASH_CTX_MGR Struct Reference

Context layer - Holds state for multi-buffer SHA512 jobs.

`#include <sha512_mb.h>`

6.29.1 Detailed Description

Context layer - Holds state for multi-buffer SHA512 jobs.

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

- sha512_mb.h

6.30 SHA512_JOB Struct Reference

Scheduler layer - Holds info describing a single SHA512 job for the multi-buffer manager.

`#include <sha512_mb.h>`

Data Fields

- uint8_t * buffer
  pointer to data buffer for this job
- uint64_t len
  length of buffer for this job in blocks.
- JOB_STS status
  output job status
- void * user_data
  pointer for user's job-related data

6.30.1 Detailed Description

Scheduler layer - Holds info describing a single SHA512 job for the multi-buffer manager.

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

- sha512_mb.h

6.31 SHA512_LANE_DATA Struct Reference

Scheduler layer - Lane data.

`#include <sha512_mb.h>`

6.31.1 Detailed Description

Scheduler layer - Lane data.

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

- sha512_mb.h

6.32 SHA512_MB_ARGS_X8 Struct Reference

Scheduler layer - Holds arguments for submitted SHA512 job.

`#include <sha512_mb.h>`

6.32.1 Detailed Description

Scheduler layer - Holds arguments for submitted SHA512 job.

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

- sha512_mb.h

6.33 SHA512_MB_JOB_MGR Struct Reference

Scheduler layer - Holds state for multi-buffer SHA512 jobs.

`#include <sha512_mb.h>`

Data Fields

- uint64_t unused_lanes

  each byte is index (00, 01 or 00...03) of unused lanes, byte 2 or 4 is set to FF as a flag

6.33.1 Detailed Description

Scheduler layer - Holds state for multi-buffer SHA512 jobs.

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

- sha512_mb.h

CHAPTER 7
FILE DOCUMENTATION

7.1 aes_cbc.h File Reference

AES CBC encryption/decryption function prototypes.

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

Data Structures

- struct cbc_key_data

  holds intermediate key data used in encryption/decryption

Functions

- int aes_cbc_precomp (uint8_t *key, int key_size, struct cbc_key_data *keys_blk)

  CBC-AES key pre-computation done once for a key.
- void aes_cbc_dec_128 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)

  CBC-AES 128 bit key Decryption.
- void aes_cbc_dec_192 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)

  CBC-AES 192 bit key Decryption.
- void aes_cbc_dec_256 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)

  CBC-AES 256 bit key Decryption.
- int aes_cbc_enc_128 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)

  CBC-AES 128 bit key Encryption.
- int aes_cbc_enc_192 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)

  CBC-AES 192 bit key Encryption.
- int aes_cbc_enc_256 (void *in, uint8_t *IV, uint8_t *keys, void *out, uint64_t len_bytes)

  CBC-AES 256 bit key Encryption.

7.1.1 Detailed Description

AES CBC encryption/decryption function prototypes. ; References:

7.1.2 Function Documentation

7.1.2.1 void aes_cbc_dec_128 (void * *in*, uint8_t * *IV*, uint8_t * *keys*, void * *out*, uint64_t *len_bytes*)

CBC-AES 128 bit key Decryption.

Requires SSE4.1 and AESNI

arg 1: in: pointer to input (cipher text) arg 2: IV: pointer to IV, Must be 16 bytes aligned to a 16 byte boundary arg 3: keys: pointer to keys, Must be on a 16 byte boundary and length of key size * key rounds arg 4: OUT: pointer to output (plain text ... in-place allowed) arg 5: len_bytes: length in bytes (multiple of 16)Must be a multiple of 16 bytes

Parameters

IV	Must be 16 bytes aligned to a 16 byte boundary
keys	Must be on a 16 byte boundary and length of key size * key rounds or dec_keys of cbc_key_data

7.1.2.2 void aes_cbc_dec_192 (void * *in*, uint8_t * *IV*, uint8_t * *keys*, void * *out*, uint64_t *len_bytes*)

CBC-AES 192 bit key Decryption.

Requires SSE4.1 and AESNI

Must be a multiple of 16 bytes

Parameters

IV	Must be 16 bytes aligned to a 16 byte boundary
keys	Must be on a 16 byte boundary and length of key size * key rounds or dec_keys of cbc_key_data

7.1.2.3 void aes_cbc_dec_256 (void * *in*, uint8_t * *IV*, uint8_t * *keys*, void * *out*, uint64_t *len_bytes*)

CBC-AES 256 bit key Decryption.

Requires SSE4.1 and AESNI

Must be a multiple of 16 bytes

Parameters

IV	Must be 16 bytes aligned to a 16 byte boundary
keys	Must be on a 16 byte boundary and length of key size * key rounds or dec_keys of cbc_key_data

7.1.2.4 int aes_cbc_enc_128 (void * *in*, uint8_t * *IV*, uint8_t * *keys*, void * *out*, uint64_t *len_bytes*)

CBC-AES 128 bit key Encryption.

Requires SSE4.1 and AESNI

arg 1: in: pointer to input (plain text) arg 2: IV: pointer to IV, Must be 16 bytes aligned to a 16 byte boundary arg 3: keys: pointer to keys, Must be on a 16 byte boundary and length of key size * key rounds arg 4: OUT: pointer to output (cipher text ... in-place allowed) arg 5: len_bytes: length in bytes (multiple of 16)Must be a multiple of 16 bytes

Parameters

IV	Must be 16 bytes aligned to a 16 byte boundary
keys	Must be on a 16 byte boundary and length of key size * key rounds or enc_keys of cbc_key_data

7.1.2.5 int aes_cbc_enc_192 (void * *in*, uint8_t * *IV*, uint8_t * *keys*, void * *out*, uint64_t *len_bytes*)

CBC-AES 192 bit key Encryption.

Requires SSE4.1 and AESNI

Must be a multiple of 16 bytes

Parameters

IV	Must be 16 bytes aligned to a 16 byte boundary
keys	Must be on a 16 byte boundary and length of key size * key rounds or enc_keys of cbc_key_data

7.1.2.6 int aes_cbc_enc_256 (void * *in*, uint8_t * *IV*, uint8_t * *keys*, void * *out*, uint64_t *len_bytes*)

CBC-AES 256 bit key Encryption.

Requires SSE4.1 and AESNI

Must be a multiple of 16 bytes

Parameters

IV	Must be 16 bytes aligned to a 16 byte boundary
keys	Must be on a 16 byte boundary and length of key size * key rounds or enc_keys of cbc_key_data

7.1.2.7 int aes_cbc_precomp (uint8_t * *key*, int *key_size*, struct cbc_key_data * *keys_blk*)

CBC-AES key pre-computation done once for a key.

Requires SSE4.1 and AESNI

arg 1: in: pointer to key arg 2: OUT: pointer to a key expanded data

7.2 aes_gcm.h File Reference

AES GCM encryption/decryption function prototypes.

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

Data Structures

- struct gcm_data

  holds intermediate key data needed to improve performance
- struct gcm_key_data

  holds intermediate key data needed to improve performance
- struct gcm_context_data

  holds GCM operation context

Functions

- void aes_gcm_enc_128 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, uint8_t const *in, uint64_t len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)

  GCM-AES Encryption using 128 bit keys.
- void aes_gcm_enc_256 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, uint8_t const *in, uint64_t len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)

  GCM-AES Encryption using 256 bit keys.
- void aes_gcm_dec_128 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, uint8_t const *in, uint64_t len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)

  GCM-AES Decryption using 128 bit keys.
- void aes_gcm_dec_256 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, uint8_t const *in, uint64_t len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)

GCM-AES Decryption using 128 bit keys.

- void aes_gcm_init_128 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *iv, uint8_t const *aad, uint64_t aad_len)

  Start a AES-GCM Encryption message 128 bit key.
- void aes_gcm_init_256 (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *iv, uint8_t const *aad, uint64_t aad_len)

  Start a AES-GCM Encryption message 256 bit key.
- void aes_gcm_enc_128_update (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, const uint8_t *in, uint64_t len)

  Encrypt a block of a AES-128-GCM Encryption message.
- void aes_gcm_enc_256_update (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, const uint8_t *in, uint64_t len)

  Encrypt a block of a AES-256-GCM Encryption message.
- void aes_gcm_dec_128_update (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, const uint8_t *in, uint64_t len)

  Decrypt a block of a AES-128-GCM Encryption message.
- void aes_gcm_dec_256_update (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *out, const uint8_t *in, uint64_t len)

  Decrypt a block of a AES-256-GCM Encryption message.
- void aes_gcm_enc_128_finalize (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *auth_tag, uint64_t auth_tag_len)

  End encryption of a AES-128-GCM Encryption message.
- void aes_gcm_enc_256_finalize (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *auth_tag, uint64_t auth_tag_len)

  End encryption of a AES-256-GCM Encryption message.
- void aes_gcm_dec_128_finalize (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *auth_tag, uint64_t auth_tag_len)

  End decryption of a AES-128-GCM Encryption message.
- void aes_gcm_dec_256_finalize (const struct gcm_key_data *key_data, struct gcm_context_data *context_data, uint8_t *auth_tag, uint64_t auth_tag_len)

  End decryption of a AES-256-GCM Encryption message.
- void aes_gcm_pre_128 (const void *key, struct gcm_key_data *key_data)

  Pre-processes GCM key data 128 bit.
- void aes_gcm_pre_256 (const void *key, struct gcm_key_data *key_data)

  Pre-processes GCM key data 128 bit.
- void aesni_gcm128_enc (struct gcm_data *my_ctx_data, uint8_t *out, uint8_t const *in, uint64_t plaintext_len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)

  GCM-AES Encryption using 128 bit keys - older interface.
- void aesni_gcm128_dec (struct gcm_data *my_ctx_data, uint8_t *out, uint8_t const *in, uint64_t plaintext_len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)

  GCM-AES Decryption using 128 bit keys - older interface.
- void aesni_gcm128_init (struct gcm_data *my_ctx_data, uint8_t *iv, uint8_t const *aad, uint64_t aad_len)

start a AES-128-GCM Encryption message - older interface

- void aesni_gcm128_enc_update (struct gcm_data *my_ctx_data, uint8_t *out, const uint8_t *in, uint64_t plaintext_len)

  encrypt a block of a AES-128-GCM Encryption message - older interface
- void aesni_gcm128_dec_update (struct gcm_data *my_ctx_data, uint8_t *out, const uint8_t *in, uint64_t plaintext_len)

  decrypt a block of a AES-128-GCM Encryption message - older interface
- void aesni_gcm128_enc_finalize (struct gcm_data *my_ctx_data, uint8_t *auth_tag, uint64_t auth_tag_len)

  End encryption of a AES-128-GCM Encryption message - older interface.
- void aesni_gcm128_dec_finalize (struct gcm_data *my_ctx_data, uint8_t *auth_tag, uint64_t auth_tag_len)

  End decryption of a AES-128-GCM Encryption message - older interface.
- void aesni_gcm128_pre (uint8_t *key, struct gcm_data *gdata)

  pre-processes key data - older interface
- void aesni_gcm256_enc (struct gcm_data *my_ctx_data, uint8_t *out, uint8_t const *in, uint64_t plaintext_len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)

  GCM-AES Encryption using 256 bit keys.
- void aesni_gcm256_dec (struct gcm_data *my_ctx_data, uint8_t *out, uint8_t const *in, uint64_t plaintext_len, uint8_t *iv, uint8_t const *aad, uint64_t aad_len, uint8_t *auth_tag, uint64_t auth_tag_len)

  GCM-AES Decryption using 256 bit keys - older interface.
- void aesni_gcm256_init (struct gcm_data *my_ctx_data, uint8_t *iv, uint8_t const *aad, uint64_t aad_len)

  start a AES-256-GCM Encryption message - older interface
- void aesni_gcm256_enc_update (struct gcm_data *my_ctx_data, uint8_t *out, const uint8_t *in, uint64_t plaintext_len)

  encrypt a block of a AES-256-GCM Encryption message - older interface
- void aesni_gcm256_dec_update (struct gcm_data *my_ctx_data, uint8_t *out, const uint8_t *in, uint64_t plaintext_len)

  decrypt a block of a AES-256-GCM Encryption message - older interface
- void aesni_gcm256_enc_finalize (struct gcm_data *my_ctx_data, uint8_t *auth_tag, uint64_t auth_tag_len)

  End encryption of a AES-256-GCM Encryption message - older interface.
- void aesni_gcm256_dec_finalize (struct gcm_data *my_ctx_data, uint8_t *auth_tag, uint64_t auth_tag_len)

  End decryption of a AES-256-GCM Encryption message - older interface.
- void aesni_gcm256_pre (uint8_t *key, struct gcm_data *gdata)

  pre-processes key data - older interface

7.2.1 Detailed Description

AES GCM encryption/decryption function prototypes. At build time there is an option to use non-temporal loads and stores selected by defining the compile time option NT_LDST. The use of this option places the following restriction on the gcm encryption functions:

- The plaintext and cyphertext buffers must be aligned on a 16 byte boundary.

- When using the streaming API, all partial input buffers must be a multiple of 16 bytes long except for the last input buffer.
- In-place encryption/decryption is not recommended.

