

**NAME**

**SHA224\_Init**, **SHA224\_Update**, **SHA224\_Final**, **SHA224\_End**, **SHA224\_File**, **SHA224\_FileChunk**, **SHA224\_Data**, **SHA256\_Init**, **SHA256\_Update**, **SHA256\_Final**, **SHA256\_End**, **SHA256\_File**, **SHA256\_FileChunk**, **SHA256\_Data** - calculate the FIPS 180-2 ‘SHA-256’ (or SHA-224) message digest

**LIBRARY**

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

**SYNOPSIS**

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

```
#include <sha224.h>
```

*void*

```
SHA224_Init(SHA224_CTX *context);
```

*void*

```
SHA224_Update(SHA224_CTX *context, const unsigned char *data, size_t len);
```

*void*

```
SHA224_Final(unsigned char digest[32], SHA224_CTX *context);
```

*char \**

```
SHA224_End(SHA224_CTX *context, char *buf);
```

*char \**

```
SHA224_File(const char *filename, char *buf);
```

*char \**

```
SHA224_FileChunk(const char *filename, char *buf, off_t offset, off_t length);
```

*char \**

```
SHA224_Data(const unsigned char *data, unsigned int len, char *buf);
```

```
#include <sha256.h>
```

*void*

```
SHA256_Init(SHA256_CTX *context);
```

*void*

**SHA256\_Update**(*SHA256\_CTX* \*context, const unsigned char \*data, size\_t len);

void

**SHA256\_Final**(unsigned char digest[32], *SHA256\_CTX* \*context);

char \*

**SHA256\_End**(*SHA256\_CTX* \*context, char \*buf);

char \*

**SHA256\_File**(const char \*filename, char \*buf);

char \*

**SHA256\_FileChunk**(const char \*filename, char \*buf, off\_t offset, off\_t length);

char \*

**SHA256\_Data**(const unsigned char \*data, unsigned int len, char \*buf);

## DESCRIPTION

The **SHA256\_** functions calculate a 256-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 **SHA256\_Init()**, **SHA256\_Update()**, and **SHA256\_Final()** functions are the core functions. Allocate an *SHA256\_CTX*, initialize it with **SHA256\_Init()**, run over the data with **SHA256\_Update()**, and finally extract the result using **SHA256\_Final()**, which will also erase the *SHA256\_CTX*.

**SHA256\_End()** is a wrapper for **SHA256\_Final()** which converts the return value to a 65-character (including the terminating `'\0'`) ASCII string which represents the 256 bits in hexadecimal.

**SHA256\_File()** calculates the digest of a file, and uses **SHA256\_End()** to return the result. If the file cannot be opened, a null pointer is returned. **SHA256\_FileChunk()** is similar to **SHA256\_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, **SHA256\_FileChunk()** calculates the digest from *offset* to the end of file. **SHA256\_Data()** calculates the digest of a chunk of data in memory, and uses **SHA256\_End()** to return the result.

When using **SHA256\_End()**, **SHA256\_File()**, or **SHA256\_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 65 characters

of buffer space.

SHA224 is identical SHA256, except it has slightly different initialization vectors, and is truncated to a shorter digest.

## ERRORS

The **SHA256\_End()** function called with a null buf argument may fail and return NULL if:

[ENOMEM]           Insufficient storage space is available.

The **SHA256\_File()** and **SHA256\_FileChunk()** may return NULL when underlying **open(2)**, **fstat(2)**, **lseek(2)**, or **SHA256\_End(3)** fail.

## SEE ALSO

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

## HISTORY

These functions appeared in FreeBSD 6.0.

## 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.