AMCL

Generated by Doxygen 1.8.13

Contents

Chapter 1

Apache Milagro Crypto Library (AMCL)

AMCL is a standards compliant C cryptographic library with no external dependencies, specifically designed to support the Internet of Things (IoT).

AMCL is provided in *C* language but includes a `Python` wrapper.for some components as an aid for development work.

1.1 Project page

The official project page is hosted at `Apache Milagro (incubating)`

1.2 License

1.3 Platforms

The software can be compiled and installed for these operating systems;

- Linux
- Windows
- Mac OS

1.4 Downloads

The source code is available from here;

git clone `https://github.com/milagro-crypto/milagro-crypto-c`

1.5 Installation

There are instructions for building for Linux, Mac OS and Windows.

Chapter 2

Linux

Software dependencies

CMake is required to build the library and can usually be installed from the operating system package manager.

- sudo apt-get install cmake

If not, then you can download it from www.cmake.org

In order to use the Python language wrapper install `Python`

The C Foreign Function Interface for Python `CFFI` module is also required if you wish to use the Python module.

- sudo pip install cffi

In order to build the documentation `doxygen` is required.

Quick Start

A Makefile is present at the project root that reads the options defined in config.mk. Change these options and then type `make` to build and test the library.

If `docker` is installed then type `make dbuild` to build and test the library in a docker container.

Manual build

The default build is for 64 bit machines, Elliptic curve BN254CX and curve type Weierstrass

1. mkdir target/build
2. cd target/build
3. cmake -D CMAKE_INSTALL_PREFIX=/opt/amcl ../..
4. export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./
5. make
6. make test
7. make doc
8. sudo make install

The build can be configured using by setting flags on the command line i.e.

1. cmake -D CMAKE_INSTALL_PREFIX=/opt/amcl -D WORD_LENGTH=32 ../..

list available CMake options

1. cmake -LH

Uninstall software

- sudo make uninstall

Building an installer

After having built the libraries you can build a binary installer and a source distribution by running this command

- make package

Chapter 3

Mac OS

Software dependencies

Install Homebrew

- ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

Install cmake

- brew install cmake

In order to use the Python language wrapper install Python

The C Foreign Function Interface for Python CFFI module is also required if you wish to use the Python module.

- brew install pkg-config libffi
- sudo pip install cffi

In order to build the documentation doxygen is required.

- brew install doxygen

Build Instructions

The default build is for 64 bit machines, Elliptic curve BN254CX and curve type Weierstrass

1. mkdir -p target/build
2. cd target/build
3. cmake ../..
4. make
5. make test
6. make doc
7. sudo make install

The build can be configured using by setting flags on the command line i.e.

1. cmake -DWORD_LENGTH=32 ../..

Uninstall software

- sudo make uninstall

Chapter 4

Windows

Software dependencies

Minimalist GNU for Windows MinGW provides the tool set used to build the library and should be installed. When the MinGW installer starts select the mingw32-base and mingw32-gcc-g++ components. From the menu select "Installation" -> "Apply Changes", then click "Apply". Finally add C:\MinGW\bin to the PATH variable.

CMake is required to build the library and can be downloaded from www.cmake.org

In order to use the Python language wrapper install Python

The C Foreign Function Interface for Python CFFI module is also required, if you wish to use the Python module.

- pip install cffi

In order to build the documentation doxygen is required.

Build Instructions

Start a command prompt as an administrator

The default build is for 64 bit machines, Elliptic curve BN254CX and curve type Weierstrass

1. mkdir target\build
2. cd target\build
3. cmake -G "MinGW Makefiles" ..\..
4. mingw32-make
5. mingw32-make test
6. mingw32-make doc
7. mingw32-make install

Post install append the PATH system variable to point to the install ./lib.

My Computer -> Properties -> Advanced > Environment Variables

The build can be configured using by setting flags on the command line i.e.

1. cmake -G "MinGW Makefiles" -DWORD_LENGTH=32 ../..

Uninstall software

- mingw32-make uninstall

Building an installer

After having built the libraries you can build a Windows installer using this command

- sudo mingw32-make package

In order for this to work `NSSI` has to have been installed

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

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

Chapter 7

Data Structure Documentation

7.1 amcl_aes Struct Reference

AES instance.

```
#include <amcl.h>
```

Data Fields

- int Nk
- int Nr
- int mode
- unsign32 fkey [60]
- unsign32 rkey [60]
- char f [16]

7.1.1 Field Documentation

7.1.1.1 f

```
char amcl_aes::f[16]
```

buffer for chaining vector

7.1.1.2 fkey

```
unsign32 amcl_aes::fkey[60]
```

subkeys for encrypton

7.1.1.3 mode

```
int amcl_aes::mode
```

AES mode of operation

7.1.1.4 Nk

```
int amcl_aes::Nk
```

AES Key Length

7.1.1.5 Nr

```
int amcl_aes::Nr
```

AES Number of rounds

7.1.1.6 rkey

```
unsign32 amcl_aes::rkey[60]
```

subkeys for decrypton

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

- amcl.h

7.2 csprng Struct Reference

Cryptographically secure pseudo-random number generator instance.

```
#include <amcl.h>
```

Data Fields

- unsign32 ira [NK]
- int rndptr
- unsign32 borrow
- int pool_ptr
- char pool [32]

7.2.1 Field Documentation

7.2.1.1 borrow

`unsign32 csprng::borrow`

borrow as a result of subtraction

7.2.1.2 ira

`unsign32 csprng::ira[NK]`

random number array

7.2.1.3 pool

`char csprng::pool[32]`

random pool

7.2.1.4 pool_ptr

`int csprng::pool_ptr`

pointer into random pool

7.2.1.5 rndptr

`int csprng::rndptr`

pointer into array

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

- amcl.h

7.3 ECP2_BLS381 Struct Reference

ECP2 Structure - Elliptic Curve Point over quadratic extension field.

`#include <ecp2_BLS381.h>`

Data Fields

- FP2_BLS381 x
- FP2_BLS381 y
- FP2_BLS381 z

7.3.1 Field Documentation

7.3.1.1 x

`FP2_BLS381 ECP2_BLS381::x`

x-coordinate of point

7.3.1.2 y

`FP2_BLS381 ECP2_BLS381::y`

y-coordinate of point

7.3.1.3 z

`FP2_BLS381 ECP2_BLS381::z`

z-coordinate of point

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

- ecp2_BLS381.h

7.4 ECP_BLS381 Struct Reference

ECP structure - Elliptic Curve Point over base field.

```
#include <ecp_BLS381.h>
```

Data Fields

- FP_BLS381 x
- FP_BLS381 y
- FP_BLS381 z

7.4.1 Field Documentation

7.4.1.1 x

`FP_BLS381 ECP_BLS381::x`

x-coordinate of point

7.4.1.2 y

`FP_BLS381 ECP_BLS381::y`

y-coordinate of point. Not needed for Montgomery representation

7.4.1.3 z

`FP_BLS381 ECP_BLS381::z`

z-coordinate of point

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

- ecp_BLS381.h

7.5 ECP_ED25519 Struct Reference

ECP structure - Elliptic Curve Point over base field.

`#include <ecp_ED25519.h>`

Data Fields

- FP_25519 x
- FP_25519 y
- FP_25519 z

7.5.1 Field Documentation

7.5.1.1 x

`FP_25519 ECP_ED25519::x`

x-coordinate of point

7.5.1.2 y

`FP_25519 ECP_ED25519::y`

y-coordinate of point. Not needed for Montgomery representation

7.5.1.3 z

`FP_25519 ECP_ED25519::z`

z-coordinate of point

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

- ecp_ED25519.h

7.6 ECP_GOLDILOCKS Struct Reference

ECP structure - Elliptic Curve Point over base field.

```
#include <ecp_GOLDILOCKS.h>
```

Data Fields

- FP_GOLDILOCKS x
- FP_GOLDILOCKS y
- FP_GOLDILOCKS z

7.6.1 Field Documentation

7.6.1.1 x

`FP_GOLDILOCKS ECP_GOLDILOCKS::x`

x-coordinate of point

7.6.1.2 y

`FP_GOLDILOCKS ECP_GOLDILOCKS::y`

y-coordinate of point. Not needed for Montgomery representation

7.6.1.3 z

`FP_GOLDILOCKS ECP_GOLDILOCKS::z`

z-coordinate of point

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

- ecp_GOLDILOCKS.h

7.7 ECP_NIST256 Struct Reference

ECP structure - Elliptic Curve Point over base field.

`#include <ecp_NIST256.h>`

Data Fields

- FP_NIST256 x
- FP_NIST256 y
- FP_NIST256 z

7.7.1 Field Documentation

7.7.1.1 x

`FP_NIST256 ECP_NIST256::x`

x-coordinate of point

7.7.1.2 y

`FP_NIST256 ECP_NIST256::y`

y-coordinate of point. Not needed for Montgomery representation

7.7.1.3 z

`FP_NIST256 ECP_NIST256::z`

z-coordinate of point

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

- ecp_NIST256.h

7.8 FP12_BLS381 Struct Reference

FP12 Structure - towered over three FP4.

`#include <fp12_BLS381.h>`

Data Fields

- FP4_BLS381 a
- FP4_BLS381 b
- FP4_BLS381 c
- int type

7.8.1 Field Documentation

7.8.1.1 a

`FP4_BLS381 FP12_BLS381::a`

first part of FP12

7.8.1.2 b

`FP4_BLS381 FP12_BLS381::b`

second part of FP12

7.8.1.3 c

`FP4_BLS381 FP12_BLS381::c`

third part of FP12

7.8.1.4 type

`int FP12_BLS381::type`

Type

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

- fp12_BLS381.h

7.9 FP2_BLS381 Struct Reference

FP2 Structure - quadratic extension field.

`#include <fp2_BLS381.h>`

Data Fields

- FP_BLS381 a
- FP_BLS381 b

7.9.1 Field Documentation

7.9.1.1 a

`FP_BLS381 FP2_BLS381::a`

real part of FP2

7.9.1.2 b

`FP_BLS381 FP2_BLS381::b`

imaginary part of FP2

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

- fp2_BLS381.h

7.10 FP4_BLS381 Struct Reference

FP4 Structure - towered over two FP2.

`#include <fp4_BLS381.h>`

Data Fields

- FP2_BLS381 a
- FP2_BLS381 b

7.10.1 Field Documentation

7.10.1.1 a

`FP2_BLS381 FP4_BLS381::a`

real part of FP4

7.10.1.2 b

`FP2_BLS381 FP4_BLS381::b`

imaginary part of FP4

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

- fp4_BLS381.h

7.11 FP_25519 Struct Reference

FP Structure - quadratic extension field.

`#include <fp_25519.h>`

Data Fields

- BIG_256_56 g
- sign32 XES

7.11.1 Field Documentation

7.11.1.1 g

`BIG_256_56 FP_25519::g`

Big representation of field element

7.11.1.2 XES

`sign32 FP_25519::XES`

Excess

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

- fp_25519.h

7.12 FP_BLS381 Struct Reference

FP Structure - quadratic extension field.

```
#include <fp_BLS381.h>
```

Data Fields

- BIG_384_58 g
- sign32 XES

7.12.1 Field Documentation

7.12.1.1 g

```
BIG_384_58 FP_BLS381::g
```

Big representation of field element

7.12.1.2 XES

```
sign32 FP_BLS381::XES
```

Excess

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

- fp_BLS381.h

7.13 FP_GOLDILOCKS Struct Reference

FP Structure - quadratic extension field.

```
#include <fp_GOLDILOCKS.h>
```

Data Fields

- BIG_448_58 g
- sign32 XES

7.13.1 Field Documentation

7.13.1.1 g

`BIG_448_58 FP_GOLDILOCKS::g`

Big representation of field element

7.13.1.2 XES

`sign32 FP_GOLDILOCKS::XES`

Excess

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

- fp_GOLDILOCKS.h

7.14 FP_NIST256 Struct Reference

FP Structure - quadratic extension field.

`#include <fp_NIST256.h>`

Data Fields

- BIG_256_56 g
- sign32 XES

7.14.1 Field Documentation

7.14.1.1 g

`BIG_256_56 FP_NIST256::g`

Big representation of field element

7.14.1.2 XES

`sign32 FP_NIST256::XES`

Excess

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

- fp_NIST256.h

7.15 gcm Struct Reference

GCM mode instance, using AES internally.

```
#include <amcl.h>
```

Data Fields

- unsign32 table [128][4]
- uchar stateX [16]
- uchar Y_0 [16]
- unsign32 lenA [2]
- unsign32 lenC [2]
- int status
- amcl_aes a

7.15.1 Field Documentation

7.15.1.1 a

```
amcl_aes gcm::a
```

Internal Instance of AMCL_AES cipher

7.15.1.2 lenA

```
unsign32 gcm::lenA[2]
```

GCM 64-bit length of header

7.15.1.3 lenC

```
unsign32 gcm::lenC[2]
```

GCM 64-bit length of ciphertext

7.15.1.4 stateX

```
uchar gcm::stateX[16]
```

GCM Internal State

7.15.1.5 status

```
int gcm::status
```

GCM Status

7.15.1.6 table

```
unsign32 gcm::table[128][4]
```

2k byte table

7.15.1.7 Y_0

```
uchar gcm::Y_0[16]
```

GCM Internal State

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

- amcl.h

7.16 hash256 Struct Reference

SHA256 hash function instance.

```
#include <amcl.h>
```

Data Fields

- unsign32 length [2]
- unsign32 h [8]
- unsign32 w [80]
- int hlen

7.16.1 Field Documentation

7.16.1.1 h

```
unsign32 hash256::h[8]
```

Internal state

7.16.1.2 hlen

```
int hash256::hlen
```

Hash length in bytes

7.16.1.3 length

```
unsign32 hash256::length[2]
```

64-bit input length

7.16.1.4 w

```
unsign32 hash256::w[80]
```

Internal state

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

- amcl.h

7.17 hash512 Struct Reference

SHA384-512 hash function instance.

```
#include <amcl.h>
```

Data Fields

- unsign64 length [2]
- unsign64 h [8]
- unsign64 w [80]
- int hlen

7.17.1 Field Documentation

7.17.1.1 h

```
unsign64 hash512::h[8]
```

Internal state

7.17.1.2 hlen

```
int hash512::hlen
```

Hash length in bytes

7.17.1.3 length

```
unsign64 hash512::length[2]
```

64-bit input length

7.17.1.4 w

```
unsign64 hash512::w[80]
```

Internal state

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

- amcl.h

7.18 octet Struct Reference

Portable representation of a big positive number.

```
#include <amcl.h>
```

Data Fields

- int len
- int max
- char * val

7.18.1 Field Documentation

7.18.1.1 len

```
int octet::len
```

length in bytes

7.18.1.2 max

```
int octet::max
```

max length allowed - enforce truncation

7.18.1.3 val

```
char* octet::val
```

byte array

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

- amcl.h

7.19 pktype Struct Reference

Public key type.

```
#include <x509.h>
```

Data Fields

- int type
- int hash
- int curve

7.19.1 Field Documentation

7.19.1.1 curve

```
int pktype::curve
```

elliptic curve used or RSA key length in bits

7.19.1.2 hash

```
int pktype::hash
```

hash type

7.19.1.3 type

```
int pktype::type
```

signature type (ECC or RSA)

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

- x509.h

7.20 rsa_private_key_2048 Struct Reference

Integer Factorisation Private Key.

```
#include <rsa_2048.h>
```

Data Fields

- BIG_1024_58 p [FFLEN_2048/2]
- BIG_1024_58 q [FFLEN_2048/2]
- BIG_1024_58 dp [FFLEN_2048/2]
- BIG_1024_58 dq [FFLEN_2048/2]
- BIG_1024_58 c [FFLEN_2048/2]

7.20.1 Field Documentation

7.20.1.1 c

```
BIG_1024_58 rsa_private_key_2048::c[FFLEN_2048/2]
```

1/p mod q

7.20.1.2 dp

```
BIG_1024_58 rsa_private_key_2048::dp[FFLEN_2048/2]
```

decrypting exponent mod (p-1)

7.20.1.3 dq

```
BIG_1024_58 rsa_private_key_2048::dq[FFLEN_2048/2]
```

decrypting exponent mod (q-1)

7.20.1.4 p

`BIG_1024_58 rsa_private_key_2048::p[FFLEN_2048/2]`

secret prime p

7.20.1.5 q

`BIG_1024_58 rsa_private_key_2048::q[FFLEN_2048/2]`

secret prime q

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

- rsa_2048.h

7.21 rsa_private_key_3072 Struct Reference

Integer Factorisation Private Key.

`#include <rsa_3072.h>`

Data Fields

- BIG_384_56 p [FFLEN_3072/2]
- BIG_384_56 q [FFLEN_3072/2]
- BIG_384_56 dp [FFLEN_3072/2]
- BIG_384_56 dq [FFLEN_3072/2]
- BIG_384_56 c [FFLEN_3072/2]

7.21.1 Field Documentation

7.21.1.1 c

`BIG_384_56 rsa_private_key_3072::c[FFLEN_3072/2]`

1/p mod q

7.21.1.2 dp

`BIG_384_56 rsa_private_key_3072::dp[FFLEN_3072/2]`

decrypting exponent mod (p-1)

7.21.1.3 dq

```
BIG_384_56 rsa_private_key_3072::dq[FFLEN_3072/2]
```

decrypting exponent mod (q-1)

7.21.1.4 p

```
BIG_384_56 rsa_private_key_3072::p[FFLEN_3072/2]
```

secret prime p

7.21.1.5 q

```
BIG_384_56 rsa_private_key_3072::q[FFLEN_3072/2]
```

secret prime q

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

- rsa_3072.h

7.22 rsa_public_key_2048 Struct Reference

Integer Factorisation Public Key.

```
#include <rsa_2048.h>
```

Data Fields

- sign32 e
- BIG_1024_58 n [FFLEN_2048]

7.22.1 Field Documentation

7.22.1.1 e

```
sign32 rsa_public_key_2048::e
```

RSA exponent (typically 65537)

7.22.1.2 n

```
BIG_1024_58 rsa_public_key_2048::n[FFLEN_2048]
```

An array of BIGs to store public key

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

- rsa_2048.h

7.23 rsa_public_key_3072 Struct Reference

Integer Factorisation Public Key.

```
#include <rsa_3072.h>
```

Data Fields

- sign32 e
- BIG_384_56 n [FFLEN_3072]

7.23.1 Field Documentation

7.23.1.1 e

```
sign32 rsa_public_key_3072::e
```

RSA exponent (typically 65537)

7.23.1.2 n

```
BIG_384_56 rsa_public_key_3072::n[FFLEN_3072]
```

An array of BIGs to store public key

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

- rsa_3072.h

7.24 sha3 Struct Reference

SHA3 hash function instance.

```
#include <amcl.h>
```

Data Fields

- unsign64 length
- unsign64 S [5][5]
- int rate
- int len

7.24.1 Field Documentation

7.24.1.1 len

```
int sha3::len
```

Hash length in bytes

7.24.1.2 length

```
unsign64 sha3::length
```

64-bit input length

7.24.1.3 rate

```
int sha3::rate
```

TODO

7.24.1.4 S

```
unsign64 sha3::S[5][5]
```

Internal state

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

- amcl.h

Chapter 8

File Documentation

8.1 arch.h File Reference

Architecture Header File.

Macros

- #define CHUNK 64
- #define byte unsigned char
- #define sign32 __int32
- #define sign8 signed char
- #define sign16 short int
- #define sign64 long long
- #define unsign32 unsigned __int32
- #define unsign64 unsigned long long
- #define uchar unsigned char
- #define chunk __int64

8.1.1 Detailed Description

Author

Mike Scott

Date

23rd February 2016 Specify Processor Architecture

8.1.2 Macro Definition Documentation

8.1.2.1 byte

```
#define byte unsigned char
```

8-bit unsigned integer

8.1.2.2 CHUNK

```
#define CHUNK 64
```

size of chunk in bits = wordlength of computer = 16, 32 or 64. Note not all curve options are supported on 16-bit processors - see rom.c

8.1.2.3 chunk

```
#define chunk __int64
```

C type corresponding to word length Note - no 128-bit type available

8.1.2.4 sign16

```
#define sign16 short int
```

16-bit signed integer

8.1.2.5 sign32

```
#define sign32 __int32
```

32-bit signed integer

8.1.2.6 sign64

```
#define sign64 long long
```

64-bit signed integer

8.1.2.7 sign8

```
#define sign8 signed char
```

8-bit signed integer

8.1.2.8 uchar

```
#define uchar unsigned char
```

Unsigned char

8.1.2.9 unsign32

```
#define unsign32 unsigned __int32
```

32-bit unsigned integer

8.1.2.10 unsign64

```
#define unsign64 unsigned long long
```

64-bit unsigned integer

8.2 big_1024_58.h File Reference

BIG Header File.

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "arch.h"
#include "amcl.h"
#include "config_big_1024_58.h"
```

Macros

- #define BIGBITS_1024_58 (8∗MODBYTES_1024_58)
- #define NLEN_1024_58 (1+((8∗MODBYTES_1024_58-1)/BASEBITS_1024_58))
- #define DNLEN_1024_58 2∗NLEN_1024_58
- #define BMASK_1024_58 (((chunk)1<<BASEBITS_1024_58)-1)
- #define NEXCESS_1024_58 (1<<(CHUNK-BASEBITS_1024_58-1))
- #define HBITS_1024_58 (BASEBITS_1024_58/2)
- #define HMASK_1024_58 (((chunk)1<<HBITS_1024_58)-1)

Typedefs

- typedef chunk BIG_1024_58[NLEN_1024_58]
- typedef chunk DBIG_1024_58[DNLEN_1024_58]

Functions

- int BIG_1024_58_iszilch (BIG_1024_58 x)

  Tests for BIG equal to zero.
- int BIG_1024_58_isunity (BIG_1024_58 x)

  Tests for BIG equal to one.
- int BIG_1024_58_diszilch (DBIG_1024_58 x)

  Tests for DBIG equal to zero.
- void BIG_1024_58_output (BIG_1024_58 x)

  Outputs a BIG number to the console.
- void BIG_1024_58_rawoutput (BIG_1024_58 x)

  Outputs a BIG number to the console in raw form (for debugging)
- void BIG_1024_58_cswap (BIG_1024_58 x, BIG_1024_58 y, int s)

  Conditional constant time swap of two BIG numbers.
- void BIG_1024_58_cmove (BIG_1024_58 x, BIG_1024_58 y, int s)

  Conditional copy of BIG number.
- void BIG_1024_58_dcmove (BIG_1024_58 x, BIG_1024_58 y, int s)

  Conditional copy of DBIG number.
- void BIG_1024_58_toBytes (char ∗a, BIG_1024_58 x)

  Convert from BIG number to byte array.
- void BIG_1024_58_fromBytes (BIG_1024_58 x, char ∗a)

  Convert to BIG number from byte array.
- void BIG_1024_58_fromBytesLen (BIG_1024_58 x, char ∗a, int s)

  Convert to BIG number from byte array of given length.
- void BIG_1024_58_dfromBytesLen (DBIG_1024_58 x, char ∗a, int s)

  Convert to DBIG number from byte array of given length.
- void BIG_1024_58_doutput (DBIG_1024_58 x)

  Outputs a DBIG number to the console.
- void BIG_1024_58_drawoutput (DBIG_1024_58 x)

  Outputs a DBIG number to the console.
- void BIG_1024_58_rcopy (BIG_1024_58 x, const BIG_1024_58 y)

  Copy BIG from Read-Only Memory to a BIG.
- void BIG_1024_58_copy (BIG_1024_58 x, BIG_1024_58 y)

  Copy BIG to another BIG.
- void BIG_1024_58_dcopy (DBIG_1024_58 x, DBIG_1024_58 y)

  Copy DBIG to another DBIG.
- void BIG_1024_58_dsucopy (DBIG_1024_58 x, BIG_1024_58 y)

  Copy BIG to upper half of DBIG.
- void BIG_1024_58_dscopy (DBIG_1024_58 x, BIG_1024_58 y)

  Copy BIG to lower half of DBIG.
- void BIG_1024_58_sdcopy (BIG_1024_58 x, DBIG_1024_58 y)

  Copy lower half of DBIG to a BIG.
- void BIG_1024_58_sducopy (BIG_1024_58 x, DBIG_1024_58 y)

  Copy upper half of DBIG to a BIG.
- void BIG_1024_58_zero (BIG_1024_58 x)

  Set BIG to zero.
- void BIG_1024_58_dzero (DBIG_1024_58 x)

  Set DBIG to zero.
- void BIG_1024_58_one (BIG_1024_58 x)

  Set BIG to one (unity)
- void BIG_1024_58_invmod2m (BIG_1024_58 x)

Set BIG to inverse mod 2^ 256.
- void BIG_1024_58_add (BIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 z)

  Set BIG to sum of two BIGs - output not normalised.
- void BIG_1024_58_or (BIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 z)

  Set BIG to logical or of two BIGs - output normalised.
- void BIG_1024_58_inc (BIG_1024_58 x, int i)

  Increment BIG by a small integer - output not normalised.
- void BIG_1024_58_sub (BIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 z)

  Set BIG to difference of two BIGs.
- void BIG_1024_58_dec (BIG_1024_58 x, int i)

  Decrement BIG by a small integer - output not normalised.
- void BIG_1024_58_dadd (DBIG_1024_58 x, DBIG_1024_58 y, DBIG_1024_58 z)

  Set DBIG to sum of two DBIGs.
- void BIG_1024_58_dsub (DBIG_1024_58 x, DBIG_1024_58 y, DBIG_1024_58 z)

  Set DBIG to difference of two DBIGs.
- void BIG_1024_58_imul (BIG_1024_58 x, BIG_1024_58 y, int i)

  Multiply BIG by a small integer - output not normalised.
- chunk BIG_1024_58_pmul (BIG_1024_58 x, BIG_1024_58 y, int i)

  Multiply BIG by not-so-small small integer - output normalised.
- int BIG_1024_58_div3 (BIG_1024_58 x)

  Divide BIG by 3 - output normalised.
- void BIG_1024_58_pxmul (DBIG_1024_58 x, BIG_1024_58 y, int i)

  Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.
- void BIG_1024_58_mul (DBIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 z)

  Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.
- void BIG_1024_58_smul (BIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 z)

  Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.
- void BIG_1024_58_sqr (DBIG_1024_58 x, BIG_1024_58 y)

  Square BIG resulting in a DBIG - input normalised and output normalised.
- void BIG_1024_58_monty (BIG_1024_58 a, BIG_1024_58 md, chunk MC, DBIG_1024_58 d)

  Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.
- void BIG_1024_58_shl (BIG_1024_58 x, int s)

  Shifts a BIG left by any number of bits - input must be normalised, output normalised.
- int BIG_1024_58_fshl (BIG_1024_58 x, int s)

  Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised.
- void BIG_1024_58_dshl (DBIG_1024_58 x, int s)

  Shifts a DBIG left by any number of bits - input must be normalised, output normalised.
- void BIG_1024_58_shr (BIG_1024_58 x, int s)

  Shifts a BIG right by any number of bits - input must be normalised, output normalised.
- int BIG_1024_58_ssn (BIG_1024_58 r, BIG_1024_58 a, BIG_1024_58 m)

  Fast time-critical combined shift by 1 bit, subtract and normalise.
- int BIG_1024_58_fshr (BIG_1024_58 x, int s)

  Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised.
- void BIG_1024_58_dshr (DBIG_1024_58 x, int s)

  Shifts a DBIG right by any number of bits - input must be normalised, output normalised.
- chunk BIG_1024_58_split (BIG_1024_58 x, BIG_1024_58 y, DBIG_1024_58 z, int s)

  Splits a DBIG into two BIGs - input must be normalised, outputs normalised.
- chunk BIG_1024_58_norm (BIG_1024_58 x)

  Normalizes a BIG number - output normalised.
- void BIG_1024_58_dnorm (DBIG_1024_58 x)

  Normalizes a DBIG number - output normalised.

- int BIG_1024_58_comp (BIG_1024_58 x, BIG_1024_58 y)

  Compares two BIG numbers. Inputs must be normalised externally.
- int BIG_1024_58_dcomp (DBIG_1024_58 x, DBIG_1024_58 y)

  Compares two DBIG numbers. Inputs must be normalised externally.
- int BIG_1024_58_nbits (BIG_1024_58 x)

  Calculate number of bits in a BIG - output normalised.
- int BIG_1024_58_dnbits (DBIG_1024_58 x)

  Calculate number of bits in a DBIG - output normalised.
- void BIG_1024_58_mod (BIG_1024_58 x, BIG_1024_58 n)

  Reduce x mod n - input and output normalised.
- void BIG_1024_58_sdiv (BIG_1024_58 x, BIG_1024_58 n)

  Divide x by n - output normalised.
- void BIG_1024_58_dmod (BIG_1024_58 x, DBIG_1024_58 y, BIG_1024_58 n)

  x=y mod n - output normalised
- void BIG_1024_58_ddiv (BIG_1024_58 x, DBIG_1024_58 y, BIG_1024_58 n)

  x=y/n - output normalised
- int BIG_1024_58_parity (BIG_1024_58 x)

  return parity of BIG, that is the least significant bit
- int BIG_1024_58_bit (BIG_1024_58 x, int i)

  return i-th of BIG
- int BIG_1024_58_lastbits (BIG_1024_58 x, int n)

  return least significant bits of a BIG
- void BIG_1024_58_random (BIG_1024_58 x, csprng ∗r)

  Create a random BIG from a random number generator.
- void BIG_1024_58_randomnum (BIG_1024_58 x, BIG_1024_58 n, csprng ∗r)

  Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.
- void BIG_1024_58_modmul (BIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 z, BIG_1024_58 n)

  Calculate x=y∗z mod n.
- void BIG_1024_58_moddiv (BIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 z, BIG_1024_58 n)

  Calculate x=y/z mod n.
- void BIG_1024_58_modsqr (BIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 n)

  Calculate x=y^2 mod n.
- void BIG_1024_58_modneg (BIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 n)

  Calculate x=-y mod n.
- int BIG_1024_58_jacobi (BIG_1024_58 x, BIG_1024_58 y)

  Calculate jacobi Symbol (x/y)
- void BIG_1024_58_invmodp (BIG_1024_58 x, BIG_1024_58 y, BIG_1024_58 n)

  Calculate x=1/y mod n.
- void BIG_1024_58_mod2m (BIG_1024_58 x, int m)

  Calculate x=x mod 2^m.
- void BIG_1024_58_dmod2m (DBIG_1024_58 x, int m)

  Calculate x=x mod 2^m.

8.2.1 Detailed Description

Author

Mike Scott

8.2.2 Macro Definition Documentation

8.2.2.1 BIGBITS_1024_58

```
#define BIGBITS_1024_58 (8*MODBYTES_1024_58)
```

Length in bits

8.2.2.2 BMASK_1024_58

```
#define BMASK_1024_58 (((chunk)1<<BASEBITS_1024_58)-1)
```

Mask = 2^BASEBITS-1

8.2.2.3 DNLEN_1024_58

```
#define DNLEN_1024_58 2*NLEN_1024_58
```

Double length in bytes

8.2.2.4 HBITS_1024_58

```
#define HBITS_1024_58 (BASEBITS_1024_58/2)
```

Number of bits in number base divided by 2

8.2.2.5 HMASK_1024_58

```
#define HMASK_1024_58 (((chunk)1<<HBITS_1024_58)-1)
```

Mask = 2^HBITS-1

8.2.2.6 NEXCESS_1024_58

```
#define NEXCESS_1024_58 (1<<(CHUNK-BASEBITS_1024_58-1))
```

2^(CHUNK-BASEBITS-1) - digit cannot be multiplied by more than this before normalisation

8.2.2.7 NLEN_1024_58

```
#define NLEN_1024_58 (1+((8*MODBYTES_1024_58-1)/BASEBITS_1024_58))
```

length in bytes

8.2.3 Typedef Documentation

8.2.3.1 BIG_1024_58

```
typedef chunk BIG_1024_58[NLEN_1024_58]
```

Define type BIG as array of chunks

8.2.3.2 DBIG_1024_58

```
typedef chunk DBIG_1024_58[DNLEN_1024_58]
```

Define type DBIG as array of chunks

8.2.4 Function Documentation

8.2.4.1 BIG_1024_58_add()

```
void BIG_1024_58_add (
            BIG_1024_58 x,
            BIG_1024_58 y,
            BIG_1024_58 z )
```

Parameters

x	BIG number, sum of other two
y	BIG number
z	BIG number

8.2.4.2 BIG_1024_58_bit()

```
int BIG_1024_58_bit (
            BIG_1024_58 x,
            int i )
```

Parameters

x	BIG number
i	the bit of x to be returned

Returns

0 or 1

8.2.4.3 BIG_1024_58_cmove()

```
void BIG_1024_58_cmove (
            BIG_1024_58 x,
            BIG_1024_58 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a BIG number
y	another BIG number
s	copy takes place if not equal to 0