7.2.2 Function Documentation

7.2.2.1 void aes_gcm_dec_128 (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *out*, uint8_t const * *in*, uint64_t *len*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

GCM-AES Decryption using 128 bit keys.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
out	Plaintext output. Decrypt in-place is allowed
in	Ciphertext input
len	Length of data in Bytes for decryption
iv	iv pointer to 12 byte IV structure. Internally, library concates 0x00000001 value to it.
aad	Additional Authentication Data (AAD)
aad_len	Length of AAD
auth_tag	Authenticated Tag output
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8

7.2.2.2 void aes_gcm_dec_128_finalize (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

End decryption of a AES-128-GCM Encryption message.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
auth_tag	Authenticated Tag output
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8

7.2.2.3 void aes_gcm_dec_128_update (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *out*, const uint8_t * *in*, uint64_t *len*)

Decrypt a block of a AES-128-GCM Encryption message.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
out	Plaintext output. Decrypt in-place is allowed.
in	Ciphertext input
len	Length of data in Bytes for decryption

7.2.2.4 void aes_gcm_dec_256 (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *out*, uint8_t const * *in*, uint64_t *len*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

GCM-AES Decryption using 128 bit keys.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
out	Plaintext output. Decrypt in-place is allowed
in	Ciphertext input
len	Length of data in Bytes for decryption
iv	iv pointer to 12 byte IV structure. Internally, library concates 0x00000001 value to it.
aad	Additional Authentication Data (AAD)
aad_len	Length of AAD
auth_tag	Authenticated Tag output
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8

7.2.2.5 void aes_gcm_dec_256_finalize (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

End decryption of a AES-256-GCM Encryption message.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
auth_tag	Authenticated Tag output
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8

7.2.2.6 void aes_gcm_dec_256_update (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *out*, const uint8_t * *in*, uint64_t *len*)

Decrypt a block of a AES-256-GCM Encryption message.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
out	Plaintext output. Decrypt in-place is allowed.
in	Ciphertext input
len	Length of data in Bytes for decryption

7.2.2.7 void aes_gcm_enc_128 (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *out*, uint8_t const * *in*, uint64_t *len*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

GCM-AES Encryption using 128 bit keys.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
out	Ciphertext output. Encrypt in-place is allowed
in	Plaintext input
len	Length of data in Bytes for encryption
iv	iv pointer to 12 byte IV structure. Internally, library concates 0x00000001 value to it.

aad	Additional Authentication Data (AAD)
aad_len	Length of AAD
auth_tag	Authenticated Tag output
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8

7.2.2.8 void aes_gcm_enc_128_finalize (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

End encryption of a AES-128-GCM Encryption message.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
auth_tag	Authenticated Tag output
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8

7.2.2.9 void aes_gcm_enc_128_update (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *out*, const uint8_t * *in*, uint64_t *len*)

Encrypt a block of a AES-128-GCM Encryption message.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
out	Ciphertext output. Encrypt in-place is allowed.
in	Plaintext input
len	Length of data in Bytes for encryption

7.2.2.10 void aes_gcm_enc_256 (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *out*, uint8_t const * *in*, uint64_t *len*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

GCM-AES Encryption using 256 bit keys.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
out	Ciphertext output. Encrypt in-place is allowed
in	Plaintext input
len	Length of data in Bytes for encryption
iv	iv pointer to 12 byte IV structure. Internally, library concates 0x00000001 value to it.
aad	Additional Authentication Data (AAD)
aad_len	Length of AAD
auth_tag	Authenticated Tag output
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8

7.2.2.11 void aes_gcm_enc_256_finalize (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

End encryption of a AES-256-GCM Encryption message.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
auth_tag	Authenticated Tag output
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8

7.2.2.12 void aes_gcm_enc_256_update (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *out*, const uint8_t * *in*, uint64_t *len*)

Encrypt a block of a AES-256-GCM Encryption message.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
out	Ciphertext output. Encrypt in-place is allowed.
in	Plaintext input
len	Length of data in Bytes for encryption

7.2.2.13 void aes_gcm_init_128 (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*)

Start a AES-GCM Encryption message 128 bit key.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
iv	Pointer to 12 byte IV structure Internally, library concates 0x00000001 value to it
aad	Additional Authentication Data (AAD)
aad_len	Length of AAD

7.2.2.14 void aes_gcm_init_256 (const struct gcm_key_data * *key_data*, struct gcm_context_data * *context_data*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*)

Start a AES-GCM Encryption message 256 bit key.

Requires SSE4.1 and AESNI

Parameters

key_data	GCM expanded key data
context_data	GCM operation context data
iv	Pointer to 12 byte IV structure Internally, library concates 0x00000001 value to it
aad	Additional Authentication Data (AAD)
aad_len	Length of AAD

7.2.2.15 void aes_gcm_pre_128 (const void * *key*, struct gcm_key_data * *key_data*)

Pre-processes GCM key data 128 bit.

Prefills the gcm key data with key values for each round and the initial sub hash key for tag encoding

Requires SSE4.1 and AESNI

Parameters

key	Pointer to key data
key_data	GCM expanded key data

7.2.2.16 void aes_gcm_pre_256 (const void * *key*, struct gcm_key_data * *key_data*)

Pre-processes GCM key data 128 bit.

Prefills the gcm key data with key values for each round and the initial sub hash key for tag encoding

Requires SSE4.1 and AESNI

Parameters

key	Pointer to key data
key_data	GCM expanded key data

7.2.2.17 void aesni_gcm128_dec (struct gcm_data * *my_ctx_data*, uint8_t * *out*, uint8_t const * *in*, uint64_t *plaintext_len*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

GCM-AES Decryption using 128 bit keys - older interface.

Requires SSE4.1 and AESNI

Parameters

out	Plaintext output. Decrypt in-place is allowed.
in	Ciphertext input
plaintext_len	Length of data in Bytes for encryption.
iv	Pre-counter block j0: 4 byte salt (from Security Association) concatenated with 8 byte Initialisation Vector (from IPSec ESP Payload) concatenated with 0x00000001. 16-byte pointer.
aad	Additional Authentication Data (AAD).
aad_len	Length of AAD.

auth_tag	Authenticated Tag output.
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8.

7.2.2.18 void aesni_gcm128_dec_finalize (struct gcm_data * *my_ctx_data*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

End decryption of a AES-128-GCM Encryption message - older interface.

Requires SSE4.1 and AESNI

Parameters

auth_tag	Authenticated Tag output.
auth_tag_len	Authenticated Tag Length in bytes. Valid values are 16 (most likely), 12 or 8.

7.2.2.19 void aesni_gcm128_dec_update (struct gcm_data * *my_ctx_data*, uint8_t * *out*, const uint8_t * *in*, uint64_t *plaintext_len*)

decrypt a block of a AES-128-GCM Encryption message - older interface

Requires SSE4.1 and AESNI

Parameters

out	Ciphertext output. Encrypt in-place is allowed.
in	Plaintext input
plaintext_len	Length of data in Bytes for encryption.

7.2.2.20 void aesni_gcm128_enc (struct gcm_data * *my_ctx_data*, uint8_t * *out*, uint8_t const * *in*, uint64_t *plaintext_len*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

GCM-AES Encryption using 128 bit keys - older interface.

Requires SSE4.1 and AESNI

Parameters

out	Ciphertext output. Encrypt in-place is allowed.
in	Plaintext input
plaintext_len	Length of data in Bytes for encryption.
iv	Pre-counter block j0: 4 byte salt (from Security Association) concatenated with 8 byte Initialization Vector (from IPSec ESP Payload) concatenated with 0x00000001. 16-byte pointer.
aad	Additional Authentication Data (AAD).
aad_len	Length of AAD.
auth_tag	Authenticated Tag output.
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8.

7.2.2.21 void aesni_gcm128_enc_finalize (struct gcm_data * *my_ctx_data*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

End encryption of a AES-128-GCM Encryption message - older interface.

Requires SSE4.1 and AESNI

Parameters

auth_tag	Authenticated Tag output.
auth_tag_len	Authenticated Tag Length in bytes. Valid values are 16 (most likely), 12 or 8.

7.2.2.22 void aesni_gcm128_enc_update (struct gcm_data * *my_ctx_data*, uint8_t * *out*, const uint8_t * *in*, uint64_t *plaintext_len*)

encrypt a block of a AES-128-GCM Encryption message - older interface

Requires SSE4.1 and AESNI

Parameters

out	Ciphertext output. Encrypt in-place is allowed.
in	Plaintext input
plaintext_len	Length of data in Bytes for encryption.

7.2.2.23 void aesni_gcm128_init (struct gcm_data * *my_ctx_data*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*)

start a AES-128-GCM Encryption message - older interface

Requires SSE4.1 and AESNI

Parameters

iv	Pre-counter block j0: 4 byte salt (from Security Association) concatenated with 8 byte Initialization Vector (from IPSec ESP Payload) concatenated with 0x00000001. 16-byte pointer.
aad	Additional Authentication Data (AAD).
aad_len	Length of AAD.

7.2.2.24 void aesni_gcm128_pre (uint8_t * *key*, struct gcm_data * *gdata*)

pre-processes key data - older interface

Prefills the gcm data with key values for each round and the initial sub hash key for tag encoding

7.2.2.25 void aesni_gcm256_dec (struct gcm_data * *my_ctx_data*, uint8_t * *out*, uint8_t const * *in*, uint64_t *plaintext_len*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

GCM-AES Decryption using 256 bit keys - older interface.

Requires SSE4.1 and AESNI

Parameters

out	Plaintext output. Decrypt in-place is allowed.
in	Ciphertext input
plaintext_len	Length of data in Bytes for encryption.
iv	Pre-counter block j0: 4 byte salt (from Security Association) concatenated with 8 byte Initialisation Vector (from IPSec ESP Payload) concatenated with 0x00000001. 16-byte pointer.
aad	Additional Authentication Data (AAD).
aad_len	Length of AAD.
auth_tag	Authenticated Tag output.
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8.

7.2.2.26 void aesni_gcm256_dec_finalize (struct gcm_data * *my_ctx_data*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

End decryption of a AES-256-GCM Encryption message - older interface.

Requires SSE4.1 and AESNI

Parameters

auth_tag	Authenticated Tag output.
auth_tag_len	Authenticated Tag Length in bytes. Valid values are 16 (most likely), 12 or 8.

7.2.2.27 void aesni_gcm256_dec_update (struct gcm_data * *my_ctx_data*, uint8_t * *out*, const uint8_t * *in*, uint64_t *plaintext_len*)

decrypt a block of a AES-256-GCM Encryption message - older interface

Requires SSE4.1 and AESNI

Parameters

out	Ciphertext output. Encrypt in-place is allowed.
in	Plaintext input
plaintext_len	Length of data in Bytes for encryption.

7.2.2.28 void aesni_gcm256_enc (struct gcm_data * *my_ctx_data*, uint8_t * *out*, uint8_t const * *in*, uint64_t *plaintext_len*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

GCM-AES Encryption using 256 bit keys.

Requires SSE4.1 and AESNI

Parameters

out	Ciphertext output. Encrypt in-place is allowed.
in	Plaintext input
plaintext_len	Length of data in Bytes for encryption.
iv	Pre-counter block j0: 4 byte salt (from Security Association) concatenated with 8 byte Initialization Vector (from IPSec ESP Payload) concatenated with 0x00000001. 16-byte pointer.
aad	Additional Authentication Data (AAD).
aad_len	Length of AAD.
auth_tag	Authenticated Tag output.
auth_tag_len	Authenticated Tag Length in bytes (must be a multiple of 4 bytes). Valid values are 16 (most likely), 12 or 8.

7.2.2.29 void aesni_gcm256_enc_finalize (struct gcm_data * *my_ctx_data*, uint8_t * *auth_tag*, uint64_t *auth_tag_len*)

End encryption of a AES-256-GCM Encryption message - older interface.

Requires SSE4.1 and AESNI

Parameters

auth_tag	Authenticated Tag output.
auth_tag_len	Authenticated Tag Length in bytes. Valid values are 16 (most likely), 12 or 8.

7.2.2.30 void aesni_gcm256_enc_update (struct gcm_data * *my_ctx_data*, uint8_t * *out*, const uint8_t * *in*, uint64_t *plaintext_len*)

encrypt a block of a AES-256-GCM Encryption message - older interface

Requires SSE4.1 and AESNI

Parameters

out	Ciphertext output. Encrypt in-place is allowed.
in	Plaintext input
plaintext_len	Length of data in Bytes for encryption.

7.2.2.31 void aesni_gcm256_init (struct gcm_data * *my_ctx_data*, uint8_t * *iv*, uint8_t const * *aad*, uint64_t *aad_len*)

start a AES-256-GCM Encryption message - older interface

Requires SSE4.1 and AESNI

Parameters

iv	Pre-counter block j0: 4 byte salt (from Security Association) concatenated with 8 byte Initialization Vector (from IPSec ESP Payload) concatenated with 0x00000001. 16-byte pointer.
aad	Additional Authentication Data (AAD).
aad_len	Length of AAD.

7.2.2.32 void aesni_gcm256_pre (uint8_t * *key*, struct gcm_data * *gdata*)

pre-processes key data - older interface

Prefills the gcm data with key values for each round and the initial sub hash key for tag encoding

7.3 aes_keyexp.h File Reference

AES key expansion functions.

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

Functions

- void aes_keyexp_128 (uint8_t *key, uint8_t *exp_key_enc, uint8_t *exp_key_dec)
  AES key expansion 128 bit.
- void aes_keyexp_192 (uint8_t *key, uint8_t *exp_key_enc, uint8_t *exp_key_dec)
  AES key expansion 192 bit.
- void aes_keyexp_256 (uint8_t *key, uint8_t *exp_key_enc, uint8_t *exp_key_dec)
  AES key expansion 256 bit.

7.3.1 Detailed Description

AES key expansion functions. This defines the interface to key expansion functions.

7.3.2 Function Documentation

7.3.2.1 void aes_keyexp_128 (uint8_t * *key*, uint8_t * *exp_key_enc*, uint8_t * *exp_key_dec*)

AES key expansion 128 bit.

Requires SSE4.1

Parameters

key	input key for AES-128, 16 bytes
exp_key_enc	expanded encryption keys, 16*11 bytes
exp_key_dec	expanded decryption keys, 16*11 bytes

7.3.2.2 void aes_keyexp_192 (uint8_t * *key*, uint8_t * *exp_key_enc*, uint8_t * *exp_key_dec*)

AES key expansion 192 bit.

Requires SSE4.1

Parameters

key	input key for AES-192, 16∗1.5 bytes
exp_key_enc	expanded encryption keys, 16∗13 bytes
exp_key_dec	expanded decryption keys, 16∗13 bytes

7.3.2.3 void aes_keyexp_256 (uint8_t ∗ *key*, uint8_t ∗ *exp_key_enc*, uint8_t ∗ *exp_key_dec*)

AES key expansion 256 bit.

Requires SSE4.1

Parameters

key	input key for AES-256, 16∗2 bytes
exp_key_enc	expanded encryption keys, 16∗15 bytes
exp_key_dec	expanded decryption keys, 16∗15 bytes

7.4 aes_xts.h File Reference

AES XTS encryption function prototypes.

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

Functions

- void XTS_AES_128_enc (uint8_t ∗k2, uint8_t ∗k1, uint8_t ∗TW_initial, uint64_t N, const uint8_t ∗pt, uint8_t ∗ct)

  XTS-AES-128 Encryption.
- void XTS_AES_128_enc_expanded_key (uint8_t ∗k2, uint8_t ∗k1, uint8_t ∗TW_initial, uint64_t N, const uint8_t ∗pt, uint8_t ∗ct)

  XTS-AES-128 Encryption with pre-expanded keys.
- void XTS_AES_128_dec (uint8_t ∗k2, uint8_t ∗k1, uint8_t ∗TW_initial, uint64_t N, const uint8_t ∗ct, uint8_t ∗pt)

  XTS-AES-128 Decryption.
- void XTS_AES_128_dec_expanded_key (uint8_t ∗k2, uint8_t ∗k1, uint8_t ∗TW_initial, uint64_t N, const uint8_t ∗ct, uint8_t ∗pt)

  XTS-AES-128 Decryption with pre-expanded keys.
- void XTS_AES_256_enc (uint8_t ∗k2, uint8_t ∗k1, uint8_t ∗TW_initial, uint64_t N, const uint8_t ∗pt, uint8_t ∗ct)

XTS-AES-256 Encryption.

