

**NAME**

**SHA512\_Init**, **SHA512\_Update**, **SHA512\_Final**, **SHA512\_End**, **SHA512\_File**, **SHA512\_FileChunk**, **SHA512\_Data**, **SHA384\_Init**, **SHA384\_Update**, **SHA384\_Final**, **SHA384\_End**, **SHA384\_File**, **SHA384\_FileChunk**, **SHA384\_Data**, **SHA512\_224\_Init**, **SHA512\_224\_Update**, **SHA512\_224\_Final**, **SHA512\_224\_End**, **SHA512\_224\_File**, **SHA512\_224\_FileChunk**, **SHA512\_224\_Data**, **SHA512\_256\_Init**, **SHA512\_256\_Update**, **SHA512\_256\_Final**, **SHA512\_256\_End**, **SHA512\_256\_File**, **SHA512\_256\_FileChunk**, **SHA512\_256\_Data** - calculate the FIPS 180-4 “SHA-512” family of message digests

**LIBRARY**

Message Digest (MD4, MD5, etc.) Support Library (libmd, -lmd)

**SYNOPSIS**

```
#include <sys/types.h>
```

```
#include <sha512.h>
```

*void*

```
SHA512_Init(SHA512_CTX *context);
```

*void*

```
SHA512_Update(SHA512_CTX *context, const unsigned char *data, size_t len);
```

*void*

```
SHA512_Final(unsigned char digest[64], SHA512_CTX *context);
```

*char \**

```
SHA512_End(SHA512_CTX *context, char *buf);
```

*char \**

```
SHA512_File(const char *filename, char *buf);
```

*char \**

```
SHA512_FileChunk(const char *filename, char *buf, off_t offset, off_t length);
```

*char \**

```
SHA512_Data(const unsigned char *data, unsigned int len, char *buf);
```

```
#include <sha384.h>
```

*void*

**SHA384\_Init**(*SHA384\_CTX* \*context);

*void*

**SHA384\_Update**(*SHA384\_CTX* \*context, const unsigned char \*data, size\_t len);

*void*

**SHA384\_Final**(unsigned char digest[48], *SHA384\_CTX* \*context);

*char* \*

**SHA384\_End**(*SHA384\_CTX* \*context, char \*buf);

*char* \*

**SHA384\_File**(const char \*filename, char \*buf);

*char* \*

**SHA384\_FileChunk**(const char \*filename, char \*buf, off\_t offset, off\_t length);

*char* \*

**SHA384\_Data**(const unsigned char \*data, unsigned int len, char \*buf);

**#include** <sha512t.h>

*void*

**SHA512\_224\_Init**(*SHA512\_CTX* \*context);

*void*

**SHA512\_224\_Update**(*SHA512\_CTX* \*context, const unsigned char \*data, size\_t len);

*void*

**SHA512\_224\_Final**(unsigned char digest[32], *SHA512\_CTX* \*context);

*char* \*

**SHA512\_224\_End**(*SHA512\_CTX* \*context, char \*buf);

*char* \*

**SHA512\_224\_File**(const char \*filename, char \*buf);

*char* \*

**SHA512\_224\_FileChunk**(const char \*filename, char \*buf, off\_t offset, off\_t length);

*char \**

**SHA512\_224\_Data**(*const unsigned char \*data, unsigned int len, char \*buf*);

*void*

**SHA512\_256\_Init**(*SHA512\_CTX \*context*);

*void*

**SHA512\_256\_Update**(*SHA512\_CTX \*context, const unsigned char \*data, size\_t len*);

*void*

**SHA512\_256\_Final**(*unsigned char digest[32], SHA512\_CTX \*context*);

*char \**

**SHA512\_256\_End**(*SHA512\_CTX \*context, char \*buf*);

*char \**

**SHA512\_256\_File**(*const char \*filename, char \*buf*);

*char \**

**SHA512\_256\_FileChunk**(*const char \*filename, char \*buf, off\_t offset, off\_t length*);

*char \**

**SHA512\_256\_Data**(*const unsigned char \*data, unsigned int len, char \*buf*);

## DESCRIPTION

The **SHA512\_** functions calculate a 512-bit cryptographic checksum (digest) for any number of input bytes. A cryptographic checksum is a one-way hash function; that is, it is computationally impractical to find the input corresponding to a particular output. This net result is a "fingerprint" of the input-data, which does not disclose the actual input.

The **SHA512\_Init()**, **SHA512\_Update()**, and **SHA512\_Final()** functions are the core functions. Allocate an *SHA512\_CTX*, initialize it with **SHA512\_Init()**, run over the data with **SHA512\_Update()**, and finally extract the result using **SHA512\_Final()**, which will also erase the *SHA512\_CTX*.

**SHA512\_End()** is a wrapper for **SHA512\_Final()** which converts the return value to a 129-character (including the terminating `'\0'`) ASCII string which represents the 512 bits in hexadecimal.

**SHA512\_File()** calculates the digest of a file, and uses **SHA512\_End()** to return the result. If the file cannot be opened, a null pointer is returned. **SHA512\_FileChunk()** is similar to **SHA512\_File()**, but it only calculates the digest over a byte-range of the file specified, starting at *offset* and spanning *length*

bytes. If the *length* parameter is specified as 0, or more than the length of the remaining part of the file, **SHA512\_FileChunk()** calculates the digest from *offset* to the end of file. **SHA512\_Data()** calculates the digest of a chunk of data in memory, and uses **SHA512\_End()** to return the result.

When using **SHA512\_End()**, **SHA512\_File()**, or **SHA512\_Data()**, the *buf* argument can be a null pointer, in which case the returned string is allocated with `malloc(3)` and subsequently must be explicitly deallocated using `free(3)` after use. If the *buf* argument is non-null it must point to at least 129 characters of buffer space.

The `SHA384_`, `SHA512_224_`, and `SHA512_256_` functions are identical to the `SHA512_` functions except they use a different initial hash value and the output is truncated to 384, 224, and 256 bits respectively.

**SHA384\_End()** is a wrapper for **SHA384\_Final()** which converts the return value to a 97-character (including the terminating `'\0'`) ASCII string which represents the 384 bits in hexadecimal.

**SHA512\_224\_End()** is a wrapper for **SHA512\_Final()** which converts the return value to a 57-character (including the terminating `'\0'`) ASCII string which represents the 224 bits in hexadecimal.

**SHA512\_224\_End()** is a wrapper for **SHA512\_Final()** which converts the return value to a 57-character (including the terminating `'\0'`) ASCII string which represents the 224 bits in hexadecimal.

**SHA512\_256\_End()** is a wrapper for **SHA512\_Final()** which converts the return value to a 65-character (including the terminating `'\0'`) ASCII string which represents the 256 bits in hexadecimal.

## ERRORS

The **SHA512\_End()** function called with a null *buf* argument may fail and return NULL if:

[ENOMEM]            Insufficient storage space is available.

The **SHA512\_File()** and **SHA512\_FileChunk()** may return NULL when underlying `open(2)`, `fstat(2)`, `lseek(2)`, or `SHA512_End(3)` fail.

## SEE ALSO

`md4(3)`, `md5(3)`, `ripemd(3)`, `sha(3)`, `sha256(3)`, `sha512(3)`, `skein(3)`

## HISTORY

These functions appeared in FreeBSD 9.0.

## AUTHORS

The core hash routines were implemented by Colin Percival based on the published FIPS 180-2 standard.

**BUGS**

No method is known to exist which finds two files having the same hash value, nor to find a file with a specific hash value. There is on the other hand no guarantee that such a method does not exist.