8.2.4.4 BIG_1024_58_comp()

```
int BIG_1024_58_comp (
            BIG_1024_58 x,
            BIG_1024_58 y )
```

Parameters

x	first BIG number to be compared
y	second BIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.2.4.5 BIG_1024_58_copy()

```
void BIG_1024_58_copy (
            BIG_1024_58 x,
            BIG_1024_58 y )
```

Parameters

x	BIG number
y	BIG number to be copied

8.2.4.6 BIG_1024_58_cswap()

```
void BIG_1024_58_cswap (
            BIG_1024_58 x,
            BIG_1024_58 y,
            int s )
```

Conditionally swaps parameters in constant time (without branching)

Parameters

x	a BIG number
y	another BIG number
s	swap takes place if not equal to 0

8.2.4.7 BIG_1024_58_dadd()

```
void BIG_1024_58_dadd (
            DBIG_1024_58 x,
            DBIG_1024_58 y,
            DBIG_1024_58 z )
```

Parameters

x	DBIG number, sum of other two - output not normalised
y	DBIG number
z	DBIG number

8.2.4.8 BIG_1024_58_dcmove()

```
void BIG_1024_58_dcmove (
            BIG_1024_58 x,
            BIG_1024_58 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a DBIG number
y	another DBIG number
s	copy takes place if not equal to 0

8.2.4.9 BIG_1024_58_dcomp()

```
int BIG_1024_58_dcomp (
            DBIG_1024_58 x,
            DBIG_1024_58 y )
```

Parameters

x	first DBIG number to be compared
y	second DBIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.2.4.10 BIG_1024_58_dcopy()

```
void BIG_1024_58_dcopy (
            DBIG_1024_58 x,
            DBIG_1024_58 y )
```

Parameters

x	DBIG number
y	DBIG number to be copied

8.2.4.11 BIG_1024_58_ddiv()

```
void BIG_1024_58_ddiv (
            BIG_1024_58 x,
            DBIG_1024_58 y,
            BIG_1024_58 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y/n
y	DBIG number
n	Modulus

8.2.4.12 BIG_1024_58_dec()

```
void BIG_1024_58_dec (
            BIG_1024_58 x,
            int i )
```

Parameters

x	BIG number to be decremented
i	integer

8.2.4.13 BIG_1024_58_dfromBytesLen()

```
void BIG_1024_58_dfromBytesLen (
            DBIG_1024_58 x,
            char * a,
            int s )
```

Parameters

x	DBIG number
a	byte array
s	byte array length

8.2.4.14 BIG_1024_58_diszilch()

```
int BIG_1024_58_diszilch (
            DBIG_1024_58 x )
```

Parameters

x	a DBIG number

Returns

1 if zero, else returns 0

8.2.4.15 BIG_1024_58_div3()

```
int BIG_1024_58_div3 (
            BIG_1024_58 x )
```

Parameters

x	BIG number

Returns

Remainder

8.2.4.16 BIG_1024_58_dmod()

```
void BIG_1024_58_dmod (
            BIG_1024_58 x,
            DBIG_1024_58 y,
            BIG_1024_58 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y mod n
y	DBIG number
n	Modulus

8.2.4.17 BIG_1024_58_dmod2m()

```
void BIG_1024_58_dmod2m (
            DBIG_1024_58 x,
            int m )
```

Truncation

Parameters

x	DBIG number, on reduced mod 2^m
m	new truncated size

8.2.4.18 BIG_1024_58_dnbits()

```
int BIG_1024_58_dnbits (
            DBIG_1024_58 x )
```

Parameters

x	DBIG number

Returns

Number of bits in x

8.2.4.19 BIG_1024_58_dnorm()

```
void BIG_1024_58_dnorm (
            DBIG_1024_58 x )
```

All digits of the input DBIG are reduced mod 2^BASEBITS

Parameters

x	DBIG number to be normalised

8.2.4.20 BIG_1024_58_doutput()

```
void BIG_1024_58_doutput (
            DBIG_1024_58 x )
```

Parameters

x	a DBIG number

8.2.4.21 BIG_1024_58_drawoutput()

```
void BIG_1024_58_drawoutput (
            DBIG_1024_58 x )
```

Parameters

x	a DBIG number

8.2.4.22 BIG_1024_58_dscopy()

```
void BIG_1024_58_dscopy (
            DBIG_1024_58 x,
            BIG_1024_58 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.2.4.23 BIG_1024_58_dshl()

```
void BIG_1024_58_dshl (
            DBIG_1024_58 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.2.4.24 BIG_1024_58_dshr()

```
void BIG_1024_58_dshr (
            DBIG_1024_58 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.2.4.25 BIG_1024_58_dsub()

```
void BIG_1024_58_dsub (
            DBIG_1024_58 x,
            DBIG_1024_58 y,
            DBIG_1024_58 z )
```

Parameters

x	DBIG number, difference of other two - output not normalised
y	DBIG number
z	DBIG number

8.2.4.26 BIG_1024_58_dsucopy()

```
void BIG_1024_58_dsucopy (
            DBIG_1024_58 x,
            BIG_1024_58 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.2.4.27 BIG_1024_58_dzero()

```
void BIG_1024_58_dzero (
            DBIG_1024_58 x )
```

Parameters

x	DBIG number to be set to zero

8.2.4.28 BIG_1024_58_fromBytes()

```
void BIG_1024_58_fromBytes (
            BIG_1024_58 x,
            char * a )
```

Parameters

x	BIG number
a	byte array

8.2.4.29 BIG_1024_58_fromBytesLen()

```
void BIG_1024_58_fromBytesLen (
            BIG_1024_58 x,
            char * a,
            int s )
```

Parameters

x	BIG number
a	byte array
s	byte array length

8.2.4.30 BIG_1024_58_fshl()

```
int BIG_1024_58_fshl (
            BIG_1024_58 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Overflow bits

8.2.4.31 BIG_1024_58_fshr()

```
int BIG_1024_58_fshr (
            BIG_1024_58 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Shifted out bits

8.2.4.32 BIG_1024_58_imul()

```
void BIG_1024_58_imul (
             BIG_1024_58 x,
             BIG_1024_58 y,
             int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

8.2.4.33 BIG_1024_58_inc()

```
void BIG_1024_58_inc (
             BIG_1024_58 x,
             int i )
```

Parameters

x	BIG number to be incremented
i	integer

8.2.4.34 BIG_1024_58_invmod2m()

```
void BIG_1024_58_invmod2m (
             BIG_1024_58 x )
```

Parameters

x	BIG number to be inverted

8.2.4.35 BIG_1024_58_invmodp()

```
void BIG_1024_58_invmodp (
```

```
BIG_1024_58 x,
BIG_1024_58 y,
BIG_1024_58 n )
```

Modular Inversion - This is slow. Uses binary method.

Parameters

x	BIG number, on exit = 1/y mod n
y	BIG number
n	The BIG Modulus

8.2.4.36 BIG_1024_58_isunity()

```
int BIG_1024_58_isunity (
            BIG_1024_58 x )
```

Parameters

x	a BIG number

Returns

1 if one, else returns 0

8.2.4.37 BIG_1024_58_iszilch()

```
int BIG_1024_58_iszilch (
            BIG_1024_58 x )
```

Parameters

x	a BIG number

Returns

1 if zero, else returns 0

8.2.4.38 BIG_1024_58_jacobi()

```
int BIG_1024_58_jacobi (
            BIG_1024_58 x,
            BIG_1024_58 y )
```

Parameters

x	BIG number
y	BIG number

Returns

Jacobi symbol, -1,0 or 1

8.2.4.39 BIG_1024_58_lastbits()

```
int BIG_1024_58_lastbits (
            BIG_1024_58 x,
            int n )
```

Parameters

x	BIG number
n	number of bits to return. Assumed to be less than BASEBITS.

Returns

least significant n bits as an integer

8.2.4.40 BIG_1024_58_mod()

```
void BIG_1024_58_mod (
            BIG_1024_58 x,
            BIG_1024_58 n )
```

Slow but rarely used

Parameters

x	BIG number to be reduced mod n
n	The modulus

8.2.4.41 BIG_1024_58_mod2m()

```
void BIG_1024_58_mod2m (
            BIG_1024_58 x,
            int m )
```

Truncation

Parameters

x	BIG number, on reduced mod 2^m
m	new truncated size

8.2.4.42 BIG_1024_58_moddiv()

```
void BIG_1024_58_moddiv (
            BIG_1024_58 x,
            BIG_1024_58 y,
            BIG_1024_58 z,
            BIG_1024_58 n )
```

Slow method for modular division

Parameters

x	BIG number, on exit = y/z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.2.4.43 BIG_1024_58_modmul()

```
void BIG_1024_58_modmul (
            BIG_1024_58 x,
            BIG_1024_58 y,
            BIG_1024_58 z,
            BIG_1024_58 n )
```

brief return NAF (Non-Adjacent-Form) value as +/- 1, 3 or 5, inputs must be normalised

Given x and 3∗x extracts NAF value from given bit position, and returns number of bits processed, and number of trailing zeros detected if any param x BIG number param x3 BIG number, three times x param i bit position param nbs pointer to integer returning number of bits processed param nzs pointer to integer returning number of trailing 0s return + or - 1, 3 or 5Slow method for modular multiplication

Parameters

x	BIG number, on exit = y∗z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.2.4.44 BIG_1024_58_modneg()

```
void BIG_1024_58_modneg (
            BIG_1024_58 x,
            BIG_1024_58 y,
            BIG_1024_58 n )
```

Modular negation

Parameters

x	BIG number, on exit = -y mod n
y	BIG number
n	The BIG Modulus

8.2.4.45 BIG_1024_58_modsqr()

```
void BIG_1024_58_modsqr (
            BIG_1024_58 x,
            BIG_1024_58 y,
            BIG_1024_58 n )
```

Slow method for modular squaring

Parameters

x	BIG number, on exit = y^2 mod n
y	BIG number
n	The BIG Modulus

8.2.4.46 BIG_1024_58_monty()

```
void BIG_1024_58_monty (
            BIG_1024_58 a,
            BIG_1024_58 md,
            chunk MC,
            DBIG_1024_58 d )
```

Parameters

a	BIG number, reduction of a BIG
md	BIG number, the modulus
MC	the Montgomery Constant
d	DBIG number to be reduced

8.2.4.47 BIG_1024_58_mul()

```
void BIG_1024_58_mul (
            DBIG_1024_58 x,
            BIG_1024_58 y,
            BIG_1024_58 z )
```

Parameters

x	DBIG number, product of other two
y	BIG number
z	BIG number

8.2.4.48 BIG_1024_58_nbits()

```
int BIG_1024_58_nbits (
            BIG_1024_58 x )
```

Parameters

x	BIG number

Returns

Number of bits in x

8.2.4.49 BIG_1024_58_norm()

```
chunk BIG_1024_58_norm (
            BIG_1024_58 x )
```

All digits of the input BIG are reduced mod 2^BASEBITS

Parameters

x	BIG number to be normalised

8.2.4.50 BIG_1024_58_one()

```
void BIG_1024_58_one (
            BIG_1024_58 x )
```

Parameters

x	BIG number to be set to one.

8.2.4.51 BIG_1024_58_or()

```
void BIG_1024_58_or (
            BIG_1024_58 x,
            BIG_1024_58 y,
            BIG_1024_58 z )
```

Parameters

x	BIG number, or of other two
y	BIG number
z	BIG number

8.2.4.52 BIG_1024_58_output()

```
void BIG_1024_58_output (
            BIG_1024_58 x )
```

Parameters

x	a BIG number

8.2.4.53 BIG_1024_58_parity()

```
int BIG_1024_58_parity (
            BIG_1024_58 x )
```

Parameters

x	BIG number

Returns

0 or 1

8.2.4.54 BIG_1024_58_pmul()

```
chunk BIG_1024_58_pmul (
            BIG_1024_58 x,
            BIG_1024_58 y,
            int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

Returns

Overflowing bits

8.2.4.55 BIG_1024_58_pxmul()

```
void BIG_1024_58_pxmul (
            DBIG_1024_58 x,
            BIG_1024_58 y,
            int i )
```

Parameters

x	DBIG number, product of other two
y	BIG number
i	small integer

8.2.4.56 BIG_1024_58_random()

```
void BIG_1024_58_random (
            BIG_1024_58 x,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
r	A pointer to a Cryptographically Secure Random Number Generator

8.2.4.57 BIG_1024_58_randomnum()

```
void BIG_1024_58_randomnum (
            BIG_1024_58 x,
            BIG_1024_58 n,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
n	The modulus
r	A pointer to a Cryptographically Secure Random Number Generator

8.2.4.58 BIG_1024_58_rawoutput()

```
void BIG_1024_58_rawoutput (
            BIG_1024_58 x )
```

Parameters

x	a BIG number

8.2.4.59 BIG_1024_58_rcopy()

```
void BIG_1024_58_rcopy (
            BIG_1024_58 x,
            const BIG_1024_58 y )
```

Parameters

x	BIG number
y	BIG number in ROM

8.2.4.60 BIG_1024_58_sdcopy()

```
void BIG_1024_58_sdcopy (
            BIG_1024_58 x,
            DBIG_1024_58 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.2.4.61 BIG_1024_58_sdiv()

```
void BIG_1024_58_sdiv (
            BIG_1024_58 x,
            BIG_1024_58 n )
```

Slow but rarely used

Parameters

x	BIG number to be divided by n
n	The Divisor

8.2.4.62 BIG_1024_58_sducopy()

```
void BIG_1024_58_sducopy (
            BIG_1024_58 x,
            DBIG_1024_58 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.2.4.63 BIG_1024_58_shl()

```
void BIG_1024_58_shl (
            BIG_1024_58 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.2.4.64 BIG_1024_58_shr()

```
void BIG_1024_58_shr (
            BIG_1024_58 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.2.4.65 BIG_1024_58_smul()

```
void BIG_1024_58_smul (
            BIG_1024_58 x,
            BIG_1024_58 y,
            BIG_1024_58 z )
```

Note that the product must fit into a BIG, and x must be distinct from y and z

Parameters

x	BIG number, product of other two
y	BIG number
z	BIG number

8.2.4.66 BIG_1024_58_split()

```
chunk BIG_1024_58_split (
            BIG_1024_58 x,
            BIG_1024_58 y,
            DBIG_1024_58 z,
            int s )
```

Internal function. The value of s must be approximately in the middle of the DBIG. Typically used to extract z mod 2^MODBITS and z/2^MODBITS

Parameters

x	BIG number, top half of z
y	BIG number, bottom half of z
z	DBIG number to be split in two.
s	Bit position at which to split

Returns

carry-out from top half

8.2.4.67 BIG_1024_58_sqr()

```
void BIG_1024_58_sqr (
            DBIG_1024_58 x,
            BIG_1024_58 y )
```

Parameters

x	DBIG number, square of a BIG
y	BIG number to be squared

8.2.4.68 BIG_1024_58_ssn()

```
int BIG_1024_58_ssn (
            BIG_1024_58 r,
            BIG_1024_58 a,
            BIG_1024_58 m )
```

Parameters

r	BIG number normalised output
a	BIG number to be subtracted from
m	BIG number to be shifted and subtracted

Returns

sign of r

8.2.4.69 BIG_1024_58_sub()

```
void BIG_1024_58_sub (
            BIG_1024_58 x,
```

```
            BIG_1024_58 y,
            BIG_1024_58 z )
```

Parameters

x	BIG number, difference of other two - output not normalised
y	BIG number
z	BIG number

8.2.4.70 BIG_1024_58_toBytes()

```
void BIG_1024_58_toBytes (
            char * a,
            BIG_1024_58 x )
```

Parameters

a	byte array
x	BIG number

8.2.4.71 BIG_1024_58_zero()

```
void BIG_1024_58_zero (
            BIG_1024_58 x )
```

Parameters

x	BIG number to be set to zero

8.3 big_256_56.h File Reference

BIG Header File.

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "arch.h"
#include "amcl.h"
#include "config_big_256_56.h"
```

Macros

- #define BIGBITS_256_56 (8∗MODBYTES_256_56)
- #define NLEN_256_56 (1+((8∗MODBYTES_256_56-1)/BASEBITS_256_56))
- #define DNLEN_256_56 2∗NLEN_256_56
- #define BMASK_256_56 (((chunk)1<<BASEBITS_256_56)-1)
- #define NEXCESS_256_56 (1<<(CHUNK-BASEBITS_256_56-1))
- #define HBITS_256_56 (BASEBITS_256_56/2)
- #define HMASK_256_56 (((chunk)1<<HBITS_256_56)-1)

Typedefs

- typedef chunk BIG_256_56[NLEN_256_56]
- typedef chunk DBIG_256_56[DNLEN_256_56]

Functions

- int BIG_256_56_iszilch (BIG_256_56 x)

  Tests for BIG equal to zero.
- int BIG_256_56_isunity (BIG_256_56 x)

  Tests for BIG equal to one.
- int BIG_256_56_diszilch (DBIG_256_56 x)

  Tests for DBIG equal to zero.
- void BIG_256_56_output (BIG_256_56 x)

  Outputs a BIG number to the console.
- void BIG_256_56_rawoutput (BIG_256_56 x)

  Outputs a BIG number to the console in raw form (for debugging)
- void BIG_256_56_cswap (BIG_256_56 x, BIG_256_56 y, int s)

  Conditional constant time swap of two BIG numbers.
- void BIG_256_56_cmove (BIG_256_56 x, BIG_256_56 y, int s)

  Conditional copy of BIG number.
- void BIG_256_56_dcmove (BIG_256_56 x, BIG_256_56 y, int s)

  Conditional copy of DBIG number.
- void BIG_256_56_toBytes (char ∗a, BIG_256_56 x)

  Convert from BIG number to byte array.
- void BIG_256_56_fromBytes (BIG_256_56 x, char ∗a)

  Convert to BIG number from byte array.
- void BIG_256_56_fromBytesLen (BIG_256_56 x, char ∗a, int s)

  Convert to BIG number from byte array of given length.
- void BIG_256_56_dfromBytesLen (DBIG_256_56 x, char ∗a, int s)

  Convert to DBIG number from byte array of given length.
- void BIG_256_56_doutput (DBIG_256_56 x)

  Outputs a DBIG number to the console.
- void BIG_256_56_drawoutput (DBIG_256_56 x)

  Outputs a DBIG number to the console.
- void BIG_256_56_rcopy (BIG_256_56 x, const BIG_256_56 y)

  Copy BIG from Read-Only Memory to a BIG.
- void BIG_256_56_copy (BIG_256_56 x, BIG_256_56 y)

  Copy BIG to another BIG.
- void BIG_256_56_dcopy (DBIG_256_56 x, DBIG_256_56 y)

Copy DBIG to another DBIG.
- void BIG_256_56_dsucopy (DBIG_256_56 x, BIG_256_56 y)

  Copy BIG to upper half of DBIG.
- void BIG_256_56_dscopy (DBIG_256_56 x, BIG_256_56 y)

  Copy BIG to lower half of DBIG.
- void BIG_256_56_sdcopy (BIG_256_56 x, DBIG_256_56 y)

  Copy lower half of DBIG to a BIG.
- void BIG_256_56_sducopy (BIG_256_56 x, DBIG_256_56 y)

  Copy upper half of DBIG to a BIG.
- void BIG_256_56_zero (BIG_256_56 x)

  Set BIG to zero.
- void BIG_256_56_dzero (DBIG_256_56 x)

  Set DBIG to zero.
- void BIG_256_56_one (BIG_256_56 x)

  Set BIG to one (unity)
- void BIG_256_56_invmod2m (BIG_256_56 x)

  Set BIG to inverse mod 2^256.
- void BIG_256_56_add (BIG_256_56 x, BIG_256_56 y, BIG_256_56 z)

  Set BIG to sum of two BIGs - output not normalised.
- void BIG_256_56_or (BIG_256_56 x, BIG_256_56 y, BIG_256_56 z)

  Set BIG to logical or of two BIGs - output normalised.
- void BIG_256_56_inc (BIG_256_56 x, int i)

  Increment BIG by a small integer - output not normalised.
- void BIG_256_56_sub (BIG_256_56 x, BIG_256_56 y, BIG_256_56 z)

  Set BIG to difference of two BIGs.
- void BIG_256_56_dec (BIG_256_56 x, int i)

  Decrement BIG by a small integer - output not normalised.
- void BIG_256_56_dadd (DBIG_256_56 x, DBIG_256_56 y, DBIG_256_56 z)

  Set DBIG to sum of two DBIGs.
- void BIG_256_56_dsub (DBIG_256_56 x, DBIG_256_56 y, DBIG_256_56 z)

  Set DBIG to difference of two DBIGs.
- void BIG_256_56_imul (BIG_256_56 x, BIG_256_56 y, int i)

  Multiply BIG by a small integer - output not normalised.
- chunk BIG_256_56_pmul (BIG_256_56 x, BIG_256_56 y, int i)

  Multiply BIG by not-so-small small integer - output normalised.
- int BIG_256_56_div3 (BIG_256_56 x)

  Divide BIG by 3 - output normalised.
- void BIG_256_56_pxmul (DBIG_256_56 x, BIG_256_56 y, int i)

  Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.
- void BIG_256_56_mul (DBIG_256_56 x, BIG_256_56 y, BIG_256_56 z)

  Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.
- void BIG_256_56_smul (BIG_256_56 x, BIG_256_56 y, BIG_256_56 z)

  Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.
- void BIG_256_56_sqr (DBIG_256_56 x, BIG_256_56 y)

  Square BIG resulting in a DBIG - input normalised and output normalised.
- void BIG_256_56_monty (BIG_256_56 a, BIG_256_56 md, chunk MC, DBIG_256_56 d)

  Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.
- void BIG_256_56_shl (BIG_256_56 x, int s)

  Shifts a BIG left by any number of bits - input must be normalised, output normalised.
- int BIG_256_56_fshl (BIG_256_56 x, int s)

  Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised.

- void BIG_256_56_dshl (DBIG_256_56 x, int s)

  Shifts a DBIG left by any number of bits - input must be normalised, output normalised.
- void BIG_256_56_shr (BIG_256_56 x, int s)

  Shifts a BIG right by any number of bits - input must be normalised, output normalised.
- int BIG_256_56_ssn (BIG_256_56 r, BIG_256_56 a, BIG_256_56 m)

  Fast time-critical combined shift by 1 bit, subtract and normalise.
- int BIG_256_56_fshr (BIG_256_56 x, int s)

  Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised.
- void BIG_256_56_dshr (DBIG_256_56 x, int s)

  Shifts a DBIG right by any number of bits - input must be normalised, output normalised.
- chunk BIG_256_56_split (BIG_256_56 x, BIG_256_56 y, DBIG_256_56 z, int s)

  Splits a DBIG into two BIGs - input must be normalised, outputs normalised.
- chunk BIG_256_56_norm (BIG_256_56 x)

  Normalizes a BIG number - output normalised.
- void BIG_256_56_dnorm (DBIG_256_56 x)

  Normalizes a DBIG number - output normalised.
- int BIG_256_56_comp (BIG_256_56 x, BIG_256_56 y)

  Compares two BIG numbers. Inputs must be normalised externally.
- int BIG_256_56_dcomp (DBIG_256_56 x, DBIG_256_56 y)

  Compares two DBIG numbers. Inputs must be normalised externally.
- int BIG_256_56_nbits (BIG_256_56 x)

  Calculate number of bits in a BIG - output normalised.
- int BIG_256_56_dnbits (DBIG_256_56 x)

  Calculate number of bits in a DBIG - output normalised.
- void BIG_256_56_mod (BIG_256_56 x, BIG_256_56 n)

  Reduce x mod n - input and output normalised.
- void BIG_256_56_sdiv (BIG_256_56 x, BIG_256_56 n)

  Divide x by n - output normalised.
- void BIG_256_56_dmod (BIG_256_56 x, DBIG_256_56 y, BIG_256_56 n)

  x=y mod n - output normalised
- void BIG_256_56_ddiv (BIG_256_56 x, DBIG_256_56 y, BIG_256_56 n)

  x=y/n - output normalised
- int BIG_256_56_parity (BIG_256_56 x)

  return parity of BIG, that is the least significant bit
- int BIG_256_56_bit (BIG_256_56 x, int i)

  return i-th of BIG
- int BIG_256_56_lastbits (BIG_256_56 x, int n)

  return least significant bits of a BIG
- void BIG_256_56_random (BIG_256_56 x, csprng ∗r)

  Create a random BIG from a random number generator.
- void BIG_256_56_randomnum (BIG_256_56 x, BIG_256_56 n, csprng ∗r)

  Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.
- void BIG_256_56_modmul (BIG_256_56 x, BIG_256_56 y, BIG_256_56 z, BIG_256_56 n)

  Calculate x=y∗z mod n.
- void BIG_256_56_moddiv (BIG_256_56 x, BIG_256_56 y, BIG_256_56 z, BIG_256_56 n)

  Calculate x=y/z mod n.
- void BIG_256_56_modsqr (BIG_256_56 x, BIG_256_56 y, BIG_256_56 n)

  Calculate x=y^2 mod n.
- void BIG_256_56_modneg (BIG_256_56 x, BIG_256_56 y, BIG_256_56 n)

  Calculate x=-y mod n.
- int BIG_256_56_jacobi (BIG_256_56 x, BIG_256_56 y)

Calculate jacobi Symbol (x/y)
- void BIG_256_56_invmodp (BIG_256_56 x, BIG_256_56 y, BIG_256_56 n)

  Calculate x=1/y mod n.
- void BIG_256_56_mod2m (BIG_256_56 x, int m)

  Calculate x=x mod 2^m.
- void BIG_256_56_dmod2m (DBIG_256_56 x, int m)

  Calculate x=x mod 2^m.

8.3.1 Detailed Description

Author

Mike Scott

8.3.2 Macro Definition Documentation

8.3.2.1 BIGBITS_256_56

```
#define BIGBITS_256_56 (8*MODBYTES_256_56)
```

Length in bits

8.3.2.2 BMASK_256_56

```
#define BMASK_256_56 (((chunk)1<<BASEBITS_256_56)-1)
```

Mask = 2^BASEBITS-1

8.3.2.3 DNLEN_256_56

```
#define DNLEN_256_56 2*NLEN_256_56
```

Double length in bytes

8.3.2.4 HBITS_256_56

```
#define HBITS_256_56 (BASEBITS_256_56/2)
```

Number of bits in number base divided by 2

8.3.2.5 HMASK_256_56

```
#define HMASK_256_56 (((chunk)1<<HBITS_256_56)-1)
```

Mask = 2^HBITS-1

8.3.2.6 NEXCESS_256_56

```
#define NEXCESS_256_56 (1<<(CHUNK-BASEBITS_256_56-1))
```

2^(CHUNK-BASEBITS-1) - digit cannot be multiplied by more than this before normalisation

8.3.2.7 NLEN_256_56

```
#define NLEN_256_56 (1+((8*MODBYTES_256_56-1)/BASEBITS_256_56))
```

length in bytes

8.3.3 Typedef Documentation

8.3.3.1 BIG_256_56

```
typedef chunk BIG_256_56[NLEN_256_56]
```

Define type BIG as array of chunks

8.3.3.2 DBIG_256_56

```
typedef chunk DBIG_256_56[DNLEN_256_56]
```

Define type DBIG as array of chunks

8.3.4 Function Documentation

8.3.4.1 BIG_256_56_add()

```
void BIG_256_56_add (
            BIG_256_56 x,
            BIG_256_56 y,
            BIG_256_56 z )
```

Parameters

x	BIG number, sum of other two
y	BIG number
z	BIG number

8.3.4.2 BIG_256_56_bit()

```
int BIG_256_56_bit (
            BIG_256_56 x,
            int i )
```

Parameters

x	BIG number
i	the bit of x to be returned

Returns

0 or 1

8.3.4.3 BIG_256_56_cmove()

```
void BIG_256_56_cmove (
            BIG_256_56 x,
            BIG_256_56 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a BIG number
y	another BIG number
s	copy takes place if not equal to 0

8.3.4.4 BIG_256_56_comp()

```
int BIG_256_56_comp (
            BIG_256_56 x,
            BIG_256_56 y )
```

Parameters

x	first BIG number to be compared
y	second BIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.3.4.5 BIG_256_56_copy()

```
void BIG_256_56_copy (
            BIG_256_56 x,
            BIG_256_56 y )
```

Parameters

x	BIG number
y	BIG number to be copied

8.3.4.6 BIG_256_56_cswap()

```
void BIG_256_56_cswap (
            BIG_256_56 x,
            BIG_256_56 y,
            int s )
```

Conditionally swaps parameters in constant time (without branching)

Parameters

x	a BIG number
y	another BIG number
s	swap takes place if not equal to 0

8.3.4.7 BIG_256_56_dadd()

```
void BIG_256_56_dadd (
            DBIG_256_56 x,
            DBIG_256_56 y,
            DBIG_256_56 z )
```

Parameters

x	DBIG number, sum of other two - output not normalised
y	DBIG number
z	DBIG number

8.3.4.8 BIG_256_56_dcmove()

```
void BIG_256_56_dcmove (
            BIG_256_56 x,
            BIG_256_56 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a DBIG number
y	another DBIG number
s	copy takes place if not equal to 0

8.3.4.9 BIG_256_56_dcomp()

```
int BIG_256_56_dcomp (
            DBIG_256_56 x,
            DBIG_256_56 y )
```

Parameters

x	first DBIG number to be compared
y	second DBIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.3.4.10 BIG_256_56_dcopy()

```
void BIG_256_56_dcopy (
            DBIG_256_56 x,
            DBIG_256_56 y )
```

Parameters

x	DBIG number
y	DBIG number to be copied

8.3.4.11 BIG_256_56_ddiv()

```
void BIG_256_56_ddiv (
            BIG_256_56 x,
            DBIG_256_56 y,
            BIG_256_56 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y/n
y	DBIG number
n	Modulus

8.3.4.12 BIG_256_56_dec()

```
void BIG_256_56_dec (
            BIG_256_56 x,
            int i )
```

Parameters

x	BIG number to be decremented
i	integer

8.3.4.13 BIG_256_56_dfromBytesLen()

```
void BIG_256_56_dfromBytesLen (
            DBIG_256_56 x,
            char * a,
            int s )
```

Parameters

x	DBIG number
a	byte array
s	byte array length

8.3.4.14 BIG_256_56_diszilch()

```
int BIG_256_56_diszilch (
            DBIG_256_56 x )
```

Parameters

x	a DBIG number

Returns

1 if zero, else returns 0

8.3.4.15 BIG_256_56_div3()

```
int BIG_256_56_div3 (
            BIG_256_56 x )
```

Parameters

x	BIG number

Returns

Remainder

8.3.4.16 BIG_256_56_dmod()

```
void BIG_256_56_dmod (
            BIG_256_56 x,
            DBIG_256_56 y,
            BIG_256_56 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y mod n
y	DBIG number
n	Modulus

8.3.4.17 BIG_256_56_dmod2m()

```
void BIG_256_56_dmod2m (
            DBIG_256_56 x,
            int m )
```

Truncation

Parameters

x	DBIG number, on reduced mod 2^m
m	new truncated size

8.3.4.18 BIG_256_56_dnbits()

```
int BIG_256_56_dnbits (
            DBIG_256_56 x )
```

Parameters

x	DBIG number

Returns

Number of bits in x

8.3.4.19 BIG_256_56_dnorm()

```
void BIG_256_56_dnorm (
            DBIG_256_56 x )
```

All digits of the input DBIG are reduced mod 2^BASEBITS

Parameters

x	DBIG number to be normalised

8.3.4.20 BIG_256_56_doutput()

```
void BIG_256_56_doutput (
            DBIG_256_56 x )
```

Parameters

x	a DBIG number

8.3.4.21 BIG_256_56_drawoutput()

```
void BIG_256_56_drawoutput (
            DBIG_256_56 x )
```

Parameters

x	a DBIG number

8.3.4.22 BIG_256_56_dscopy()

```
void BIG_256_56_dscopy (
            DBIG_256_56 x,
            BIG_256_56 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.3.4.23 BIG_256_56_dshl()

```
void BIG_256_56_dshl (
            DBIG_256_56 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.3.4.24 BIG_256_56_dshr()

```
void BIG_256_56_dshr (
```

```
            DBIG_256_56 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.3.4.25 BIG_256_56_dsub()

```
void BIG_256_56_dsub (
            DBIG_256_56 x,
            DBIG_256_56 y,
            DBIG_256_56 z )
```

Parameters

x	DBIG number, difference of other two - output not normalised
y	DBIG number
z	DBIG number

8.3.4.26 BIG_256_56_dsucopy()

```
void BIG_256_56_dsucopy (
            DBIG_256_56 x,
            BIG_256_56 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.3.4.27 BIG_256_56_dzero()

```
void BIG_256_56_dzero (
            DBIG_256_56 x )
```