- void XTS_AES_256_enc_expanded_key (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *pt, uint8_t *ct)

  XTS-AES-256 Encryption with pre-expanded keys.
- void XTS_AES_256_dec (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *ct, uint8_t *pt)

  XTS-AES-256 Decryption.
- void XTS_AES_256_dec_expanded_key (uint8_t *k2, uint8_t *k1, uint8_t *TW_initial, uint64_t N, const uint8_t *ct, uint8_t *pt)

  XTS-AES-256 Decryption with pre-expanded keys.

7.4.1 Detailed Description

AES XTS encryption function prototypes. This defines the interface to optimized AES XTS functions

Pre-expanded keys

For key encryption, pre-expanded keys are stored in the order that they will be used. As an example, if Key[0] is the 128-bit initial key used for an AES-128 encryption, the rest of the keys are stored as follows:

- Key[0] : Initial encryption key
- Key[1] : Round 1 encryption key
- Key[2] : Round 2 encryption key
- ...
- Key[10] : Round 10 encryption key

For decryption, the order of keys is reversed. However, we apply the necessary aesimc instructions before storing the expanded keys. For the same key used above, the pre-expanded keys will be stored as follows:

- Key[0] : Round 10 encryption key
- Key[1] : aesimc(Round 9 encryption key)
- Key[2] : aesimc(Round 8 encryption key)
- ...
- Key[9] : aesimc(Round 1 encryption key)
- Key[10] : Initial encryption key

Note: The expanded key decryption requires a decryption key only for the block decryption step. The tweak step in the expanded key decryption requires the same expanded encryption key that is used in the expanded key encryption.

Input and Output Buffers

The input and output buffers can be overlapping as long as the output buffer pointer is not less than the input buffer pointer. If the two pointers are the same, then encryption/decryption will occur in-place.

Data Length

- The functions support data length of any bytes greater than or equal to 16 bytes.
- Data length is a 64-bit value, which makes the largest possible data length 2^64 - 1 bytes.
- For data lengths from 0 to 15 bytes, the functions return without any error codes, without reading or writing any data.
- The functions only support byte lengths, not bits.

Initial Tweak

The functions accept a 128-bit initial tweak value. The user is responsible for padding the initial tweak value to this length.

Data Alignment

The input and output buffers, keys, pre-expanded keys and initial tweak value are not required to be aligned to 16 bytes, any alignment works.

7.4.2 Function Documentation

7.4.2.1 void XTS_AES_128_dec (uint8_t * *k2*, uint8_t * *k1*, uint8_t * *TW_initial*, uint64_t *N*, const uint8_t * *ct*, uint8_t * *pt*)

XTS-AES-128 Decryption.

Requires AES-NI

Parameters

k2	key used for tweaking, 16 bytes
k1	key used for decryption of tweaked ciphertext, 16 bytes
TW_initial	initial tweak value, 16 bytes
N	sector size, in bytes
ct	ciphertext sector input data
pt	plaintext sector output data

7.4.2.2 void XTS_AES_128_dec_expanded_key (uint8_t * *k2*, uint8_t * *k1*, uint8_t * *TW_initial*, uint64_t *N*, const uint8_t * *ct*, uint8_t * *pt*)

XTS-AES-128 Decryption with pre-expanded keys.

Requires AES-NI

Parameters

k2	expanded key used for tweaking, 16∗11 bytes - encryption key is used
k1	expanded decryption key used for decryption of tweaked ciphertext, 16∗11 bytes
TW_initial	initial tweak value, 16 bytes
N	sector size, in bytes
ct	ciphertext sector input data
pt	plaintext sector output data

7.4.2.3 void XTS_AES_128_enc (uint8_t * *k2*, uint8_t * *k1*, uint8_t * *TW_initial*, uint64_t *N*, const uint8_t * *pt*, uint8_t * *ct*)

XTS-AES-128 Encryption.

Requires AES-NI

Parameters

k2	key used for tweaking, 16 bytes
k1	key used for encryption of tweaked plaintext, 16 bytes
TW_initial	initial tweak value, 16 bytes
N	sector size, in bytes
pt	plaintext sector input data
ct	ciphertext sector output data

7.4.2.4 void XTS_AES_128_enc_expanded_key (uint8_t * *k2*, uint8_t * *k1*, uint8_t * *TW_initial*, uint64_t *N*, const uint8_t * *pt*, uint8_t * *ct*)

XTS-AES-128 Encryption with pre-expanded keys.

Requires AES-NI

Parameters

k2	expanded key used for tweaking, 16∗11 bytes
k1	expanded key used for encryption of tweaked plaintext, 16∗11 bytes
TW_initial	initial tweak value, 16 bytes
N	sector size, in bytes
pt	plaintext sector input data
ct	ciphertext sector output data

7.4.2.5 void XTS_AES_256_dec (uint8_t ∗ *k2*, uint8_t ∗ *k1*, uint8_t ∗ *TW_initial*, uint64_t *N*, const uint8_t ∗ *ct*, uint8_t ∗ *pt*)

XTS-AES-256 Decryption.

Requires AES-NI

Parameters

k2	key used for tweaking, 16∗2 bytes
k1	key used for decryption of tweaked ciphertext, 16∗2 bytes
TW_initial	initial tweak value, 16 bytes
N	sector size, in bytes
ct	ciphertext sector input data
pt	plaintext sector output data

7.4.2.6 void XTS_AES_256_dec_expanded_key (uint8_t ∗ *k2*, uint8_t ∗ *k1*, uint8_t ∗ *TW_initial*, uint64_t *N*, const uint8_t ∗ *ct*, uint8_t ∗ *pt*)

XTS-AES-256 Decryption with pre-expanded keys.

Requires AES-NI

Parameters

k2	expanded key used for tweaking, 16∗15 bytes - encryption key is used
k1	expanded decryption key used for decryption of tweaked ciphertext, 16∗15 bytes
TW_initial	initial tweak value, 16 bytes
N	sector size, in bytes
ct	ciphertext sector input data
pt	plaintext sector output data

7.4.2.7 void XTS_AES_256_enc (uint8_t * *k2*, uint8_t * *k1*, uint8_t * *TW_initial*, uint64_t *N*, const uint8_t * *pt*, uint8_t * *ct*)

XTS-AES-256 Encryption.

Requires AES-NI

Parameters

k2	key used for tweaking, 16*2 bytes
k1	key used for encryption of tweaked plaintext, 16*2 bytes
TW_initial	initial tweak value, 16 bytes
N	sector size, in bytes
pt	plaintext sector input data
ct	ciphertext sector output data

7.4.2.8 void XTS_AES_256_enc_expanded_key (uint8_t * *k2*, uint8_t * *k1*, uint8_t * *TW_initial*, uint64_t *N*, const uint8_t * *pt*, uint8_t * *ct*)

XTS-AES-256 Encryption with pre-expanded keys.

Requires AES-NI

Parameters

k2	expanded key used for tweaking, 16*15 bytes
k1	expanded key used for encryption of tweaked plaintext, 16*15 bytes
TW_initial	initial tweak value, 16 bytes
N	sector size, in bytes
pt	plaintext sector input data
ct	ciphertext sector output data

7.5 md5_mb.h File Reference

Multi-buffer CTX API MD5 function prototypes and structures.

```
#include <stdint.h>
#include "multi_buffer.h"
#include "types.h"
```

Data Structures

- struct MD5_JOB

  Scheduler layer - Holds info describing a single MD5 job for the multi-buffer manager.
- struct MD5_MB_ARGS_X32

  Scheduler layer - Holds arguments for submitted MD5 job.
- struct MD5_LANE_DATA

  Scheduler layer - Lane data.
- struct MD5_MB_JOB_MGR

  Scheduler layer - Holds state for multi-buffer MD5 jobs.
- struct MD5_HASH_CTX_MGR

  Context layer - Holds state for multi-buffer MD5 jobs.
- struct MD5_HASH_CTX

  Context layer - Holds info describing a single MD5 job for the multi-buffer CTX manager.

Functions

- void md5_ctx_mgr_init_sse (MD5_HASH_CTX_MGR *mgr)

  Initialize the context level MD5 multi-buffer manager structure.
- MD5_HASH_CTX * md5_ctx_mgr_submit_sse (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new MD5 job to the context level multi-buffer manager.
- MD5_HASH_CTX * md5_ctx_mgr_flush_sse (MD5_HASH_CTX_MGR *mgr)

  Finish all submitted MD5 jobs and return when complete.
- void md5_ctx_mgr_init_avx (MD5_HASH_CTX_MGR *mgr)

  Initialize the MD5 multi-buffer manager structure.
- MD5_HASH_CTX * md5_ctx_mgr_submit_avx (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new MD5 job to the multi-buffer manager.
- MD5_HASH_CTX * md5_ctx_mgr_flush_avx (MD5_HASH_CTX_MGR *mgr)

  Finish all submitted MD5 jobs and return when complete.
- void md5_ctx_mgr_init_avx2 (MD5_HASH_CTX_MGR *mgr)

  Initialize the MD5 multi-buffer manager structure.
- MD5_HASH_CTX * md5_ctx_mgr_submit_avx2 (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new MD5 job to the multi-buffer manager.
- MD5_HASH_CTX * md5_ctx_mgr_flush_avx2 (MD5_HASH_CTX_MGR *mgr)

  Finish all submitted MD5 jobs and return when complete.
- void md5_ctx_mgr_init_avx512 (MD5_HASH_CTX_MGR *mgr)

  Initialize the MD5 multi-buffer manager structure.

- MD5_HASH_CTX * md5_ctx_mgr_submit_avx512 (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new MD5 job to the multi-buffer manager.
- MD5_HASH_CTX * md5_ctx_mgr_flush_avx512 (MD5_HASH_CTX_MGR *mgr)

  Finish all submitted MD5 jobs and return when complete.
- void md5_ctx_mgr_init (MD5_HASH_CTX_MGR *mgr)

  Initialize the MD5 multi-buffer manager structure.
- MD5_HASH_CTX * md5_ctx_mgr_submit (MD5_HASH_CTX_MGR *mgr, MD5_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new MD5 job to the multi-buffer manager.
- MD5_HASH_CTX * md5_ctx_mgr_flush (MD5_HASH_CTX_MGR *mgr)

  Finish all submitted MD5 jobs and return when complete.

7.5.1 Detailed Description

Multi-buffer CTX API MD5 function prototypes and structures. Interface for multi-buffer MD5 functions

Multi-buffer MD5 Entire or First-Update..Update-Last

The interface to this multi-buffer hashing code is carried out through the context-level (CTX) init, submit and flush functions and the MD5_HASH_CTX_MGR and MD5_HASH_CTX objects. Numerous MD5_HASH_CTX objects may be instantiated by the application for use with a single MD5_HASH_CTX_MGR.

The CTX interface functions carry out the initialization and padding of the jobs entered by the user and add them to the multi-buffer manager. The lower level "scheduler" layer then processes the jobs in an out-of-order manner. The scheduler layer functions are internal and are not intended to be invoked directly. Jobs can be submitted to a CTX as a complete buffer to be hashed, using the HASH_ENTIRE flag, or as partial jobs which can be started using the HASH_FIRST flag, and later resumed or finished using the HASH_UPDATE and HASH_LAST flags respectively.

Note: The submit function does not require data buffers to be block sized.

The MD5 CTX interface functions are available for 4 architectures: SSE, AVX, AVX2 and AVX512. In addition, a multibinary interface is provided, which selects the appropriate architecture-specific function at runtime.

Usage: The application creates a MD5_HASH_CTX_MGR object and initializes it with a call to md5_ctx_mgr_-init*() function, where henceforth "*" stands for the relevant suffix for each architecture; _sse, _avx, _avx2, _avx512 (or no suffix for the multibinary version). The MD5_HASH_CTX_MGR object will be used to schedule processor resources, with up to 8 MD5_HASH_CTX objects (or 16 in AVX2 case, 32 in AVX512 case) being processed at a time.

Each MD5_HASH_CTX must be initialized before first use by the hash_ctx_init macro defined in multi_buffer.h. After initialization, the application may begin computing a hash by giving the MD5_HASH_CTX to a MD5_HASH_-CTX_MGR using the submit functions md5_ctx_mgr_submit*() with the HASH_FIRST flag set. When the MD5_H-ASH_CTX is returned to the application (via this or a later call to md5_ctx_mgr_submit*() or md5_ctx_mgr_flush*()), the application can then re-submit it with another call to md5_ctx_mgr_submit*(), but without the HASH_FIRST flag set.

Ideally, on the last buffer for that hash, md5_ctx_mgr_submit_sse is called with HASH_LAST, although it is also possible to submit the hash with HASH_LAST and a zero length if necessary. When a MD5_HASH_CTX is returned

after having been submitted with HASH_LAST, it will contain a valid hash. The MD5_HASH_CTX can be reused immediately by submitting with HASH_FIRST.

For example, you would submit hashes with the following flags for the following numbers of buffers:

- one buffer: HASH_FIRST | HASH_LAST (or, equivalently, HASH_ENTIRE)
- two buffers: HASH_FIRST, HASH_LAST
- three buffers: HASH_FIRST, HASH_UPDATE, HASH_LAST etc.

The order in which MD5_CTX objects are returned is in general different from the order in which they are submitted.

A few possible error conditions exist:

- Submitting flags other than the allowed entire/first/update/last values
- Submitting a context that is currently being managed by a MD5_HASH_CTX_MGR.
- Submitting a context after HASH_LAST is used but before HASH_FIRST is set.

These error conditions are reported by returning the MD5_HASH_CTX immediately after a submit with its error member set to a non-zero error code (defined in multi_buffer.h). No changes are made to the MD5_HASH_CTX_MGR in the case of an error; no processing is done for other hashes.

7.5.2 Function Documentation

7.5.2.1 MD5_HASH_CTX* md5_ctx_mgr_flush (MD5_HASH_CTX_MGR * *mgr*)

Finish all submitted MD5 jobs and return when complete.

Requires SSE4.1 or AVX or AVX2 or AVX512

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.5.2.2 MD5_HASH_CTX* md5_ctx_mgr_flush_avx (MD5_HASH_CTX_MGR * *mgr*)

Finish all submitted MD5 jobs and return when complete.

Requires AVX

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.5.2.3 MD5_HASH_CTX* md5_ctx_mgr_flush_avx2 (MD5_HASH_CTX_MGR * *mgr*)

Finish all submitted MD5 jobs and return when complete.

Requires AVX2

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.5.2.4 MD5_HASH_CTX* md5_ctx_mgr_flush_avx512 (MD5_HASH_CTX_MGR * *mgr*)

Finish all submitted MD5 jobs and return when complete.

Requires AVX512

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.5.2.5 MD5_HASH_CTX* md5_ctx_mgr_flush_sse (MD5_HASH_CTX_MGR * *mgr*)

Finish all submitted MD5 jobs and return when complete.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.5.2.6 void md5_ctx_mgr_init (MD5_HASH_CTX_MGR * *mgr*)

Initialize the MD5 multi-buffer manager structure.

