NAME

libmp - traditional BSD multiple precision integer arithmetic library

SYNOPSIS

#include <mp.h>

Function prototypes are given in the main body of the text.

Applications using this interface must be linked with **-l***mp* (this library) and **-l***crypto* (crypto(3)).

DESCRIPTION

This interface is obsolete in favor of the crypto(3) *BIGNUM* **library.**

libmp is the traditional BSD multiple precision integer arithmetic library. It has a number of problems, and is unsuitable for use in any programs where reliability is a concern. It is provided here for compatibility only.

These routines perform arithmetic on integers of arbitrary precision stored using the defined type *MINT*. Pointers to *MINT* are initialized using **mp_itom**() or **mp_xtom**(), and must be recycled with **mp_mfree**() when they are no longer needed. Routines which store a result in one of their arguments expect that the latter has also been initialized prior to being passed to it. The following routines are defined and implemented:

MINT ***mp_itom**(*short n*);

MINT ***mp_xtom**(*const char *s*);

char ***mp_mtox**(*const MINT *mp*);

void **mp_mfree**(*MINT *mp*);

mp_itom() returns an *MINT* with the value of *n*. **mp_xtom**() returns an *MINT* with the value of *s*, which is treated to be in hexadecimal. The return values from **mp_itom**() and **mp_xtom**() must be released with **mp_mfree**() when they are no longer needed. **mp_mtox**() returns a null-terminated hexadecimal string having the value of *mp*; its return value must be released with **free**() (free(3)) when it is no longer needed.

void **mp_madd**(*const MINT *mp1*, *const MINT *mp2*, *MINT *rmp*);

void **mp_msub**(*const MINT *mp1*, *const MINT *mp2*, *MINT *rmp*);

void **mp_mult**(*const MINT *mp1*, *const MINT *mp2*, *MINT *rmp*);

mp_madd(), **mp_msub**(), and **mp_mult**() store the sum, difference, or product, respectively, of *mp1* and *mp2* in *rmp*.

void **mp_mdiv**(*const MINT *nmp*, *const MINT *dmp*, *MINT *qmp*, *MINT *rmp*);

void **mp_sdiv**(*const MINT *nmp*, *short d*, *MINT *qmp*, *short *ro*);

mp_mdiv() computes the quotient and remainder of *nmp* and *dmp* and stores the result in *qmp* and *rmp*, respectively. **mp_sdiv**() is similar to **mp_mdiv**() except the divisor (*dmp* or *d*) and remainder (*rmp* or *ro*) are ordinary integers.

void **mp_pow**(*const MINT *bmp*, *const MINT *emp*, *const MINT *mmp*, *MINT *rmp*);

void **mp_rpow**(*const MINT *bmp*, *short e*, *MINT *rmp*);

mp_rpow() computes the result of *bmp* raised to the *emp*th power and reduced modulo *mmp*; the result is stored in *rmp*. **mp_pow**() computes the result of *bmp* raised to the *e*th power and stores the result in *rmp*.

void **mp_min**(*MINT *mp*);

void **mp_mout**(*const MINT *mp*);

mp_min() reads a line from standard input, tries to interpret it as a decimal number, and if successful, stores the result in *mp*. **mp_mout**() prints the value, in decimal, of *mp* to standard output (without a trailing newline).

void **mp_gcd**(*const MINT *mp1*, *const MINT *mp2*, *MINT *rmp*);

mp_gcd() computes the greatest common divisor of *mp1* and *mp2* and stores the result in *rmp*.

int **mp_mcmp**(*const MINT *mp1*, *const MINT *mp2*);

mcmp compares the values of *mp1* and *mp2* and returns 0 if the two values are equal, a value greater than 0 if *mp1* is greater than *mp2*, and a value less than 0 if *mp2* is greater than *mp1*.

void **mp_move**(*const MINT *smp*, *MINT *tmp*);

mp_move() copies the value of *smp* to *tmp* (both values must be initialized).

void **mp_msqrt**(*const MINT *nmp, MINT *xmp, MINT *rmp*);

mp_msqrt() computes the square root and remainder of *nmp* and stores them in *xmp* and *rmp*, respectively.

IMPLEMENTATION NOTES

This version of **libmp** is implemented in terms of the crypto(3) *BIGNUM* library.

DIAGNOSTICS

Running out of memory or illegal operations result in error messages on standard error and a call to abort(3).

SEE ALSO

abort(3), bn(3), crypto(3), free(3), malloc(3), math(3)

HISTORY

A **libmp** library appeared in 4.3BSD. FreeBSD 2.2 shipped with a **libmp** implemented in terms of **libgmp**. This implementation appeared in FreeBSD 5.0.

BUGS

Errors are reported via output to standard error and abnormal program termination instead of via return values. The application cannot control this behavior.

It is not clear whether the string returned by **mp_mtox**() may be written to by the caller. This implementation allows it, but others may not. Ideally, **mp_mtox**() would take a pointer to a buffer to fill in.

It is not clear whether using the same variable as both source and destination in a single invocation is permitted. Some of the calls in this implementation allow this, while others do not.