Parameters

x	DBIG number to be set to zero

8.3.4.28 BIG_256_56_fromBytes()

```
void BIG_256_56_fromBytes (
            BIG_256_56 x,
            char * a )
```

Parameters

x	BIG number
a	byte array

8.3.4.29 BIG_256_56_fromBytesLen()

```
void BIG_256_56_fromBytesLen (
            BIG_256_56 x,
            char * a,
            int s )
```

Parameters

x	BIG number
a	byte array
s	byte array length

8.3.4.30 BIG_256_56_fshl()

```
int BIG_256_56_fshl (
            BIG_256_56 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Overflow bits

8.3.4.31 BIG_256_56_fshr()

```
int BIG_256_56_fshr (
            BIG_256_56 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Shifted out bits

8.3.4.32 BIG_256_56_imul()

```
void BIG_256_56_imul (
            BIG_256_56 x,
            BIG_256_56 y,
            int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

8.3.4.33 BIG_256_56_inc()

```
void BIG_256_56_inc (
            BIG_256_56 x,
            int i )
```

Parameters

x	BIG number to be incremented
i	integer

8.3.4.34 BIG_256_56_invmod2m()

```
void BIG_256_56_invmod2m (
            BIG_256_56 x )
```

Parameters

x	BIG number to be inverted

8.3.4.35 BIG_256_56_invmodp()

```
void BIG_256_56_invmodp (
            BIG_256_56 x,
            BIG_256_56 y,
            BIG_256_56 n )
```

Modular Inversion - This is slow. Uses binary method.

Parameters

x	BIG number, on exit = 1/y mod n
y	BIG number
n	The BIG Modulus

8.3.4.36 BIG_256_56_isunity()

```
int BIG_256_56_isunity (
            BIG_256_56 x )
```

Parameters

x	a BIG number

Returns

1 if one, else returns 0

8.3.4.37 BIG_256_56_iszilch()

```
int BIG_256_56_iszilch (
            BIG_256_56 x )
```

Parameters

x	a BIG number

Returns

1 if zero, else returns 0

8.3.4.38 BIG_256_56_jacobi()

```
int BIG_256_56_jacobi (
            BIG_256_56 x,
            BIG_256_56 y )
```

Parameters

x	BIG number
y	BIG number

Returns

Jacobi symbol, -1,0 or 1

8.3.4.39 BIG_256_56_lastbits()

```
int BIG_256_56_lastbits (
            BIG_256_56 x,
            int n )
```

Parameters

x	BIG number
n	number of bits to return. Assumed to be less than BASEBITS.

Returns

least significant n bits as an integer

8.3.4.40 BIG_256_56_mod()

```
void BIG_256_56_mod (
            BIG_256_56 x,
            BIG_256_56 n )
```

Slow but rarely used

Parameters

x	BIG number to be reduced mod n
n	The modulus

8.3.4.41 BIG_256_56_mod2m()

```
void BIG_256_56_mod2m (
            BIG_256_56 x,
            int m )
```

Truncation

Parameters

x	BIG number, on reduced mod 2^m
m	new truncated size

8.3.4.42 BIG_256_56_moddiv()

```
void BIG_256_56_moddiv (
            BIG_256_56 x,
            BIG_256_56 y,
            BIG_256_56 z,
            BIG_256_56 n )
```

Slow method for modular division

Parameters

x	BIG number, on exit = y/z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.3.4.43 BIG_256_56_modmul()

```
void BIG_256_56_modmul (
            BIG_256_56 x,
            BIG_256_56 y,
```

```
            BIG_256_56 z,
            BIG_256_56 n )
```

brief return NAF (Non-Adjacent-Form) value as +/- 1, 3 or 5, inputs must be normalised

Given x and 3∗x extracts NAF value from given bit position, and returns number of bits processed, and number of trailing zeros detected if any param x BIG number param x3 BIG number, three times x param i bit position param nbs pointer to integer returning number of bits processed param nzs pointer to integer returning number of trailing 0s return + or - 1, 3 or 5Slow method for modular multiplication

Parameters

x	BIG number, on exit = y∗z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.3.4.44 BIG_256_56_modneg()

```
void BIG_256_56_modneg (
            BIG_256_56 x,
            BIG_256_56 y,
            BIG_256_56 n )
```

Modular negation

Parameters

x	BIG number, on exit = -y mod n
y	BIG number
n	The BIG Modulus

8.3.4.45 BIG_256_56_modsqr()

```
void BIG_256_56_modsqr (
            BIG_256_56 x,
            BIG_256_56 y,
            BIG_256_56 n )
```

Slow method for modular squaring

Parameters

x	BIG number, on exit = y^2 mod n
y	BIG number
n	The BIG Modulus

8.3.4.46 BIG_256_56_monty()

```
void BIG_256_56_monty (
            BIG_256_56 a,
            BIG_256_56 md,
            chunk MC,
            DBIG_256_56 d )
```

Parameters

a	BIG number, reduction of a BIG
md	BIG number, the modulus
MC	the Montgomery Constant
d	DBIG number to be reduced

8.3.4.47 BIG_256_56_mul()

```
void BIG_256_56_mul (
            DBIG_256_56 x,
            BIG_256_56 y,
            BIG_256_56 z )
```

Parameters

x	DBIG number, product of other two
y	BIG number
z	BIG number

8.3.4.48 BIG_256_56_nbits()

```
int BIG_256_56_nbits (
            BIG_256_56 x )
```

Parameters

x	BIG number

Returns

Number of bits in x

8.3.4.49 BIG_256_56_norm()

```
chunk BIG_256_56_norm (
            BIG_256_56 x )
```

All digits of the input BIG are reduced mod 2^BASEBITS

Parameters

x	BIG number to be normalised

8.3.4.50 BIG_256_56_one()

```
void BIG_256_56_one (
            BIG_256_56 x )
```

Parameters

x	BIG number to be set to one.

8.3.4.51 BIG_256_56_or()

```
void BIG_256_56_or (
            BIG_256_56 x,
            BIG_256_56 y,
            BIG_256_56 z )
```

Parameters

x	BIG number, or of other two
y	BIG number
z	BIG number

8.3.4.52 BIG_256_56_output()

```
void BIG_256_56_output (
            BIG_256_56 x )
```

Parameters

x	a BIG number

8.3.4.53 BIG_256_56_parity()

```
int BIG_256_56_parity (
            BIG_256_56 x )
```

Parameters

x	BIG number

Returns

0 or 1

8.3.4.54 BIG_256_56_pmul()

```
chunk BIG_256_56_pmul (
            BIG_256_56 x,
            BIG_256_56 y,
            int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

Returns

Overflowing bits

8.3.4.55 BIG_256_56_pxmul()

```
void BIG_256_56_pxmul (
            DBIG_256_56 x,
            BIG_256_56 y,
            int i )
```

Parameters

x	DBIG number, product of other two
y	BIG number
i	small integer

8.3.4.56 BIG_256_56_random()

```
void BIG_256_56_random (
            BIG_256_56 x,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
r	A pointer to a Cryptographically Secure Random Number Generator

8.3.4.57 BIG_256_56_randomnum()

```
void BIG_256_56_randomnum (
            BIG_256_56 x,
            BIG_256_56 n,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
n	The modulus
r	A pointer to a Cryptographically Secure Random Number Generator

8.3.4.58 BIG_256_56_rawoutput()

```
void BIG_256_56_rawoutput (
            BIG_256_56 x )
```

Parameters

x	a BIG number

8.3.4.59 BIG_256_56_rcopy()

```
void BIG_256_56_rcopy (
```

```
            BIG_256_56 x,
            const BIG_256_56 y )
```

Parameters

x	BIG number
y	BIG number in ROM

8.3.4.60 BIG_256_56_sdcopy()

```
void BIG_256_56_sdcopy (
            BIG_256_56 x,
            DBIG_256_56 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.3.4.61 BIG_256_56_sdiv()

```
void BIG_256_56_sdiv (
            BIG_256_56 x,
            BIG_256_56 n )
```

Slow but rarely used

Parameters

x	BIG number to be divided by n
n	The Divisor

8.3.4.62 BIG_256_56_sducopy()

```
void BIG_256_56_sducopy (
            BIG_256_56 x,
            DBIG_256_56 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.3.4.63 BIG_256_56_shl()

```
void BIG_256_56_shl (
            BIG_256_56 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.3.4.64 BIG_256_56_shr()

```
void BIG_256_56_shr (
            BIG_256_56 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.3.4.65 BIG_256_56_smul()

```
void BIG_256_56_smul (
            BIG_256_56 x,
            BIG_256_56 y,
            BIG_256_56 z )
```

Note that the product must fit into a BIG, and x must be distinct from y and z

Parameters

x	BIG number, product of other two
y	BIG number
z	BIG number

8.3.4.66 BIG_256_56_split()

```
chunk BIG_256_56_split (
```

```
            BIG_256_56 x,
            BIG_256_56 y,
            DBIG_256_56 z,
            int s )
```

Internal function. The value of s must be approximately in the middle of the DBIG. Typically used to extract z mod 2^MODBITS and z/2^MODBITS

Parameters

x	BIG number, top half of z
y	BIG number, bottom half of z
z	DBIG number to be split in two.
s	Bit position at which to split

Returns

carry-out from top half

8.3.4.67 BIG_256_56_sqr()

```
void BIG_256_56_sqr (
            DBIG_256_56 x,
            BIG_256_56 y )
```

Parameters

x	DBIG number, square of a BIG
y	BIG number to be squared

8.3.4.68 BIG_256_56_ssn()

```
int BIG_256_56_ssn (
            BIG_256_56 r,
            BIG_256_56 a,
            BIG_256_56 m )
```

Parameters

r	BIG number normalised output
a	BIG number to be subtracted from
m	BIG number to be shifted and subtracted

Returns

sign of r

8.3.4.69 BIG_256_56_sub()

```
void BIG_256_56_sub (
            BIG_256_56 x,
            BIG_256_56 y,
            BIG_256_56 z )
```

Parameters

x	BIG number, difference of other two - output not normalised
y	BIG number
z	BIG number

8.3.4.70 BIG_256_56_toBytes()

```
void BIG_256_56_toBytes (
            char * a,
            BIG_256_56 x )
```

Parameters

a	byte array
x	BIG number

8.3.4.71 BIG_256_56_zero()

```
void BIG_256_56_zero (
            BIG_256_56 x )
```

Parameters

x	BIG number to be set to zero

8.4 big_384_56.h File Reference

BIG Header File.

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "arch.h"
#include "amcl.h"
#include "config_big_384_56.h"
```

Macros

- #define BIGBITS_384_56 (8∗MODBYTES_384_56)
- #define NLEN_384_56 (1+((8∗MODBYTES_384_56-1)/BASEBITS_384_56))
- #define DNLEN_384_56 2∗NLEN_384_56
- #define BMASK_384_56 (((chunk)1<<BASEBITS_384_56)-1)
- #define NEXCESS_384_56 (1<<(CHUNK-BASEBITS_384_56-1))
- #define HBITS_384_56 (BASEBITS_384_56/2)
- #define HMASK_384_56 (((chunk)1<<HBITS_384_56)-1)

Typedefs

- typedef chunk BIG_384_56[NLEN_384_56]
- typedef chunk DBIG_384_56[DNLEN_384_56]

Functions

- int BIG_384_56_iszilch (BIG_384_56 x)

  Tests for BIG equal to zero.
- int BIG_384_56_isunity (BIG_384_56 x)

  Tests for BIG equal to one.
- int BIG_384_56_diszilch (DBIG_384_56 x)

  Tests for DBIG equal to zero.
- void BIG_384_56_output (BIG_384_56 x)

  Outputs a BIG number to the console.
- void BIG_384_56_rawoutput (BIG_384_56 x)

  Outputs a BIG number to the console in raw form (for debugging)
- void BIG_384_56_cswap (BIG_384_56 x, BIG_384_56 y, int s)

  Conditional constant time swap of two BIG numbers.
- void BIG_384_56_cmove (BIG_384_56 x, BIG_384_56 y, int s)

  Conditional copy of BIG number.
- void BIG_384_56_dcmove (BIG_384_56 x, BIG_384_56 y, int s)

  Conditional copy of DBIG number.
- void BIG_384_56_toBytes (char ∗a, BIG_384_56 x)

  Convert from BIG number to byte array.
- void BIG_384_56_fromBytes (BIG_384_56 x, char ∗a)

  Convert to BIG number from byte array.
- void BIG_384_56_fromBytesLen (BIG_384_56 x, char ∗a, int s)

  Convert to BIG number from byte array of given length.
- void BIG_384_56_dfromBytesLen (DBIG_384_56 x, char ∗a, int s)

  Convert to DBIG number from byte array of given length.
- void BIG_384_56_doutput (DBIG_384_56 x)

Outputs a DBIG number to the console.
- void BIG_384_56_drawoutput (DBIG_384_56 x)

  Outputs a DBIG number to the console.
- void BIG_384_56_rcopy (BIG_384_56 x, const BIG_384_56 y)

  Copy BIG from Read-Only Memory to a BIG.
- void BIG_384_56_copy (BIG_384_56 x, BIG_384_56 y)

  Copy BIG to another BIG.
- void BIG_384_56_dcopy (DBIG_384_56 x, DBIG_384_56 y)

  Copy DBIG to another DBIG.
- void BIG_384_56_dsucopy (DBIG_384_56 x, BIG_384_56 y)

  Copy BIG to upper half of DBIG.
- void BIG_384_56_dscopy (DBIG_384_56 x, BIG_384_56 y)

  Copy BIG to lower half of DBIG.
- void BIG_384_56_sdcopy (BIG_384_56 x, DBIG_384_56 y)

  Copy lower half of DBIG to a BIG.
- void BIG_384_56_sducopy (BIG_384_56 x, DBIG_384_56 y)

  Copy upper half of DBIG to a BIG.
- void BIG_384_56_zero (BIG_384_56 x)

  Set BIG to zero.
- void BIG_384_56_dzero (DBIG_384_56 x)

  Set DBIG to zero.
- void BIG_384_56_one (BIG_384_56 x)

  Set BIG to one (unity)
- void BIG_384_56_invmod2m (BIG_384_56 x)

  Set BIG to inverse mod 2^256.
- void BIG_384_56_add (BIG_384_56 x, BIG_384_56 y, BIG_384_56 z)

  Set BIG to sum of two BIGs - output not normalised.
- void BIG_384_56_or (BIG_384_56 x, BIG_384_56 y, BIG_384_56 z)

  Set BIG to logical or of two BIGs - output normalised.
- void BIG_384_56_inc (BIG_384_56 x, int i)

  Increment BIG by a small integer - output not normalised.
- void BIG_384_56_sub (BIG_384_56 x, BIG_384_56 y, BIG_384_56 z)

  Set BIG to difference of two BIGs.
- void BIG_384_56_dec (BIG_384_56 x, int i)

  Decrement BIG by a small integer - output not normalised.
- void BIG_384_56_dadd (DBIG_384_56 x, DBIG_384_56 y, DBIG_384_56 z)

  Set DBIG to sum of two DBIGs.
- void BIG_384_56_dsub (DBIG_384_56 x, DBIG_384_56 y, DBIG_384_56 z)

  Set DBIG to difference of two DBIGs.
- void BIG_384_56_imul (BIG_384_56 x, BIG_384_56 y, int i)

  Multiply BIG by a small integer - output not normalised.
- chunk BIG_384_56_pmul (BIG_384_56 x, BIG_384_56 y, int i)

  Multiply BIG by not-so-small small integer - output normalised.
- int BIG_384_56_div3 (BIG_384_56 x)

  Divide BIG by 3 - output normalised.
- void BIG_384_56_pxmul (DBIG_384_56 x, BIG_384_56 y, int i)

  Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.
- void BIG_384_56_mul (DBIG_384_56 x, BIG_384_56 y, BIG_384_56 z)

  Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.
- void BIG_384_56_smul (BIG_384_56 x, BIG_384_56 y, BIG_384_56 z)

  Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.

- void BIG_384_56_sqr (DBIG_384_56 x, BIG_384_56 y)

    Square BIG resulting in a DBIG - input normalised and output normalised.
- void BIG_384_56_monty (BIG_384_56 a, BIG_384_56 md, chunk MC, DBIG_384_56 d)

    Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.
- void BIG_384_56_shl (BIG_384_56 x, int s)

    Shifts a BIG left by any number of bits - input must be normalised, output normalised.
- int BIG_384_56_fshl (BIG_384_56 x, int s)

    Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised.
- void BIG_384_56_dshl (DBIG_384_56 x, int s)

    Shifts a DBIG left by any number of bits - input must be normalised, output normalised.
- void BIG_384_56_shr (BIG_384_56 x, int s)

    Shifts a BIG right by any number of bits - input must be normalised, output normalised.
- int BIG_384_56_ssn (BIG_384_56 r, BIG_384_56 a, BIG_384_56 m)

    Fast time-critical combined shift by 1 bit, subtract and normalise.
- int BIG_384_56_fshr (BIG_384_56 x, int s)

    Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised.
- void BIG_384_56_dshr (DBIG_384_56 x, int s)

    Shifts a DBIG right by any number of bits - input must be normalised, output normalised.
- chunk BIG_384_56_split (BIG_384_56 x, BIG_384_56 y, DBIG_384_56 z, int s)

    Splits a DBIG into two BIGs - input must be normalised, outputs normalised.
- chunk BIG_384_56_norm (BIG_384_56 x)

    Normalizes a BIG number - output normalised.
- void BIG_384_56_dnorm (DBIG_384_56 x)

    Normalizes a DBIG number - output normalised.
- int BIG_384_56_comp (BIG_384_56 x, BIG_384_56 y)

    Compares two BIG numbers. Inputs must be normalised externally.
- int BIG_384_56_dcomp (DBIG_384_56 x, DBIG_384_56 y)

    Compares two DBIG numbers. Inputs must be normalised externally.
- int BIG_384_56_nbits (BIG_384_56 x)

    Calculate number of bits in a BIG - output normalised.
- int BIG_384_56_dnbits (DBIG_384_56 x)

    Calculate number of bits in a DBIG - output normalised.
- void BIG_384_56_mod (BIG_384_56 x, BIG_384_56 n)

    Reduce x mod n - input and output normalised.
- void BIG_384_56_sdiv (BIG_384_56 x, BIG_384_56 n)

    Divide x by n - output normalised.
- void BIG_384_56_dmod (BIG_384_56 x, DBIG_384_56 y, BIG_384_56 n)

    x=y mod n - output normalised
- void BIG_384_56_ddiv (BIG_384_56 x, DBIG_384_56 y, BIG_384_56 n)

    x=y/n - output normalised
- int BIG_384_56_parity (BIG_384_56 x)

    return parity of BIG, that is the least significant bit
- int BIG_384_56_bit (BIG_384_56 x, int i)

    return i-th of BIG
- int BIG_384_56_lastbits (BIG_384_56 x, int n)

    return least significant bits of a BIG
- void BIG_384_56_random (BIG_384_56 x, csprng ∗r)

    Create a random BIG from a random number generator.
- void BIG_384_56_randomnum (BIG_384_56 x, BIG_384_56 n, csprng ∗r)

    Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.
- void BIG_384_56_modmul (BIG_384_56 x, BIG_384_56 y, BIG_384_56 z, BIG_384_56 n)

*Calculate x=y*z mod n.*

- void BIG_384_56_moddiv (BIG_384_56 x, BIG_384_56 y, BIG_384_56 z, BIG_384_56 n)

    Calculate x=y/z mod n.

- void BIG_384_56_modsqr (BIG_384_56 x, BIG_384_56 y, BIG_384_56 n)

    Calculate x=y^2 mod n.

- void BIG_384_56_modneg (BIG_384_56 x, BIG_384_56 y, BIG_384_56 n)

    Calculate x=-y mod n.

- int BIG_384_56_jacobi (BIG_384_56 x, BIG_384_56 y)

    Calculate jacobi Symbol (x/y)

- void BIG_384_56_invmodp (BIG_384_56 x, BIG_384_56 y, BIG_384_56 n)

    Calculate x=1/y mod n.

- void BIG_384_56_mod2m (BIG_384_56 x, int m)

    Calculate x=x mod 2^m.

- void BIG_384_56_dmod2m (DBIG_384_56 x, int m)

    Calculate x=x mod 2^m.

8.4.1 Detailed Description

Author

Mike Scott

8.4.2 Macro Definition Documentation

8.4.2.1 BIGBITS_384_56

```
#define BIGBITS_384_56 (8*MODBYTES_384_56)
```

Length in bits

8.4.2.2 BMASK_384_56

```
#define BMASK_384_56 (((chunk)1<<BASEBITS_384_56)-1)
```

Mask = 2^BASEBITS-1

8.4.2.3 DNLEN_384_56

```
#define DNLEN_384_56 2*NLEN_384_56
```

Double length in bytes

8.4.2.4 HBITS_384_56

```
#define HBITS_384_56 (BASEBITS_384_56/2)
```

Number of bits in number base divided by 2

8.4.2.5 HMASK_384_56

```
#define HMASK_384_56 (((chunk)1<<HBITS_384_56)-1)
```

Mask = 2^HBITS-1

8.4.2.6 NEXCESS_384_56

```
#define NEXCESS_384_56 (1<<(CHUNK-BASEBITS_384_56-1))
```

2^(CHUNK-BASEBITS-1) - digit cannot be multiplied by more than this before normalisation

8.4.2.7 NLEN_384_56

```
#define NLEN_384_56 (1+((8*MODBYTES_384_56-1)/BASEBITS_384_56))
```

length in bytes

8.4.3 Typedef Documentation

8.4.3.1 BIG_384_56

```
typedef chunk BIG_384_56[NLEN_384_56]
```

Define type BIG as array of chunks

8.4.3.2 DBIG_384_56

```
typedef chunk DBIG_384_56[DNLEN_384_56]
```

Define type DBIG as array of chunks

8.4.4 Function Documentation

8.4.4.1 BIG_384_56_add()

```
void BIG_384_56_add (
            BIG_384_56 x,
            BIG_384_56 y,
            BIG_384_56 z )
```

Parameters

x	BIG number, sum of other two
y	BIG number
z	BIG number

8.4.4.2 BIG_384_56_bit()

```
int BIG_384_56_bit (
            BIG_384_56 x,
            int i )
```

Parameters

x	BIG number
i	the bit of x to be returned

Returns

0 or 1

8.4.4.3 BIG_384_56_cmove()

```
void BIG_384_56_cmove (
            BIG_384_56 x,
            BIG_384_56 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a BIG number
y	another BIG number
s	copy takes place if not equal to 0

8.4.4.4 BIG_384_56_comp()

```
int BIG_384_56_comp (
            BIG_384_56 x,
            BIG_384_56 y )
```

Parameters

x	first BIG number to be compared
y	second BIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.4.4.5 BIG_384_56_copy()

```
void BIG_384_56_copy (
            BIG_384_56 x,
            BIG_384_56 y )
```

Parameters

x	BIG number
y	BIG number to be copied

8.4.4.6 BIG_384_56_cswap()

```
void BIG_384_56_cswap (
            BIG_384_56 x,
            BIG_384_56 y,
            int s )
```

Conditionally swaps parameters in constant time (without branching)

Parameters

x	a BIG number
y	another BIG number
s	swap takes place if not equal to 0

8.4.4.7 BIG_384_56_dadd()

```
void BIG_384_56_dadd (
            DBIG_384_56 x,
            DBIG_384_56 y,
            DBIG_384_56 z )
```

Parameters

x	DBIG number, sum of other two - output not normalised
y	DBIG number
z	DBIG number

8.4.4.8 BIG_384_56_dcmove()

```
void BIG_384_56_dcmove (
            BIG_384_56 x,
            BIG_384_56 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a DBIG number
y	another DBIG number
s	copy takes place if not equal to 0

8.4.4.9 BIG_384_56_dcomp()

```
int BIG_384_56_dcomp (
            DBIG_384_56 x,
            DBIG_384_56 y )
```

Parameters

x	first DBIG number to be compared
y	second DBIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.4.4.10 BIG_384_56_dcopy()

```
void BIG_384_56_dcopy (
            DBIG_384_56 x,
            DBIG_384_56 y )
```

Parameters

x	DBIG number
y	DBIG number to be copied

8.4.4.11 BIG_384_56_ddiv()

```
void BIG_384_56_ddiv (
            BIG_384_56 x,
            DBIG_384_56 y,
            BIG_384_56 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y/n
y	DBIG number
n	Modulus

8.4.4.12 BIG_384_56_dec()

```
void BIG_384_56_dec (
            BIG_384_56 x,
            int i )
```

Parameters

x	BIG number to be decremented
i	integer

8.4.4.13 BIG_384_56_dfromBytesLen()

```
void BIG_384_56_dfromBytesLen (
            DBIG_384_56 x,
            char * a,
            int s )
```

Parameters

x	DBIG number
a	byte array
s	byte array length

8.4.4.14 BIG_384_56_diszilch()

```
int BIG_384_56_diszilch (
            DBIG_384_56 x )
```

Parameters

x	a DBIG number

Returns

1 if zero, else returns 0

8.4.4.15 BIG_384_56_div3()

```
int BIG_384_56_div3 (
            BIG_384_56 x )
```

Parameters

x	BIG number

Returns

Remainder

8.4.4.16 BIG_384_56_dmod()

```
void BIG_384_56_dmod (
            BIG_384_56 x,
            DBIG_384_56 y,
            BIG_384_56 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y mod n
y	DBIG number
n	Modulus

8.4.4.17 BIG_384_56_dmod2m()

```
void BIG_384_56_dmod2m (
            DBIG_384_56 x,
            int m )
```

Truncation

Parameters

x	DBIG number, on reduced mod 2^m
m	new truncated size

8.4.4.18 BIG_384_56_dnbits()

```
int BIG_384_56_dnbits (
            DBIG_384_56 x )
```

Parameters

x	DBIG number

Returns

Number of bits in x

8.4.4.19 BIG_384_56_dnorm()

```
void BIG_384_56_dnorm (
            DBIG_384_56 x )
```

All digits of the input DBIG are reduced mod 2^BASEBITS

Parameters

x	DBIG number to be normalised

8.4.4.20 BIG_384_56_doutput()

```
void BIG_384_56_doutput (
```

```
            DBIG_384_56 x )
```

Parameters

x	a DBIG number

8.4.4.21 BIG_384_56_drawoutput()

```
void BIG_384_56_drawoutput (
            DBIG_384_56 x )
```

Parameters

x	a DBIG number

8.4.4.22 BIG_384_56_dscopy()

```
void BIG_384_56_dscopy (
            DBIG_384_56 x,
            BIG_384_56 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.4.4.23 BIG_384_56_dshl()

```
void BIG_384_56_dshl (
            DBIG_384_56 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.4.4.24 BIG_384_56_dshr()

```
void BIG_384_56_dshr (
            DBIG_384_56 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.4.4.25 BIG_384_56_dsub()

```
void BIG_384_56_dsub (
            DBIG_384_56 x,
            DBIG_384_56 y,
            DBIG_384_56 z )
```

Parameters

x	DBIG number, difference of other two - output not normalised
y	DBIG number
z	DBIG number

8.4.4.26 BIG_384_56_dsucopy()

```
void BIG_384_56_dsucopy (
            DBIG_384_56 x,
            BIG_384_56 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.4.4.27 BIG_384_56_dzero()

```
void BIG_384_56_dzero (
            DBIG_384_56 x )
```

Parameters

x	DBIG number to be set to zero

8.4.4.28 BIG_384_56_fromBytes()

```
void BIG_384_56_fromBytes (
            BIG_384_56 x,
            char * a )
```

Parameters

x	BIG number
a	byte array

8.4.4.29 BIG_384_56_fromBytesLen()

```
void BIG_384_56_fromBytesLen (
            BIG_384_56 x,
            char * a,
            int s )
```

Parameters

x	BIG number
a	byte array
s	byte array length

8.4.4.30 BIG_384_56_fshl()

```
int BIG_384_56_fshl (
            BIG_384_56 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Overflow bits

8.4.4.31 BIG_384_56_fshr()

```
int BIG_384_56_fshr (
            BIG_384_56 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Shifted out bits

8.4.4.32 BIG_384_56_imul()

```
void BIG_384_56_imul (
            BIG_384_56 x,
            BIG_384_56 y,
            int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

8.4.4.33 BIG_384_56_inc()

```
void BIG_384_56_inc (
            BIG_384_56 x,
            int i )
```

Parameters

x	BIG number to be incremented
i	integer

8.4.4.34 BIG_384_56_invmod2m()

```
void BIG_384_56_invmod2m (
            BIG_384_56 x )
```

Parameters

x	BIG number to be inverted

8.4.4.35 BIG_384_56_invmodp()

```
void BIG_384_56_invmodp (
            BIG_384_56 x,
            BIG_384_56 y,
            BIG_384_56 n )
```

Modular Inversion - This is slow. Uses binary method.

Parameters

x	BIG number, on exit = 1/y mod n
y	BIG number
n	The BIG Modulus

8.4.4.36 BIG_384_56_isunity()

```
int BIG_384_56_isunity (
            BIG_384_56 x )
```

Parameters

x	a BIG number

Returns

1 if one, else returns 0

8.4.4.37 BIG_384_56_iszilch()

```
int BIG_384_56_iszilch (
            BIG_384_56 x )
```

Parameters

x	a BIG number

Returns

1 if zero, else returns 0

8.4.4.38 BIG_384_56_jacobi()

```
int BIG_384_56_jacobi (
            BIG_384_56 x,
            BIG_384_56 y )
```

Parameters

x	BIG number
y	BIG number

Returns

Jacobi symbol, -1,0 or 1

8.4.4.39 BIG_384_56_lastbits()

```
int BIG_384_56_lastbits (
            BIG_384_56 x,
            int n )
```

Parameters

x	BIG number
n	number of bits to return. Assumed to be less than BASEBITS.

Returns

least significant n bits as an integer

8.4.4.40 BIG_384_56_mod()

```
void BIG_384_56_mod (
            BIG_384_56 x,
            BIG_384_56 n )
```

Slow but rarely used

Parameters

x	BIG number to be reduced mod n
n	The modulus

8.4.4.41 BIG_384_56_mod2m()

```
void BIG_384_56_mod2m (
            BIG_384_56 x,
            int m )
```

Truncation

Parameters

x	BIG number, on reduced mod 2^m
m	new truncated size

8.4.4.42 BIG_384_56_moddiv()

```
void BIG_384_56_moddiv (
            BIG_384_56 x,
            BIG_384_56 y,
            BIG_384_56 z,
            BIG_384_56 n )
```

Slow method for modular division

Parameters

x	BIG number, on exit = y/z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.4.4.43 BIG_384_56_modmul()

```
void BIG_384_56_modmul (
            BIG_384_56 x,
            BIG_384_56 y,
```

```
BIG_384_56 z,
BIG_384_56 n )
```

brief return NAF (Non-Adjacent-Form) value as +/- 1, 3 or 5, inputs must be normalised

Given x and 3*x extracts NAF value from given bit position, and returns number of bits processed, and number of trailing zeros detected if any param x BIG number param x3 BIG number, three times x param i bit position param nbs pointer to integer returning number of bits processed param nzs pointer to integer returning number of trailing 0s return + or - 1, 3 or 5Slow method for modular multiplication

Parameters

x	BIG number, on exit = y*z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.4.4.44 BIG_384_56_modneg()

```
void BIG_384_56_modneg (
            BIG_384_56 x,
            BIG_384_56 y,
            BIG_384_56 n )
```

Modular negation

Parameters

x	BIG number, on exit = -y mod n
y	BIG number
n	The BIG Modulus

8.4.4.45 BIG_384_56_modsqr()

```
void BIG_384_56_modsqr (
            BIG_384_56 x,
            BIG_384_56 y,
            BIG_384_56 n )
```

Slow method for modular squaring

Parameters

x	BIG number, on exit = y^2 mod n
y	BIG number
n	The BIG Modulus

8.4.4.46 BIG_384_56_monty()

```
void BIG_384_56_monty (
            BIG_384_56 a,
            BIG_384_56 md,
            chunk MC,
            DBIG_384_56 d )
```

Parameters

a	BIG number, reduction of a BIG
md	BIG number, the modulus
MC	the Montgomery Constant
d	DBIG number to be reduced

8.4.4.47 BIG_384_56_mul()

```
void BIG_384_56_mul (
            DBIG_384_56 x,
            BIG_384_56 y,
            BIG_384_56 z )
```

Parameters

x	DBIG number, product of other two
y	BIG number
z	BIG number

8.4.4.48 BIG_384_56_nbits()

```
int BIG_384_56_nbits (
            BIG_384_56 x )
```

Parameters

x	BIG number

Returns

Number of bits in x

8.4.4.49 BIG_384_56_norm()

```
chunk BIG_384_56_norm (
            BIG_384_56 x )
```

All digits of the input BIG are reduced mod 2^BASEBITS

Parameters

x	BIG number to be normalised

8.4.4.50 BIG_384_56_one()

```
void BIG_384_56_one (
            BIG_384_56 x )
```

Parameters

x	BIG number to be set to one.