Requires SSE4.1 or AVX or AVX2 or AVX512

Parameters

mgr	Structure holding context level state info

Returns

void

7.5.2.7 void md5_ctx_mgr_init_avx (MD5_HASH_CTX_MGR * *mgr*)

Initialize the MD5 multi-buffer manager structure.

Requires AVX

Parameters

mgr	Structure holding context level state info

Returns

void

7.5.2.8 void md5_ctx_mgr_init_avx2 (MD5_HASH_CTX_MGR * *mgr*)

Initialize the MD5 multi-buffer manager structure.

Requires AVX2

Parameters

mgr	Structure holding context level state info

Returns

void

7.5.2.9 void md5_ctx_mgr_init_avx512 (MD5_HASH_CTX_MGR * *mgr*)

Initialize the MD5 multi-buffer manager structure.

Requires AVX512

Parameters

mgr	Structure holding context level state info

Returns

void

7.5.2.10 void md5_ctx_mgr_init_sse (MD5_HASH_CTX_MGR * *mgr*)

Initialize the context level MD5 multi-buffer manager structure.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info

Returns

void

7.5.2.11 MD5_HASH_CTX* md5_ctx_mgr_submit (MD5_HASH_CTX_MGR * *mgr*, MD5_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new MD5 job to the multi-buffer manager.

Requires SSE4.1 or AVX or AVX2 or AVX512

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.5.2.12 MD5_HASH_CTX* md5_ctx_mgr_submit_avx (MD5_HASH_CTX_MGR * *mgr*, MD5_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new MD5 job to the multi-buffer manager.

Requires AVX

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.5.2.13 MD5_HASH_CTX* md5_ctx_mgr_submit_avx2 (MD5_HASH_CTX_MGR * *mgr*, MD5_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new MD5 job to the multi-buffer manager.

Requires AVX2

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.5.2.14 MD5_HASH_CTX* md5_ctx_mgr_submit_avx512 (MD5_HASH_CTX_MGR * *mgr*, MD5_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new MD5 job to the multi-buffer manager.

Requires AVX512

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.5.2.15 MD5_HASH_CTX* md5_ctx_mgr_submit_sse (MD5_HASH_CTX_MGR * *mgr*, MD5_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new MD5 job to the context level multi-buffer manager.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.6 mh_sha1.h File Reference

mh_sha1 function prototypes and structures

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

Data Structures

- struct mh_sha1_ctx

  Holds info describing a single mh_sha1.

Enumerations

- enum mh_sha1_ctx_error { MH_SHA1_CTX_ERROR_NONE = 0, MH_SHA1_CTX_ERROR_NULL = -1 }

  CTX error flags.

Functions

- int mh_sha1_init (struct mh_sha1_ctx *ctx)

  Initialize the mh_sha1_ctx structure.

- int mh_sha1_update (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha1 update.
- int mh_sha1_finalize (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)

  Finalize the message digests for multi-hash sha1.
- int mh_sha1_update_base (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha1 update.
- int mh_sha1_update_sse (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha1 update.
- int mh_sha1_update_avx (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha1 update.
- int mh_sha1_update_avx2 (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha1 update.
- int mh_sha1_update_avx512 (struct mh_sha1_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha1 update.
- int mh_sha1_finalize_base (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)

  Finalize the message digests for multi-hash sha1.
- int mh_sha1_finalize_sse (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha1_finalize_avx (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha1_finalize_avx2 (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha1_finalize_avx512 (struct mh_sha1_ctx *ctx, void *mh_sha1_digest)

  Finalize the message digests for combined multi-hash and murmur.

7.6.1 Detailed Description

mh_sha1 function prototypes and structures Interface for mh_sha1 functions

mh_sha1 Init-Update..Update-Finalize

This file defines the interface to optimized functions used in mh_sha1. The definition of multi-hash SHA1(mh_sha1, for short) is: Pad the buffer in SHA1 style until the total length is a multiple of 4*16*16 (words-width * parallel-segments * block-size); Hash the buffer in parallel, generating digests of 4*16*5 (words-width*parallel-segments* digest-size); Treat the set of digests as another data buffer, and generate a final SHA1 digest for it.

Example

```
uint32_t mh_sha1_digest[SHA1_DIGEST_WORDS];
struct mh_sha1_ctx *ctx;

ctx = malloc(sizeof(struct mh_sha1_ctx));
mh_sha1_init(ctx);
mh_sha1_update(ctx, buff, block_len);
mh_sha1_finalize(ctx, mh_sha1_digest);
```

7.6.2 Enumeration Type Documentation

7.6.2.1 enum mh_sha1_ctx_error

CTX error flags.

Enumerator

MH_SHA1_CTX_ERROR_NONE MH_SHA1_CTX_ERROR_NONE.

MH_SHA1_CTX_ERROR_NULL MH_SHA1_CTX_ERROR_NULL.

7.6.3 Function Documentation

7.6.3.1 int mh_sha1_finalize (struct mh_sha1_ctx * *ctx*, void * *mh_sha1_digest*)

Finalize the message digests for multi-hash sha1.

Place the message digest in mh_sha1_digest which must have enough space for the outputs. This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

ctx	Structure holding mh_sha1 info
mh_sha1_digest	The digest of mh_sha1

Returns

int Return 0 if the function runs without errors

7.6.3.2 int mh_sha1_finalize_avx (struct mh_sha1_ctx * *ctx*, void * *mh_sha1_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digest in mh_sha1_digest which must have enough space for the outputs.

Requires AVX

Parameters

ctx	Structure holding mh_sha1 info
mh_sha1_digest	The digest of mh_sha1

Returns

int Return 0 if the function runs without errors

7.6.3.3 int mh_sha1_finalize_avx2 (struct mh_sha1_ctx * *ctx*, void * *mh_sha1_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digest in mh_sha1_digest which must have enough space for the outputs.

Requires AVX2

Parameters

ctx	Structure holding mh_sha1 info
mh_sha1_digest	The digest of mh_sha1

Returns

int Return 0 if the function runs without errors

7.6.3.4 int mh_sha1_finalize_avx512 (struct mh_sha1_ctx * *ctx*, void * *mh_sha1_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digest in mh_sha1_digest which must have enough space for the outputs.

Requires AVX512

Parameters

ctx	Structure holding mh_sha1 info
mh_sha1_digest	The digest of mh_sha1

Returns

int Return 0 if the function runs without errors

7.6.3.5 int mh_sha1_finalize_base (struct mh_sha1_ctx * *ctx*, void * *mh_sha1_digest*)

Finalize the message digests for multi-hash sha1.

Place the message digests in mh_sha1_digest, which must have enough space for the outputs. Base Finalize() function that does not require SIMD support.

Parameters

ctx	Structure holding mh_sha1 info
mh_sha1_digest	The digest of mh_sha1

Returns

int Return 0 if the function runs without errors

7.6.3.6 int mh_sha1_finalize_sse (struct mh_sha1_ctx * *ctx*, void * *mh_sha1_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digest in mh_sha1_digest which must have enough space for the outputs.

Requires SSE

Parameters

ctx	Structure holding mh_sha1 info
mh_sha1_digest	The digest of mh_sha1

Returns

int Return 0 if the function runs without errors

7.6.3.7 int mh_sha1_init (struct mh_sha1_ctx * *ctx*)

Initialize the mh_sha1_ctx structure.

Parameters

ctx	Structure holding mh_sha1 info

Returns

int Return 0 if the function runs without errors

7.6.3.8 int mh_sha1_update (struct mh_sha1_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha1 update.

Can be called repeatedly to update hashes with new input data. This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

ctx	Structure holding mh_sha1 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.6.3.9 int mh_sha1_update_avx (struct mh_sha1_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha1 update.

Can be called repeatedly to update hashes with new input data.

Requires AVX

Parameters

ctx	Structure holding mh_sha1 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.6.3.10 int mh_sha1_update_avx2 (struct mh_sha1_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha1 update.

Can be called repeatedly to update hashes with new input data.

Requires AVX2

Parameters

ctx	Structure holding mh_sha1 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.6.3.11 int mh_sha1_update_avx512 (struct mh_sha1_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha1 update.

Can be called repeatedly to update hashes with new input data.

Requires AVX512

Parameters

ctx	Structure holding mh_sha1 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.6.3.12 int mh_sha1_update_base (struct mh_sha1_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha1 update.

Can be called repeatedly to update hashes with new input data. Base update() function that does not require SIMD support.

Parameters

ctx	Structure holding mh_sha1 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.6.3.13 int mh_sha1_update_sse (struct mh_sha1_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha1 update.

Can be called repeatedly to update hashes with new input data.

Requires SSE

Parameters

ctx	Structure holding mh_sha1 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.7 mh_sha1_murmur3_x64_128.h File Reference

mh_sha1_murmur3_x64_128 function prototypes and structures

```
#include <stdint.h>
#include "mh_sha1.h"
```

Data Structures

- struct mh_sha1_murmur3_x64_128_ctx

  Holds info describing a single mh_sha1_murmur3_x64_128.

Enumerations

- enum mh_sha1_murmur3_ctx_error { MH_SHA1_MURMUR3_CTX_ERROR_NONE = 0, MH_SHA1_MURMUR3_CTX_ERROR_NULL = -1 }

  CTX error flags.

Functions

- int mh_sha1_murmur3_x64_128_init (struct mh_sha1_murmur3_x64_128_ctx *ctx, uint64_t murmur_seed)

  Initialize the mh_sha1_murmur3_x64_128_ctx structure.
- int mh_sha1_murmur3_x64_128_update (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)

  Combined multi-hash and murmur hash update.
- int mh_sha1_murmur3_x64_128_finalize (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha1_murmur3_x64_128_update_base (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)

  Combined multi-hash and murmur hash update.
- int mh_sha1_murmur3_x64_128_update_sse (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)

  Combined multi-hash and murmur hash update.
- int mh_sha1_murmur3_x64_128_update_avx (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)

  Combined multi-hash and murmur hash update.
- int mh_sha1_murmur3_x64_128_update_avx2 (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)

  Combined multi-hash and murmur hash update.
- int mh_sha1_murmur3_x64_128_update_avx512 (struct mh_sha1_murmur3_x64_128_ctx *ctx, const void *buffer, uint32_t len)

  Combined multi-hash and murmur hash update.
- int mh_sha1_murmur3_x64_128_finalize_base (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha1_murmur3_x64_128_finalize_sse (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha1_murmur3_x64_128_finalize_avx (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha1_murmur3_x64_128_finalize_avx2 (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha1_murmur3_x64_128_finalize_avx512 (struct mh_sha1_murmur3_x64_128_ctx *ctx, void *mh_sha1_digest, void *murmur3_x64_128_digest)

  Finalize the message digests for combined multi-hash and murmur.

7.7.1 Detailed Description

mh_sha1_murmur3_x64_128 function prototypes and structures Interface for mh_sha1_murmur3_x64_128 functions

mh_sha1_murmur3_x64_128 Init-Update..Update-Finalize

This file defines the interface to optimized functions used in mh_sha1 and mh_sha1_murmur3_x64_128. The definition of multi-hash SHA1(mh_sha1, for short) is: Pad the buffer in SHA1 style until the total length is a multiple of 4∗16∗16(words-width ∗ parallel-segments ∗ block-size); Hash the buffer in parallel, generating digests of 4∗16∗5 (words-width∗parallel-segments∗ digest-size); Treat the set of digests as another data buffer, and generate a final SHA1 digest for it. mh_sha1_murmur3_x64_128 is a stitching function which will get a murmur3_x64_128 digest while generate mh_sha1 digest.

Example

```
uint32_t mh_sha1_digest[SHA1_DIGEST_WORDS];
uint32_t murmur_digest[MURMUR3_x64_128_DIGEST_WORDS];
struct mh_sha1_murmur3_x64_128_ctx *ctx;

ctx = malloc(sizeof(struct mh_sha1_murmur3_x64_128_ctx));
mh_sha1_murmur3_x64_128_init(ctx, 0);
mh_sha1_murmur3_x64_128_update(ctx, buff, block_len);
mh_sha1_murmur3_x64_128_finalize(ctx, mh_sha1_digest,
murmur_digest);
```

7.7.2 Enumeration Type Documentation

7.7.2.1 enum mh_sha1_murmur3_ctx_error

CTX error flags.

Enumerator

MH_SHA1_MURMUR3_CTX_ERROR_NONE MH_SHA1_MURMUR3_CTX_ERROR_NONE.

MH_SHA1_MURMUR3_CTX_ERROR_NULL MH_SHA1_MURMUR3_CTX_ERROR_NULL.

7.7.3 Function Documentation

7.7.3.1 int mh_sha1_murmur3_x64_128_finalize (struct mh_sha1_murmur3_x64_128_ctx ∗ *ctx*, void ∗ *mh_sha1_digest*, void ∗ *murmur3_x64_128_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digests in mh_sha1_digest and murmur3_x64_128_digest, which must have enough space for the outputs. This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
mh_sha1_digest	The digest of mh_sha1
murmur3_x64_-128_digest	The digest of murmur3_x64_128

Returns

int Return 0 if the function runs without errors

7.7.3.2 int mh_sha1_murmur3_x64_128_finalize_avx (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, void * *mh_sha1_digest*, void * *murmur3_x64_128_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digests in mh_sha1_digest and murmur3_x64_128_digest, which must have enough space for the outputs.

Requires AVX

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
mh_sha1_digest	The digest of mh_sha1
murmur3_x64_-128_digest	The digest of murmur3_x64_128

Returns

int Return 0 if the function runs without errors

7.7.3.3 int mh_sha1_murmur3_x64_128_finalize_avx2 (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, void * *mh_sha1_digest*, void * *murmur3_x64_128_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digests in mh_sha1_digest and murmur3_x64_128_digest, which must have enough space for the outputs.

Requires AVX2

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
mh_sha1_digest	The digest of mh_sha1
murmur3_x64_-128_digest	The digest of murmur3_x64_128

Returns

int Return 0 if the function runs without errors

7.7.3.4 int mh_sha1_murmur3_x64_128_finalize_avx512 (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, void * *mh_sha1_digest*, void * *murmur3_x64_128_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digests in mh_sha1_digest and murmur3_x64_128_digest, which must have enough space for the outputs.

Requires AVX512

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
mh_sha1_digest	The digest of mh_sha1
murmur3_x64_-128_digest	The digest of murmur3_x64_128

Returns

int Return 0 if the function runs without errors

7.7.3.5 int mh_sha1_murmur3_x64_128_finalize_base (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, void * *mh_sha1_digest*, void * *murmur3_x64_128_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digests in mh_sha1_digest and murmur3_x64_128_digest, which must have enough space for the outputs. Base Finalize() function that does not require SIMD support.

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
mh_sha1_digest	The digest of mh_sha1

murmur3_x64_-128_digest	The digest of murmur3_x64_128

Returns

int Return 0 if the function runs without errors

7.7.3.6 int mh_sha1_murmur3_x64_128_finalize_sse (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, void * *mh_sha1_digest*, void * *murmur3_x64_128_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digests in mh_sha1_digest and murmur3_x64_128_digest, which must have enough space for the outputs.

Requires SSE

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
mh_sha1_digest	The digest of mh_sha1
murmur3_x64_-128_digest	The digest of murmur3_x64_128

Returns

int Return 0 if the function runs without errors

7.7.3.7 int mh_sha1_murmur3_x64_128_init (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, uint64_t *murmur_seed*)

Initialize the mh_sha1_murmur3_x64_128_ctx structure.

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
murmur_seed	Seed as an initial digest of murmur3

Returns

int Return 0 if the function runs without errors

7.7.3.8 int mh_sha1_murmur3_x64_128_update (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Combined multi-hash and murmur hash update.

Can be called repeatedly to update hashes with new input data. This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.7.3.9 int mh_sha1_murmur3_x64_128_update_avx (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Combined multi-hash and murmur hash update.

Can be called repeatedly to update hashes with new input data.

Requires AVX

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.7.3.10 int mh_sha1_murmur3_x64_128_update_avx2 (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Combined multi-hash and murmur hash update.

Can be called repeatedly to update hashes with new input data.

Requires AVX2

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.7.3.11 int mh_sha1_murmur3_x64_128_update_avx512 (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Combined multi-hash and murmur hash update.

Can be called repeatedly to update hashes with new input data.

Requires AVX512

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.7.3.12 int mh_sha1_murmur3_x64_128_update_base (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Combined multi-hash and murmur hash update.

Can be called repeatedly to update hashes with new input data. Base update() function that does not require SIMD support.

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.7.3.13 int mh_sha1_murmur3_x64_128_update_sse (struct mh_sha1_murmur3_x64_128_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Combined multi-hash and murmur hash update.

Can be called repeatedly to update hashes with new input data.

Requires SSE

Parameters

ctx	Structure holding mh_sha1_murmur3_x64_128 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.8 mh_sha256.h File Reference

mh_sha256 function prototypes and structures

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

Data Structures

- struct mh_sha256_ctx

  Holds info describing a single mh_sha256.

Enumerations

- enum mh_sha256_ctx_error { MH_SHA256_CTX_ERROR_NONE = 0, MH_SHA256_CTX_ERROR_NULL = -1 }

  CTX error flags.

Functions

- int mh_sha256_init (struct mh_sha256_ctx *ctx)

  Initialize the mh_sha256_ctx structure.
- int mh_sha256_update (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha256 update.
- int mh_sha256_finalize (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)

  Finalize the message digests for multi-hash sha256.
- int mh_sha256_update_base (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha256 update.
- int mh_sha256_update_sse (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha256 update.
- int mh_sha256_update_avx (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha256 update.
- int mh_sha256_update_avx2 (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha256 update.
- int mh_sha256_update_avx512 (struct mh_sha256_ctx *ctx, const void *buffer, uint32_t len)

  Multi-hash sha256 update.
- int mh_sha256_finalize_base (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)

  Finalize the message digests for multi-hash sha256.
- int mh_sha256_finalize_sse (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha256_finalize_avx (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha256_finalize_avx2 (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)

  Finalize the message digests for combined multi-hash and murmur.
- int mh_sha256_finalize_avx512 (struct mh_sha256_ctx *ctx, void *mh_sha256_digest)

  Finalize the message digests for combined multi-hash and murmur.

7.8.1 Detailed Description

mh_sha256 function prototypes and structures Interface for mh_sha256 functions

mh_sha256 Init-Update..Update-Finalize

This file defines the interface to optimized functions used in mh_sha256. The definition of multi-hash SHA256(mh-_sha256, for short) is: Pad the buffer in SHA256 style until the total length is a multiple of 4*16*16 (words-width * parallel-segments * block-size); Hash the buffer in parallel, generating digests of 4*16*8 (words-width*parallel-segments* digest-size); Treat the set of digests as another data buffer, and generate a final SHA256 digest for it.

Example

```
uint32_t mh_sha256_digest[SHA256_DIGEST_WORDS];
struct mh_sha256_ctx *ctx;

ctx = malloc(sizeof(struct mh_sha256_ctx));
mh_sha256_init(ctx);
mh_sha256_update(ctx, buff, block_len);
mh_sha256_finalize(ctx, mh_sha256_digest);
```

7.8.2 Enumeration Type Documentation

7.8.2.1 enum mh_sha256_ctx_error

CTX error flags.

Enumerator

MH_SHA256_CTX_ERROR_NONE MH_SHA256_CTX_ERROR_NONE.

MH_SHA256_CTX_ERROR_NULL MH_SHA256_CTX_ERROR_NULL.

7.8.3 Function Documentation

7.8.3.1 int mh_sha256_finalize (struct mh_sha256_ctx * *ctx*, void * *mh_sha256_digest*)

Finalize the message digests for multi-hash sha256.

Place the message digest in mh_sha256_digest which must have enough space for the outputs. This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

ctx	Structure holding mh_sha256 info
mh_sha256_digest	The digest of mh_sha256

Returns

int Return 0 if the function runs without errors

7.8.3.2 int mh_sha256_finalize_avx (struct mh_sha256_ctx * *ctx*, void * *mh_sha256_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digest in mh_sha256_digest which must have enough space for the outputs.

Requires AVX

Parameters

ctx	Structure holding mh_sha256 info
mh_sha256_-digest	The digest of mh_sha256

Returns

int Return 0 if the function runs without errors

7.8.3.3 int mh_sha256_finalize_avx2 (struct mh_sha256_ctx * *ctx*, void * *mh_sha256_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digest in mh_sha256_digest which must have enough space for the outputs.

Requires AVX2

Parameters

ctx	Structure holding mh_sha256 info
mh_sha256_-digest	The digest of mh_sha256

Returns

int Return 0 if the function runs without errors

7.8.3.4 int mh_sha256_finalize_avx512 (struct mh_sha256_ctx * *ctx*, void * *mh_sha256_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digest in mh_sha256_digest which must have enough space for the outputs.

Requires AVX512

Parameters

ctx	Structure holding mh_sha256 info
mh_sha256_-digest	The digest of mh_sha256

Returns

int Return 0 if the function runs without errors

7.8.3.5 int mh_sha256_finalize_base (struct mh_sha256_ctx * *ctx*, void * *mh_sha256_digest*)

Finalize the message digests for multi-hash sha256.

Place the message digests in mh_sha256_digest, which must have enough space for the outputs. Base Finalize() function that does not require SIMD support.

Parameters

ctx	Structure holding mh_sha256 info
mh_sha256_digest	The digest of mh_sha256

Returns

int Return 0 if the function runs without errors

7.8.3.6 int mh_sha256_finalize_sse (struct mh_sha256_ctx * *ctx*, void * *mh_sha256_digest*)

Finalize the message digests for combined multi-hash and murmur.

Place the message digest in mh_sha256_digest which must have enough space for the outputs.

Requires SSE

Parameters

ctx	Structure holding mh_sha256 info
mh_sha256_digest	The digest of mh_sha256

Returns

int Return 0 if the function runs without errors

7.8.3.7 int mh_sha256_init (struct mh_sha256_ctx * *ctx*)

Initialize the mh_sha256_ctx structure.

Parameters

ctx	Structure holding mh_sha256 info

Returns

int Return 0 if the function runs without errors

7.8.3.8 int mh_sha256_update (struct mh_sha256_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha256 update.

Can be called repeatedly to update hashes with new input data. This function determines what instruction sets are enabled and selects the appropriate version at runtime.

Parameters

ctx	Structure holding mh_sha256 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.8.3.9 int mh_sha256_update_avx (struct mh_sha256_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha256 update.

Can be called repeatedly to update hashes with new input data.

Requires AVX

Parameters

ctx	Structure holding mh_sha256 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.8.3.10 int mh_sha256_update_avx2 (struct mh_sha256_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha256 update.

Can be called repeatedly to update hashes with new input data.

Requires AVX2

Parameters

ctx	Structure holding mh_sha256 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.8.3.11 int mh_sha256_update_avx512 (struct mh_sha256_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha256 update.

Can be called repeatedly to update hashes with new input data.

Requires AVX512

Parameters

ctx	Structure holding mh_sha256 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.8.3.12 int mh_sha256_update_base (struct mh_sha256_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha256 update.

Can be called repeatedly to update hashes with new input data. Base update() function that does not require SIMD support.

Parameters

ctx	Structure holding mh_sha256 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.8.3.13 int mh_sha256_update_sse (struct mh_sha256_ctx * *ctx*, const void * *buffer*, uint32_t *len*)

Multi-hash sha256 update.

Can be called repeatedly to update hashes with new input data.

Requires SSE

Parameters

ctx	Structure holding mh_sha256 info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed

Returns

int Return 0 if the function runs without errors

7.9 multi_buffer.h File Reference

Multi-buffer common fields.

Enumerations

- enum JOB_STS { STS_UNKNOWN = 0, STS_BEING_PROCESSED = 1, STS_COMPLETED = 2, STS_INTERNAL_ERROR, STS_ERROR }

  Job return codes.
- enum HASH_CTX_FLAG { HASH_UPDATE = 0x00, HASH_FIRST = 0x01, HASH_LAST = 0x02, HASH_ENTIRE = 0x03 }

  CTX job type.

- enum HASH_CTX_STS { HASH_CTX_STS_IDLE = 0x00, HASH_CTX_STS_PROCESSING = 0x01, HASH_CTX_STS_LAST = 0x02, HASH_CTX_STS_COMPLETE = 0x04 }

  CTX status flags.
- enum HASH_CTX_ERROR { HASH_CTX_ERROR_NONE = 0, HASH_CTX_ERROR_INVALID_FLAGS = -1, HASH_CTX_ERROR_ALREADY_PROCESSING = -2, HASH_CTX_ERROR_ALREADY_COMPLETED = -3 }

  CTX error flags.

7.9.1 Detailed Description

Multi-buffer common fields.

7.9.2 Enumeration Type Documentation

7.9.2.1 enum HASH_CTX_ERROR

CTX error flags.

Enumerator

HASH_CTX_ERROR_NONE HASH_CTX_ERROR_NONE.

HASH_CTX_ERROR_INVALID_FLAGS HASH_CTX_ERROR_INVALID_FLAGS.

HASH_CTX_ERROR_ALREADY_PROCESSING HASH_CTX_ERROR_ALREADY_PROCESSING.

HASH_CTX_ERROR_ALREADY_COMPLETED HASH_CTX_ERROR_ALREADY_COMPLETED.

7.9.2.2 enum HASH_CTX_FLAG

CTX job type.

Enumerator

HASH_UPDATE HASH_UPDATE.

HASH_FIRST HASH_FIRST.

HASH_LAST HASH_LAST.

HASH_ENTIRE HASH_ENTIRE.

7.9.2.3 enum HASH_CTX_STS

CTX status flags.

Enumerator

HASH_CTX_STS_IDLE HASH_CTX_STS_IDLE.
HASH_CTX_STS_PROCESSING HASH_CTX_STS_PROCESSING.
HASH_CTX_STS_LAST HASH_CTX_STS_LAST.
HASH_CTX_STS_COMPLETE HASH_CTX_STS_COMPLETE.

7.9.2.4 enum JOB_STS

Job return codes.

Enumerator

STS_UNKNOWN STS_UNKNOWN.
STS_BEING_PROCESSED STS_BEING_PROCESSED.
STS_COMPLETED STS_COMPLETED.
STS_INTERNAL_ERROR STS_INTERNAL_ERROR.
STS_ERROR STS_ERROR.

7.10 rolling_hashx.h File Reference

Fingerprint functions based on rolling hash.

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

Data Structures

- struct rh_state1
  Context for rolling_hash1 functions.
- struct rh_state2
  Context for rolling_hash2 functions.

Enumerations

- enum { FINGERPRINT_RET_HIT = 0, FINGERPRINT_RET_MAX, FINGERPRINT_RET_OTHER }
  rolling hash return values

Functions

- int rolling_hash1_init (struct rh_state1 *state, uint32_t w)

  Initialize state object for rolling hash1.
- void rolling_hash1_reset (struct rh_state1 *state, uint8_t *init_bytes)

  Reset the hash state history.
- int rolling_hash1_run (struct rh_state1 *state, uint8_t *buffer, uint32_t buffer_length, uint32_t mask, uint32_t trigger, uint32_t *offset)

  Run rolling hash function until trigger met or max length reached.
- int rolling_hash2_init (struct rh_state2 *state, uint32_t w)

  Initialize state object for rolling hash2.
- void rolling_hash2_reset (struct rh_state2 *state, uint8_t *init_bytes)

  Reset the hash state history.
- int rolling_hash2_run (struct rh_state2 *state, uint8_t *buffer, uint32_t buffer_length, uint32_t mask, uint32_t trigger, uint32_t *offset)

  Run rolling hash function until trigger met or max length reached.
- uint32_t rolling_hashx_mask_gen (long mean, int shift)

  Generate an appropriate mask to target mean hit rate.

7.10.1 Detailed Description

Fingerprint functions based on rolling hash. Includes interfaces to two different rolling hashes: rolling_hash1 - checks hash in a sliding window based on CRC. rolling_hash2 - checks hash in a sliding window based on random 64-bit hash.

7.10.2 Enumeration Type Documentation

7.10.2.1 anonymous enum

rolling hash return values

Enumerator

FINGERPRINT_RET_HIT Fingerprint trigger hit.

FINGERPRINT_RET_MAX Fingerprint max length reached before hit.

FINGERPRINT_RET_OTHER Fingerprint function error returned.

7.10.3 Function Documentation

7.10.3.1 int rolling_hash1_init (struct rh_state1 * *state*, uint32_t *w*)

Initialize state object for rolling hash1.

Parameters

state	Structure holding state info on current rolling hash
w	Window width (1 <= w <= 32)

Returns

0 - success, -1 - failure

7.10.3.2 void rolling_hash1_reset (struct rh_state1 * *state*, uint8_t * *init_bytes*)

Reset the hash state history.

Parameters

state	Structure holding state info on current rolling hash
init_bytes	Optional window size buffer to pre-init hash

Returns

none

7.10.3.3 int rolling_hash1_run (struct rh_state1 * *state*, uint8_t * *buffer*, uint32_t *buffer_length*, uint32_t *mask*, uint32_t *trigger*, uint32_t * *offset*)

Run rolling hash function until trigger met or max length reached.

Checks for trigger based on CRC in a sliding window.

Requires SSE4.2

Parameters

state	Structure holding state info on current rolling hash
buffer	Pointer to input buffer to run windowed hash on
max_len	Max length to run over input
mask	Mask bits ORed with hash before test with trigger
trigger	Match value to compare with windowed hash at each input byte
offset	Offset from buffer to match, set if match found

Returns

FINGERPRINT_RET_HIT - match found, FINGERPRINT_RET_MAX - exceeded max length

7.10.3.4 int rolling_hash2_init (struct rh_state2 * *state*, uint32_t *w*)

Initialize state object for rolling hash2.