8.4.4.51 BIG_384_56_or()

```
void BIG_384_56_or (
            BIG_384_56 x,
            BIG_384_56 y,
            BIG_384_56 z )
```

Parameters

x	BIG number, or of other two
y	BIG number
z	BIG number

8.4.4.52 BIG_384_56_output()

```
void BIG_384_56_output (
            BIG_384_56 x )
```

Parameters

x	a BIG number

8.4.4.53 BIG_384_56_parity()

```
int BIG_384_56_parity (
            BIG_384_56 x )
```

Parameters

x	BIG number

Returns

0 or 1

8.4.4.54 BIG_384_56_pmul()

```
chunk BIG_384_56_pmul (
            BIG_384_56 x,
            BIG_384_56 y,
            int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

Returns

Overflowing bits

8.4.4.55 BIG_384_56_pxmul()

```
void BIG_384_56_pxmul (
            DBIG_384_56 x,
            BIG_384_56 y,
            int i )
```

Parameters

x	DBIG number, product of other two
y	BIG number
i	small integer

8.4.4.56 BIG_384_56_random()

```
void BIG_384_56_random (
            BIG_384_56 x,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
r	A pointer to a Cryptographically Secure Random Number Generator

8.4.4.57 BIG_384_56_randomnum()

```
void BIG_384_56_randomnum (
            BIG_384_56 x,
            BIG_384_56 n,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
n	The modulus
r	A pointer to a Cryptographically Secure Random Number Generator

8.4.4.58 BIG_384_56_rawoutput()

```
void BIG_384_56_rawoutput (
            BIG_384_56 x )
```

Parameters

x	a BIG number

8.4.4.59 BIG_384_56_rcopy()

```
void BIG_384_56_rcopy (
```

```
            BIG_384_56 x,
            const BIG_384_56 y )
```

Parameters

x	BIG number
y	BIG number in ROM

8.4.4.60 BIG_384_56_sdcopy()

```
void BIG_384_56_sdcopy (
            BIG_384_56 x,
            DBIG_384_56 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.4.4.61 BIG_384_56_sdiv()

```
void BIG_384_56_sdiv (
            BIG_384_56 x,
            BIG_384_56 n )
```

Slow but rarely used

Parameters

x	BIG number to be divided by n
n	The Divisor

8.4.4.62 BIG_384_56_sducopy()

```
void BIG_384_56_sducopy (
            BIG_384_56 x,
            DBIG_384_56 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.4.4.63 BIG_384_56_shl()

```
void BIG_384_56_shl (
            BIG_384_56 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.4.4.64 BIG_384_56_shr()

```
void BIG_384_56_shr (
            BIG_384_56 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.4.4.65 BIG_384_56_smul()

```
void BIG_384_56_smul (
            BIG_384_56 x,
            BIG_384_56 y,
            BIG_384_56 z )
```

Note that the product must fit into a BIG, and x must be distinct from y and z

Parameters

x	BIG number, product of other two
y	BIG number
z	BIG number

8.4.4.66 BIG_384_56_split()

```
chunk BIG_384_56_split (
```

```
            BIG_384_56 x,
            BIG_384_56 y,
            DBIG_384_56 z,
            int s )
```

Internal function. The value of s must be approximately in the middle of the DBIG. Typically used to extract z mod 2^MODBITS and z/2^MODBITS

Parameters

x	BIG number, top half of z
y	BIG number, bottom half of z
z	DBIG number to be split in two.
s	Bit position at which to split

Returns

carry-out from top half

8.4.4.67 BIG_384_56_sqr()

```
void BIG_384_56_sqr (
            DBIG_384_56 x,
            BIG_384_56 y )
```

Parameters

x	DBIG number, square of a BIG
y	BIG number to be squared

8.4.4.68 BIG_384_56_ssn()

```
int BIG_384_56_ssn (
            BIG_384_56 r,
            BIG_384_56 a,
            BIG_384_56 m )
```

Parameters

r	BIG number normalised output
a	BIG number to be subtracted from
m	BIG number to be shifted and subtracted

Returns

sign of r

8.4.4.69 BIG_384_56_sub()

```
void BIG_384_56_sub (
            BIG_384_56 x,
            BIG_384_56 y,
            BIG_384_56 z )
```

Parameters

x	BIG number, difference of other two - output not normalised
y	BIG number
z	BIG number

8.4.4.70 BIG_384_56_toBytes()

```
void BIG_384_56_toBytes (
            char * a,
            BIG_384_56 x )
```

Parameters

a	byte array
x	BIG number

8.4.4.71 BIG_384_56_zero()

```
void BIG_384_56_zero (
            BIG_384_56 x )
```

Parameters

x	BIG number to be set to zero

8.5 big_384_58.h File Reference

BIG Header File.

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "arch.h"
#include "amcl.h"
#include "config_big_384_58.h"
```

Macros

- #define BIGBITS_384_58 (8*MODBYTES_384_58)
- #define NLEN_384_58 (1+((8*MODBYTES_384_58-1)/BASEBITS_384_58))
- #define DNLEN_384_58 2*NLEN_384_58
- #define BMASK_384_58 (((chunk)1<<BASEBITS_384_58)-1)
- #define NEXCESS_384_58 (1<<(CHUNK-BASEBITS_384_58-1))
- #define HBITS_384_58 (BASEBITS_384_58/2)
- #define HMASK_384_58 (((chunk)1<<HBITS_384_58)-1)

Typedefs

- typedef chunk BIG_384_58[NLEN_384_58]
- typedef chunk DBIG_384_58[DNLEN_384_58]

Functions

- int BIG_384_58_iszilch (BIG_384_58 x)

    Tests for BIG equal to zero.
- int BIG_384_58_isunity (BIG_384_58 x)

    Tests for BIG equal to one.
- int BIG_384_58_diszilch (DBIG_384_58 x)

    Tests for DBIG equal to zero.
- void BIG_384_58_output (BIG_384_58 x)

    Outputs a BIG number to the console.
- void BIG_384_58_rawoutput (BIG_384_58 x)

    Outputs a BIG number to the console in raw form (for debugging)
- void BIG_384_58_cswap (BIG_384_58 x, BIG_384_58 y, int s)

    Conditional constant time swap of two BIG numbers.
- void BIG_384_58_cmove (BIG_384_58 x, BIG_384_58 y, int s)

    Conditional copy of BIG number.
- void BIG_384_58_dcmove (BIG_384_58 x, BIG_384_58 y, int s)

    Conditional copy of DBIG number.
- void BIG_384_58_toBytes (char *a, BIG_384_58 x)

    Convert from BIG number to byte array.
- void BIG_384_58_fromBytes (BIG_384_58 x, char *a)

    Convert to BIG number from byte array.
- void BIG_384_58_fromBytesLen (BIG_384_58 x, char *a, int s)

    Convert to BIG number from byte array of given length.
- void BIG_384_58_dfromBytesLen (DBIG_384_58 x, char *a, int s)

    Convert to DBIG number from byte array of given length.
- void BIG_384_58_doutput (DBIG_384_58 x)

Outputs a DBIG number to the console.

- void BIG_384_58_drawoutput (DBIG_384_58 x)

  Outputs a DBIG number to the console.
- void BIG_384_58_rcopy (BIG_384_58 x, const BIG_384_58 y)

  Copy BIG from Read-Only Memory to a BIG.
- void BIG_384_58_copy (BIG_384_58 x, BIG_384_58 y)

  Copy BIG to another BIG.
- void BIG_384_58_dcopy (DBIG_384_58 x, DBIG_384_58 y)

  Copy DBIG to another DBIG.
- void BIG_384_58_dsucopy (DBIG_384_58 x, BIG_384_58 y)

  Copy BIG to upper half of DBIG.
- void BIG_384_58_dscopy (DBIG_384_58 x, BIG_384_58 y)

  Copy BIG to lower half of DBIG.
- void BIG_384_58_sdcopy (BIG_384_58 x, DBIG_384_58 y)

  Copy lower half of DBIG to a BIG.
- void BIG_384_58_sducopy (BIG_384_58 x, DBIG_384_58 y)

  Copy upper half of DBIG to a BIG.
- void BIG_384_58_zero (BIG_384_58 x)

  Set BIG to zero.
- void BIG_384_58_dzero (DBIG_384_58 x)

  Set DBIG to zero.
- void BIG_384_58_one (BIG_384_58 x)

  Set BIG to one (unity)
- void BIG_384_58_invmod2m (BIG_384_58 x)

  Set BIG to inverse mod 2^256.
- void BIG_384_58_add (BIG_384_58 x, BIG_384_58 y, BIG_384_58 z)

  Set BIG to sum of two BIGs - output not normalised.
- void BIG_384_58_or (BIG_384_58 x, BIG_384_58 y, BIG_384_58 z)

  Set BIG to logical or of two BIGs - output normalised.
- void BIG_384_58_inc (BIG_384_58 x, int i)

  Increment BIG by a small integer - output not normalised.
- void BIG_384_58_sub (BIG_384_58 x, BIG_384_58 y, BIG_384_58 z)

  Set BIG to difference of two BIGs.
- void BIG_384_58_dec (BIG_384_58 x, int i)

  Decrement BIG by a small integer - output not normalised.
- void BIG_384_58_dadd (DBIG_384_58 x, DBIG_384_58 y, DBIG_384_58 z)

  Set DBIG to sum of two DBIGs.
- void BIG_384_58_dsub (DBIG_384_58 x, DBIG_384_58 y, DBIG_384_58 z)

  Set DBIG to difference of two DBIGs.
- void BIG_384_58_imul (BIG_384_58 x, BIG_384_58 y, int i)

  Multiply BIG by a small integer - output not normalised.
- chunk BIG_384_58_pmul (BIG_384_58 x, BIG_384_58 y, int i)

  Multiply BIG by not-so-small small integer - output normalised.
- int BIG_384_58_div3 (BIG_384_58 x)

  Divide BIG by 3 - output normalised.
- void BIG_384_58_pxmul (DBIG_384_58 x, BIG_384_58 y, int i)

  Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.
- void BIG_384_58_mul (DBIG_384_58 x, BIG_384_58 y, BIG_384_58 z)

  Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.
- void BIG_384_58_smul (BIG_384_58 x, BIG_384_58 y, BIG_384_58 z)

  Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.

- void BIG_384_58_sqr (DBIG_384_58 x, BIG_384_58 y)

  Square BIG resulting in a DBIG - input normalised and output normalised.
- void BIG_384_58_monty (BIG_384_58 a, BIG_384_58 md, chunk MC, DBIG_384_58 d)

  Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.
- void BIG_384_58_shl (BIG_384_58 x, int s)

  Shifts a BIG left by any number of bits - input must be normalised, output normalised.
- int BIG_384_58_fshl (BIG_384_58 x, int s)

  Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised.
- void BIG_384_58_dshl (DBIG_384_58 x, int s)

  Shifts a DBIG left by any number of bits - input must be normalised, output normalised.
- void BIG_384_58_shr (BIG_384_58 x, int s)

  Shifts a BIG right by any number of bits - input must be normalised, output normalised.
- int BIG_384_58_ssn (BIG_384_58 r, BIG_384_58 a, BIG_384_58 m)

  Fast time-critical combined shift by 1 bit, subtract and normalise.
- int BIG_384_58_fshr (BIG_384_58 x, int s)

  Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised.
- void BIG_384_58_dshr (DBIG_384_58 x, int s)

  Shifts a DBIG right by any number of bits - input must be normalised, output normalised.
- chunk BIG_384_58_split (BIG_384_58 x, BIG_384_58 y, DBIG_384_58 z, int s)

  Splits a DBIG into two BIGs - input must be normalised, outputs normalised.
- chunk BIG_384_58_norm (BIG_384_58 x)

  Normalizes a BIG number - output normalised.
- void BIG_384_58_dnorm (DBIG_384_58 x)

  Normalizes a DBIG number - output normalised.
- int BIG_384_58_comp (BIG_384_58 x, BIG_384_58 y)

  Compares two BIG numbers. Inputs must be normalised externally.
- int BIG_384_58_dcomp (DBIG_384_58 x, DBIG_384_58 y)

  Compares two DBIG numbers. Inputs must be normalised externally.
- int BIG_384_58_nbits (BIG_384_58 x)

  Calculate number of bits in a BIG - output normalised.
- int BIG_384_58_dnbits (DBIG_384_58 x)

  Calculate number of bits in a DBIG - output normalised.
- void BIG_384_58_mod (BIG_384_58 x, BIG_384_58 n)

  Reduce x mod n - input and output normalised.
- void BIG_384_58_sdiv (BIG_384_58 x, BIG_384_58 n)

  Divide x by n - output normalised.
- void BIG_384_58_dmod (BIG_384_58 x, DBIG_384_58 y, BIG_384_58 n)

  x=y mod n - output normalised
- void BIG_384_58_ddiv (BIG_384_58 x, DBIG_384_58 y, BIG_384_58 n)

  x=y/n - output normalised
- int BIG_384_58_parity (BIG_384_58 x)

  return parity of BIG, that is the least significant bit
- int BIG_384_58_bit (BIG_384_58 x, int i)

  return i-th of BIG
- int BIG_384_58_lastbits (BIG_384_58 x, int n)

  return least significant bits of a BIG
- void BIG_384_58_random (BIG_384_58 x, csprng *r)

  Create a random BIG from a random number generator.
- void BIG_384_58_randomnum (BIG_384_58 x, BIG_384_58 n, csprng *r)

  Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.
- void BIG_384_58_modmul (BIG_384_58 x, BIG_384_58 y, BIG_384_58 z, BIG_384_58 n)

*Calculate x=y*z mod n.*

- void BIG_384_58_moddiv (BIG_384_58 x, BIG_384_58 y, BIG_384_58 z, BIG_384_58 n)

  Calculate x=y/z mod n.
- void BIG_384_58_modsqr (BIG_384_58 x, BIG_384_58 y, BIG_384_58 n)

  Calculate x=y^2 mod n.
- void BIG_384_58_modneg (BIG_384_58 x, BIG_384_58 y, BIG_384_58 n)

  Calculate x=-y mod n.
- int BIG_384_58_jacobi (BIG_384_58 x, BIG_384_58 y)

  Calculate jacobi Symbol (x/y)
- void BIG_384_58_invmodp (BIG_384_58 x, BIG_384_58 y, BIG_384_58 n)

  Calculate x=1/y mod n.
- void BIG_384_58_mod2m (BIG_384_58 x, int m)

  Calculate x=x mod 2^m.
- void BIG_384_58_dmod2m (DBIG_384_58 x, int m)

  Calculate x=x mod 2^m.

8.5.1 Detailed Description

Author

Mike Scott

8.5.2 Macro Definition Documentation

8.5.2.1 BIGBITS_384_58

```
#define BIGBITS_384_58 (8*MODBYTES_384_58)
```

Length in bits

8.5.2.2 BMASK_384_58

```
#define BMASK_384_58 (((chunk)1<<BASEBITS_384_58)-1)
```

Mask = 2^BASEBITS-1

8.5.2.3 DNLEN_384_58

```
#define DNLEN_384_58 2*NLEN_384_58
```

Double length in bytes

8.5.2.4 HBITS_384_58

```
#define HBITS_384_58 (BASEBITS_384_58/2)
```

Number of bits in number base divided by 2

8.5.2.5 HMASK_384_58

```
#define HMASK_384_58 (((chunk)1<<HBITS_384_58)-1)
```

Mask = 2^HBITS-1

8.5.2.6 NEXCESS_384_58

```
#define NEXCESS_384_58 (1<<(CHUNK-BASEBITS_384_58-1))
```

2^(CHUNK-BASEBITS-1) - digit cannot be multiplied by more than this before normalisation

8.5.2.7 NLEN_384_58

```
#define NLEN_384_58 (1+((8*MODBYTES_384_58-1)/BASEBITS_384_58))
```

length in bytes

8.5.3 Typedef Documentation

8.5.3.1 BIG_384_58

```
typedef chunk BIG_384_58[NLEN_384_58]
```

Define type BIG as array of chunks

8.5.3.2 DBIG_384_58

```
typedef chunk DBIG_384_58[DNLEN_384_58]
```

Define type DBIG as array of chunks

8.5.4 Function Documentation

8.5.4.1 BIG_384_58_add()

```
void BIG_384_58_add (
            BIG_384_58 x,
            BIG_384_58 y,
            BIG_384_58 z )
```

Parameters

x	BIG number, sum of other two
y	BIG number
z	BIG number

8.5.4.2 BIG_384_58_bit()

```
int BIG_384_58_bit (
            BIG_384_58 x,
            int i )
```

Parameters

x	BIG number
i	the bit of x to be returned

Returns

0 or 1

8.5.4.3 BIG_384_58_cmove()

```
void BIG_384_58_cmove (
            BIG_384_58 x,
            BIG_384_58 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a BIG number
y	another BIG number
s	copy takes place if not equal to 0

8.5.4.4 BIG_384_58_comp()

```
int BIG_384_58_comp (
            BIG_384_58 x,
            BIG_384_58 y )
```

Parameters

x	first BIG number to be compared
y	second BIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.5.4.5 BIG_384_58_copy()

```
void BIG_384_58_copy (
            BIG_384_58 x,
            BIG_384_58 y )
```

Parameters

x	BIG number
y	BIG number to be copied

8.5.4.6 BIG_384_58_cswap()

```
void BIG_384_58_cswap (
            BIG_384_58 x,
            BIG_384_58 y,
            int s )
```

Conditionally swaps parameters in constant time (without branching)

Parameters

x	a BIG number
y	another BIG number
s	swap takes place if not equal to 0

8.5.4.7 BIG_384_58_dadd()

```
void BIG_384_58_dadd (
            DBIG_384_58 x,
            DBIG_384_58 y,
            DBIG_384_58 z )
```

Parameters

x	DBIG number, sum of other two - output not normalised
y	DBIG number
z	DBIG number

8.5.4.8 BIG_384_58_dcmove()

```
void BIG_384_58_dcmove (
            BIG_384_58 x,
            BIG_384_58 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a DBIG number
y	another DBIG number
s	copy takes place if not equal to 0

8.5.4.9 BIG_384_58_dcomp()

```
int BIG_384_58_dcomp (
            DBIG_384_58 x,
            DBIG_384_58 y )
```

Parameters

x	first DBIG number to be compared
y	second DBIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.5.4.10 BIG_384_58_dcopy()

```
void BIG_384_58_dcopy (
            DBIG_384_58 x,
            DBIG_384_58 y )
```

Parameters

x	DBIG number
y	DBIG number to be copied

8.5.4.11 BIG_384_58_ddiv()

```
void BIG_384_58_ddiv (
            BIG_384_58 x,
            DBIG_384_58 y,
            BIG_384_58 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y/n
y	DBIG number
n	Modulus

8.5.4.12 BIG_384_58_dec()

```
void BIG_384_58_dec (
            BIG_384_58 x,
            int i )
```

Parameters

x	BIG number to be decremented
i	integer

8.5.4.13 BIG_384_58_dfromBytesLen()

```
void BIG_384_58_dfromBytesLen (
            DBIG_384_58 x,
            char * a,
            int s )
```

Parameters

x	DBIG number
a	byte array
s	byte array length

8.5.4.14 BIG_384_58_diszilch()

```
int BIG_384_58_diszilch (
            DBIG_384_58 x )
```

Parameters

x	a DBIG number

Returns

1 if zero, else returns 0

8.5.4.15 BIG_384_58_div3()

```
int BIG_384_58_div3 (
            BIG_384_58 x )
```

Parameters

x	BIG number

Returns

Remainder

8.5.4.16 BIG_384_58_dmod()

```
void BIG_384_58_dmod (
            BIG_384_58 x,
            DBIG_384_58 y,
            BIG_384_58 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y mod n
y	DBIG number
n	Modulus

8.5.4.17 BIG_384_58_dmod2m()

```
void BIG_384_58_dmod2m (
            DBIG_384_58 x,
            int m )
```

Truncation

Parameters

x	DBIG number, on reduced mod 2^m
m	new truncated size

8.5.4.18 BIG_384_58_dnbits()

```
int BIG_384_58_dnbits (
            DBIG_384_58 x )
```

Parameters

x	DBIG number

Returns

Number of bits in x

8.5.4.19 BIG_384_58_dnorm()

```
void BIG_384_58_dnorm (
            DBIG_384_58 x )
```

All digits of the input DBIG are reduced mod 2^BASEBITS

Parameters

x	DBIG number to be normalised

8.5.4.20 BIG_384_58_doutput()

```
void BIG_384_58_doutput (
```

```
            DBIG_384_58 x )
```

Parameters

x	a DBIG number

8.5.4.21 BIG_384_58_drawoutput()

```
void BIG_384_58_drawoutput (
            DBIG_384_58 x )
```

Parameters

x	a DBIG number

8.5.4.22 BIG_384_58_dscopy()

```
void BIG_384_58_dscopy (
            DBIG_384_58 x,
            BIG_384_58 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.5.4.23 BIG_384_58_dshl()

```
void BIG_384_58_dshl (
            DBIG_384_58 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.5.4.24 BIG_384_58_dshr()

```
void BIG_384_58_dshr (
            DBIG_384_58 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.5.4.25 BIG_384_58_dsub()

```
void BIG_384_58_dsub (
            DBIG_384_58 x,
            DBIG_384_58 y,
            DBIG_384_58 z )
```

Parameters

x	DBIG number, difference of other two - output not normalised
y	DBIG number
z	DBIG number

8.5.4.26 BIG_384_58_dsucopy()

```
void BIG_384_58_dsucopy (
            DBIG_384_58 x,
            BIG_384_58 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.5.4.27 BIG_384_58_dzero()

```
void BIG_384_58_dzero (
            DBIG_384_58 x )
```

Parameters

x	DBIG number to be set to zero

8.5.4.28 BIG_384_58_fromBytes()

```
void BIG_384_58_fromBytes (
            BIG_384_58 x,
            char * a )
```

Parameters

x	BIG number
a	byte array

8.5.4.29 BIG_384_58_fromBytesLen()

```
void BIG_384_58_fromBytesLen (
            BIG_384_58 x,
            char * a,
            int s )
```

Parameters

x	BIG number
a	byte array
s	byte array length

8.5.4.30 BIG_384_58_fshl()

```
int BIG_384_58_fshl (
            BIG_384_58 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Overflow bits

8.5.4.31 BIG_384_58_fshr()

```
int BIG_384_58_fshr (
            BIG_384_58 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Shifted out bits

8.5.4.32 BIG_384_58_imul()

```
void BIG_384_58_imul (
            BIG_384_58 x,
            BIG_384_58 y,
            int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

8.5.4.33 BIG_384_58_inc()

```
void BIG_384_58_inc (
            BIG_384_58 x,
            int i )
```

Parameters

x	BIG number to be incremented
i	integer

8.5.4.34 BIG_384_58_invmod2m()

```
void BIG_384_58_invmod2m (
            BIG_384_58 x )
```

Parameters

x	BIG number to be inverted

8.5.4.35 BIG_384_58_invmodp()

```
void BIG_384_58_invmodp (
            BIG_384_58 x,
            BIG_384_58 y,
            BIG_384_58 n )
```

Modular Inversion - This is slow. Uses binary method.

Parameters

x	BIG number, on exit = 1/y mod n
y	BIG number
n	The BIG Modulus

8.5.4.36 BIG_384_58_isunity()

```
int BIG_384_58_isunity (
            BIG_384_58 x )
```

Parameters

x	a BIG number

Returns

1 if one, else returns 0

8.5.4.37 BIG_384_58_iszilch()

```
int BIG_384_58_iszilch (
            BIG_384_58 x )
```

Parameters

x	a BIG number

Returns

1 if zero, else returns 0

8.5.4.38 BIG_384_58_jacobi()

```
int BIG_384_58_jacobi (
            BIG_384_58 x,
            BIG_384_58 y )
```

Parameters

x	BIG number
y	BIG number

Returns

Jacobi symbol, -1,0 or 1

8.5.4.39 BIG_384_58_lastbits()

```
int BIG_384_58_lastbits (
            BIG_384_58 x,
            int n )
```

Parameters

x	BIG number
n	number of bits to return. Assumed to be less than BASEBITS.

Returns

least significant n bits as an integer

8.5.4.40 BIG_384_58_mod()

```
void BIG_384_58_mod (
            BIG_384_58 x,
            BIG_384_58 n )
```

Slow but rarely used

Parameters

x	BIG number to be reduced mod n
n	The modulus

8.5.4.41 BIG_384_58_mod2m()

```
void BIG_384_58_mod2m (
            BIG_384_58 x,
            int m )
```

Truncation

Parameters

x	BIG number, on reduced mod 2^m
m	new truncated size

8.5.4.42 BIG_384_58_moddiv()

```
void BIG_384_58_moddiv (
            BIG_384_58 x,
            BIG_384_58 y,
            BIG_384_58 z,
            BIG_384_58 n )
```

Slow method for modular division

Parameters

x	BIG number, on exit = y/z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.5.4.43 BIG_384_58_modmul()

```
void BIG_384_58_modmul (
            BIG_384_58 x,
            BIG_384_58 y,
```

```
            BIG_384_58 z,
            BIG_384_58 n )
```

brief return NAF (Non-Adjacent-Form) value as +/- 1, 3 or 5, inputs must be normalised

Given x and 3∗x extracts NAF value from given bit position, and returns number of bits processed, and number of trailing zeros detected if any param x BIG number param x3 BIG number, three times x param i bit position param nbs pointer to integer returning number of bits processed param nzs pointer to integer returning number of trailing 0s return + or - 1, 3 or 5Slow method for modular multiplication

Parameters

x	BIG number, on exit = y∗z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.5.4.44 BIG_384_58_modneg()

```
void BIG_384_58_modneg (
            BIG_384_58 x,
            BIG_384_58 y,
            BIG_384_58 n )
```

Modular negation

Parameters

x	BIG number, on exit = -y mod n
y	BIG number
n	The BIG Modulus

8.5.4.45 BIG_384_58_modsqr()

```
void BIG_384_58_modsqr (
            BIG_384_58 x,
            BIG_384_58 y,
            BIG_384_58 n )
```

Slow method for modular squaring

Parameters

x	BIG number, on exit = y^2 mod n
y	BIG number
n	The BIG Modulus

8.5.4.46 BIG_384_58_monty()

```
void BIG_384_58_monty (
            BIG_384_58 a,
            BIG_384_58 md,
            chunk MC,
            DBIG_384_58 d )
```

Parameters

a	BIG number, reduction of a BIG
md	BIG number, the modulus
MC	the Montgomery Constant
d	DBIG number to be reduced

8.5.4.47 BIG_384_58_mul()

```
void BIG_384_58_mul (
            DBIG_384_58 x,
            BIG_384_58 y,
            BIG_384_58 z )
```

Parameters

x	DBIG number, product of other two
y	BIG number
z	BIG number

8.5.4.48 BIG_384_58_nbits()

```
int BIG_384_58_nbits (
            BIG_384_58 x )
```

Parameters

x	BIG number

Returns

Number of bits in x

8.5.4.49 BIG_384_58_norm()

```
chunk BIG_384_58_norm (
            BIG_384_58 x )
```

All digits of the input BIG are reduced mod 2^BASEBITS

Parameters

x	BIG number to be normalised

8.5.4.50 BIG_384_58_one()

```
void BIG_384_58_one (
            BIG_384_58 x )
```

Parameters

x	BIG number to be set to one.

8.5.4.51 BIG_384_58_or()

```
void BIG_384_58_or (
            BIG_384_58 x,
            BIG_384_58 y,
            BIG_384_58 z )
```

Parameters

x	BIG number, or of other two
y	BIG number
z	BIG number

8.5.4.52 BIG_384_58_output()

```
void BIG_384_58_output (
            BIG_384_58 x )
```

Parameters

x	a BIG number

8.5.4.53 BIG_384_58_parity()

```
int BIG_384_58_parity (
            BIG_384_58 x )
```

Parameters

x	BIG number

Returns

0 or 1

8.5.4.54 BIG_384_58_pmul()

```
chunk BIG_384_58_pmul (
            BIG_384_58 x,
            BIG_384_58 y,
            int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

Returns

Overflowing bits

8.5.4.55 BIG_384_58_pxmul()

```
void BIG_384_58_pxmul (
            DBIG_384_58 x,
            BIG_384_58 y,
            int i )
```

Parameters

x	DBIG number, product of other two
y	BIG number
i	small integer

8.5.4.56 BIG_384_58_random()

```
void BIG_384_58_random (
            BIG_384_58 x,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
r	A pointer to a Cryptographically Secure Random Number Generator

8.5.4.57 BIG_384_58_randomnum()

```
void BIG_384_58_randomnum (
            BIG_384_58 x,
            BIG_384_58 n,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
n	The modulus
r	A pointer to a Cryptographically Secure Random Number Generator

8.5.4.58 BIG_384_58_rawoutput()

```
void BIG_384_58_rawoutput (
            BIG_384_58 x )
```

Parameters

x	a BIG number

8.5.4.59 BIG_384_58_rcopy()

```
void BIG_384_58_rcopy (
```

```
BIG_384_58 x,
const BIG_384_58 y )
```

Parameters

x	BIG number
y	BIG number in ROM

8.5.4.60 BIG_384_58_sdcopy()

```
void BIG_384_58_sdcopy (
            BIG_384_58 x,
            DBIG_384_58 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.5.4.61 BIG_384_58_sdiv()

```
void BIG_384_58_sdiv (
            BIG_384_58 x,
            BIG_384_58 n )
```

Slow but rarely used

Parameters

x	BIG number to be divided by n
n	The Divisor

8.5.4.62 BIG_384_58_sducopy()

```
void BIG_384_58_sducopy (
            BIG_384_58 x,
            DBIG_384_58 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.5.4.63 BIG_384_58_shl()

```
void BIG_384_58_shl (
            BIG_384_58 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.5.4.64 BIG_384_58_shr()

```
void BIG_384_58_shr (
            BIG_384_58 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.5.4.65 BIG_384_58_smul()

```
void BIG_384_58_smul (
            BIG_384_58 x,
            BIG_384_58 y,
            BIG_384_58 z )
```

Note that the product must fit into a BIG, and x must be distinct from y and z

Parameters

x	BIG number, product of other two
y	BIG number
z	BIG number

8.5.4.66 BIG_384_58_split()

```
chunk BIG_384_58_split (
```

```
            BIG_384_58 x,
            BIG_384_58 y,
            DBIG_384_58 z,
            int s )
```

Internal function. The value of s must be approximately in the middle of the DBIG. Typically used to extract z mod 2^MODBITS and z/2^MODBITS

Parameters

x	BIG number, top half of z
y	BIG number, bottom half of z
z	DBIG number to be split in two.
s	Bit position at which to split

Returns

carry-out from top half

8.5.4.67 BIG_384_58_sqr()

```
void BIG_384_58_sqr (
            DBIG_384_58 x,
            BIG_384_58 y )
```

Parameters

x	DBIG number, square of a BIG
y	BIG number to be squared

8.5.4.68 BIG_384_58_ssn()

```
int BIG_384_58_ssn (
            BIG_384_58 r,
            BIG_384_58 a,
            BIG_384_58 m )
```

Parameters

r	BIG number normalised output
a	BIG number to be subtracted from
m	BIG number to be shifted and subtracted

Returns

sign of r

8.5.4.69 BIG_384_58_sub()

```
void BIG_384_58_sub (
            BIG_384_58 x,
            BIG_384_58 y,
            BIG_384_58 z )
```

Parameters

x	BIG number, difference of other two - output not normalised
y	BIG number
z	BIG number

8.5.4.70 BIG_384_58_toBytes()

```
void BIG_384_58_toBytes (
            char * a,
            BIG_384_58 x )
```

Parameters

a	byte array
x	BIG number

8.5.4.71 BIG_384_58_zero()

```
void BIG_384_58_zero (
            BIG_384_58 x )
```

Parameters

x	BIG number to be set to zero

8.6 big_448_58.h File Reference

BIG Header File.

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "arch.h"
#include "amcl.h"
#include "config_big_448_58.h"
```

Macros

- #define BIGBITS_448_58 (8∗MODBYTES_448_58)
- #define NLEN_448_58 (1+((8∗MODBYTES_448_58-1)/BASEBITS_448_58))
- #define DNLEN_448_58 2∗NLEN_448_58
- #define BMASK_448_58 (((chunk)1<<BASEBITS_448_58)-1)
- #define NEXCESS_448_58 (1<<(CHUNK-BASEBITS_448_58-1))
- #define HBITS_448_58 (BASEBITS_448_58/2)
- #define HMASK_448_58 (((chunk)1<<HBITS_448_58)-1)

Typedefs

- typedef chunk BIG_448_58[NLEN_448_58]
- typedef chunk DBIG_448_58[DNLEN_448_58]

Functions

- int BIG_448_58_iszilch (BIG_448_58 x)

  Tests for BIG equal to zero.
- int BIG_448_58_isunity (BIG_448_58 x)

  Tests for BIG equal to one.
- int BIG_448_58_diszilch (DBIG_448_58 x)

  Tests for DBIG equal to zero.
- void BIG_448_58_output (BIG_448_58 x)

  Outputs a BIG number to the console.
- void BIG_448_58_rawoutput (BIG_448_58 x)

  Outputs a BIG number to the console in raw form (for debugging)
- void BIG_448_58_cswap (BIG_448_58 x, BIG_448_58 y, int s)

  Conditional constant time swap of two BIG numbers.
- void BIG_448_58_cmove (BIG_448_58 x, BIG_448_58 y, int s)

  Conditional copy of BIG number.
- void BIG_448_58_dcmove (BIG_448_58 x, BIG_448_58 y, int s)

  Conditional copy of DBIG number.
- void BIG_448_58_toBytes (char ∗a, BIG_448_58 x)

  Convert from BIG number to byte array.
- void BIG_448_58_fromBytes (BIG_448_58 x, char ∗a)

  Convert to BIG number from byte array.
- void BIG_448_58_fromBytesLen (BIG_448_58 x, char ∗a, int s)

  Convert to BIG number from byte array of given length.
- void BIG_448_58_dfromBytesLen (DBIG_448_58 x, char ∗a, int s)

  Convert to DBIG number from byte array of given length.
- void BIG_448_58_doutput (DBIG_448_58 x)

Outputs a DBIG number to the console.
- void BIG_448_58_drawoutput (DBIG_448_58 x)

  Outputs a DBIG number to the console.
- void BIG_448_58_rcopy (BIG_448_58 x, const BIG_448_58 y)

  Copy BIG from Read-Only Memory to a BIG.
- void BIG_448_58_copy (BIG_448_58 x, BIG_448_58 y)

  Copy BIG to another BIG.
- void BIG_448_58_dcopy (DBIG_448_58 x, DBIG_448_58 y)

  Copy DBIG to another DBIG.
- void BIG_448_58_dsucopy (DBIG_448_58 x, BIG_448_58 y)

  Copy BIG to upper half of DBIG.
- void BIG_448_58_dscopy (DBIG_448_58 x, BIG_448_58 y)

  Copy BIG to lower half of DBIG.
- void BIG_448_58_sdcopy (BIG_448_58 x, DBIG_448_58 y)

  Copy lower half of DBIG to a BIG.
- void BIG_448_58_sducopy (BIG_448_58 x, DBIG_448_58 y)

  Copy upper half of DBIG to a BIG.
- void BIG_448_58_zero (BIG_448_58 x)

  Set BIG to zero.
- void BIG_448_58_dzero (DBIG_448_58 x)

  Set DBIG to zero.
- void BIG_448_58_one (BIG_448_58 x)

  Set BIG to one (unity)
- void BIG_448_58_invmod2m (BIG_448_58 x)

  Set BIG to inverse mod 2^256.
- void BIG_448_58_add (BIG_448_58 x, BIG_448_58 y, BIG_448_58 z)

  Set BIG to sum of two BIGs - output not normalised.
- void BIG_448_58_or (BIG_448_58 x, BIG_448_58 y, BIG_448_58 z)

  Set BIG to logical or of two BIGs - output normalised.
- void BIG_448_58_inc (BIG_448_58 x, int i)

  Increment BIG by a small integer - output not normalised.
- void BIG_448_58_sub (BIG_448_58 x, BIG_448_58 y, BIG_448_58 z)

  Set BIG to difference of two BIGs.
- void BIG_448_58_dec (BIG_448_58 x, int i)

  Decrement BIG by a small integer - output not normalised.
- void BIG_448_58_dadd (DBIG_448_58 x, DBIG_448_58 y, DBIG_448_58 z)

  Set DBIG to sum of two DBIGs.
- void BIG_448_58_dsub (DBIG_448_58 x, DBIG_448_58 y, DBIG_448_58 z)

  Set DBIG to difference of two DBIGs.
- void BIG_448_58_imul (BIG_448_58 x, BIG_448_58 y, int i)

  Multiply BIG by a small integer - output not normalised.
- chunk BIG_448_58_pmul (BIG_448_58 x, BIG_448_58 y, int i)

  Multiply BIG by not-so-small small integer - output normalised.
- int BIG_448_58_div3 (BIG_448_58 x)

  Divide BIG by 3 - output normalised.
- void BIG_448_58_pxmul (DBIG_448_58 x, BIG_448_58 y, int i)

  Multiply BIG by even bigger small integer resulting in a DBIG - output normalised.
- void BIG_448_58_mul (DBIG_448_58 x, BIG_448_58 y, BIG_448_58 z)

  Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised.
- void BIG_448_58_smul (BIG_448_58 x, BIG_448_58 y, BIG_448_58 z)

  Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised.

- void BIG_448_58_sqr (DBIG_448_58 x, BIG_448_58 y)

  Square BIG resulting in a DBIG - input normalised and output normalised.
- void BIG_448_58_monty (BIG_448_58 a, BIG_448_58 md, chunk MC, DBIG_448_58 d)

  Montgomery reduction of a DBIG to a BIG - input normalised and output normalised.
- void BIG_448_58_shl (BIG_448_58 x, int s)

  Shifts a BIG left by any number of bits - input must be normalised, output normalised.
- int BIG_448_58_fshl (BIG_448_58 x, int s)

  Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised.
- void BIG_448_58_dshl (DBIG_448_58 x, int s)

  Shifts a DBIG left by any number of bits - input must be normalised, output normalised.
- void BIG_448_58_shr (BIG_448_58 x, int s)

  Shifts a BIG right by any number of bits - input must be normalised, output normalised.
- int BIG_448_58_ssn (BIG_448_58 r, BIG_448_58 a, BIG_448_58 m)

  Fast time-critical combined shift by 1 bit, subtract and normalise.
- int BIG_448_58_fshr (BIG_448_58 x, int s)

  Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised.
- void BIG_448_58_dshr (DBIG_448_58 x, int s)

  Shifts a DBIG right by any number of bits - input must be normalised, output normalised.
- chunk BIG_448_58_split (BIG_448_58 x, BIG_448_58 y, DBIG_448_58 z, int s)

  Splits a DBIG into two BIGs - input must be normalised, outputs normalised.
- chunk BIG_448_58_norm (BIG_448_58 x)

  Normalizes a BIG number - output normalised.
- void BIG_448_58_dnorm (DBIG_448_58 x)

  Normalizes a DBIG number - output normalised.
- int BIG_448_58_comp (BIG_448_58 x, BIG_448_58 y)

  Compares two BIG numbers. Inputs must be normalised externally.
- int BIG_448_58_dcomp (DBIG_448_58 x, DBIG_448_58 y)

  Compares two DBIG numbers. Inputs must be normalised externally.
- int BIG_448_58_nbits (BIG_448_58 x)

  Calculate number of bits in a BIG - output normalised.
- int BIG_448_58_dnbits (DBIG_448_58 x)

  Calculate number of bits in a DBIG - output normalised.
- void BIG_448_58_mod (BIG_448_58 x, BIG_448_58 n)

  Reduce x mod n - input and output normalised.
- void BIG_448_58_sdiv (BIG_448_58 x, BIG_448_58 n)

  Divide x by n - output normalised.
- void BIG_448_58_dmod (BIG_448_58 x, DBIG_448_58 y, BIG_448_58 n)

  x=y mod n - output normalised
- void BIG_448_58_ddiv (BIG_448_58 x, DBIG_448_58 y, BIG_448_58 n)

  x=y/n - output normalised
- int BIG_448_58_parity (BIG_448_58 x)

  return parity of BIG, that is the least significant bit
- int BIG_448_58_bit (BIG_448_58 x, int i)

  return i-th of BIG
- int BIG_448_58_lastbits (BIG_448_58 x, int n)

  return least significant bits of a BIG
- void BIG_448_58_random (BIG_448_58 x, csprng *r)

  Create a random BIG from a random number generator.
- void BIG_448_58_randomnum (BIG_448_58 x, BIG_448_58 n, csprng *r)

  Create an unbiased random BIG from a random number generator, reduced with respect to a modulus.
- void BIG_448_58_modmul (BIG_448_58 x, BIG_448_58 y, BIG_448_58 z, BIG_448_58 n)

Calculate x=y∗z mod n.
- void BIG_448_58_moddiv (BIG_448_58 x, BIG_448_58 y, BIG_448_58 z, BIG_448_58 n)

  Calculate x=y/z mod n.
- void BIG_448_58_modsqr (BIG_448_58 x, BIG_448_58 y, BIG_448_58 n)

  Calculate x=y^2 mod n.
- void BIG_448_58_modneg (BIG_448_58 x, BIG_448_58 y, BIG_448_58 n)

  Calculate x=-y mod n.
- int BIG_448_58_jacobi (BIG_448_58 x, BIG_448_58 y)

  Calculate jacobi Symbol (x/y)
- void BIG_448_58_invmodp (BIG_448_58 x, BIG_448_58 y, BIG_448_58 n)

  Calculate x=1/y mod n.
- void BIG_448_58_mod2m (BIG_448_58 x, int m)

  Calculate x=x mod 2^m.
- void BIG_448_58_dmod2m (DBIG_448_58 x, int m)

  Calculate x=x mod 2^m.

8.6.1 Detailed Description

Author

Mike Scott

8.6.2 Macro Definition Documentation

8.6.2.1 BIGBITS_448_58

```
#define BIGBITS_448_58 (8*MODBYTES_448_58)
```

Length in bits

8.6.2.2 BMASK_448_58

```
#define BMASK_448_58 (((chunk)1<<BASEBITS_448_58)-1)
```

Mask = 2^BASEBITS-1

8.6.2.3 DNLEN_448_58

```
#define DNLEN_448_58 2*NLEN_448_58
```

Double length in bytes

8.6.2.4 HBITS_448_58

```
#define HBITS_448_58 (BASEBITS_448_58/2)
```

Number of bits in number base divided by 2

8.6.2.5 HMASK_448_58

```
#define HMASK_448_58 (((chunk)1<<HBITS_448_58)-1)
```

Mask = 2^HBITS-1

8.6.2.6 NEXCESS_448_58

```
#define NEXCESS_448_58 (1<<(CHUNK-BASEBITS_448_58-1))
```

2^(CHUNK-BASEBITS-1) - digit cannot be multiplied by more than this before normalisation

8.6.2.7 NLEN_448_58

```
#define NLEN_448_58 (1+((8*MODBYTES_448_58-1)/BASEBITS_448_58))
```

length in bytes

8.6.3 Typedef Documentation

8.6.3.1 BIG_448_58

```
typedef chunk BIG_448_58[NLEN_448_58]
```

Define type BIG as array of chunks

8.6.3.2 DBIG_448_58

```
typedef chunk DBIG_448_58[DNLEN_448_58]
```

Define type DBIG as array of chunks

8.6.4 Function Documentation

8.6.4.1 BIG_448_58_add()

```
void BIG_448_58_add (
            BIG_448_58 x,
            BIG_448_58 y,
            BIG_448_58 z )
```

Parameters

x	BIG number, sum of other two
y	BIG number
z	BIG number

8.6.4.2 BIG_448_58_bit()

```
int BIG_448_58_bit (
            BIG_448_58 x,
            int i )
```

Parameters

x	BIG number
i	the bit of x to be returned

Returns

0 or 1

8.6.4.3 BIG_448_58_cmove()

```
void BIG_448_58_cmove (
            BIG_448_58 x,
            BIG_448_58 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a BIG number
y	another BIG number
s	copy takes place if not equal to 0

8.6.4.4 BIG_448_58_comp()

```
int BIG_448_58_comp (
            BIG_448_58 x,
            BIG_448_58 y )
```

Parameters

x	first BIG number to be compared
y	second BIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.6.4.5 BIG_448_58_copy()

```
void BIG_448_58_copy (
            BIG_448_58 x,
            BIG_448_58 y )
```

Parameters

x	BIG number
y	BIG number to be copied

8.6.4.6 BIG_448_58_cswap()

```
void BIG_448_58_cswap (
            BIG_448_58 x,
            BIG_448_58 y,
            int s )
```

Conditionally swaps parameters in constant time (without branching)

Parameters

x	a BIG number
y	another BIG number
s	swap takes place if not equal to 0

8.6.4.7 BIG_448_58_dadd()

```
void BIG_448_58_dadd (
            DBIG_448_58 x,
            DBIG_448_58 y,
            DBIG_448_58 z )
```

Parameters

x	DBIG number, sum of other two - output not normalised
y	DBIG number
z	DBIG number

8.6.4.8 BIG_448_58_dcmove()

```
void BIG_448_58_dcmove (
            BIG_448_58 x,
            BIG_448_58 y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	a DBIG number
y	another DBIG number
s	copy takes place if not equal to 0

8.6.4.9 BIG_448_58_dcomp()

```
int BIG_448_58_dcomp (
            DBIG_448_58 x,
            DBIG_448_58 y )
```

Parameters

x	first DBIG number to be compared
y	second DBIG number to be compared

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.6.4.10 BIG_448_58_dcopy()

```
void BIG_448_58_dcopy (
            DBIG_448_58 x,
            DBIG_448_58 y )
```

Parameters

x	DBIG number
y	DBIG number to be copied

8.6.4.11 BIG_448_58_ddiv()

```
void BIG_448_58_ddiv (
            BIG_448_58 x,
            DBIG_448_58 y,
            BIG_448_58 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y/n
y	DBIG number
n	Modulus

8.6.4.12 BIG_448_58_dec()

```
void BIG_448_58_dec (
            BIG_448_58 x,
            int i )
```

Parameters

x	BIG number to be decremented
i	integer

8.6.4.13 BIG_448_58_dfromBytesLen()

```
void BIG_448_58_dfromBytesLen (
            DBIG_448_58 x,
            char * a,
            int s )
```

Parameters

x	DBIG number
a	byte array
s	byte array length

8.6.4.14 BIG_448_58_diszilch()

```
int BIG_448_58_diszilch (
            DBIG_448_58 x )
```

Parameters

x	a DBIG number

Returns

1 if zero, else returns 0

8.6.4.15 BIG_448_58_div3()

```
int BIG_448_58_div3 (
            BIG_448_58 x )
```

Parameters

x	BIG number

Returns

Remainder

8.6.4.16 BIG_448_58_dmod()

```
void BIG_448_58_dmod (
            BIG_448_58 x,
            DBIG_448_58 y,
            BIG_448_58 n )
```

Slow but rarely used. y is destroyed.

Parameters

x	BIG number, on exit = y mod n
y	DBIG number
n	Modulus

8.6.4.17 BIG_448_58_dmod2m()

```
void BIG_448_58_dmod2m (
            DBIG_448_58 x,
            int m )
```

Truncation

Parameters

x	DBIG number, on reduced mod 2^m
m	new truncated size

8.6.4.18 BIG_448_58_dnbits()

```
int BIG_448_58_dnbits (
            DBIG_448_58 x )
```

Parameters

x	DBIG number

Returns

Number of bits in x

8.6.4.19 BIG_448_58_dnorm()

```
void BIG_448_58_dnorm (
            DBIG_448_58 x )
```

All digits of the input DBIG are reduced mod 2^BASEBITS

Parameters

x	DBIG number to be normalised

8.6.4.20 BIG_448_58_doutput()

```
void BIG_448_58_doutput (
```

```
            DBIG_448_58 x )
```

Parameters

x	a DBIG number

8.6.4.21 BIG_448_58_drawoutput()

```
void BIG_448_58_drawoutput (
            DBIG_448_58 x )
```

Parameters

x	a DBIG number

8.6.4.22 BIG_448_58_dscopy()

```
void BIG_448_58_dscopy (
            DBIG_448_58 x,
            BIG_448_58 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.6.4.23 BIG_448_58_dshl()

```
void BIG_448_58_dshl (
            DBIG_448_58 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.6.4.24 BIG_448_58_dshr()

```
void BIG_448_58_dshr (
            DBIG_448_58 x,
            int s )
```

Parameters

x	DBIG number to be shifted
s	Number of bits to shift

8.6.4.25 BIG_448_58_dsub()

```
void BIG_448_58_dsub (
            DBIG_448_58 x,
            DBIG_448_58 y,
            DBIG_448_58 z )
```

Parameters

x	DBIG number, difference of other two - output not normalised
y	DBIG number
z	DBIG number

8.6.4.26 BIG_448_58_dsucopy()

```
void BIG_448_58_dsucopy (
            DBIG_448_58 x,
            BIG_448_58 y )
```

Parameters

x	DBIG number
y	BIG number to be copied

8.6.4.27 BIG_448_58_dzero()

```
void BIG_448_58_dzero (
            DBIG_448_58 x )
```

Parameters

x	DBIG number to be set to zero

8.6.4.28 BIG_448_58_fromBytes()

```
void BIG_448_58_fromBytes (
            BIG_448_58 x,
            char * a )
```

Parameters

x	BIG number
a	byte array

8.6.4.29 BIG_448_58_fromBytesLen()

```
void BIG_448_58_fromBytesLen (
            BIG_448_58 x,
            char * a,
            int s )
```

Parameters

x	BIG number
a	byte array
s	byte array length

8.6.4.30 BIG_448_58_fshl()

```
int BIG_448_58_fshl (
            BIG_448_58 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Overflow bits

8.6.4.31 BIG_448_58_fshr()

```
int BIG_448_58_fshr (
            BIG_448_58 x,
            int s )
```

The number of bits to be shifted must be less than BASEBITS

Parameters

x	BIG number to be shifted
s	Number of bits to shift

Returns

Shifted out bits

8.6.4.32 BIG_448_58_imul()

```
void BIG_448_58_imul (
            BIG_448_58 x,
            BIG_448_58 y,
            int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

8.6.4.33 BIG_448_58_inc()

```
void BIG_448_58_inc (
            BIG_448_58 x,
            int i )
```

Parameters

x	BIG number to be incremented
i	integer

8.6.4.34 BIG_448_58_invmod2m()

```
void BIG_448_58_invmod2m (
            BIG_448_58 x )
```

Parameters

x	BIG number to be inverted

8.6.4.35 BIG_448_58_invmodp()

```
void BIG_448_58_invmodp (
            BIG_448_58 x,
            BIG_448_58 y,
            BIG_448_58 n )
```

Modular Inversion - This is slow. Uses binary method.

Parameters

x	BIG number, on exit = 1/y mod n
y	BIG number
n	The BIG Modulus

8.6.4.36 BIG_448_58_isunity()

```
int BIG_448_58_isunity (
            BIG_448_58 x )
```

Parameters

x	a BIG number

Returns

1 if one, else returns 0

8.6.4.37 BIG_448_58_iszilch()

```
int BIG_448_58_iszilch (
            BIG_448_58 x )
```

Parameters

x	a BIG number

Returns

1 if zero, else returns 0

8.6.4.38 BIG_448_58_jacobi()

```
int BIG_448_58_jacobi (
            BIG_448_58 x,
            BIG_448_58 y )
```

Parameters

x	BIG number
y	BIG number

Returns

Jacobi symbol, -1,0 or 1

8.6.4.39 BIG_448_58_lastbits()

```
int BIG_448_58_lastbits (
            BIG_448_58 x,
            int n )
```

Parameters

x	BIG number
n	number of bits to return. Assumed to be less than BASEBITS.

Returns

least significant n bits as an integer

8.6.4.40 BIG_448_58_mod()

```
void BIG_448_58_mod (
            BIG_448_58 x,
            BIG_448_58 n )
```

Slow but rarely used

Parameters

x	BIG number to be reduced mod n
n	The modulus

8.6.4.41 BIG_448_58_mod2m()

```
void BIG_448_58_mod2m (
            BIG_448_58 x,
            int m )
```

Truncation

Parameters

x	BIG number, on reduced mod 2^m
m	new truncated size

8.6.4.42 BIG_448_58_moddiv()

```
void BIG_448_58_moddiv (
            BIG_448_58 x,
            BIG_448_58 y,
            BIG_448_58 z,
            BIG_448_58 n )
```

Slow method for modular division

Parameters

x	BIG number, on exit = y/z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.6.4.43 BIG_448_58_modmul()

```
void BIG_448_58_modmul (
            BIG_448_58 x,
            BIG_448_58 y,
```

```
BIG_448_58 z,
BIG_448_58 n )
```

brief return NAF (Non-Adjacent-Form) value as +/- 1, 3 or 5, inputs must be normalised

Given x and 3∗x extracts NAF value from given bit position, and returns number of bits processed, and number of trailing zeros detected if any param x BIG number param x3 BIG number, three times x param i bit position param nbs pointer to integer returning number of bits processed param nzs pointer to integer returning number of trailing 0s return + or - 1, 3 or 5Slow method for modular multiplication

Parameters

x	BIG number, on exit = y∗z mod n
y	BIG number
z	BIG number
n	The BIG Modulus

8.6.4.44 BIG_448_58_modneg()

```
void BIG_448_58_modneg (
            BIG_448_58 x,
            BIG_448_58 y,
            BIG_448_58 n )
```

Modular negation

Parameters

x	BIG number, on exit = -y mod n
y	BIG number
n	The BIG Modulus

8.6.4.45 BIG_448_58_modsqr()

```
void BIG_448_58_modsqr (
            BIG_448_58 x,
            BIG_448_58 y,
            BIG_448_58 n )
```

Slow method for modular squaring

Parameters

x	BIG number, on exit = y^2 mod n
y	BIG number
n	The BIG Modulus

8.6.4.46 BIG_448_58_monty()

```
void BIG_448_58_monty (
            BIG_448_58 a,
            BIG_448_58 md,
            chunk MC,
            DBIG_448_58 d )
```

Parameters

a	BIG number, reduction of a BIG
md	BIG number, the modulus
MC	the Montgomery Constant
d	DBIG number to be reduced

8.6.4.47 BIG_448_58_mul()

```
void BIG_448_58_mul (
            DBIG_448_58 x,
            BIG_448_58 y,
            BIG_448_58 z )
```

Parameters

x	DBIG number, product of other two
y	BIG number
z	BIG number

8.6.4.48 BIG_448_58_nbits()

```
int BIG_448_58_nbits (
            BIG_448_58 x )
```

Parameters

x	BIG number

Returns

Number of bits in x

8.6.4.49 BIG_448_58_norm()

```
chunk BIG_448_58_norm (
            BIG_448_58 x )
```

All digits of the input BIG are reduced mod 2^BASEBITS

Parameters

x	BIG number to be normalised

8.6.4.50 BIG_448_58_one()

```
void BIG_448_58_one (
            BIG_448_58 x )
```

Parameters

x	BIG number to be set to one.

8.6.4.51 BIG_448_58_or()

```
void BIG_448_58_or (
            BIG_448_58 x,
            BIG_448_58 y,
            BIG_448_58 z )
```

Parameters

x	BIG number, or of other two
y	BIG number
z	BIG number

8.6.4.52 BIG_448_58_output()

```
void BIG_448_58_output (
            BIG_448_58 x )
```

Parameters

x	a BIG number

8.6.4.53 BIG_448_58_parity()

```
int BIG_448_58_parity (
            BIG_448_58 x )
```

Parameters

x	BIG number

Returns

0 or 1

8.6.4.54 BIG_448_58_pmul()

```
chunk BIG_448_58_pmul (
            BIG_448_58 x,
            BIG_448_58 y,
            int i )
```

Parameters

x	BIG number, product of other two
y	BIG number
i	small integer

Returns

Overflowing bits

8.6.4.55 BIG_448_58_pxmul()

```
void BIG_448_58_pxmul (
            DBIG_448_58 x,
            BIG_448_58 y,
            int i )
```

Parameters

x	DBIG number, product of other two
y	BIG number
i	small integer

8.6.4.56 BIG_448_58_random()

```
void BIG_448_58_random (
            BIG_448_58 x,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
r	A pointer to a Cryptographically Secure Random Number Generator

8.6.4.57 BIG_448_58_randomnum()

```
void BIG_448_58_randomnum (
            BIG_448_58 x,
            BIG_448_58 n,
            csprng * r )
```

Assumes that the random number generator has been suitably initialised

Parameters

x	BIG number, on exit a random number
n	The modulus
r	A pointer to a Cryptographically Secure Random Number Generator

8.6.4.58 BIG_448_58_rawoutput()

```
void BIG_448_58_rawoutput (
            BIG_448_58 x )
```

Parameters

x	a BIG number

8.6.4.59 BIG_448_58_rcopy()

```
void BIG_448_58_rcopy (
```

```
            BIG_448_58 x,
            const BIG_448_58 y )
```

Parameters

x	BIG number
y	BIG number in ROM

8.6.4.60 BIG_448_58_sdcopy()

```
void BIG_448_58_sdcopy (
            BIG_448_58 x,
            DBIG_448_58 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.6.4.61 BIG_448_58_sdiv()

```
void BIG_448_58_sdiv (
            BIG_448_58 x,
            BIG_448_58 n )
```

Slow but rarely used

Parameters

x	BIG number to be divided by n
n	The Divisor

8.6.4.62 BIG_448_58_sducopy()

```
void BIG_448_58_sducopy (
            BIG_448_58 x,
            DBIG_448_58 y )
```

Parameters

x	BIG number
y	DBIG number to be copied

8.6.4.63 BIG_448_58_shl()

```
void BIG_448_58_shl (
            BIG_448_58 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.6.4.64 BIG_448_58_shr()

```
void BIG_448_58_shr (
            BIG_448_58 x,
            int s )
```

Parameters

x	BIG number to be shifted
s	Number of bits to shift

8.6.4.65 BIG_448_58_smul()

```
void BIG_448_58_smul (
            BIG_448_58 x,
            BIG_448_58 y,
            BIG_448_58 z )
```

Note that the product must fit into a BIG, and x must be distinct from y and z

Parameters

x	BIG number, product of other two
y	BIG number
z	BIG number

8.6.4.66 BIG_448_58_split()

```
chunk BIG_448_58_split (
```

```
            BIG_448_58 x,
            BIG_448_58 y,
            DBIG_448_58 z,
            int s )
```

Internal function. The value of s must be approximately in the middle of the DBIG. Typically used to extract z mod 2^MODBITS and z/2^MODBITS

Parameters

x	BIG number, top half of z
y	BIG number, bottom half of z
z	DBIG number to be split in two.
s	Bit position at which to split

Returns

carry-out from top half

8.6.4.67 BIG_448_58_sqr()

```
void BIG_448_58_sqr (
            DBIG_448_58 x,
            BIG_448_58 y )
```

Parameters

x	DBIG number, square of a BIG
y	BIG number to be squared

8.6.4.68 BIG_448_58_ssn()

```
int BIG_448_58_ssn (
            BIG_448_58 r,
            BIG_448_58 a,
            BIG_448_58 m )
```

Parameters

r	BIG number normalised output
a	BIG number to be subtracted from
m	BIG number to be shifted and subtracted

Returns

sign of r

8.6.4.69 BIG_448_58_sub()

```
void BIG_448_58_sub (
            BIG_448_58 x,
            BIG_448_58 y,
            BIG_448_58 z )
```

Parameters

x	BIG number, difference of other two - output not normalised
y	BIG number
z	BIG number

8.6.4.70 BIG_448_58_toBytes()

```
void BIG_448_58_toBytes (
            char * a,
            BIG_448_58 x )
```

Parameters

a	byte array
x	BIG number

8.6.4.71 BIG_448_58_zero()

```
void BIG_448_58_zero (
            BIG_448_58 x )
```

Parameters

x	BIG number to be set to zero

8.7 bls_BLS381.h File Reference

BLS Header file.

```
#include "pair_BLS381.h"
```

Macros

- #define BGS_BLS381 MODBYTES_384_58
- #define BFS_BLS381 MODBYTES_384_58
- #define BLS_OK 0
- #define BLS_FAIL 41
- #define BLS_INVALID_G1 42
- #define BLS_INVALID_G2 43

Functions

- int BLS_BLS381_KEY_PAIR_GENERATE (csprng ∗RNG, octet ∗S, octet ∗W)

  Generate Key Pair.
- int BLS_BLS381_SIGN (octet ∗SIG, char ∗m, octet ∗S)

  Calculate a signature.
- int BLS_BLS381_VERIFY (octet ∗SIG, char ∗m, octet ∗W)

  Verify a signature.
- int BLS_BLS381_ADD_G1 (octet ∗R1, octet ∗R2, octet ∗R)

  Add two members from the group G1.
- int BLS_BLS381_ADD_G2 (octet ∗W1, octet ∗W2, octet ∗W)

  Add two members from the group G2.

8.7.1 Detailed Description

Author

Mike Scott

Date

28th Novemebr 2018 Allows some user configuration defines structures declares functions

8.7.2 Macro Definition Documentation

8.7.2.1 BFS_BLS381

```
#define BFS_BLS381 MODBYTES_384_58
```

BLS Field Size

8.7.2.2 BGS_BLS381

```
#define BGS_BLS381 MODBYTES_384_58
```

BLS Group Size

8.7.2.3 BLS_FAIL

```
#define BLS_FAIL 41
```

Invalid signature

8.7.2.4 BLS_INVALID_G1

```
#define BLS_INVALID_G1 42
```

Not a valid G1 point on the curve

8.7.2.5 BLS_INVALID_G2

```
#define BLS_INVALID_G2 43
```

Not a valid G2 point on the curve

8.7.2.6 BLS_OK

```
#define BLS_OK 0
```

Function completed without error

8.7.3 Function Documentation

8.7.3.1 BLS_BLS381_ADD_G1()

```
int BLS_BLS381_ADD_G1 (
            octet * R1,
            octet * R2,
            octet * R )
```

Parameters

R1	member of G1
R2	member of G1
R	member of G1. R = R1+R2

Returns

Zero for success or else an error code

8.7.3.2 BLS_BLS381_ADD_G2()

```
int BLS_BLS381_ADD_G2 (
            octet * W1,
            octet * W2,
            octet * W )
```

Parameters

W1	member of G2
W2	member of G2
W	member of G2. W = W1+W2

Returns

Zero for success or else an error code

8.7.3.3 BLS_BLS381_KEY_PAIR_GENERATE()

```
int BLS_BLS381_KEY_PAIR_GENERATE (
            csprng * RNG,
            octet * S,
            octet * W )
```

Parameters

RNG	Pointer to a cryptographically secure random number generator
S	Private key
W	Public Key. W = S∗G, where G is fixed generator

Returns

Zero for success or else an error code

8.7.3.4 BLS_BLS381_SIGN()

```
int BLS_BLS381_SIGN (
            octet * SIG,
            char * m,
            octet * S )
```

Parameters

SIG	signature
m	message to be signed
S	Private key

Returns

Zero for success or else an error code

8.7.3.5 BLS_BLS381_VERIFY()

```
int BLS_BLS381_VERIFY (
            octet * SIG,
            char * m,
            octet * W )
```

Parameters

SIG	signature
m	message whose signature is to be verified.
W	Public key

Returns

Zero for success or else an error code

8.8 config_big_1024_58.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

Macros

- #define MODBYTES_1024_58 128
- #define BASEBITS_1024_58 58

8.8.1 Detailed Description

Author

Mike Scott

8.8.2 Macro Definition Documentation

8.8.2.1 BASEBITS_1024_58

```
#define BASEBITS_1024_58 58
```

Numbers represented to base 2*BASEBITS

8.8.2.2 MODBYTES_1024_58

```
#define MODBYTES_1024_58 128
```

Number of bytes in Modulus

8.9 config_big_256_56.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

Macros

- #define MODBYTES_256_56 32
- #define BASEBITS_256_56 56

8.9.1 Detailed Description

Author

Mike Scott

8.9.2 Macro Definition Documentation

8.9.2.1 BASEBITS_256_56

```
#define BASEBITS_256_56 56
```

Numbers represented to base 2*BASEBITS

8.9.2.2 MODBYTES_256_56

```
#define MODBYTES_256_56 32
```

Number of bytes in Modulus

8.10 config_big_384_56.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

Macros

- #define MODBYTES_384_56 48
- #define BASEBITS_384_56 56

8.10.1 Detailed Description

Author

Mike Scott

8.10.2 Macro Definition Documentation

8.10.2.1 BASEBITS_384_56

```
#define BASEBITS_384_56 56
```

Numbers represented to base 2*BASEBITS

8.10.2.2 MODBYTES_384_56

```
#define MODBYTES_384_56 48
```

Number of bytes in Modulus

8.11 config_big_384_58.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

Macros

- #define MODBYTES_384_58 48
- #define BASEBITS_384_58 58

8.11.1 Detailed Description

Author

Mike Scott

8.11.2 Macro Definition Documentation

8.11.2.1 BASEBITS_384_58

```
#define BASEBITS_384_58 58
```

Numbers represented to base 2*BASEBITS

8.11.2.2 MODBYTES_384_58

```
#define MODBYTES_384_58 48
```

Number of bytes in Modulus

8.12 config_big_448_58.h File Reference

Config BIG Header File.