Parameters

state	Structure holding state info on current rolling hash
w	Window width (1 <= w <= 32)

Returns

0 - success, -1 - failure

7.10.3.5 void rolling_hash2_reset (struct rh_state2 * *state*, uint8_t * *init_bytes*)

Reset the hash state history.

Parameters

state	Structure holding state info on current rolling hash
init_bytes	Optional window size buffer to pre-init hash

Returns

none

7.10.3.6 int rolling_hash2_run (struct rh_state2 * *state*, uint8_t * *buffer*, uint32_t *buffer_length*, uint32_t *mask*, uint32_t *trigger*, uint32_t * *offset*)

Run rolling hash function until trigger met or max length reached.

Checks for trigger based on a random hash in a sliding window.

Parameters

state	Structure holding state info on current rolling hash
buffer	Pointer to input buffer to run windowed hash on
max_len	Max length to run over input
mask	Mask bits ORed with hash before test with trigger
trigger	Match value to compare with windowed hash at each input byte
offset	Offset from buffer to match, set if match found

Returns

FINGERPRINT_RET_HIT - match found, FINGERPRINT_RET_MAX - exceeded max length

7.10.3.7 uint32_t rolling_hashx_mask_gen (long *mean*, int *shift*)

Generate an appropriate mask to target mean hit rate.

Parameters

mean	Target chunk size in bytes
shift	Bits to rotate result to get independent masks

Returns

32-bit mask value

7.11 sha.h File Reference

SHA1 functions.

```
#include <stdlib.h>
```

Functions

- void sha1_update (unsigned int *digest, unsigned char *input, size_t num_blocks)

  Part of the SHA1 hash algorithm that can be run repeatedly on message blocks of 64 bytes to update the hash value.
- void sha1_opt (unsigned char *input, unsigned int *digest, int len)

  Performs complete SHA1 algorithm using optimized sha1_update routine.

7.11.1 Detailed Description

SHA1 functions.

7.11.2 Function Documentation

7.11.2.1 void sha1_opt (unsigned char * *input*, unsigned int * *digest*, int *len*)

Performs complete SHA1 algorithm using optimized sha1_update routine.

Requires SSE3

Parameters

input	Pointer to buffer containing the input message.
digest	Pointer to digest to update.
len	Length of buffer.

Returns

None

7.11.2.2 void sha1_update (unsigned int * *digest*, unsigned char * *input*, size_t *num_blocks*)

Part of the SHA1 hash algorithm that can be run repeatedly on message blocks of 64 bytes to update the hash value.

Requires SSE3

Parameters

digest	Pointer to digest to update.
input	Pointer to buffer containing the input message in 64 byte blocks.
num_blocks	Number of 64 byte blocks to incorporate in hash update.

Returns

None

7.12 sha1_mb.h File Reference

Multi-buffer CTX API SHA1 function prototypes and structures.

```
#include <stdint.h>
#include "multi_buffer.h"
#include "types.h"
#include <stdbool.h>
```

Data Structures

- struct SHA1_JOB

  Scheduler layer - Holds info describing a single SHA1 job for the multi-buffer manager.
- struct SHA1_MB_ARGS_X16

  Scheduler layer - Holds arguments for submitted SHA1 job.

- struct SHA1_LANE_DATA

  Scheduler layer - Lane data.
- struct SHA1_MB_JOB_MGR

  Scheduler layer - Holds state for multi-buffer SHA1 jobs.
- struct SHA1_HASH_CTX_MGR

  Context layer - Holds state for multi-buffer SHA1 jobs.
- struct SHA1_HASH_CTX

  Context layer - Holds info describing a single SHA1 job for the multi-buffer CTX manager.

Functions

- void sha1_ctx_mgr_init (SHA1_HASH_CTX_MGR *mgr)

  Initialize the SHA1 multi-buffer manager structure.
- SHA1_HASH_CTX * sha1_ctx_mgr_submit (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA1 job to the multi-buffer manager.
- SHA1_HASH_CTX * sha1_ctx_mgr_flush (SHA1_HASH_CTX_MGR *mgr)

  Finish all submitted SHA1 jobs and return when complete.
- void sha1_ctx_mgr_init_sse (SHA1_HASH_CTX_MGR *mgr)

  Initialize the context level SHA1 multi-buffer manager structure.
- SHA1_HASH_CTX * sha1_ctx_mgr_submit_sse (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA1 job to the context level multi-buffer manager.
- SHA1_HASH_CTX * sha1_ctx_mgr_flush_sse (SHA1_HASH_CTX_MGR *mgr)

  Finish all submitted SHA1 jobs and return when complete.
- void sha1_ctx_mgr_init_sse_ni (SHA1_HASH_CTX_MGR *mgr)

  Initialize the context level SHA1 multi-buffer manager structure.
- SHA1_HASH_CTX * sha1_ctx_mgr_submit_sse_ni (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA1 job to the context level multi-buffer manager.
- SHA1_HASH_CTX * sha1_ctx_mgr_flush_sse_ni (SHA1_HASH_CTX_MGR *mgr)

  Finish all submitted SHA1 jobs and return when complete.
- void sha1_ctx_mgr_init_avx (SHA1_HASH_CTX_MGR *mgr)

  Initialize the SHA1 multi-buffer manager structure.
- SHA1_HASH_CTX * sha1_ctx_mgr_submit_avx (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA1 job to the multi-buffer manager.
- SHA1_HASH_CTX * sha1_ctx_mgr_flush_avx (SHA1_HASH_CTX_MGR *mgr)

  Finish all submitted SHA1 jobs and return when complete.
- void sha1_ctx_mgr_init_avx2 (SHA1_HASH_CTX_MGR *mgr)

Initialize the SHA1 multi-buffer manager structure.

- SHA1_HASH_CTX * sha1_ctx_mgr_submit_avx2 (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA1 job to the multi-buffer manager.
- SHA1_HASH_CTX * sha1_ctx_mgr_flush_avx2 (SHA1_HASH_CTX_MGR *mgr)

  Finish all submitted SHA1 jobs and return when complete.
- void sha1_ctx_mgr_init_avx512 (SHA1_HASH_CTX_MGR *mgr)

  Initialize the SHA1 multi-buffer manager structure.
- SHA1_HASH_CTX * sha1_ctx_mgr_submit_avx512 (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA1 job to the multi-buffer manager.
- SHA1_HASH_CTX * sha1_ctx_mgr_flush_avx512 (SHA1_HASH_CTX_MGR *mgr)

  Finish all submitted SHA1 jobs and return when complete.
- void sha1_ctx_mgr_init_avx512_ni (SHA1_HASH_CTX_MGR *mgr)

  Initialize the SHA1 multi-buffer manager structure.
- SHA1_HASH_CTX * sha1_ctx_mgr_submit_avx512_ni (SHA1_HASH_CTX_MGR *mgr, SHA1_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA1 job to the multi-buffer manager.
- SHA1_HASH_CTX * sha1_ctx_mgr_flush_avx512_ni (SHA1_HASH_CTX_MGR *mgr)

  Finish all submitted SHA1 jobs and return when complete.

7.12.1 Detailed Description

Multi-buffer CTX API SHA1 function prototypes and structures. Interface for multi-buffer SHA1 functions

Multi-buffer SHA1 Entire or First-Update..Update-Last

The interface to this multi-buffer hashing code is carried out through the context-level (CTX) init, submit and flush functions and the SHA1_HASH_CTX_MGR and SHA1_HASH_CTX objects. Numerous SHA1_HASH_CTX objects may be instantiated by the application for use with a single SHA1_HASH_CTX_MGR.

The CTX interface functions carry out the initialization and padding of the jobs entered by the user and add them to the multi-buffer manager. The lower level "scheduler" layer then processes the jobs in an out-of-order manner. The scheduler layer functions are internal and are not intended to be invoked directly. Jobs can be submitted to a CTX as a complete buffer to be hashed, using the HASH_ENTIRE flag, or as partial jobs which can be started using the HASH_FIRST flag, and later resumed or finished using the HASH_UPDATE and HASH_LAST flags respectively.

Note: The submit function does not require data buffers to be block sized.

The SHA1 CTX interface functions are available for 4 architectures: SSE, AVX, AVX2 and AVX512. In addition, a multibinary interface is provided, which selects the appropriate architecture-specific function at runtime.

Usage: The application creates a SHA1_HASH_CTX_MGR object and initializes it with a call to sha1_ctx_mgr_-init*() function, where henceforth "*" stands for the relevant suffix for each architecture; _sse, _avx, _avx2, _avx512(or no suffix for the multibinary version). The SHA1_HASH_CTX_MGR object will be used to schedule processor

resources, with up to 4 SHA1_HASH_CTX objects (or 8 in the AVX2 case, 16 in the AVX512) being processed at a time.

Each SHA1_HASH_CTX must be initialized before first use by the hash_ctx_init macro defined in multi_buffer.h. After initialization, the application may begin computing a hash by giving the SHA1_HASH_CTX to a SHA1_HASH_CTX_MGR using the submit functions sha1_ctx_mgr_submit∗() with the HASH_FIRST flag set. When the SHA1_HASH_CTX is returned to the application (via this or a later call to sha1_ctx_mgr_submit∗() or sha1_ctx_mgr_flush∗()), the application can then re-submit it with another call to sha1_ctx_mgr_submit∗(), but without the HASH_FIRST flag set.

Ideally, on the last buffer for that hash, sha1_ctx_mgr_submit_sse is called with HASH_LAST, although it is also possible to submit the hash with HASH_LAST and a zero length if necessary. When a SHA1_HASH_CTX is returned after having been submitted with HASH_LAST, it will contain a valid hash. The SHA1_HASH_CTX can be reused immediately by submitting with HASH_FIRST.

For example, you would submit hashes with the following flags for the following numbers of buffers:

- one buffer: HASH_FIRST | HASH_LAST (or, equivalently, HASH_ENTIRE)
- two buffers: HASH_FIRST, HASH_LAST
- three buffers: HASH_FIRST, HASH_UPDATE, HASH_LAST etc.

The order in which SHA1_CTX objects are returned is in general different from the order in which they are submitted.

A few possible error conditions exist:

- Submitting flags other than the allowed entire/first/update/last values
- Submitting a context that is currently being managed by a SHA1_HASH_CTX_MGR.
- Submitting a context after HASH_LAST is used but before HASH_FIRST is set.

These error conditions are reported by returning the SHA1_HASH_CTX immediately after a submit with its error member set to a non-zero error code (defined in multi_buffer.h). No changes are made to the SHA1_HASH_CTX_MGR in the case of an error; no processing is done for other hashes.

7.12.2 Function Documentation

7.12.2.1 SHA1_HASH_CTX∗ sha1_ctx_mgr_flush (SHA1_HASH_CTX_MGR ∗ *mgr*)

Finish all submitted SHA1 jobs and return when complete.

Requires SSE4.1 or AVX or AVX2 or AVX512

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

Examples:

sha1_multi_buffer_example.c.

7.12.2.2 SHA1_HASH_CTX* sha1_ctx_mgr_flush_avx (SHA1_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA1 jobs and return when complete.

Requires AVX

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.12.2.3 SHA1_HASH_CTX* sha1_ctx_mgr_flush_avx2 (SHA1_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA1 jobs and return when complete.

Requires AVX2

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.12.2.4 SHA1_HASH_CTX* sha1_ctx_mgr_flush_avx512 (SHA1_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA1 jobs and return when complete.

Requires AVX512

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.12.2.5 SHA1_HASH_CTX* sha1_ctx_mgr_flush_avx512_ni (SHA1_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA1 jobs and return when complete.

Requires AVX512 and SHANI

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.12.2.6 SHA1_HASH_CTX* sha1_ctx_mgr_flush_sse (SHA1_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA1 jobs and return when complete.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.12.2.7 SHA1_HASH_CTX* sha1_ctx_mgr_flush_sse_ni (SHA1_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA1 jobs and return when complete.

Requires SSE4.1 and SHANI

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.12.2.8 void sha1_ctx_mgr_init (SHA1_HASH_CTX_MGR * *mgr*)

Initialize the SHA1 multi-buffer manager structure.

Requires SSE4.1 or AVX or AVX2 or AVX512

Parameters

mgr	Structure holding context level state info

Returns

void

Examples:

sha1_multi_buffer_example.c.

7.12.2.9 void sha1_ctx_mgr_init_avx (SHA1_HASH_CTX_MGR * *mgr*)

Initialize the SHA1 multi-buffer manager structure.

Requires AVX

Parameters

mgr	Structure holding context level state info

Returns

void

7.12.2.10 void sha1_ctx_mgr_init_avx2 (SHA1_HASH_CTX_MGR * *mgr*)

Initialize the SHA1 multi-buffer manager structure.

Requires AVX2

Parameters

mgr	Structure holding context level state info

Returns

void

7.12.2.11 void sha1_ctx_mgr_init_avx512 (SHA1_HASH_CTX_MGR * *mgr*)

Initialize the SHA1 multi-buffer manager structure.

Requires AVX512

Parameters

mgr	Structure holding context level state info

Returns

void

7.12.2.12 void sha1_ctx_mgr_init_avx512_ni (SHA1_HASH_CTX_MGR * *mgr*)

Initialize the SHA1 multi-buffer manager structure.

Requires AVX512 and SHANI

Parameters

mgr	Structure holding context level state info

Returns

void

7.12.2.13 void sha1_ctx_mgr_init_sse (SHA1_HASH_CTX_MGR * *mgr*)

Initialize the context level SHA1 multi-buffer manager structure.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info

Returns

void

7.12.2.14 void sha1_ctx_mgr_init_sse_ni (SHA1_HASH_CTX_MGR * *mgr*)

Initialize the context level SHA1 multi-buffer manager structure.

Requires SSE4.1 and SHANI

Parameters

mgr	Structure holding context level state info

Returns

void

7.12.2.15 SHA1_HASH_CTX* sha1_ctx_mgr_submit (SHA1_HASH_CTX_MGR * *mgr*, SHA1_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA1 job to the multi-buffer manager.

Requires SSE4.1 or AVX or AVX2 or AVX512

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

Examples:

sha1_multi_buffer_example.c.

7.12.2.16 SHA1_HASH_CTX* sha1_ctx_mgr_submit_avx (SHA1_HASH_CTX_MGR * *mgr*, SHA1_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA1 job to the multi-buffer manager.

Requires AVX

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.12.2.17 SHA1_HASH_CTX* sha1_ctx_mgr_submit_avx2 (SHA1_HASH_CTX_MGR * *mgr*, SHA1_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA1 job to the multi-buffer manager.

Requires AVX2

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.12.2.18 SHA1_HASH_CTX* sha1_ctx_mgr_submit_avx512 (SHA1_HASH_CTX_MGR * *mgr*, SHA1_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA1 job to the multi-buffer manager.