```
#include "amcl.h"
```

Macros

- #define MODBYTES_448_58 56
- #define BASEBITS_448_58 58

8.12.1 Detailed Description

Author

Mike Scott

8.12.2 Macro Definition Documentation

8.12.2.1 BASEBITS_448_58

```
#define BASEBITS_448_58 58
```

Numbers represented to base 2*BASEBITS

8.12.2.2 MODBYTES_448_58

```
#define MODBYTES_448_58 56
```

Number of bytes in Modulus

8.13 config_ff_2048.h File Reference

COnfig FF Header File.

```
#include "amcl.h"
#include "config_big_1024_58.h"
```

Macros

- #define FFLEN_2048 2

8.13.1 Detailed Description

Author

Mike Scott

8.13.2 Macro Definition Documentation

8.13.2.1 FFLEN_2048

```
#define FFLEN_2048 2
```

2^n multiplier of BIGBITS to specify supported Finite Field size, e.g 2048=256*2^3 where BIGBITS=256

8.14 config_ff_3072.h File Reference

COnfig FF Header File.

```
#include "amcl.h"
#include "config_big_384_56.h"
```

Macros

- #define FFLEN_3072 8

8.14.1 Detailed Description

Author

Mike Scott

8.14.2 Macro Definition Documentation

8.14.2.1 FFLEN_3072

```
#define FFLEN_3072 8
```

2^n multiplier of BIGBITS to specify supported Finite Field size, e.g 2048=256∗2^3 where BIGBITS=256

8.15 ecdh_BLS381.h File Reference

ECDH Header file for implementation of standard EC protocols.

```
#include "ecp_BLS381.h"
#include "ecdh_support.h"
```

Macros

- #define EGS_BLS381 MODBYTES_384_58
- #define EFS_BLS381 MODBYTES_384_58
- #define ECDH_OK 0
- #define ECDH_INVALID_PUBLIC_KEY -2
- #define ECDH_ERROR -3
- #define ECDH_INVALID -4

Functions

- int ECP_BLS381_KEY_PAIR_GENERATE (csprng *R, octet *s, octet *W)

  Generate an ECC public/private key pair.
- int ECP_BLS381_PUBLIC_KEY_VALIDATE (octet *W)

  Validate an ECC public key.
- int ECP_BLS381_SVDP_DH (octet *s, octet *W, octet *K)

  Generate Diffie-Hellman shared key.
- void ECP_BLS381_ECIES_ENCRYPT (int h, octet *P1, octet *P2, csprng *R, octet *W, octet *M, int len, octet *V, octet *C, octet *T)

  ECIES Encryption.
- int ECP_BLS381_ECIES_DECRYPT (int h, octet *P1, octet *P2, octet *V, octet *C, octet *T, octet *U, octet *M)

  ECIES Decryption.
- int ECP_BLS381_SP_DSA (int h, csprng *R, octet *k, octet *s, octet *M, octet *c, octet *d)

  ECDSA Signature.
- int ECP_BLS381_VP_DSA (int h, octet *W, octet *M, octet *c, octet *d)

  ECDSA Signature Verification.

8.15.1 Detailed Description

Author

Mike Scott

8.15.2 Macro Definition Documentation

8.15.2.1 ECDH_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

8.15.2.2 ECDH_INVALID

```
#define ECDH_INVALID -4
```

ECDH Internal Error

8.15.2.3 ECDH_INVALID_PUBLIC_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

8.15.2.4 ECDH_OK

```
#define ECDH_OK 0
```

Function completed without error

8.15.2.5 EFS_BLS381

```
#define EFS_BLS381 MODBYTES_384_58
```

ECC Field Size in bytes

8.15.2.6 EGS_BLS381

```
#define EGS_BLS381 MODBYTES_384_58
```

ECC Group Size in bytes

8.15.3 Function Documentation

8.15.3.1 ECP_BLS381_ECIES_DECRYPT()

```
int ECP_BLS381_ECIES_DECRYPT (
            int h,
            octet * P1,
            octet * P2,
            octet * V,
            octet * C,
            octet * T,
            octet * U,
            octet * M )
```

IEEE-1363 ECIES Decryption

Parameters

h	is the hash type
P1	input Key Derivation parameters
P2	input Encoding parameters
V	component of the input ciphertext
C	the input ciphertext
T	the input HMAC tag, part of the ciphertext
U	the input private key for decryption
M	the output plaintext message

Returns

1 if successful, else 0

8.15.3.2 ECP_BLS381_ECIES_ENCRYPT()

```
void ECP_BLS381_ECIES_ENCRYPT (
            int h,
            octet * P1,
            octet * P2,
            csprng * R,
            octet * W,
            octet * M,
            int len,
            octet * V,
            octet * C,
            octet * T )
```

IEEE-1363 ECIES Encryption

Parameters

h	is the hash type
P1	input Key Derivation parameters
P2	input Encoding parameters
R	is a pointer to a cryptographically secure random number generator
W	the input public key of the recieving party
M	is the plaintext message to be encrypted
len	the length of the HMAC tag
V	component of the output ciphertext
C	the output ciphertext
T	the output HMAC tag, part of the ciphertext

8.15.3.3 ECP_BLS381_KEY_PAIR_GENERATE()

```
int ECP_BLS381_KEY_PAIR_GENERATE (
            csprng * R,
            octet * s,
            octet * W )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
s	the private key, an output internally randomly generated if R!=NULL, otherwise must be provided as an input
W	the output public key, which is s.G, where G is a fixed generator

Returns

0 or an error code

8.15.3.4 ECP_BLS381_PUBLIC_KEY_VALIDATE()

```
int ECP_BLS381_PUBLIC_KEY_VALIDATE (
            octet * W )
```

Parameters

W	the input public key to be validated

Returns

0 if public key is OK, or an error code

8.15.3.5 ECP_BLS381_SP_DSA()

```
int ECP_BLS381_SP_DSA (
            int h,
            csprng * R,
            octet * k,
            octet * s,
            octet * M,
            octet * c,
            octet * d )
```

IEEE-1363 ECDSA Signature

Parameters

h	is the hash type
R	is a pointer to a cryptographically secure random number generator
k	Ephemeral key. This value is used when R=NULL
s	the input private signing key
M	the input message to be signed
c	component of the output signature
d	component of the output signature

8.15.3.6 ECP_BLS381_SVDP_DH()

```
int ECP_BLS381_SVDP_DH (
            octet * s,
```

```
            octet * W,
            octet * K )
```

IEEE-1363 Diffie-Hellman shared secret calculation

Parameters

s	is the input private key,
W	the input public key of the other party
K	the output shared key, in fact the x-coordinate of s.W

Returns

0 or an error code

8.15.3.7 ECP_BLS381_VP_DSA()

```
int ECP_BLS381_VP_DSA (
            int h,
            octet * W,
            octet * M,
            octet * c,
            octet * d )
```

IEEE-1363 ECDSA Signature Verification

Parameters

h	is the hash type
W	the input public key
M	the input message
c	component of the input signature
d	component of the input signature

Returns

0 or an error code

8.16 ecdh_ED25519.h File Reference

ECDH Header file for implementation of standard EC protocols.

```
#include "ecp_ED25519.h"
#include "ecdh_support.h"
```

Macros

- #define EGS_ED25519 MODBYTES_256_56
- #define EFS_ED25519 MODBYTES_256_56
- #define ECDH_OK 0
- #define ECDH_INVALID_PUBLIC_KEY -2
- #define ECDH_ERROR -3
- #define ECDH_INVALID -4

Functions

- int ECP_ED25519_KEY_PAIR_GENERATE (csprng ∗R, octet ∗s, octet ∗W)

  Generate an ECC public/private key pair.
- int ECP_ED25519_PUBLIC_KEY_VALIDATE (octet ∗W)

  Validate an ECC public key.
- int ECP_ED25519_SVDP_DH (octet ∗s, octet ∗W, octet ∗K)

  Generate Diffie-Hellman shared key.
- void ECP_ED25519_ECIES_ENCRYPT (int h, octet ∗P1, octet ∗P2, csprng ∗R, octet ∗W, octet ∗M, int len, octet ∗V, octet ∗C, octet ∗T)

  ECIES Encryption.
- int ECP_ED25519_ECIES_DECRYPT (int h, octet ∗P1, octet ∗P2, octet ∗V, octet ∗C, octet ∗T, octet ∗U, octet ∗M)

  ECIES Decryption.
- int ECP_ED25519_SP_DSA (int h, csprng ∗R, octet ∗k, octet ∗s, octet ∗M, octet ∗c, octet ∗d)

  ECDSA Signature.
- int ECP_ED25519_VP_DSA (int h, octet ∗W, octet ∗M, octet ∗c, octet ∗d)

  ECDSA Signature Verification.

8.16.1 Detailed Description

Author

Mike Scott

8.16.2 Macro Definition Documentation

8.16.2.1 ECDH_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

8.16.2.2 ECDH_INVALID

```
#define ECDH_INVALID -4
```

ECDH Internal Error

8.16.2.3 ECDH_INVALID_PUBLIC_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

8.16.2.4 ECDH_OK

```
#define ECDH_OK 0
```

Function completed without error

8.16.2.5 EFS_ED25519

```
#define EFS_ED25519 MODBYTES_256_56
```

ECC Field Size in bytes

8.16.2.6 EGS_ED25519

```
#define EGS_ED25519 MODBYTES_256_56
```

ECC Group Size in bytes

8.16.3 Function Documentation

8.16.3.1 ECP_ED25519_ECIES_DECRYPT()

```
int ECP_ED25519_ECIES_DECRYPT (
            int h,
            octet * P1,
            octet * P2,
            octet * V,
            octet * C,
            octet * T,
            octet * U,
            octet * M )
```

IEEE-1363 ECIES Decryption

Parameters

h	is the hash type
P1	input Key Derivation parameters
P2	input Encoding parameters
V	component of the input ciphertext
C	the input ciphertext
T	the input HMAC tag, part of the ciphertext
U	the input private key for decryption
M	the output plaintext message

Returns

1 if successful, else 0

8.16.3.2 ECP_ED25519_ECIES_ENCRYPT()

```
void ECP_ED25519_ECIES_ENCRYPT (
            int h,
            octet * P1,
            octet * P2,
            csprng * R,
            octet * W,
            octet * M,
            int len,
            octet * V,
            octet * C,
            octet * T )
```

IEEE-1363 ECIES Encryption

Parameters

h	is the hash type
P1	input Key Derivation parameters
P2	input Encoding parameters
R	is a pointer to a cryptographically secure random number generator
W	the input public key of the recieving party
M	is the plaintext message to be encrypted
len	the length of the HMAC tag
V	component of the output ciphertext
C	the output ciphertext
T	the output HMAC tag, part of the ciphertext

8.16.3.3 ECP_ED25519_KEY_PAIR_GENERATE()

```
int ECP_ED25519_KEY_PAIR_GENERATE (
            csprng * R,
            octet * s,
            octet * W )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
s	the private key, an output internally randomly generated if R!=NULL, otherwise must be provided as an input
W	the output public key, which is s.G, where G is a fixed generator

Returns

0 or an error code

8.16.3.4 ECP_ED25519_PUBLIC_KEY_VALIDATE()

```
int ECP_ED25519_PUBLIC_KEY_VALIDATE (
            octet * W )
```

Parameters

W	the input public key to be validated

Returns

0 if public key is OK, or an error code

8.16.3.5 ECP_ED25519_SP_DSA()

```
int ECP_ED25519_SP_DSA (
            int h,
            csprng * R,
            octet * k,
            octet * s,
            octet * M,
            octet * c,
            octet * d )
```

IEEE-1363 ECDSA Signature

Parameters

h	is the hash type
R	is a pointer to a cryptographically secure random number generator
k	Ephemeral key. This value is used when R=NULL
s	the input private signing key
M	the input message to be signed
c	component of the output signature
d	component of the output signature

8.16.3.6 ECP_ED25519_SVDP_DH()

```
int ECP_ED25519_SVDP_DH (
            octet * s,
```

```
            octet * W,
            octet * K )
```

IEEE-1363 Diffie-Hellman shared secret calculation

Parameters

s	is the input private key,
W	the input public key of the other party
K	the output shared key, in fact the x-coordinate of s.W

Returns

0 or an error code

8.16.3.7 ECP_ED25519_VP_DSA()

```
int ECP_ED25519_VP_DSA (
            int h,
            octet * W,
            octet * M,
            octet * c,
            octet * d )
```

IEEE-1363 ECDSA Signature Verification

Parameters

h	is the hash type
W	the input public key
M	the input message
c	component of the input signature
d	component of the input signature

Returns

0 or an error code

8.17 ecdh_GOLDILOCKS.h File Reference

ECDH Header file for implementation of standard EC protocols.

```
#include "ecp_GOLDILOCKS.h"
#include "ecdh_support.h"
```

Macros

- #define EGS_GOLDILOCKS MODBYTES_448_58
- #define EFS_GOLDILOCKS MODBYTES_448_58
- #define ECDH_OK 0
- #define ECDH_INVALID_PUBLIC_KEY -2
- #define ECDH_ERROR -3
- #define ECDH_INVALID -4

Functions

- int ECP_GOLDILOCKS_KEY_PAIR_GENERATE (csprng ∗R, octet ∗s, octet ∗W)

  Generate an ECC public/private key pair.
- int ECP_GOLDILOCKS_PUBLIC_KEY_VALIDATE (octet ∗W)

  Validate an ECC public key.
- int ECP_GOLDILOCKS_SVDP_DH (octet ∗s, octet ∗W, octet ∗K)

  Generate Diffie-Hellman shared key.
- void ECP_GOLDILOCKS_ECIES_ENCRYPT (int h, octet ∗P1, octet ∗P2, csprng ∗R, octet ∗W, octet ∗M, int len, octet ∗V, octet ∗C, octet ∗T)

  ECIES Encryption.
- int ECP_GOLDILOCKS_ECIES_DECRYPT (int h, octet ∗P1, octet ∗P2, octet ∗V, octet ∗C, octet ∗T, octet ∗U, octet ∗M)

  ECIES Decryption.
- int ECP_GOLDILOCKS_SP_DSA (int h, csprng ∗R, octet ∗k, octet ∗s, octet ∗M, octet ∗c, octet ∗d)

  ECDSA Signature.
- int ECP_GOLDILOCKS_VP_DSA (int h, octet ∗W, octet ∗M, octet ∗c, octet ∗d)

  ECDSA Signature Verification.

8.17.1 Detailed Description

Author

Mike Scott

8.17.2 Macro Definition Documentation

8.17.2.1 ECDH_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

8.17.2.2 ECDH_INVALID

```
#define ECDH_INVALID -4
```

ECDH Internal Error

8.17.2.3 ECDH_INVALID_PUBLIC_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

8.17.2.4 ECDH_OK

```
#define ECDH_OK 0
```

Function completed without error

8.17.2.5 EFS_GOLDILOCKS

```
#define EFS_GOLDILOCKS MODBYTES_448_58
```

ECC Field Size in bytes

8.17.2.6 EGS_GOLDILOCKS

```
#define EGS_GOLDILOCKS MODBYTES_448_58
```

ECC Group Size in bytes

8.17.3 Function Documentation

8.17.3.1 ECP_GOLDILOCKS_ECIES_DECRYPT()

```
int ECP_GOLDILOCKS_ECIES_DECRYPT (
            int h,
            octet * P1,
            octet * P2,
            octet * V,
            octet * C,
            octet * T,
            octet * U,
            octet * M )
```

IEEE-1363 ECIES Decryption

Parameters

h	is the hash type
P1	input Key Derivation parameters
P2	input Encoding parameters
V	component of the input ciphertext
C	the input ciphertext
T	the input HMAC tag, part of the ciphertext
U	the input private key for decryption
M	the output plaintext message

Returns

1 if successful, else 0

8.17.3.2 ECP_GOLDILOCKS_ECIES_ENCRYPT()

```
void ECP_GOLDILOCKS_ECIES_ENCRYPT (
            int h,
            octet * P1,
            octet * P2,
            csprng * R,
            octet * W,
            octet * M,
            int len,
            octet * V,
            octet * C,
            octet * T )
```

IEEE-1363 ECIES Encryption

Parameters

h	is the hash type
P1	input Key Derivation parameters
P2	input Encoding parameters
R	is a pointer to a cryptographically secure random number generator
W	the input public key of the recieving party
M	is the plaintext message to be encrypted
len	the length of the HMAC tag
V	component of the output ciphertext
C	the output ciphertext
T	the output HMAC tag, part of the ciphertext

8.17.3.3 ECP_GOLDILOCKS_KEY_PAIR_GENERATE()

```
int ECP_GOLDILOCKS_KEY_PAIR_GENERATE (
            csprng * R,
            octet * s,
            octet * W )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
s	the private key, an output internally randomly generated if R!=NULL, otherwise must be provided as an input
W	the output public key, which is s.G, where G is a fixed generator

Returns

0 or an error code

8.17.3.4 ECP_GOLDILOCKS_PUBLIC_KEY_VALIDATE()

```
int ECP_GOLDILOCKS_PUBLIC_KEY_VALIDATE (
            octet * W )
```

Parameters

W	the input public key to be validated

Returns

0 if public key is OK, or an error code

8.17.3.5 ECP_GOLDILOCKS_SP_DSA()

```
int ECP_GOLDILOCKS_SP_DSA (
            int h,
            csprng * R,
            octet * k,
            octet * s,
            octet * M,
            octet * c,
            octet * d )
```

IEEE-1363 ECDSA Signature

Parameters

h	is the hash type
R	is a pointer to a cryptographically secure random number generator
k	Ephemeral key. This value is used when R=NULL
s	the input private signing key
M	the input message to be signed
c	component of the output signature
d	component of the output signature

8.17.3.6 ECP_GOLDILOCKS_SVDP_DH()

```
int ECP_GOLDILOCKS_SVDP_DH (
            octet * s,
```

```
            octet * W,
            octet * K )
```

IEEE-1363 Diffie-Hellman shared secret calculation

Parameters

s	is the input private key,
W	the input public key of the other party
K	the output shared key, in fact the x-coordinate of s.W

Returns

0 or an error code

8.17.3.7 ECP_GOLDILOCKS_VP_DSA()

```
int ECP_GOLDILOCKS_VP_DSA (
            int h,
            octet * W,
            octet * M,
            octet * c,
            octet * d )
```

IEEE-1363 ECDSA Signature Verification

Parameters

h	is the hash type
W	the input public key
M	the input message
c	component of the input signature
d	component of the input signature

Returns

0 or an error code

8.18 ecdh_NIST256.h File Reference

ECDH Header file for implementation of standard EC protocols.

```
#include "ecp_NIST256.h"
#include "ecdh_support.h"
```

Macros

- #define EGS_NIST256 MODBYTES_256_56
- #define EFS_NIST256 MODBYTES_256_56
- #define ECDH_OK 0
- #define ECDH_INVALID_PUBLIC_KEY -2
- #define ECDH_ERROR -3
- #define ECDH_INVALID -4

Functions

- int ECP_NIST256_KEY_PAIR_GENERATE (csprng ∗R, octet ∗s, octet ∗W)

  Generate an ECC public/private key pair.
- int ECP_NIST256_PUBLIC_KEY_VALIDATE (octet ∗W)

  Validate an ECC public key.
- int ECP_NIST256_SVDP_DH (octet ∗s, octet ∗W, octet ∗K)

  Generate Diffie-Hellman shared key.
- void ECP_NIST256_ECIES_ENCRYPT (int h, octet ∗P1, octet ∗P2, csprng ∗R, octet ∗W, octet ∗M, int len, octet ∗V, octet ∗C, octet ∗T)

  ECIES Encryption.
- int ECP_NIST256_ECIES_DECRYPT (int h, octet ∗P1, octet ∗P2, octet ∗V, octet ∗C, octet ∗T, octet ∗U, octet ∗M)

  ECIES Decryption.
- int ECP_NIST256_SP_DSA (int h, csprng ∗R, octet ∗k, octet ∗s, octet ∗M, octet ∗c, octet ∗d)

  ECDSA Signature.
- int ECP_NIST256_VP_DSA (int h, octet ∗W, octet ∗M, octet ∗c, octet ∗d)

  ECDSA Signature Verification.

8.18.1 Detailed Description

Author

Mike Scott

8.18.2 Macro Definition Documentation

8.18.2.1 ECDH_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

8.18.2.2 ECDH_INVALID

```
#define ECDH_INVALID -4
```

ECDH Internal Error

8.18.2.3 ECDH_INVALID_PUBLIC_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

8.18.2.4 ECDH_OK

```
#define ECDH_OK 0
```

Function completed without error

8.18.2.5 EFS_NIST256

```
#define EFS_NIST256 MODBYTES_256_56
```

ECC Field Size in bytes

8.18.2.6 EGS_NIST256

```
#define EGS_NIST256 MODBYTES_256_56
```

ECC Group Size in bytes

8.18.3 Function Documentation

8.18.3.1 ECP_NIST256_ECIES_DECRYPT()

```
int ECP_NIST256_ECIES_DECRYPT (
            int h,
            octet * P1,
            octet * P2,
            octet * V,
            octet * C,
            octet * T,
            octet * U,
            octet * M )
```

IEEE-1363 ECIES Decryption

Parameters

h	is the hash type
P1	input Key Derivation parameters
P2	input Encoding parameters
V	component of the input ciphertext
C	the input ciphertext
T	the input HMAC tag, part of the ciphertext
U	the input private key for decryption
M	the output plaintext message

Returns

1 if successful, else 0

8.18.3.2 ECP_NIST256_ECIES_ENCRYPT()

```
void ECP_NIST256_ECIES_ENCRYPT (
            int h,
            octet * P1,
            octet * P2,
            csprng * R,
            octet * W,
            octet * M,
            int len,
            octet * V,
            octet * C,
            octet * T )
```

IEEE-1363 ECIES Encryption

Parameters

h	is the hash type
P1	input Key Derivation parameters
P2	input Encoding parameters
R	is a pointer to a cryptographically secure random number generator
W	the input public key of the recieving party
M	is the plaintext message to be encrypted
len	the length of the HMAC tag
V	component of the output ciphertext
C	the output ciphertext
T	the output HMAC tag, part of the ciphertext

8.18.3.3 ECP_NIST256_KEY_PAIR_GENERATE()

```
int ECP_NIST256_KEY_PAIR_GENERATE (
            csprng * R,
            octet * s,
            octet * W )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
s	the private key, an output internally randomly generated if R!=NULL, otherwise must be provided as an input
W	the output public key, which is s.G, where G is a fixed generator

Returns

0 or an error code

8.18.3.4 ECP_NIST256_PUBLIC_KEY_VALIDATE()

```
int ECP_NIST256_PUBLIC_KEY_VALIDATE (
            octet * W )
```

Parameters

W	the input public key to be validated

Returns

0 if public key is OK, or an error code

8.18.3.5 ECP_NIST256_SP_DSA()

```
int ECP_NIST256_SP_DSA (
            int h,
            csprng * R,
            octet * k,
            octet * s,
            octet * M,
            octet * c,
            octet * d )
```

IEEE-1363 ECDSA Signature

Parameters

h	is the hash type
R	is a pointer to a cryptographically secure random number generator
k	Ephemeral key. This value is used when R=NULL
s	the input private signing key
M	the input message to be signed
c	component of the output signature
d	component of the output signature

8.18.3.6 ECP_NIST256_SVDP_DH()

```
int ECP_NIST256_SVDP_DH (
            octet * s,
```

```
            octet * W,
            octet * K )
```

IEEE-1363 Diffie-Hellman shared secret calculation

Parameters

s	is the input private key,
W	the input public key of the other party
K	the output shared key, in fact the x-coordinate of s.W

Returns

0 or an error code

8.18.3.7 ECP_NIST256_VP_DSA()

```
int ECP_NIST256_VP_DSA (
            int h,
            octet * W,
            octet * M,
            octet * c,
            octet * d )
```

IEEE-1363 ECDSA Signature Verification

Parameters

h	is the hash type
W	the input public key
M	the input message
c	component of the input signature
d	component of the input signature

Returns

0 or an error code

8.19 ecdh_support.h File Reference

ECDH Support Header File.

```
#include "amcl.h"
```

Functions

- void ehashit (int sha, octet ∗p, int n, octet ∗x, octet ∗w, int pad)

  general purpose hash function w=hash(p|n|x|y)
- void HASH (int h, octet ∗I, octet ∗O)

  hash an octet into another octet
- int HMAC (int h, octet ∗M, octet ∗K, int len, octet ∗tag)

  HMAC of message M using key K to create tag of length len in octet tag.
- void KDF2 (int h, octet ∗Z, octet ∗P, int len, octet ∗K)

  Key Derivation Function - generates key K from inputs Z and P.
- void PBKDF2 (int h, octet ∗P, octet ∗S, int rep, int len, octet ∗K)

  Password Based Key Derivation Function - generates key K from password, salt and repeat counter.
- void AES_CBC_IV0_ENCRYPT (octet ∗K, octet ∗P, octet ∗C)

  AES encrypts a plaintext to a ciphtertext.
- int AES_CBC_IV0_DECRYPT (octet ∗K, octet ∗C, octet ∗P)

  AES encrypts a plaintext to a ciphtertext.

8.19.1 Detailed Description

Author

Mike Scott

8.19.2 Function Documentation

8.19.2.1 AES_CBC_IV0_DECRYPT()

```
int AES_CBC_IV0_DECRYPT (
            octet * K,
            octet * C,
            octet * P )
```

IEEE-1363 AES_CBC_IV0_DECRYPT function. Decrypts in CBC mode with a zero IV.

Parameters

K	AES key
C	input ciphertext octet
P	output plaintext octet

Returns

0 if bad input, else 1

8.19.2.2 AES_CBC_IV0_ENCRYPT()

```
void AES_CBC_IV0_ENCRYPT (
            octet * K,
            octet * P,
            octet * C )
```

IEEE-1363 AES_CBC_IV0_ENCRYPT function. Encrypts in CBC mode with a zero IV, padding as necessary to create a full final block.

Parameters

K	AES key
P	input plaintext octet
C	output ciphertext octet

8.19.2.3 ehashit()

```
void ehashit (
            int sha,
            octet * p,
            int n,
            octet * x,
            octet * w,
            int pad )
```

Parameters

sha	is the hash type
p	first octect involved in the hash
n	integer involved in the hash
x	second octect involved in the h ash
w	output
pad	padding

8.19.2.4 HASH()

```
void HASH (
            int h,
            octet * I,
            octet * O )
```

Parameters

h	is the hash type
I	input octet
O	output octet - H(I)

8.19.2.5 HMAC()

```
int HMAC (
            int h,
            octet * M,
            octet * K,
            int len,
            octet * tag )
```

IEEE-1363 MAC1 function. Uses SHA256 internally.

Parameters

h	is the hash type
M	input message octet
K	input encryption key
len	is output desired length of HMAC tag
tag	is the output HMAC

Returns

0 for bad parameters, else 1

8.19.2.6 KDF2()

```
void KDF2 (
            int h,
            octet * Z,
            octet * P,
            int len,
            octet * K )
```

IEEE-1363 KDF2 Key Derivation Function. Uses SHA256 internally.

Parameters

h	is the hash type
Z	input octet
P	input key derivation parameters - can be NULL
len	is output desired length of key
K	is the derived key

8.19.2.7 PBKDF2()

```
void PBKDF2 (
            int h,
            octet * P,
            octet * S,
            int rep,
            int len,
            octet * K )
```

PBKDF2 Password Based Key Derivation Function. Uses SHA256 internally.

Parameters

h	is the hash type
P	input password
S	input salt
rep	Number of times to be iterated.
len	is output desired length
K	is the derived key

8.20 ecp2_BLS381.h File Reference

ECP2 Header File.

```
#include "fp2_BLS381.h"
#include "config_curve_BLS381.h"
```

Data Structures

- struct ECP2_BLS381

    ECP2 Structure - Elliptic Curve Point over quadratic extension field.

Functions

- int ECP2_BLS381_isinf (ECP2_BLS381 *P)

    Tests for ECP2 point equal to infinity.
- void ECP2_BLS381_copy (ECP2_BLS381 *P, ECP2_BLS381 *Q)

    Copy ECP2 point to another ECP2 point.
- void ECP2_BLS381_inf (ECP2_BLS381 *P)

    Set ECP2 to point-at-infinity.
- int ECP2_BLS381_equals (ECP2_BLS381 *P, ECP2_BLS381 *Q)

    Tests for equality of two ECP2s.
- void ECP2_BLS381_affine (ECP2_BLS381 *P)

    Converts an ECP2 point from Projective (x,y,z) coordinates to affine (x,y) coordinates.
- int ECP2_BLS381_get (FP2_BLS381 *x, FP2_BLS381 *y, ECP2_BLS381 *P)

    Extract x and y coordinates of an ECP2 point P.

- void ECP2_BLS381_output (ECP2_BLS381 *P)

    Formats and outputs an ECP2 point to the console, converted to affine coordinates.
- void ECP2_BLS381_outputxyz (ECP2_BLS381 *P)

    Formats and outputs an ECP2 point to the console, in projective coordinates.
- void ECP2_BLS381_toOctet (octet *S, ECP2_BLS381 *P)

    Formats and outputs an ECP2 point to an octet string.
- int ECP2_BLS381_fromOctet (ECP2_BLS381 *P, octet *S)

    Creates an ECP2 point from an octet string.
- void ECP2_BLS381_rhs (FP2_BLS381 *r, FP2_BLS381 *x)

    Calculate Right Hand Side of curve equation y^2=f(x)
- int ECP2_BLS381_set (ECP2_BLS381 *P, FP2_BLS381 *x, FP2_BLS381 *y)

    Set ECP2 to point(x,y) given x and y.
- int ECP2_BLS381_setx (ECP2_BLS381 *P, FP2_BLS381 *x)

    Set ECP to point(x,[y]) given x.
- void ECP2_BLS381_neg (ECP2_BLS381 *P)

    Negation of an ECP2 point.
- int ECP2_BLS381_dbl (ECP2_BLS381 *P)

    Doubles an ECP2 instance P.
- int ECP2_BLS381_add (ECP2_BLS381 *P, ECP2_BLS381 *Q)

    Adds ECP2 instance Q to ECP2 instance P.
- void ECP2_BLS381_sub (ECP2_BLS381 *P, ECP2_BLS381 *Q)

    Subtracts ECP instance Q from ECP2 instance P.
- void ECP2_BLS381_mul (ECP2_BLS381 *P, BIG_384_58 b)

    Multiplies an ECP2 instance P by a BIG, side-channel resistant.
- void ECP2_BLS381_frob (ECP2_BLS381 *P, FP2_BLS381 *f)

    Multiplies an ECP2 instance P by the internal modulus p, using precalculated Frobenius constant f.
- void ECP2_BLS381_mul4 (ECP2_BLS381 *P, ECP2_BLS381 *Q, BIG_384_58 *b)

    *Calculates P=b[0]*Q[0]+b[1]*Q[1]+b[2]*Q[2]+b[3]*Q[3].*
- void ECP2_BLS381_mapit (ECP2_BLS381 *P, octet *w)

    Maps random BIG to curve point of correct order.
- void ECP2_BLS381_generator (ECP2_BLS381 *G)

    Get Group Generator from ROM.

Variables

- const int CURVE_A_BLS381
- const int CURVE_B_I_BLS381
- const BIG_384_58 CURVE_B_BLS381
- const BIG_384_58 CURVE_Order_BLS381
- const BIG_384_58 CURVE_Cof_BLS381
- const BIG_384_58 CURVE_Bnx_BLS381
- const BIG_384_58 Fra_BLS381
- const BIG_384_58 Frb_BLS381
- const BIG_384_58 CURVE_Gx_BLS381
- const BIG_384_58 CURVE_Gy_BLS381
- const BIG_384_58 CURVE_Pxa_BLS381
- const BIG_384_58 CURVE_Pxb_BLS381
- const BIG_384_58 CURVE_Pya_BLS381
- const BIG_384_58 CURVE_Pyb_BLS381

8.20.1 Detailed Description

Author

Mike Scott

8.20.2 Function Documentation

8.20.2.1 ECP2_BLS381_add()

```
int ECP2_BLS381_add (
            ECP2_BLS381 * P,
            ECP2_BLS381 * Q )
```

Parameters

P	ECP2 instance, on exit =P+Q
Q	ECP2 instance to be added to P

8.20.2.2 ECP2_BLS381_affine()

```
void ECP2_BLS381_affine (
            ECP2_BLS381 * P )
```

Parameters

P	ECP2 instance to be converted to affine form

8.20.2.3 ECP2_BLS381_copy()

```
void ECP2_BLS381_copy (
            ECP2_BLS381 * P,
            ECP2_BLS381 * Q )
```

Parameters

P	ECP2 instance, on exit = Q
Q	ECP2 instance to be copied

8.20.2.4 ECP2_BLS381_dbl()