Requires AVX512

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.12.2.19 SHA1_HASH_CTX* sha1_ctx_mgr_submit_avx512_ni (SHA1_HASH_CTX_MGR * *mgr*, SHA1_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA1 job to the multi-buffer manager.

Requires AVX512 and SHANI

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.12.2.20 SHA1_HASH_CTX* sha1_ctx_mgr_submit_sse (SHA1_HASH_CTX_MGR * *mgr*, SHA1_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA1 job to the context level multi-buffer manager.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.12.2.21 SHA1_HASH_CTX* sha1_ctx_mgr_submit_sse_ni (SHA1_HASH_CTX_MGR * *mgr*, SHA1_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA1 job to the context level multi-buffer manager.

Requires SSE4.1 and SHANI

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.13 sha256_mb.h File Reference

Multi-buffer CTX API SHA256 function prototypes and structures.

```
#include <stdint.h>
#include "multi_buffer.h"
#include "types.h"
#include <stdbool.h>
```

Data Structures

- struct SHA256_JOB

  Scheduler layer - Holds info describing a single SHA256 job for the multi-buffer manager.
- struct SHA256_MB_ARGS_X16

  Scheduler layer - Holds arguments for submitted SHA256 job.
- struct SHA256_LANE_DATA

  Scheduler layer - Lane data.
- struct SHA256_MB_JOB_MGR

  Scheduler layer - Holds state for multi-buffer SHA256 jobs.
- struct SHA256_HASH_CTX_MGR

  Context layer - Holds state for multi-buffer SHA256 jobs.
- struct SHA256_HASH_CTX

  Context layer - Holds info describing a single SHA256 job for the multi-buffer CTX manager.

Functions

- void sha256_ctx_mgr_init (SHA256_HASH_CTX_MGR *mgr)

  Initialize the SHA256 multi-buffer manager structure.

- SHA256_HASH_CTX * sha256_ctx_mgr_submit (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA256 job to the multi-buffer manager.
- SHA256_HASH_CTX * sha256_ctx_mgr_flush (SHA256_HASH_CTX_MGR *mgr)

  Finish all submitted SHA256 jobs and return when complete.
- void sha256_ctx_mgr_init_sse (SHA256_HASH_CTX_MGR *mgr)

  Initialize the context level SHA256 multi-buffer manager structure.
- SHA256_HASH_CTX * sha256_ctx_mgr_submit_sse (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA256 job to the context level multi-buffer manager.
- SHA256_HASH_CTX * sha256_ctx_mgr_flush_sse (SHA256_HASH_CTX_MGR *mgr)

  Finish all submitted SHA256 jobs and return when complete.
- void sha256_ctx_mgr_init_sse_ni (SHA256_HASH_CTX_MGR *mgr)

  Initialize the context level SHA256 multi-buffer manager structure.
- SHA256_HASH_CTX * sha256_ctx_mgr_submit_sse_ni (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA256 job to the context level multi-buffer manager.
- SHA256_HASH_CTX * sha256_ctx_mgr_flush_sse_ni (SHA256_HASH_CTX_MGR *mgr)

  Finish all submitted SHA256 jobs and return when complete.
- void sha256_ctx_mgr_init_avx (SHA256_HASH_CTX_MGR *mgr)

  Initialize the SHA256 multi-buffer manager structure.
- SHA256_HASH_CTX * sha256_ctx_mgr_submit_avx (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA256 job to the multi-buffer manager.
- SHA256_HASH_CTX * sha256_ctx_mgr_flush_avx (SHA256_HASH_CTX_MGR *mgr)

  Finish all submitted SHA256 jobs and return when complete.
- void sha256_ctx_mgr_init_avx2 (SHA256_HASH_CTX_MGR *mgr)

  Initialize the SHA256 multi-buffer manager structure.
- SHA256_HASH_CTX * sha256_ctx_mgr_submit_avx2 (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA256 job to the multi-buffer manager.
- SHA256_HASH_CTX * sha256_ctx_mgr_flush_avx2 (SHA256_HASH_CTX_MGR *mgr)

  Finish all submitted SHA256 jobs and return when complete.
- void sha256_ctx_mgr_init_avx512 (SHA256_HASH_CTX_MGR *mgr)

  Initialize the SHA256 multi-buffer manager structure.
- SHA256_HASH_CTX * sha256_ctx_mgr_submit_avx512 (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA256 job to the multi-buffer manager.
- SHA256_HASH_CTX * sha256_ctx_mgr_flush_avx512 (SHA256_HASH_CTX_MGR *mgr)

  Finish all submitted SHA256 jobs and return when complete.
- void sha256_ctx_mgr_init_avx512_ni (SHA256_HASH_CTX_MGR *mgr)

Initialize the SHA256 multi-buffer manager structure.

- SHA256_HASH_CTX * sha256_ctx_mgr_submit_avx512_ni (SHA256_HASH_CTX_MGR *mgr, SHA256_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA256 job to the multi-buffer manager.

- SHA256_HASH_CTX * sha256_ctx_mgr_flush_avx512_ni (SHA256_HASH_CTX_MGR *mgr)

  Finish all submitted SHA256 jobs and return when complete.

7.13.1 Detailed Description

Multi-buffer CTX API SHA256 function prototypes and structures. Interface for multi-buffer SHA256 functions

Multi-buffer SHA256 Entire or First-Update..Update-Last

The interface to this multi-buffer hashing code is carried out through the context-level (CTX) init, submit and flush functions and the SHA256_HASH_CTX_MGR and SHA256_HASH_CTX objects. Numerous SHA256_HASH_CTX objects may be instantiated by the application for use with a single SHA256_HASH_CTX_MGR.

The CTX interface functions carry out the initialization and padding of the jobs entered by the user and add them to the multi-buffer manager. The lower level "scheduler" layer then processes the jobs in an out-of-order manner. The scheduler layer functions are internal and are not intended to be invoked directly. Jobs can be submitted to a CTX as a complete buffer to be hashed, using the HASH_ENTIRE flag, or as partial jobs which can be started using the HASH_FIRST flag, and later resumed or finished using the HASH_UPDATE and HASH_LAST flags respectively.

Note: The submit function does not require data buffers to be block sized.

The SHA256 CTX interface functions are available for 4 architectures: SSE, AVX, AVX2 and AVX512. In addition, a multibinary interface is provided, which selects the appropriate architecture-specific function at runtime.

Usage: The application creates a SHA256_HASH_CTX_MGR object and initializes it with a call to sha256_ctx_mgr_init*() function, where henceforth "*" stands for the relevant suffix for each architecture; _sse, _avx, _avx2, _avx512(or no suffix for the multibinary version). The SHA256_HASH_CTX_MGR object will be used to schedule processor resources, with up to 4 SHA256_HASH_CTX objects (or 8 in the AVX2 case, 16 in the AVX512) being processed at a time.

Each SHA256_HASH_CTX must be initialized before first use by the hash_ctx_init macro defined in multi_buffer.h. After initialization, the application may begin computing a hash by giving the SHA256_HASH_CTX to a SHA256_HASH_CTX_MGR using the submit functions sha256_ctx_mgr_submit*() with the HASH_FIRST flag set. When the SHA256_HASH_CTX is returned to the application (via this or a later call to sha256_ctx_mgr_submit*() or sha256_ctx_mgr_flush*()), the application can then re-submit it with another call to sha256_ctx_mgr_submit*(), but without the HASH_FIRST flag set.

Ideally, on the last buffer for that hash, sha256_ctx_mgr_submit_sse is called with HASH_LAST, although it is also possible to submit the hash with HASH_LAST and a zero length if necessary. When a SHA256_HASH_CTX is returned after having been submitted with HASH_LAST, it will contain a valid hash. The SHA256_HASH_CTX can be reused immediately by submitting with HASH_FIRST.

For example, you would submit hashes with the following flags for the following numbers of buffers:

- one buffer: HASH_FIRST | HASH_LAST (or, equivalently, HASH_ENTIRE)

- two buffers: HASH_FIRST, HASH_LAST
- three buffers: HASH_FIRST, HASH_UPDATE, HASH_LAST etc.

The order in which SHA256_CTX objects are returned is in general different from the order in which they are submitted.

A few possible error conditions exist:

- Submitting flags other than the allowed entire/first/update/last values
- Submitting a context that is currently being managed by a SHA256_HASH_CTX_MGR.
- Submitting a context after HASH_LAST is used but before HASH_FIRST is set.

These error conditions are reported by returning the SHA256_HASH_CTX immediately after a submit with its error member set to a non-zero error code (defined in multi_buffer.h). No changes are made to the SHA256_HASH_CTX_MGR in the case of an error; no processing is done for other hashes.

7.13.2 Function Documentation

7.13.2.1 SHA256_HASH_CTX* sha256_ctx_mgr_flush (SHA256_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA256 jobs and return when complete.

Requires SSE4.1 or AVX or AVX2

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.13.2.2 SHA256_HASH_CTX* sha256_ctx_mgr_flush_avx (SHA256_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA256 jobs and return when complete.

Requires AVX

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.13.2.3 SHA256_HASH_CTX* sha256_ctx_mgr_flush_avx2 (SHA256_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA256 jobs and return when complete.

Requires AVX2

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.13.2.4 SHA256_HASH_CTX* sha256_ctx_mgr_flush_avx512 (SHA256_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA256 jobs and return when complete.

Requires AVX512

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.13.2.5 SHA256_HASH_CTX* sha256_ctx_mgr_flush_avx512_ni (SHA256_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA256 jobs and return when complete.

Requires AVX512 and SHANI

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.13.2.6 SHA256_HASH_CTX* sha256_ctx_mgr_flush_sse (SHA256_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA256 jobs and return when complete.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.13.2.7 SHA256_HASH_CTX* sha256_ctx_mgr_flush_sse_ni (SHA256_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA256 jobs and return when complete.

Requires SSE4.1 and SHANI

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.13.2.8 void sha256_ctx_mgr_init (SHA256_HASH_CTX_MGR * *mgr*)

Initialize the SHA256 multi-buffer manager structure.

Requires SSE4.1 or AVX or AVX2

Parameters

mgr	Structure holding context level state info

Returns

void

7.13.2.9 void sha256_ctx_mgr_init_avx (SHA256_HASH_CTX_MGR * *mgr*)

Initialize the SHA256 multi-buffer manager structure.

Requires AVX

Parameters

mgr	Structure holding context level state info

Returns

void

7.13.2.10 void sha256_ctx_mgr_init_avx2 (SHA256_HASH_CTX_MGR * *mgr*)

Initialize the SHA256 multi-buffer manager structure.

Requires AVX2

Parameters

mgr	Structure holding context level state info

Returns

void

7.13.2.11 void sha256_ctx_mgr_init_avx512 (SHA256_HASH_CTX_MGR * *mgr*)

Initialize the SHA256 multi-buffer manager structure.

Requires AVX512

Parameters

mgr	Structure holding context level state info

Returns

void

7.13.2.12 void sha256_ctx_mgr_init_avx512_ni (SHA256_HASH_CTX_MGR * *mgr*)

Initialize the SHA256 multi-buffer manager structure.

Requires AVX512 and SHANI

Parameters

mgr	Structure holding context level state info

Returns

void

7.13.2.13 void sha256_ctx_mgr_init_sse (SHA256_HASH_CTX_MGR * *mgr*)

Initialize the context level SHA256 multi-buffer manager structure.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info

Returns

void

7.13.2.14 void sha256_ctx_mgr_init_sse_ni (SHA256_HASH_CTX_MGR * *mgr*)

Initialize the context level SHA256 multi-buffer manager structure.

Requires SSE4.1 and SHANI

Parameters

mgr	Structure holding context level state info

Returns

void

7.13.2.15 SHA256_HASH_CTX* sha256_ctx_mgr_submit (SHA256_HASH_CTX_MGR * *mgr*, SHA256_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA256 job to the multi-buffer manager.

Requires SSE4.1 or AVX or AVX2

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.13.2.16 SHA256_HASH_CTX* sha256_ctx_mgr_submit_avx (SHA256_HASH_CTX_MGR * *mgr*, SHA256_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA256 job to the multi-buffer manager.

Requires AVX

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.13.2.17 SHA256_HASH_CTX* sha256_ctx_mgr_submit_avx2 (SHA256_HASH_CTX_MGR * *mgr*, SHA256_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA256 job to the multi-buffer manager.

Requires AVX2

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.13.2.18 SHA256_HASH_CTX* sha256_ctx_mgr_submit_avx512 (SHA256_HASH_CTX_MGR * *mgr*, SHA256_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA256 job to the multi-buffer manager.

Requires AVX512

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.13.2.19 SHA256_HASH_CTX* sha256_ctx_mgr_submit_avx512_ni (SHA256_HASH_CTX_MGR * *mgr*, SHA256_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA256 job to the multi-buffer manager.

Requires AVX512 and SHANI

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.13.2.20 SHA256_HASH_CTX* sha256_ctx_mgr_submit_sse (SHA256_HASH_CTX_MGR * *mgr*, SHA256_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA256 job to the context level multi-buffer manager.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.13.2.21 SHA256_HASH_CTX* sha256_ctx_mgr_submit_sse_ni (SHA256_HASH_CTX_MGR * *mgr*, SHA256_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA256 job to the context level multi-buffer manager.

Requires SSE4.1 and SHANI

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.14 sha512_mb.h File Reference

Single/Multi-buffer CTX API SHA512 function prototypes and structures.

```
#include <stdint.h>
#include "multi_buffer.h"
#include "types.h"
#include <stdbool.h>
```

Data Structures

- struct SHA512_JOB

  Scheduler layer - Holds info describing a single SHA512 job for the multi-buffer manager.
- struct SHA512_MB_ARGS_X8

  Scheduler layer - Holds arguments for submitted SHA512 job.
- struct SHA512_LANE_DATA

  Scheduler layer - Lane data.
- struct SHA512_MB_JOB_MGR

  Scheduler layer - Holds state for multi-buffer SHA512 jobs.
- struct SHA512_HASH_CTX_MGR

  Context layer - Holds state for multi-buffer SHA512 jobs.
- struct SHA512_HASH_CTX

  Context layer - Holds info describing a single SHA512 job for the multi-buffer CTX manager.