```
int ECP2_BLS381_dbl (
            ECP2_BLS381 * P )
```

Parameters

P	ECP2 instance, on exit =2∗P

8.20.2.5 ECP2_BLS381_equals()

```
int ECP2_BLS381_equals (
            ECP2_BLS381 * P,
            ECP2_BLS381 * Q )
```

Parameters

P	ECP2 instance to be compared
Q	ECP2 instance to be compared

Returns

1 if P=Q, else returns 0

8.20.2.6 ECP2_BLS381_frob()

```
void ECP2_BLS381_frob (
            ECP2_BLS381 * P,
            FP2_BLS381 * f )
```

Fast point multiplication using Frobenius

Parameters

P	ECP2 instance, on exit = p∗P
f	FP2 precalculated Frobenius constant

8.20.2.7 ECP2_BLS381_fromOctet()

```
int ECP2_BLS381_fromOctet (
            ECP2_BLS381 * P,
            octet * S )
```

The octet string is in the form x|y The real and imaginary parts of the x and y coordinates are in big-endian base 256 form.

Parameters

P	ECP2 instance to be created from the octet string
S	input octet string return 1 if octet string corresponds to a point on the curve, else 0

8.20.2.8 ECP2_BLS381_generator()

```
void ECP2_BLS381_generator (
            ECP2_BLS381 * G )
```

Parameters

G	ECP2 instance

8.20.2.9 ECP2_BLS381_get()

```
int ECP2_BLS381_get (
            FP2_BLS381 * x,
            FP2_BLS381 * y,
            ECP2_BLS381 * P )
```

If x=y, returns only x

Parameters

x	FP2 on exit = x coordinate of point
y	FP2 on exit = y coordinate of point (unless x=y)
P	ECP2 instance (x,y)

Returns

-1 if P is point-at-infinity, else 0

8.20.2.10 ECP2_BLS381_inf()

```
void ECP2_BLS381_inf (
            ECP2_BLS381 * P )
```

Parameters

P	ECP2 instance to be set to infinity

8.20.2.11 ECP2_BLS381_isinf()

```
int ECP2_BLS381_isinf (
            ECP2_BLS381 * P )
```

Parameters

P	ECP2 point to be tested

Returns

1 if infinity, else returns 0

8.20.2.12 ECP2_BLS381_mapit()

```
void ECP2_BLS381_mapit (
            ECP2_BLS381 * P,
            octet * w )
```

Parameters

P	ECP2 instance of correct order
w	OCTET byte array to be mapped

8.20.2.13 ECP2_BLS381_mul()

```
void ECP2_BLS381_mul (
            ECP2_BLS381 * P,
            BIG_384_58 b )
```

Uses fixed sized windows.

Parameters

P	ECP2 instance, on exit =b∗P
b	BIG number multiplier

8.20.2.14 ECP2_BLS381_mul4()

```
void ECP2_BLS381_mul4 (
            ECP2_BLS381 * P,
            ECP2_BLS381 * Q,
            BIG_384_58 * b )
```

Parameters

P	ECP2 instance, on exit = b[0]∗Q[0]+b[1]∗Q[1]+b[2]∗Q[2]+b[3]∗Q[3]
Q	ECP2 array of 4 points
b	BIG array of 4 multipliers

8.20.2.15 ECP2_BLS381_neg()

```
void ECP2_BLS381_neg (
            ECP2_BLS381 * P )
```

Parameters

P	ECP2 instance, on exit = -P

8.20.2.16 ECP2_BLS381_output()

```
void ECP2_BLS381_output (
            ECP2_BLS381 * P )
```

Parameters

P	ECP2 instance to be printed

8.20.2.17 ECP2_BLS381_outputxyz()

```
void ECP2_BLS381_outputxyz (
            ECP2_BLS381 * P )
```

Parameters

P	ECP2 instance to be printed

8.20.2.18 ECP2_BLS381_rhs()

```
void ECP2_BLS381_rhs (
            FP2_BLS381 * r,
            FP2_BLS381 * x )
```

Function f(x)=x^3+Ax+B Used internally.

Parameters

r	FP2 value of f(x)
x	FP2 instance

8.20.2.19 ECP2_BLS381_set()

```
int ECP2_BLS381_set (
            ECP2_BLS381 * P,
            FP2_BLS381 * x,
            FP2_BLS381 * y )
```

Point P set to infinity if no such point on the curve.

Parameters

P	ECP2 instance to be set (x,y)
x	FP2 x coordinate of point
y	FP2 y coordinate of point

Returns

1 if point exists, else 0

8.20.2.20 ECP2_BLS381_setx()

```
int ECP2_BLS381_setx (
            ECP2_BLS381 * P,
            FP2_BLS381 * x )
```

Point P set to infinity if no such point on the curve. Otherwise y coordinate is calculated from x.

Parameters

P	ECP instance to be set (x,[y])
x	BIG x coordinate of point

Returns

1 if point exists, else 0

8.20.2.21 ECP2_BLS381_sub()

```
void ECP2_BLS381_sub (
            ECP2_BLS381 * P,
            ECP2_BLS381 * Q )
```

Parameters

P	ECP2 instance, on exit =P-Q
Q	ECP2 instance to be subtracted from P

8.20.2.22 ECP2_BLS381_toOctet()

```
void ECP2_BLS381_toOctet (
            octet * S,
            ECP2_BLS381 * P )
```

The octet string is created in the form x|y. Convert the real and imaginary parts of the x and y coordinates to big-endian base 256 form.

Parameters

S	output octet string
P	ECP2 instance to be converted to an octet string

8.20.3 Variable Documentation

8.20.3.1 CURVE_A_BLS381

```
const int CURVE_A_BLS381
```

Elliptic curve A parameter

8.20.3.2 CURVE_B_BLS381

```
const BIG_384_58 CURVE_B_BLS381
```

Elliptic curve B parameter

8.20.3.3 CURVE_B_I_BLS381

```
const int CURVE_B_I_BLS381
```

Elliptic curve B parameter

8.20.3.4 CURVE_Bnx_BLS381

```
const BIG_384_58 CURVE_Bnx_BLS381
```

Elliptic curve parameter

8.20.3.5 CURVE_Cof_BLS381

```
const BIG_384_58 CURVE_Cof_BLS381
```

Elliptic curve cofactor

8.20.3.6 CURVE_Gx_BLS381

```
const BIG_384_58 CURVE_Gx_BLS381
```

x-coordinate of generator point in group G1

8.20.3.7 CURVE_Gy_BLS381

```
const BIG_384_58 CURVE_Gy_BLS381
```

y-coordinate of generator point in group G1

8.20.3.8 CURVE_Order_BLS381

```
const BIG_384_58 CURVE_Order_BLS381
```

Elliptic curve group order

8.20.3.9 CURVE_Pxa_BLS381

```
const BIG_384_58 CURVE_Pxa_BLS381
```

real part of x-coordinate of generator point in group G2

8.20.3.10 CURVE_Pxb_BLS381

```
const BIG_384_58 CURVE_Pxb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

8.20.3.11 CURVE_Pya_BLS381

```
const BIG_384_58 CURVE_Pya_BLS381
```

real part of y-coordinate of generator point in group G2

8.20.3.12 CURVE_Pyb_BLS381

```
const BIG_384_58 CURVE_Pyb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

8.20.3.13 Fra_BLS381

```
const BIG_384_58 Fra_BLS381
```

real part of BN curve Frobenius Constant

8.20.3.14 Frb_BLS381

```
const BIG_384_58 Frb_BLS381
```

imaginary part of BN curve Frobenius Constant

8.21 ecp_BLS381.h File Reference

ECP Header File.

```
#include "fp_BLS381.h"
#include "config_curve_BLS381.h"
```

Data Structures

- struct ECP_BLS381

  ECP structure - Elliptic Curve Point over base field.

Functions

- int ECP_BLS381_isinf (ECP_BLS381 *P)

    Tests for ECP point equal to infinity.
- int ECP_BLS381_equals (ECP_BLS381 *P, ECP_BLS381 *Q)

    Tests for equality of two ECPs.
- void ECP_BLS381_copy (ECP_BLS381 *P, ECP_BLS381 *Q)

    Copy ECP point to another ECP point.
- void ECP_BLS381_neg (ECP_BLS381 *P)

    Negation of an ECP point.
- void ECP_BLS381_inf (ECP_BLS381 *P)

    Set ECP to point-at-infinity.
- void ECP_BLS381_rhs (FP_BLS381 *r, FP_BLS381 *x)

    Calculate Right Hand Side of curve equation y^2=f(x)
- int ECP_BLS381_set (ECP_BLS381 *P, BIG_384_58 x, BIG_384_58 y)

    Set ECP to point(x,y) given x and y.
- int ECP_BLS381_get (BIG_384_58 x, BIG_384_58 y, ECP_BLS381 *P)

    Extract x and y coordinates of an ECP point P.
- void ECP_BLS381_add (ECP_BLS381 *P, ECP_BLS381 *Q)

    Adds ECP instance Q to ECP instance P.
- void ECP_BLS381_sub (ECP_BLS381 *P, ECP_BLS381 *Q)

    Subtracts ECP instance Q from ECP instance P.
- int ECP_BLS381_setx (ECP_BLS381 *P, BIG_384_58 x, int s)

    Set ECP to point(x,y) given just x and sign of y.
- void ECP_BLS381_cfp (ECP_BLS381 *Q)

    Multiplies Point by curve co-factor.
- void ECP_BLS381_mapit (ECP_BLS381 *Q, octet *w)

    Maps random BIG to curve point of correct order.
- void ECP_BLS381_affine (ECP_BLS381 *P)

    Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.
- void ECP_BLS381_outputxyz (ECP_BLS381 *P)

    Formats and outputs an ECP point to the console, in projective coordinates.
- void ECP_BLS381_output (ECP_BLS381 *P)

    Formats and outputs an ECP point to the console, converted to affine coordinates.
- void ECP_BLS381_rawoutput (ECP_BLS381 *P)

    Formats and outputs an ECP point to the console.
- void ECP_BLS381_toOctet (octet *S, ECP_BLS381 *P, bool c)

    Formats and outputs an ECP point to an octet string The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If c is true, only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd.
- int ECP_BLS381_fromOctet (ECP_BLS381 *P, octet *S)

    Creates an ECP point from an octet string.
- void ECP_BLS381_dbl (ECP_BLS381 *P)

    Doubles an ECP instance P.
- void ECP_BLS381_pinmul (ECP_BLS381 *P, int i, int b)

    Multiplies an ECP instance P by a small integer, side-channel resistant.
- void ECP_BLS381_mul (ECP_BLS381 *P, BIG_384_58 b)

    Multiplies an ECP instance P by a BIG, side-channel resistant.
- void ECP_BLS381_mul2 (ECP_BLS381 *P, ECP_BLS381 *Q, BIG_384_58 e, BIG_384_58 f)

    *Calculates double multiplication P=e*P+f*Q, side-channel resistant.*
- void ECP_BLS381_generator (ECP_BLS381 *G)

    Get Group Generator from ROM.

Variables

- const int CURVE_A_BLS381
- const int CURVE_Cof_I_BLS381
- const int CURVE_B_I_BLS381
- const BIG_384_58 CURVE_B_BLS381
- const BIG_384_58 CURVE_Order_BLS381
- const BIG_384_58 CURVE_Cof_BLS381
- const BIG_384_58 CURVE_Gx_BLS381
- const BIG_384_58 CURVE_Gy_BLS381
- const BIG_384_58 CURVE_Pxa_BLS381
- const BIG_384_58 CURVE_Pxb_BLS381
- const BIG_384_58 CURVE_Pya_BLS381
- const BIG_384_58 CURVE_Pyb_BLS381
- const BIG_384_58 CURVE_Pxaa_BLS381
- const BIG_384_58 CURVE_Pxab_BLS381
- const BIG_384_58 CURVE_Pxba_BLS381
- const BIG_384_58 CURVE_Pxbb_BLS381
- const BIG_384_58 CURVE_Pyaa_BLS381
- const BIG_384_58 CURVE_Pyab_BLS381
- const BIG_384_58 CURVE_Pyba_BLS381
- const BIG_384_58 CURVE_Pybb_BLS381
- const BIG_384_58 CURVE_Pxaaa_BLS381
- const BIG_384_58 CURVE_Pxaab_BLS381
- const BIG_384_58 CURVE_Pxaba_BLS381
- const BIG_384_58 CURVE_Pxabb_BLS381
- const BIG_384_58 CURVE_Pxbaa_BLS381
- const BIG_384_58 CURVE_Pxbab_BLS381
- const BIG_384_58 CURVE_Pxbba_BLS381
- const BIG_384_58 CURVE_Pxbbb_BLS381
- const BIG_384_58 CURVE_Pyaaa_BLS381
- const BIG_384_58 CURVE_Pyaab_BLS381
- const BIG_384_58 CURVE_Pyaba_BLS381
- const BIG_384_58 CURVE_Pyabb_BLS381
- const BIG_384_58 CURVE_Pybaa_BLS381
- const BIG_384_58 CURVE_Pybab_BLS381
- const BIG_384_58 CURVE_Pybba_BLS381
- const BIG_384_58 CURVE_Pybbb_BLS381
- const BIG_384_58 CURVE_Bnx_BLS381
- const BIG_384_58 CURVE_Cru_BLS381
- const BIG_384_58 Fra_BLS381
- const BIG_384_58 Frb_BLS381
- const BIG_384_58 CURVE_W_BLS381 [2]
- const BIG_384_58 CURVE_SB_BLS381 [2][2]
- const BIG_384_58 CURVE_WB_BLS381 [4]
- const BIG_384_58 CURVE_BB_BLS381 [4][4]

8.21.1 Detailed Description

Author

Mike Scott

8.21.2 Function Documentation

8.21.2.1 ECP_BLS381_add()

```
void ECP_BLS381_add (
            ECP_BLS381 * P,
            ECP_BLS381 * Q )
```

Parameters

P	ECP instance, on exit =P+Q
Q	ECP instance to be added to P

8.21.2.2 ECP_BLS381_affine()

```
void ECP_BLS381_affine (
            ECP_BLS381 * P )
```

Parameters

P	ECP instance to be converted to affine form

8.21.2.3 ECP_BLS381_cfp()

```
void ECP_BLS381_cfp (
            ECP_BLS381 * Q )
```

Parameters

Q	ECP instance

8.21.2.4 ECP_BLS381_copy()

```
void ECP_BLS381_copy (
            ECP_BLS381 * P,
            ECP_BLS381 * Q )
```

Parameters

P	ECP instance, on exit = Q
Q	ECP instance to be copied

8.21.2.5 ECP_BLS381_dbl()

```
void ECP_BLS381_dbl (
            ECP_BLS381 * P )
```

Parameters

P	ECP instance, on exit =2∗P

8.21.2.6 ECP_BLS381_equals()

```
int ECP_BLS381_equals (
            ECP_BLS381 * P,
            ECP_BLS381 * Q )
```

Parameters

P	ECP instance to be compared
Q	ECP instance to be compared

Returns

1 if P=Q, else returns 0

8.21.2.7 ECP_BLS381_fromOctet()

```
int ECP_BLS381_fromOctet (
            ECP_BLS381 * P,
            octet * S )
```

The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If in compressed form only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd

Parameters

P	ECP instance to be created from the octet string
S	input octet string return 1 if octet string corresponds to a point on the curve, else 0

8.21.2.8 ECP_BLS381_generator()

```
void ECP_BLS381_generator (
            ECP_BLS381 * G )
```

Parameters

G	ECP instance

8.21.2.9 ECP_BLS381_get()

```
int ECP_BLS381_get (
            BIG_384_58 x,
            BIG_384_58 y,
            ECP_BLS381 * P )
```

If x=y, returns only x

Parameters

x	BIG on exit = x coordinate of point
y	BIG on exit = y coordinate of point (unless x=y)
P	ECP instance (x,y)

Returns

sign of y, or -1 if P is point-at-infinity

8.21.2.10 ECP_BLS381_inf()

```
void ECP_BLS381_inf (
            ECP_BLS381 * P )
```

Parameters

P	ECP instance to be set to infinity

8.21.2.11 ECP_BLS381_isinf()

```
int ECP_BLS381_isinf (
            ECP_BLS381 * P )
```

Parameters

P	ECP point to be tested

Returns

1 if infinity, else returns 0

8.21.2.12 ECP_BLS381_mapit()

```
void ECP_BLS381_mapit (
            ECP_BLS381 * Q,
            octet * w )
```

Parameters

Q	ECP instance of correct order
w	OCTET byte array to be mapped

8.21.2.13 ECP_BLS381_mul()

```
void ECP_BLS381_mul (
            ECP_BLS381 * P,
            BIG_384_58 b )
```

Uses Montgomery ladder for Montgomery curves, otherwise fixed sized windows.

Parameters

P	ECP instance, on exit =b*P
b	BIG number multiplier

8.21.2.14 ECP_BLS381_mul2()

```
void ECP_BLS381_mul2 (
            ECP_BLS381 * P,
```

```
ECP_BLS381 * Q,
BIG_384_58 e,
BIG_384_58 f )
```

Parameters

P	ECP instance, on exit =e∗P+f∗Q
Q	ECP instance
e	BIG number multiplier
f	BIG number multiplier

8.21.2.15 ECP_BLS381_neg()

```
void ECP_BLS381_neg (
    ECP_BLS381 * P )
```

Parameters

P	ECP instance, on exit = -P

8.21.2.16 ECP_BLS381_output()

```
void ECP_BLS381_output (
    ECP_BLS381 * P )
```

Parameters

P	ECP instance to be printed

8.21.2.17 ECP_BLS381_outputxyz()

```
void ECP_BLS381_outputxyz (
    ECP_BLS381 * P )
```

Parameters

P	ECP instance to be printed

8.21.2.18 ECP_BLS381_pinmul()

```
void ECP_BLS381_pinmul (
            ECP_BLS381 * P,
            int i,
            int b )
```

Parameters

P	ECP instance, on exit =i*P
i	small integer multiplier
b	maximum number of bits in multiplier

8.21.2.19 ECP_BLS381_rawoutput()

```
void ECP_BLS381_rawoutput (
            ECP_BLS381 * P )
```

Parameters

P	ECP instance to be printed

8.21.2.20 ECP_BLS381_rhs()

```
void ECP_BLS381_rhs (
            FP_BLS381 * r,
            FP_BLS381 * x )
```

Function f(x) depends on form of elliptic curve, Weierstrass, Edwards or Montgomery. Used internally.

Parameters

r	BIG n-residue value of f(x)
x	BIG n-residue x

8.21.2.21 ECP_BLS381_set()

```
int ECP_BLS381_set (
            ECP_BLS381 * P,
            BIG_384_58 x,
            BIG_384_58 y )
```

Point P set to infinity if no such point on the curve.

Parameters

P	ECP instance to be set (x,y)
x	BIG x coordinate of point
y	BIG y coordinate of point

Returns

1 if point exists, else 0

8.21.2.22 ECP_BLS381_setx()

```
int ECP_BLS381_setx (
            ECP_BLS381 * P,
            BIG_384_58 x,
            int s )
```

Point P set to infinity if no such point on the curve. If x is on the curve then y is calculated from the curve equation. The correct y value (plus or minus) is selected given its sign s.

Parameters

P	ECP instance to be set (x,[y])
x	BIG x coordinate of point
s	an integer representing the "sign" of y, in fact its least significant bit.

8.21.2.23 ECP_BLS381_sub()

```
void ECP_BLS381_sub (
            ECP_BLS381 * P,
            ECP_BLS381 * Q )
```

Parameters

P	ECP instance, on exit =P-Q
Q	ECP instance to be subtracted from P

8.21.2.24 ECP_BLS381_toOctet()

```
void ECP_BLS381_toOctet (
            octet * S,
```

```
ECP_BLS381 * P,
bool c )
```

Parameters

c	compression required, true or false
S	output octet string
P	ECP instance to be converted to an octet string

8.21.3 Variable Documentation

8.21.3.1 CURVE_A_BLS381

```
const int CURVE_A_BLS381
```

Elliptic curve A parameter

8.21.3.2 CURVE_B_BLS381

```
const BIG_384_58 CURVE_B_BLS381
```

Elliptic curve B parameter

8.21.3.3 CURVE_B_I_BLS381

```
const int CURVE_B_I_BLS381
```

Elliptic curve B_i parameter

8.21.3.4 CURVE_BB_BLS381

```
const BIG_384_58 CURVE_BB_BLS381[4][4]
```

BN curve constant for GS decomposition

8.21.3.5 CURVE_Bnx_BLS381

```
const BIG_384_58 CURVE_Bnx_BLS381
```

BN curve x parameter

8.21.3.6 CURVE_Cof_BLS381

```
const BIG_384_58 CURVE_Cof_BLS381
```

Elliptic curve cofactor

8.21.3.7 CURVE_Cof_I_BLS381

```
const int CURVE_Cof_I_BLS381
```

Elliptic curve cofactor

8.21.3.8 CURVE_Cru_BLS381

```
const BIG_384_58 CURVE_Cru_BLS381
```

BN curve Cube Root of Unity

8.21.3.9 CURVE_Gx_BLS381

```
const BIG_384_58 CURVE_Gx_BLS381
```

x-coordinate of generator point in group G1

8.21.3.10 CURVE_Gy_BLS381

```
const BIG_384_58 CURVE_Gy_BLS381
```

y-coordinate of generator point in group G1

8.21.3.11 CURVE_Order_BLS381

```
const BIG_384_58 CURVE_Order_BLS381
```

Elliptic curve group order

8.21.3.12 CURVE_Pxa_BLS381

```
const BIG_384_58 CURVE_Pxa_BLS381
```

real part of x-coordinate of generator point in group G2

8.21.3.13 CURVE_Pxaa_BLS381

```
const BIG_384_58 CURVE_Pxaa_BLS381
```

real part of x-coordinate of generator point in group G2

8.21.3.14 CURVE_Pxaaa_BLS381

```
const BIG_384_58 CURVE_Pxaaa_BLS381
```

real part of x-coordinate of generator point in group G2

8.21.3.15 CURVE_Pxaab_BLS381

```
const BIG_384_58 CURVE_Pxaab_BLS381
```

imaginary part of x-coordinate of generator point in group G2

8.21.3.16 CURVE_Pxab_BLS381

```
const BIG_384_58 CURVE_Pxab_BLS381
```

imaginary part of x-coordinate of generator point in group G2

8.21.3.17 CURVE_Pxaba_BLS381

```
const BIG_384_58 CURVE_Pxaba_BLS381
```

real part of x-coordinate of generator point in group G2

8.21.3.18 CURVE_Pxabb_BLS381

```
const BIG_384_58 CURVE_Pxabb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

8.21.3.19 CURVE_Pxb_BLS381

```
const BIG_384_58 CURVE_Pxb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

8.21.3.20 CURVE_Pxba_BLS381

```
const BIG_384_58 CURVE_Pxba_BLS381
```

real part of x-coordinate of generator point in group G2

8.21.3.21 CURVE_Pxbaa_BLS381

```
const BIG_384_58 CURVE_Pxbaa_BLS381
```

real part of x-coordinate of generator point in group G2

8.21.3.22 CURVE_Pxbab_BLS381

```
const BIG_384_58 CURVE_Pxbab_BLS381
```

imaginary part of x-coordinate of generator point in group G2

8.21.3.23 CURVE_Pxbb_BLS381

```
const BIG_384_58 CURVE_Pxbb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

8.21.3.24 CURVE_Pxbba_BLS381

```
const BIG_384_58 CURVE_Pxbba_BLS381
```

real part of x-coordinate of generator point in group G2

8.21.3.25 CURVE_Pxbbb_BLS381

```
const BIG_384_58 CURVE_Pxbbb_BLS381
```

imaginary part of x-coordinate of generator point in group G2

8.21.3.26 CURVE_Pya_BLS381

```
const BIG_384_58 CURVE_Pya_BLS381
```

real part of y-coordinate of generator point in group G2

8.21.3.27 CURVE_Pyaa_BLS381

```
const BIG_384_58 CURVE_Pyaa_BLS381
```

real part of y-coordinate of generator point in group G2

8.21.3.28 CURVE_Pyaaa_BLS381

```
const BIG_384_58 CURVE_Pyaaa_BLS381
```

real part of y-coordinate of generator point in group G2

8.21.3.29 CURVE_Pyaab_BLS381

```
const BIG_384_58 CURVE_Pyaab_BLS381
```

imaginary part of y-coordinate of generator point in group G2

8.21.3.30 CURVE_Pyab_BLS381

```
const BIG_384_58 CURVE_Pyab_BLS381
```

imaginary part of y-coordinate of generator point in group G2

8.21.3.31 CURVE_Pyaba_BLS381

```
const BIG_384_58 CURVE_Pyaba_BLS381
```

real part of y-coordinate of generator point in group G2

8.21.3.32 CURVE_Pyabb_BLS381

```
const BIG_384_58 CURVE_Pyabb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

8.21.3.33 CURVE_Pyb_BLS381

```
const BIG_384_58 CURVE_Pyb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

8.21.3.34 CURVE_Pyba_BLS381

```
const BIG_384_58 CURVE_Pyba_BLS381
```

real part of y-coordinate of generator point in group G2

8.21.3.35 CURVE_Pybaa_BLS381

```
const BIG_384_58 CURVE_Pybaa_BLS381
```

real part of y-coordinate of generator point in group G2

8.21.3.36 CURVE_Pybab_BLS381

```
const BIG_384_58 CURVE_Pybab_BLS381
```

imaginary part of y-coordinate of generator point in group G2

8.21.3.37 CURVE_Pybb_BLS381

```
const BIG_384_58 CURVE_Pybb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

8.21.3.38 CURVE_Pybba_BLS381

```
const BIG_384_58 CURVE_Pybba_BLS381
```

real part of y-coordinate of generator point in group G2

8.21.3.39 CURVE_Pybbb_BLS381

```
const BIG_384_58 CURVE_Pybbb_BLS381
```

imaginary part of y-coordinate of generator point in group G2

8.21.3.40 CURVE_SB_BLS381

```
const BIG_384_58 CURVE_SB_BLS381[2][2]
```

BN curve constant for GLV decomposition

8.21.3.41 CURVE_W_BLS381

```
const BIG_384_58 CURVE_W_BLS381[2]
```

BN curve constant for GLV decomposition

8.21.3.42 CURVE_WB_BLS381

```
const BIG_384_58 CURVE_WB_BLS381[4]
```

BN curve constant for GS decomposition

8.21.3.43 Fra_BLS381

```
const BIG_384_58 Fra_BLS381
```

real part of BN curve Frobenius Constant

8.21.3.44 Frb_BLS381

```
const BIG_384_58 Frb_BLS381
```

imaginary part of BN curve Frobenius Constant

8.22 ecp_ED25519.h File Reference

ECP Header File.

```
#include "fp_25519.h"
#include "config_curve_ED25519.h"
```

Data Structures

- struct ECP_ED25519

  ECP structure - Elliptic Curve Point over base field.

Functions

- int ECP_ED25519_isinf (ECP_ED25519 ∗P)

  Tests for ECP point equal to infinity.
- int ECP_ED25519_equals (ECP_ED25519 ∗P, ECP_ED25519 ∗Q)

  Tests for equality of two ECPs.
- void ECP_ED25519_copy (ECP_ED25519 ∗P, ECP_ED25519 ∗Q)

  Copy ECP point to another ECP point.
- void ECP_ED25519_neg (ECP_ED25519 ∗P)

  Negation of an ECP point.
- void ECP_ED25519_inf (ECP_ED25519 ∗P)

  Set ECP to point-at-infinity.
- void ECP_ED25519_rhs (FP_25519 ∗r, FP_25519 ∗x)

  Calculate Right Hand Side of curve equation y^2=f(x)
- int ECP_ED25519_set (ECP_ED25519 ∗P, BIG_256_56 x, BIG_256_56 y)

  Set ECP to point(x,y) given x and y.
- int ECP_ED25519_get (BIG_256_56 x, BIG_256_56 y, ECP_ED25519 ∗P)

  Extract x and y coordinates of an ECP point P.
- void ECP_ED25519_add (ECP_ED25519 ∗P, ECP_ED25519 ∗Q)

  Adds ECP instance Q to ECP instance P.
- void ECP_ED25519_sub (ECP_ED25519 ∗P, ECP_ED25519 ∗Q)

  Subtracts ECP instance Q from ECP instance P.
- int ECP_ED25519_setx (ECP_ED25519 ∗P, BIG_256_56 x, int s)

  Set ECP to point(x,y) given just x and sign of y.
- void ECP_ED25519_cfp (ECP_ED25519 ∗Q)

  Multiplies Point by curve co-factor.
- void ECP_ED25519_mapit (ECP_ED25519 ∗Q, octet ∗w)

  Maps random BIG to curve point of correct order.
- void ECP_ED25519_affine (ECP_ED25519 ∗P)

  Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.
- void ECP_ED25519_outputxyz (ECP_ED25519 ∗P)

  Formats and outputs an ECP point to the console, in projective coordinates.
- void ECP_ED25519_output (ECP_ED25519 ∗P)

  Formats and outputs an ECP point to the console, converted to affine coordinates.
- void ECP_ED25519_rawoutput (ECP_ED25519 ∗P)

  Formats and outputs an ECP point to the console.
- void ECP_ED25519_toOctet (octet ∗S, ECP_ED25519 ∗P, bool c)

  Formats and outputs an ECP point to an octet string The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If c is true, only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd.
- int ECP_ED25519_fromOctet (ECP_ED25519 ∗P, octet ∗S)

  Creates an ECP point from an octet string.
- void ECP_ED25519_dbl (ECP_ED25519 ∗P)

  Doubles an ECP instance P.
- void ECP_ED25519_pinmul (ECP_ED25519 ∗P, int i, int b)

  Multiplies an ECP instance P by a small integer, side-channel resistant.

- void ECP_ED25519_mul (ECP_ED25519 ∗P, BIG_256_56 b)

    Multiplies an ECP instance P by a BIG, side-channel resistant.
- void ECP_ED25519_mul2 (ECP_ED25519 ∗P, ECP_ED25519 ∗Q, BIG_256_56 e, BIG_256_56 f)

    Calculates double multiplication P=e∗P+f∗Q, side-channel resistant.
- void ECP_ED25519_generator (ECP_ED25519 ∗G)

    Get Group Generator from ROM.

Variables

- const int CURVE_A_ED25519
- const int CURVE_Cof_I_ED25519
- const int CURVE_B_I_ED25519
- const BIG_256_56 CURVE_B_ED25519
- const BIG_256_56 CURVE_Order_ED25519
- const BIG_256_56 CURVE_Cof_ED25519
- const BIG_256_56 CURVE_Gx_ED25519
- const BIG_256_56 CURVE_Gy_ED25519
- const BIG_256_56 CURVE_Pxa_ED25519
- const BIG_256_56 CURVE_Pxb_ED25519
- const BIG_256_56 CURVE_Pya_ED25519
- const BIG_256_56 CURVE_Pyb_ED25519
- const BIG_256_56 CURVE_Pxaa_ED25519
- const BIG_256_56 CURVE_Pxab_ED25519
- const BIG_256_56 CURVE_Pxba_ED25519
- const BIG_256_56 CURVE_Pxbb_ED25519
- const BIG_256_56 CURVE_Pyaa_ED25519
- const BIG_256_56 CURVE_Pyab_ED25519
- const BIG_256_56 CURVE_Pyba_ED25519
- const BIG_256_56 CURVE_Pybb_ED25519
- const BIG_256_56 CURVE_Pxaaa_ED25519
- const BIG_256_56 CURVE_Pxaab_ED25519
- const BIG_256_56 CURVE_Pxaba_ED25519
- const BIG_256_56 CURVE_Pxabb_ED25519
- const BIG_256_56 CURVE_Pxbaa_ED25519
- const BIG_256_56 CURVE_Pxbab_ED25519
- const BIG_256_56 CURVE_Pxbba_ED25519
- const BIG_256_56 CURVE_Pxbbb_ED25519
- const BIG_256_56 CURVE_Pyaaa_ED25519
- const BIG_256_56 CURVE_Pyaab_ED25519
- const BIG_256_56 CURVE_Pyaba_ED25519
- const BIG_256_56 CURVE_Pyabb_ED25519
- const BIG_256_56 CURVE_Pybaa_ED25519
- const BIG_256_56 CURVE_Pybab_ED25519
- const BIG_256_56 CURVE_Pybba_ED25519
- const BIG_256_56 CURVE_Pybbb_ED25519
- const BIG_256_56 CURVE_Bnx_ED25519
- const BIG_256_56 CURVE_Cru_ED25519
- const BIG_256_56 Fra_25519
- const BIG_256_56 Frb_25519
- const BIG_256_56 CURVE_W_ED25519 [2]
- const BIG_256_56 CURVE_SB_ED25519 [2][2]
- const BIG_256_56 CURVE_WB_ED25519 [4]
- const BIG_256_56 CURVE_BB_ED25519 [4][4]

8.22.1 Detailed Description

Author

Mike Scott

8.22.2 Function Documentation

8.22.2.1 ECP_ED25519_add()