Functions

- void sha512_ctx_mgr_init_sse (SHA512_HASH_CTX_MGR *mgr)

  Initialize the context level SHA512 multi-buffer manager structure.
- SHA512_HASH_CTX * sha512_ctx_mgr_submit_sse (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA512 job to the context level multi-buffer manager.
- SHA512_HASH_CTX * sha512_ctx_mgr_flush_sse (SHA512_HASH_CTX_MGR *mgr)

  Finish all submitted SHA512 jobs and return when complete.
- void sha512_ctx_mgr_init_avx (SHA512_HASH_CTX_MGR *mgr)

  Initialize the SHA512 multi-buffer manager structure.
- SHA512_HASH_CTX * sha512_ctx_mgr_submit_avx (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA512 job to the multi-buffer manager.
- SHA512_HASH_CTX * sha512_ctx_mgr_flush_avx (SHA512_HASH_CTX_MGR *mgr)

  Finish all submitted SHA512 jobs and return when complete.
- void sha512_ctx_mgr_init_avx2 (SHA512_HASH_CTX_MGR *mgr)

  Initialize the SHA512 multi-buffer manager structure.
- SHA512_HASH_CTX * sha512_ctx_mgr_submit_avx2 (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA512 job to the multi-buffer manager.
- SHA512_HASH_CTX * sha512_ctx_mgr_flush_avx2 (SHA512_HASH_CTX_MGR *mgr)

  Finish all submitted SHA512 jobs and return when complete.
- void sha512_ctx_mgr_init_avx512 (SHA512_HASH_CTX_MGR *mgr)

  Initialize the SHA512 multi-buffer manager structure.
- SHA512_HASH_CTX * sha512_ctx_mgr_submit_avx512 (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA512 job to the multi-buffer manager.
- SHA512_HASH_CTX * sha512_ctx_mgr_flush_avx512 (SHA512_HASH_CTX_MGR *mgr)

  Finish all submitted SHA512 jobs and return when complete.
- void sha512_ctx_mgr_init_sb_sse4 (SHA512_HASH_CTX_MGR *mgr)

  Initialize the SHA512 multi-buffer manager structure.
- SHA512_HASH_CTX * sha512_ctx_mgr_submit_sb_sse4 (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

  Submit a new SHA512 job to the multi-buffer manager.
- SHA512_HASH_CTX * sha512_ctx_mgr_flush_sb_sse4 (SHA512_HASH_CTX_MGR *mgr)

  Finish all submitted SHA512 jobs and return when complete.
- void sha512_ctx_mgr_init (SHA512_HASH_CTX_MGR *mgr)

  Initialize the SHA512 multi-buffer manager structure.
- SHA512_HASH_CTX * sha512_ctx_mgr_submit (SHA512_HASH_CTX_MGR *mgr, SHA512_HASH_CTX *ctx, const void *buffer, uint32_t len, HASH_CTX_FLAG flags)

Submit a new SHA512 job to the multi-buffer manager.

- SHA512_HASH_CTX ∗ sha512_ctx_mgr_flush (SHA512_HASH_CTX_MGR ∗mgr)

  Finish all submitted SHA512 jobs and return when complete.

7.14.1 Detailed Description

Single/Multi-buffer CTX API SHA512 function prototypes and structures. Interface for single and multi-buffer SHA512 functions

Single/Multi-buffer SHA512 Entire or First-Update..Update-Last

The interface to this single/multi-buffer hashing code is carried out through the context-level (CTX) init, submit and flush functions and the SHA512_HASH_CTX_MGR and SHA512_HASH_CTX objects. Numerous SHA512_HASH_CTX objects may be instantiated by the application for use with a single SHA512_HASH_CTX_MGR.

The CTX interface functions carry out the initialization and padding of the jobs entered by the user and add them to the multi-buffer manager. The lower level "scheduler" layer then processes the jobs in an out-of-order manner. The scheduler layer functions are internal and are not intended to be invoked directly. Jobs can be submitted to a CTX as a complete buffer to be hashed, using the HASH_ENTIRE flag, or as partial jobs which can be started using the HASH_FIRST flag, and later resumed or finished using the HASH_UPDATE and HASH_LAST flags respectively.

Note: The submit function does not require data buffers to be block sized.

The SHA512 CTX interface functions are available for 5 architectures: multi-buffer SSE, AVX, AVX2, AVX512 and single-buffer SSE4 (which is used in the same way as the multi-buffer code). In addition, a multibinary interface is provided, which selects the appropriate architecture-specific function at runtime. This multibinary interface selects the single buffer SSE4 functions when the platform is detected to be Silvermont.

Usage: The application creates a SHA512_HASH_CTX_MGR object and initializes it with a call to sha512_ctx_mgr_init∗() function, where henceforth "∗" stands for the relevant suffix for each architecture; _sse, _avx, _avx2, _avx512(or no suffix for the multibinary version). The SHA512_HASH_CTX_MGR object will be used to schedule processor resources, with up to 2 SHA512_HASH_CTX objects (or 4 in the AVX2 case, 8 in the AVX512 case) being processed at a time.

Each SHA512_HASH_CTX must be initialized before first use by the hash_ctx_init macro defined in multi_buffer.h. After initialization, the application may begin computing a hash by giving the SHA512_HASH_CTX to a SHA512_HASH_CTX_MGR using the submit functions sha512_ctx_mgr_submit∗() with the HASH_FIRST flag set. When the SHA512_HASH_CTX is returned to the application (via this or a later call to sha512_ctx_mgr_submit∗() or sha512_ctx_mgr_flush∗()), the application can then re-submit it with another call to sha512_ctx_mgr_submit∗(), but without the HASH_FIRST flag set.

Ideally, on the last buffer for that hash, sha512_ctx_mgr_submit_sse is called with HASH_LAST, although it is also possible to submit the hash with HASH_LAST and a zero length if necessary. When a SHA512_HASH_CTX is returned after having been submitted with HASH_LAST, it will contain a valid hash. The SHA512_HASH_CTX can be reused immediately by submitting with HASH_FIRST.

For example, you would submit hashes with the following flags for the following numbers of buffers:

- one buffer: HASH_FIRST | HASH_LAST (or, equivalently, HASH_ENTIRE)

- two buffers: HASH_FIRST, HASH_LAST
- three buffers: HASH_FIRST, HASH_UPDATE, HASH_LAST etc.

The order in which SHA512_CTX objects are returned is in general different from the order in which they are submitted.

A few possible error conditions exist:

- Submitting flags other than the allowed entire/first/update/last values
- Submitting a context that is currently being managed by a SHA512_HASH_CTX_MGR. (Note: This error case is not applicable to the single buffer SSE4 version)
- Submitting a context after HASH_LAST is used but before HASH_FIRST is set.

These error conditions are reported by returning the SHA512_HASH_CTX immediately after a submit with its error member set to a non-zero error code (defined in multi_buffer.h). No changes are made to the SHA512_HASH_CTX_MGR in the case of an error; no processing is done for other hashes.

7.14.2 Function Documentation

7.14.2.1 SHA512_HASH_CTX* sha512_ctx_mgr_flush (SHA512_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA512 jobs and return when complete.

Requires SSE4.1 or AVX or AVX2 or AVX512

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.14.2.2 SHA512_HASH_CTX* sha512_ctx_mgr_flush_avx (SHA512_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA512 jobs and return when complete.

Requires AVX

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.14.2.3 SHA512_HASH_CTX* sha512_ctx_mgr_flush_avx2 (SHA512_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA512 jobs and return when complete.

Requires AVX2

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.14.2.4 SHA512_HASH_CTX* sha512_ctx_mgr_flush_avx512 (SHA512_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA512 jobs and return when complete.

Requires AVX512

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.14.2.5 SHA512_HASH_CTX* sha512_ctx_mgr_flush_sb_sse4 (SHA512_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA512 jobs and return when complete.

Requires SSE4

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.14.2.6 SHA512_HASH_CTX* sha512_ctx_mgr_flush_sse (SHA512_HASH_CTX_MGR * *mgr*)

Finish all submitted SHA512 jobs and return when complete.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info

Returns

NULL if no jobs to complete or pointer to jobs structure.

7.14.2.7 void sha512_ctx_mgr_init (SHA512_HASH_CTX_MGR * *mgr*)

Initialize the SHA512 multi-buffer manager structure.

Requires SSE4.1 or AVX or AVX2 or AVX512

Parameters

mgr	Structure holding context level state info

Returns

void

7.14.2.8 void sha512_ctx_mgr_init_avx (SHA512_HASH_CTX_MGR * *mgr*)

Initialize the SHA512 multi-buffer manager structure.

Requires AVX

Parameters

mgr	Structure holding context level state info

Returns

void

7.14.2.9 void sha512_ctx_mgr_init_avx2 (SHA512_HASH_CTX_MGR * *mgr*)

Initialize the SHA512 multi-buffer manager structure.

Requires AVX2

Parameters

mgr	Structure holding context level state info

Returns

void

7.14.2.10 void sha512_ctx_mgr_init_avx512 (SHA512_HASH_CTX_MGR * *mgr*)

Initialize the SHA512 multi-buffer manager structure.

Requires AVX512

Parameters

mgr	Structure holding context level state info

Returns

void

7.14.2.11 void sha512_ctx_mgr_init_sb_sse4 (SHA512_HASH_CTX_MGR * *mgr*)

Initialize the SHA512 multi-buffer manager structure.

Requires SSE4

Parameters

mgr	Structure holding context level state info

Returns

void

7.14.2.12 void sha512_ctx_mgr_init_sse (SHA512_HASH_CTX_MGR * *mgr*)

Initialize the context level SHA512 multi-buffer manager structure.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info

Returns

void

7.14.2.13 SHA512_HASH_CTX* sha512_ctx_mgr_submit (SHA512_HASH_CTX_MGR * *mgr*, SHA512_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA512 job to the multi-buffer manager.

Requires SSE4.1 or AVX or AVX2 or AVX512

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.14.2.14 SHA512_HASH_CTX* sha512_ctx_mgr_submit_avx (SHA512_HASH_CTX_MGR * *mgr*, SHA512_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA512 job to the multi-buffer manager.

Requires AVX

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.14.2.15 SHA512_HASH_CTX* sha512_ctx_mgr_submit_avx2 (SHA512_HASH_CTX_MGR * *mgr*, SHA512_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA512 job to the multi-buffer manager.

Requires AVX2

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.14.2.16 SHA512_HASH_CTX* sha512_ctx_mgr_submit_avx512 (SHA512_HASH_CTX_MGR * *mgr*, SHA512_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA512 job to the multi-buffer manager.

Requires AVX512

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.14.2.17 SHA512_HASH_CTX* sha512_ctx_mgr_submit_sb_sse4 (SHA512_HASH_CTX_MGR * *mgr*, SHA512_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA512 job to the multi-buffer manager.

Requires SSE4

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

7.14.2.18 SHA512_HASH_CTX* sha512_ctx_mgr_submit_sse (SHA512_HASH_CTX_MGR * *mgr*, SHA512_HASH_CTX * *ctx*, const void * *buffer*, uint32_t *len*, HASH_CTX_FLAG *flags*)

Submit a new SHA512 job to the context level multi-buffer manager.

Requires SSE4.1

Parameters

mgr	Structure holding context level state info
ctx	Structure holding ctx job info
buffer	Pointer to buffer to be processed
len	Length of buffer (in bytes) to be processed
flags	Input flag specifying job type (first, update, last or entire)

Returns

NULL if no jobs complete or pointer to jobs structure.

CHAPTER 8
EXAMPLE DOCUMENTATION

8.1 sha1_multi_buffer_example.c

Example of multi-buffer hashing using sha1_mb.

```
/*****************************************************************
  Copyright(c) 2011-2015 Intel Corporation All rights reserved.

  Redistribution and use in source and binary forms, with or without
  modification, are permitted provided that the following conditions
  are met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in
      the documentation and/or other materials provided with the
      distribution.
    * Neither the name of Intel Corporation nor the names of its
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*****************************************************************/
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "sha1_mb.h"

// Test messages
#define TST_STR "0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWX"
uint8_t msg1[] = "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq";
uint8_t msg2[] = "0123456789:;<=>?@ABCDEFGHIJKLMNO";
uint8_t msg3[] = TST_STR TST_STR "0123456789:;<";
uint8_t msg4[] = TST_STR TST_STR TST_STR "0123456789:;<=>?@ABCDEFGHIJKLMNOPQR";
uint8_t msg5[] = TST_STR TST_STR TST_STR TST_STR TST_STR "0123456789:;<=>?";
uint8_t msg6[] =
    TST_STR TST_STR TST_STR TST_STR TST_STR TST_STR "0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTU";
uint8_t msg7[] = "";

// Expected digests
uint32_t dgst1[] = { 0x84983E44, 0x1C3BD26E, 0xBAAE4AA1, 0xF95129E5, 0xE54670F1 };
uint32_t dgst2[] = { 0xB7C66452, 0x0FD122B3, 0x55D539F2, 0xA35E6FAA, 0xC2A5A11D };
uint32_t dgst3[] = { 0x127729B6, 0xA8B2F8A0, 0xA4DDC819, 0x08E1D8B3, 0x67CEEA55 };
uint32_t dgst4[] = { 0xFDDE2D00, 0xABD5B7A3, 0x699DE6F2, 0x3FF1D1AC, 0x3B872AC2 };
uint32_t dgst5[] = { 0xE7FCA85C, 0xA4AB3740, 0x6A180B32, 0x0B8D362C, 0x622A96E6 };
uint32_t dgst6[] = { 0x505B0686, 0xE1ACDF42, 0xB3588B5A, 0xB043D52C, 0x6D8C7444 };
uint32_t dgst7[] = { 0xDA39A3EE, 0x5E6B4B0D, 0x3255BFEF, 0x95601890, 0xAFD80709 };

uint8_t *msgs[] = { msg1, msg2, msg3, msg4, msg5, msg6, msg7 };
```

```
uint32_t *expected_digest[] = { dgst1, dgst2, dgst3, dgst4, dgst5, dgst6, dgst7 };

int check_job(uint32_t * ref, uint32_t * good, int words)
{
        int i;
        for (i = 0; i < words; i++)
                if (good[i] != ref[i])
                        return 1;

        return 0;
}

#define MAX_MSGS 7

int main(void)
{
        SHA1_HASH_CTX_MGR *mgr = NULL;
        SHA1_HASH_CTX ctxpool[MAX_MSGS];
        SHA1_HASH_CTX *p_job;
        int i, checked = 0, failed = 0;
        int n = sizeof(msgs) / sizeof(msgs[0]);

        posix_memalign((void *)&mgr, 16, sizeof(SHA1_HASH_CTX_MGR));

        // Initialize multi-buffer manager
        sha1_ctx_mgr_init(mgr);

        for (i = 0; i < n; i++) {
                hash_ctx_init(&ctxpool[i]);
                ctxpool[i].user_data = (void *)expected_digest[i];

                p_job = sha1_ctx_mgr_submit(mgr, &ctxpool[i], msgs[i],
                                            strlen((char *)msgs[i]), HASH_ENTIRE);

                if (p_job) {    // If we have finished a job, process it
                        checked++;
                        failed += check_job(p_job->job.result_digest, p_job->
      user_data,
                                            SHA1_DIGEST_NWORDS);
                }
        }

        // Finish remaining jobs
        while (NULL != (p_job = sha1_ctx_mgr_flush(mgr))) {
                checked++;
                failed += check_job(p_job->job.result_digest, p_job->user_data,
                                    SHA1_DIGEST_NWORDS);
        }

        printf("Example multi-buffer sha1 completed=%d, failed=%d\n", checked, failed);
        return failed;
}
```

INDEX