```
void ECP_ED25519_add (
            ECP_ED25519 * P,
            ECP_ED25519 * Q )
```

Parameters

P	ECP instance, on exit =P+Q
Q	ECP instance to be added to P

8.22.2.2 ECP_ED25519_affine()

```
void ECP_ED25519_affine (
            ECP_ED25519 * P )
```

Parameters

P	ECP instance to be converted to affine form

8.22.2.3 ECP_ED25519_cfp()

```
void ECP_ED25519_cfp (
            ECP_ED25519 * Q )
```

Parameters

Q	ECP instance

8.22.2.4 ECP_ED25519_copy()

```
void ECP_ED25519_copy (
            ECP_ED25519 * P,
            ECP_ED25519 * Q )
```

Parameters

P	ECP instance, on exit = Q
Q	ECP instance to be copied

8.22.2.5 ECP_ED25519_dbl()

```
void ECP_ED25519_dbl (
            ECP_ED25519 * P )
```

Parameters

P	ECP instance, on exit =2*P

8.22.2.6 ECP_ED25519_equals()

```
int ECP_ED25519_equals (
            ECP_ED25519 * P,
            ECP_ED25519 * Q )
```

Parameters

P	ECP instance to be compared
Q	ECP instance to be compared

Returns

1 if P=Q, else returns 0

8.22.2.7 ECP_ED25519_fromOctet()

```
int ECP_ED25519_fromOctet (
            ECP_ED25519 * P,
            octet * S )
```

The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If in compressed form only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd

Parameters

P	ECP instance to be created from the octet string
S	input octet string return 1 if octet string corresponds to a point on the curve, else 0

8.22.2.8 ECP_ED25519_generator()

```
void ECP_ED25519_generator (
            ECP_ED25519 * G )
```

Parameters

G	ECP instance

8.22.2.9 ECP_ED25519_get()

```
int ECP_ED25519_get (
            BIG_256_56 x,
            BIG_256_56 y,
            ECP_ED25519 * P )
```

If x=y, returns only x

Parameters

x	BIG on exit = x coordinate of point
y	BIG on exit = y coordinate of point (unless x=y)
P	ECP instance (x,y)

Returns

sign of y, or -1 if P is point-at-infinity

8.22.2.10 ECP_ED25519_inf()

```
void ECP_ED25519_inf (
            ECP_ED25519 * P )
```

Parameters

P	ECP instance to be set to infinity

8.22.2.11 ECP_ED25519_isinf()

```
int ECP_ED25519_isinf (
            ECP_ED25519 * P )
```

Parameters

P	ECP point to be tested

Returns

1 if infinity, else returns 0

8.22.2.12 ECP_ED25519_mapit()

```
void ECP_ED25519_mapit (
            ECP_ED25519 * Q,
            octet * w )
```

Parameters

Q	ECP instance of correct order
w	OCTET byte array to be mapped

8.22.2.13 ECP_ED25519_mul()

```
void ECP_ED25519_mul (
            ECP_ED25519 * P,
            BIG_256_56 b )
```

Uses Montgomery ladder for Montgomery curves, otherwise fixed sized windows.

Parameters

P	ECP instance, on exit =b∗P
b	BIG number multiplier

8.22.2.14 ECP_ED25519_mul2()

```
void ECP_ED25519_mul2 (
            ECP_ED25519 * P,
            ECP_ED25519 * Q,
            BIG_256_56 e,
            BIG_256_56 f )
```

Parameters

P	ECP instance, on exit =e∗P+f∗Q
Q	ECP instance
e	BIG number multiplier
f	BIG number multiplier

8.22.2.15 ECP_ED25519_neg()

```
void ECP_ED25519_neg (
            ECP_ED25519 * P )
```

Parameters

P	ECP instance, on exit = -P

8.22.2.16 ECP_ED25519_output()

```
void ECP_ED25519_output (
            ECP_ED25519 * P )
```

Parameters

P	ECP instance to be printed

8.22.2.17 ECP_ED25519_outputxyz()

```
void ECP_ED25519_outputxyz (
            ECP_ED25519 * P )
```

Parameters

P	ECP instance to be printed

8.22.2.18 ECP_ED25519_pinmul()

```
void ECP_ED25519_pinmul (
            ECP_ED25519 * P,
            int i,
            int b )
```

Parameters

P	ECP instance, on exit =i∗P
i	small integer multiplier
b	maximum number of bits in multiplier

8.22.2.19 ECP_ED25519_rawoutput()

```
void ECP_ED25519_rawoutput (
            ECP_ED25519 * P )
```

Parameters

P	ECP instance to be printed

8.22.2.20 ECP_ED25519_rhs()

```
void ECP_ED25519_rhs (
            FP_25519 * r,
            FP_25519 * x )
```

Function f(x) depends on form of elliptic curve, Weierstrass, Edwards or Montgomery. Used internally.

Parameters

r	BIG n-residue value of f(x)
x	BIG n-residue x

8.22.2.21 ECP_ED25519_set()

```
int ECP_ED25519_set (
            ECP_ED25519 * P,
```

```
BIG_256_56 x,
BIG_256_56 y )
```

Point P set to infinity if no such point on the curve.

Parameters

P	ECP instance to be set (x,y)
x	BIG x coordinate of point
y	BIG y coordinate of point

Returns

1 if point exists, else 0

8.22.2.22 ECP_ED25519_setx()

```
int ECP_ED25519_setx (
            ECP_ED25519 * P,
            BIG_256_56 x,
            int s )
```

Point P set to infinity if no such point on the curve. If x is on the curve then y is calculated from the curve equation. The correct y value (plus or minus) is selected given its sign s.

Parameters

P	ECP instance to be set (x,[y])
x	BIG x coordinate of point
s	an integer representing the "sign" of y, in fact its least significant bit.

8.22.2.23 ECP_ED25519_sub()

```
void ECP_ED25519_sub (
            ECP_ED25519 * P,
            ECP_ED25519 * Q )
```

Parameters

P	ECP instance, on exit =P-Q
Q	ECP instance to be subtracted from P

8.22.2.24 ECP_ED25519_toOctet()

```
void ECP_ED25519_toOctet (
            octet * S,
            ECP_ED25519 * P,
            bool c )
```

Parameters

c	compression required, true or false
S	output octet string
P	ECP instance to be converted to an octet string

8.22.3 Variable Documentation

8.22.3.1 CURVE_A_ED25519

```
const int CURVE_A_ED25519
```

Elliptic curve A parameter

8.22.3.2 CURVE_B_ED25519

```
const BIG_256_56 CURVE_B_ED25519
```

Elliptic curve B parameter

8.22.3.3 CURVE_B_I_ED25519

```
const int CURVE_B_I_ED25519
```

Elliptic curve B_i parameter

8.22.3.4 CURVE_BB_ED25519

```
const BIG_256_56 CURVE_BB_ED25519[4][4]
```

BN curve constant for GS decomposition

8.22.3.5 CURVE_Bnx_ED25519

```
const BIG_256_56 CURVE_Bnx_ED25519
```

BN curve x parameter

8.22.3.6 CURVE_Cof_ED25519

`const BIG_256_56 CURVE_Cof_ED25519`

Elliptic curve cofactor

8.22.3.7 CURVE_Cof_I_ED25519

`const int CURVE_Cof_I_ED25519`

Elliptic curve cofactor

8.22.3.8 CURVE_Cru_ED25519

`const BIG_256_56 CURVE_Cru_ED25519`

BN curve Cube Root of Unity

8.22.3.9 CURVE_Gx_ED25519

`const BIG_256_56 CURVE_Gx_ED25519`

x-coordinate of generator point in group G1

8.22.3.10 CURVE_Gy_ED25519

`const BIG_256_56 CURVE_Gy_ED25519`

y-coordinate of generator point in group G1

8.22.3.11 CURVE_Order_ED25519

`const BIG_256_56 CURVE_Order_ED25519`

Elliptic curve group order

8.22.3.12 CURVE_Pxa_ED25519

`const BIG_256_56 CURVE_Pxa_ED25519`

real part of x-coordinate of generator point in group G2

8.22.3.13 CURVE_Pxaa_ED25519

`const BIG_256_56 CURVE_Pxaa_ED25519`

real part of x-coordinate of generator point in group G2

8.22.3.14 CURVE_Pxaaa_ED25519

```
const BIG_256_56 CURVE_Pxaaa_ED25519
```

real part of x-coordinate of generator point in group G2

8.22.3.15 CURVE_Pxaab_ED25519

```
const BIG_256_56 CURVE_Pxaab_ED25519
```

imaginary part of x-coordinate of generator point in group G2

8.22.3.16 CURVE_Pxab_ED25519

```
const BIG_256_56 CURVE_Pxab_ED25519
```

imaginary part of x-coordinate of generator point in group G2

8.22.3.17 CURVE_Pxaba_ED25519

```
const BIG_256_56 CURVE_Pxaba_ED25519
```

real part of x-coordinate of generator point in group G2

8.22.3.18 CURVE_Pxabb_ED25519

```
const BIG_256_56 CURVE_Pxabb_ED25519
```

imaginary part of x-coordinate of generator point in group G2

8.22.3.19 CURVE_Pxb_ED25519

```
const BIG_256_56 CURVE_Pxb_ED25519
```

imaginary part of x-coordinate of generator point in group G2

8.22.3.20 CURVE_Pxba_ED25519

```
const BIG_256_56 CURVE_Pxba_ED25519
```

real part of x-coordinate of generator point in group G2

8.22.3.21 CURVE_Pxbaa_ED25519

```
const BIG_256_56 CURVE_Pxbaa_ED25519
```

real part of x-coordinate of generator point in group G2

8.22.3.22 CURVE_Pxbab_ED25519

```
const BIG_256_56 CURVE_Pxbab_ED25519
```

imaginary part of x-coordinate of generator point in group G2

8.22.3.23 CURVE_Pxbb_ED25519

```
const BIG_256_56 CURVE_Pxbb_ED25519
```

imaginary part of x-coordinate of generator point in group G2

8.22.3.24 CURVE_Pxbba_ED25519

```
const BIG_256_56 CURVE_Pxbba_ED25519
```

real part of x-coordinate of generator point in group G2

8.22.3.25 CURVE_Pxbbb_ED25519

```
const BIG_256_56 CURVE_Pxbbb_ED25519
```

imaginary part of x-coordinate of generator point in group G2

8.22.3.26 CURVE_Pya_ED25519

```
const BIG_256_56 CURVE_Pya_ED25519
```

real part of y-coordinate of generator point in group G2

8.22.3.27 CURVE_Pyaa_ED25519

```
const BIG_256_56 CURVE_Pyaa_ED25519
```

real part of y-coordinate of generator point in group G2

8.22.3.28 CURVE_Pyaaa_ED25519

```
const BIG_256_56 CURVE_Pyaaa_ED25519
```

real part of y-coordinate of generator point in group G2

8.22.3.29 CURVE_Pyaab_ED25519

```
const BIG_256_56 CURVE_Pyaab_ED25519
```

imaginary part of y-coordinate of generator point in group G2

8.22.3.30 CURVE_Pyab_ED25519

```
const BIG_256_56 CURVE_Pyab_ED25519
```

imaginary part of y-coordinate of generator point in group G2

8.22.3.31 CURVE_Pyaba_ED25519

```
const BIG_256_56 CURVE_Pyaba_ED25519
```

real part of y-coordinate of generator point in group G2

8.22.3.32 CURVE_Pyabb_ED25519

```
const BIG_256_56 CURVE_Pyabb_ED25519
```

imaginary part of y-coordinate of generator point in group G2

8.22.3.33 CURVE_Pyb_ED25519

```
const BIG_256_56 CURVE_Pyb_ED25519
```

imaginary part of y-coordinate of generator point in group G2

8.22.3.34 CURVE_Pyba_ED25519

```
const BIG_256_56 CURVE_Pyba_ED25519
```

real part of y-coordinate of generator point in group G2

8.22.3.35 CURVE_Pybaa_ED25519

```
const BIG_256_56 CURVE_Pybaa_ED25519
```

real part of y-coordinate of generator point in group G2

8.22.3.36 CURVE_Pybab_ED25519

```
const BIG_256_56 CURVE_Pybab_ED25519
```

imaginary part of y-coordinate of generator point in group G2

8.22.3.37 CURVE_Pybb_ED25519

```
const BIG_256_56 CURVE_Pybb_ED25519
```

imaginary part of y-coordinate of generator point in group G2

8.22.3.38 CURVE_Pybba_ED25519

```
const BIG_256_56 CURVE_Pybba_ED25519
```

real part of y-coordinate of generator point in group G2

8.22.3.39 CURVE_Pybbb_ED25519

```
const BIG_256_56 CURVE_Pybbb_ED25519
```

imaginary part of y-coordinate of generator point in group G2

8.22.3.40 CURVE_SB_ED25519

```
const BIG_256_56 CURVE_SB_ED25519[2][2]
```

BN curve constant for GLV decomposition

8.22.3.41 CURVE_W_ED25519

```
const BIG_256_56 CURVE_W_ED25519[2]
```

BN curve constant for GLV decomposition

8.22.3.42 CURVE_WB_ED25519

```
const BIG_256_56 CURVE_WB_ED25519[4]
```

BN curve constant for GS decomposition

8.22.3.43 Fra_25519

```
const BIG_256_56 Fra_25519
```

real part of BN curve Frobenius Constant

8.22.3.44 Frb_25519

```
const BIG_256_56 Frb_25519
```

imaginary part of BN curve Frobenius Constant

8.23 ecp_GOLDILOCKS.h File Reference

ECP Header File.

```
#include "fp_GOLDILOCKS.h"
#include "config_curve_GOLDILOCKS.h"
```

Data Structures

- struct ECP_GOLDILOCKS

  ECP structure - Elliptic Curve Point over base field.

Functions

- int ECP_GOLDILOCKS_isinf (ECP_GOLDILOCKS ∗P)

  Tests for ECP point equal to infinity.
- int ECP_GOLDILOCKS_equals (ECP_GOLDILOCKS ∗P, ECP_GOLDILOCKS ∗Q)

  Tests for equality of two ECPs.
- void ECP_GOLDILOCKS_copy (ECP_GOLDILOCKS ∗P, ECP_GOLDILOCKS ∗Q)

  Copy ECP point to another ECP point.
- void ECP_GOLDILOCKS_neg (ECP_GOLDILOCKS ∗P)

  Negation of an ECP point.
- void ECP_GOLDILOCKS_inf (ECP_GOLDILOCKS ∗P)

  Set ECP to point-at-infinity.
- void ECP_GOLDILOCKS_rhs (FP_GOLDILOCKS ∗r, FP_GOLDILOCKS ∗x)

  Calculate Right Hand Side of curve equation y^2=f(x)
- int ECP_GOLDILOCKS_set (ECP_GOLDILOCKS ∗P, BIG_448_58 x, BIG_448_58 y)

  Set ECP to point(x,y) given x and y.
- int ECP_GOLDILOCKS_get (BIG_448_58 x, BIG_448_58 y, ECP_GOLDILOCKS ∗P)

  Extract x and y coordinates of an ECP point P.
- void ECP_GOLDILOCKS_add (ECP_GOLDILOCKS ∗P, ECP_GOLDILOCKS ∗Q)

  Adds ECP instance Q to ECP instance P.
- void ECP_GOLDILOCKS_sub (ECP_GOLDILOCKS ∗P, ECP_GOLDILOCKS ∗Q)

  Subtracts ECP instance Q from ECP instance P.
- int ECP_GOLDILOCKS_setx (ECP_GOLDILOCKS ∗P, BIG_448_58 x, int s)

  Set ECP to point(x,y) given just x and sign of y.
- void ECP_GOLDILOCKS_cfp (ECP_GOLDILOCKS ∗Q)

  Multiplies Point by curve co-factor.
- void ECP_GOLDILOCKS_mapit (ECP_GOLDILOCKS ∗Q, octet ∗w)

  Maps random BIG to curve point of correct order.
- void ECP_GOLDILOCKS_affine (ECP_GOLDILOCKS ∗P)

  Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.
- void ECP_GOLDILOCKS_outputxyz (ECP_GOLDILOCKS ∗P)

  Formats and outputs an ECP point to the console, in projective coordinates.
- void ECP_GOLDILOCKS_output (ECP_GOLDILOCKS ∗P)

  Formats and outputs an ECP point to the console, converted to affine coordinates.
- void ECP_GOLDILOCKS_rawoutput (ECP_GOLDILOCKS ∗P)

  Formats and outputs an ECP point to the console.

- void ECP_GOLDILOCKS_toOctet (octet ∗S, ECP_GOLDILOCKS ∗P, bool c)

  Formats and outputs an ECP point to an octet string The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If c is true, only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd.
- int ECP_GOLDILOCKS_fromOctet (ECP_GOLDILOCKS ∗P, octet ∗S)

  Creates an ECP point from an octet string.
- void ECP_GOLDILOCKS_dbl (ECP_GOLDILOCKS ∗P)

  Doubles an ECP instance P.
- void ECP_GOLDILOCKS_pinmul (ECP_GOLDILOCKS ∗P, int i, int b)

  Multiplies an ECP instance P by a small integer, side-channel resistant.
- void ECP_GOLDILOCKS_mul (ECP_GOLDILOCKS ∗P, BIG_448_58 b)

  Multiplies an ECP instance P by a BIG, side-channel resistant.
- void ECP_GOLDILOCKS_mul2 (ECP_GOLDILOCKS ∗P, ECP_GOLDILOCKS ∗Q, BIG_448_58 e, BIG_448_58 f)

  Calculates double multiplication P=e∗P+f∗Q, side-channel resistant.
- void ECP_GOLDILOCKS_generator (ECP_GOLDILOCKS ∗G)

  Get Group Generator from ROM.

Variables

- const int CURVE_A_GOLDILOCKS
- const int CURVE_Cof_I_GOLDILOCKS
- const int CURVE_B_I_GOLDILOCKS
- const BIG_448_58 CURVE_B_GOLDILOCKS
- const BIG_448_58 CURVE_Order_GOLDILOCKS
- const BIG_448_58 CURVE_Cof_GOLDILOCKS
- const BIG_448_58 CURVE_Gx_GOLDILOCKS
- const BIG_448_58 CURVE_Gy_GOLDILOCKS
- const BIG_448_58 CURVE_Pxa_GOLDILOCKS
- const BIG_448_58 CURVE_Pxb_GOLDILOCKS
- const BIG_448_58 CURVE_Pya_GOLDILOCKS
- const BIG_448_58 CURVE_Pyb_GOLDILOCKS
- const BIG_448_58 CURVE_Pxaa_GOLDILOCKS
- const BIG_448_58 CURVE_Pxab_GOLDILOCKS
- const BIG_448_58 CURVE_Pxba_GOLDILOCKS
- const BIG_448_58 CURVE_Pxbb_GOLDILOCKS
- const BIG_448_58 CURVE_Pyaa_GOLDILOCKS
- const BIG_448_58 CURVE_Pyab_GOLDILOCKS
- const BIG_448_58 CURVE_Pyba_GOLDILOCKS
- const BIG_448_58 CURVE_Pybb_GOLDILOCKS
- const BIG_448_58 CURVE_Pxaaa_GOLDILOCKS
- const BIG_448_58 CURVE_Pxaab_GOLDILOCKS
- const BIG_448_58 CURVE_Pxaba_GOLDILOCKS
- const BIG_448_58 CURVE_Pxabb_GOLDILOCKS
- const BIG_448_58 CURVE_Pxbaa_GOLDILOCKS
- const BIG_448_58 CURVE_Pxbab_GOLDILOCKS
- const BIG_448_58 CURVE_Pxbba_GOLDILOCKS
- const BIG_448_58 CURVE_Pxbbb_GOLDILOCKS
- const BIG_448_58 CURVE_Pyaaa_GOLDILOCKS
- const BIG_448_58 CURVE_Pyaab_GOLDILOCKS
- const BIG_448_58 CURVE_Pyaba_GOLDILOCKS
- const BIG_448_58 CURVE_Pyabb_GOLDILOCKS
- const BIG_448_58 CURVE_Pybaa_GOLDILOCKS

- const BIG_448_58 CURVE_Pybab_GOLDILOCKS
- const BIG_448_58 CURVE_Pybba_GOLDILOCKS
- const BIG_448_58 CURVE_Pybbb_GOLDILOCKS
- const BIG_448_58 CURVE_Bnx_GOLDILOCKS
- const BIG_448_58 CURVE_Cru_GOLDILOCKS
- const BIG_448_58 Fra_GOLDILOCKS
- const BIG_448_58 Frb_GOLDILOCKS
- const BIG_448_58 CURVE_W_GOLDILOCKS [2]
- const BIG_448_58 CURVE_SB_GOLDILOCKS [2][2]
- const BIG_448_58 CURVE_WB_GOLDILOCKS [4]
- const BIG_448_58 CURVE_BB_GOLDILOCKS [4][4]

8.23.1 Detailed Description

Author

Mike Scott

8.23.2 Function Documentation

8.23.2.1 ECP_GOLDILOCKS_add()

```
void ECP_GOLDILOCKS_add (
            ECP_GOLDILOCKS * P,
            ECP_GOLDILOCKS * Q )
```

Parameters

P	ECP instance, on exit =P+Q
Q	ECP instance to be added to P

8.23.2.2 ECP_GOLDILOCKS_affine()

```
void ECP_GOLDILOCKS_affine (
            ECP_GOLDILOCKS * P )
```

Parameters

P	ECP instance to be converted to affine form

8.23.2.3 ECP_GOLDILOCKS_cfp()

```
void ECP_GOLDILOCKS_cfp (
            ECP_GOLDILOCKS * Q )
```

Parameters

Q	ECP instance

8.23.2.4 ECP_GOLDILOCKS_copy()

```
void ECP_GOLDILOCKS_copy (
            ECP_GOLDILOCKS * P,
            ECP_GOLDILOCKS * Q )
```

Parameters

P	ECP instance, on exit = Q
Q	ECP instance to be copied

8.23.2.5 ECP_GOLDILOCKS_dbl()

```
void ECP_GOLDILOCKS_dbl (
            ECP_GOLDILOCKS * P )
```

Parameters

P	ECP instance, on exit =2*P

8.23.2.6 ECP_GOLDILOCKS_equals()

```
int ECP_GOLDILOCKS_equals (
            ECP_GOLDILOCKS * P,
            ECP_GOLDILOCKS * Q )
```

Parameters

P	ECP instance to be compared
Q	ECP instance to be compared

Returns

1 if P=Q, else returns 0

8.23.2.7 ECP_GOLDILOCKS_fromOctet()

```
int ECP_GOLDILOCKS_fromOctet (
            ECP_GOLDILOCKS * P,
            octet * S )
```

The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If in compressed form only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd

Parameters

P	ECP instance to be created from the octet string
S	input octet string return 1 if octet string corresponds to a point on the curve, else 0

8.23.2.8 ECP_GOLDILOCKS_generator()

```
void ECP_GOLDILOCKS_generator (
            ECP_GOLDILOCKS * G )
```

Parameters

G	ECP instance

8.23.2.9 ECP_GOLDILOCKS_get()

```
int ECP_GOLDILOCKS_get (
            BIG_448_58 x,
            BIG_448_58 y,
            ECP_GOLDILOCKS * P )
```

If x=y, returns only x

Parameters

x	BIG on exit = x coordinate of point
y	BIG on exit = y coordinate of point (unless x=y)
P	ECP instance (x,y)

Returns

sign of y, or -1 if P is point-at-infinity

8.23.2.10 ECP_GOLDILOCKS_inf()

```
void ECP_GOLDILOCKS_inf (
            ECP_GOLDILOCKS * P )
```

Parameters

P	ECP instance to be set to infinity

8.23.2.11 ECP_GOLDILOCKS_isinf()

```
int ECP_GOLDILOCKS_isinf (
            ECP_GOLDILOCKS * P )
```

Parameters

P	ECP point to be tested

Returns

1 if infinity, else returns 0

8.23.2.12 ECP_GOLDILOCKS_mapit()

```
void ECP_GOLDILOCKS_mapit (
            ECP_GOLDILOCKS * Q,
            octet * w )
```

Parameters

Q	ECP instance of correct order
w	OCTET byte array to be mapped

8.23.2.13 ECP_GOLDILOCKS_mul()

```
void ECP_GOLDILOCKS_mul (
```

```
            ECP_GOLDILOCKS * P,
            BIG_448_58 b )
```

Uses Montgomery ladder for Montgomery curves, otherwise fixed sized windows.

Parameters

P	ECP instance, on exit =b∗P
b	BIG number multiplier

8.23.2.14 ECP_GOLDILOCKS_mul2()

```
void ECP_GOLDILOCKS_mul2 (
            ECP_GOLDILOCKS * P,
            ECP_GOLDILOCKS * Q,
            BIG_448_58 e,
            BIG_448_58 f )
```

Parameters

P	ECP instance, on exit =e∗P+f∗Q
Q	ECP instance
e	BIG number multiplier
f	BIG number multiplier

8.23.2.15 ECP_GOLDILOCKS_neg()

```
void ECP_GOLDILOCKS_neg (
            ECP_GOLDILOCKS * P )
```

Parameters

P	ECP instance, on exit = -P

8.23.2.16 ECP_GOLDILOCKS_output()

```
void ECP_GOLDILOCKS_output (
            ECP_GOLDILOCKS * P )
```

Parameters

P	ECP instance to be printed

8.23.2.17 ECP_GOLDILOCKS_outputxyz()

```
void ECP_GOLDILOCKS_outputxyz (
            ECP_GOLDILOCKS * P )
```

Parameters

P	ECP instance to be printed

8.23.2.18 ECP_GOLDILOCKS_pinmul()

```
void ECP_GOLDILOCKS_pinmul (
            ECP_GOLDILOCKS * P,
            int i,
            int b )
```

Parameters

P	ECP instance, on exit =i*P
i	small integer multiplier
b	maximum number of bits in multiplier

8.23.2.19 ECP_GOLDILOCKS_rawoutput()

```
void ECP_GOLDILOCKS_rawoutput (
            ECP_GOLDILOCKS * P )
```

Parameters

P	ECP instance to be printed

8.23.2.20 ECP_GOLDILOCKS_rhs()

```
void ECP_GOLDILOCKS_rhs (
            FP_GOLDILOCKS * r,
            FP_GOLDILOCKS * x )
```

Function f(x) depends on form of elliptic curve, Weierstrass, Edwards or Montgomery. Used internally.

Parameters

r	BIG n-residue value of f(x)
x	BIG n-residue x

8.23.2.21 ECP_GOLDILOCKS_set()

```
int ECP_GOLDILOCKS_set (
            ECP_GOLDILOCKS * P,
            BIG_448_58 x,
            BIG_448_58 y )
```

Point P set to infinity if no such point on the curve.

Parameters

P	ECP instance to be set (x,y)
x	BIG x coordinate of point
y	BIG y coordinate of point

Returns

1 if point exists, else 0

8.23.2.22 ECP_GOLDILOCKS_setx()

```
int ECP_GOLDILOCKS_setx (
            ECP_GOLDILOCKS * P,
            BIG_448_58 x,
            int s )
```

Point P set to infinity if no such point on the curve. If x is on the curve then y is calculated from the curve equation. The correct y value (plus or minus) is selected given its sign s.

Parameters

P	ECP instance to be set (x,[y])
x	BIG x coordinate of point
s	an integer representing the "sign" of y, in fact its least significant bit.

8.23.2.23 ECP_GOLDILOCKS_sub()

```
void ECP_GOLDILOCKS_sub (
            ECP_GOLDILOCKS * P,
            ECP_GOLDILOCKS * Q )
```

Parameters

P	ECP instance, on exit =P-Q
Q	ECP instance to be subtracted from P

8.23.2.24 ECP_GOLDILOCKS_toOctet()

```
void ECP_GOLDILOCKS_toOctet (
            octet * S,
            ECP_GOLDILOCKS * P,
            bool c )
```

Parameters

c	compression required, true or false
S	output octet string
P	ECP instance to be converted to an octet string

8.23.3 Variable Documentation

8.23.3.1 CURVE_A_GOLDILOCKS

```
const int CURVE_A_GOLDILOCKS
```

Elliptic curve A parameter

8.23.3.2 CURVE_B_GOLDILOCKS

```
const BIG_448_58 CURVE_B_GOLDILOCKS
```

Elliptic curve B parameter

8.23.3.3 CURVE_B_I_GOLDILOCKS

```
const int CURVE_B_I_GOLDILOCKS
```

Elliptic curve B_i parameter

8.23.3.4 CURVE_BB_GOLDILOCKS

```
const BIG_448_58 CURVE_BB_GOLDILOCKS[4][4]
```

BN curve constant for GS decomposition

8.23.3.5 CURVE_Bnx_GOLDILOCKS

```
const BIG_448_58 CURVE_Bnx_GOLDILOCKS
```

BN curve x parameter

8.23.3.6 CURVE_Cof_GOLDILOCKS

```
const BIG_448_58 CURVE_Cof_GOLDILOCKS
```

Elliptic curve cofactor

8.23.3.7 CURVE_Cof_I_GOLDILOCKS

```
const int CURVE_Cof_I_GOLDILOCKS
```

Elliptic curve cofactor

8.23.3.8 CURVE_Cru_GOLDILOCKS

```
const BIG_448_58 CURVE_Cru_GOLDILOCKS
```

BN curve Cube Root of Unity

8.23.3.9 CURVE_Gx_GOLDILOCKS

```
const BIG_448_58 CURVE_Gx_GOLDILOCKS
```

x-coordinate of generator point in group G1

8.23.3.10 CURVE_Gy_GOLDILOCKS

```
const BIG_448_58 CURVE_Gy_GOLDILOCKS
```

y-coordinate of generator point in group G1

8.23.3.11 CURVE_Order_GOLDILOCKS

```
const BIG_448_58 CURVE_Order_GOLDILOCKS
```

Elliptic curve group order

8.23.3.12 CURVE_Pxa_GOLDILOCKS

`const BIG_448_58 CURVE_Pxa_GOLDILOCKS`

real part of x-coordinate of generator point in group G2

8.23.3.13 CURVE_Pxaa_GOLDILOCKS

`const BIG_448_58 CURVE_Pxaa_GOLDILOCKS`

real part of x-coordinate of generator point in group G2

8.23.3.14 CURVE_Pxaaa_GOLDILOCKS

`const BIG_448_58 CURVE_Pxaaa_GOLDILOCKS`

real part of x-coordinate of generator point in group G2

8.23.3.15 CURVE_Pxaab_GOLDILOCKS

`const BIG_448_58 CURVE_Pxaab_GOLDILOCKS`

imaginary part of x-coordinate of generator point in group G2

8.23.3.16 CURVE_Pxab_GOLDILOCKS

`const BIG_448_58 CURVE_Pxab_GOLDILOCKS`

imaginary part of x-coordinate of generator point in group G2

8.23.3.17 CURVE_Pxaba_GOLDILOCKS

`const BIG_448_58 CURVE_Pxaba_GOLDILOCKS`

real part of x-coordinate of generator point in group G2

8.23.3.18 CURVE_Pxabb_GOLDILOCKS

`const BIG_448_58 CURVE_Pxabb_GOLDILOCKS`

imaginary part of x-coordinate of generator point in group G2

8.23.3.19 CURVE_Pxb_GOLDILOCKS

`const BIG_448_58 CURVE_Pxb_GOLDILOCKS`

imaginary part of x-coordinate of generator point in group G2

8.23.3.20 CURVE_Pxba_GOLDILOCKS

`const BIG_448_58 CURVE_Pxba_GOLDILOCKS`

real part of x-coordinate of generator point in group G2

8.23.3.21 CURVE_Pxbaa_GOLDILOCKS

`const BIG_448_58 CURVE_Pxbaa_GOLDILOCKS`

real part of x-coordinate of generator point in group G2

8.23.3.22 CURVE_Pxbab_GOLDILOCKS

`const BIG_448_58 CURVE_Pxbab_GOLDILOCKS`

imaginary part of x-coordinate of generator point in group G2

8.23.3.23 CURVE_Pxbb_GOLDILOCKS

`const BIG_448_58 CURVE_Pxbb_GOLDILOCKS`

imaginary part of x-coordinate of generator point in group G2

8.23.3.24 CURVE_Pxbba_GOLDILOCKS

`const BIG_448_58 CURVE_Pxbba_GOLDILOCKS`

real part of x-coordinate of generator point in group G2

8.23.3.25 CURVE_Pxbbb_GOLDILOCKS

`const BIG_448_58 CURVE_Pxbbb_GOLDILOCKS`

imaginary part of x-coordinate of generator point in group G2

8.23.3.26 CURVE_Pya_GOLDILOCKS

`const BIG_448_58 CURVE_Pya_GOLDILOCKS`

real part of y-coordinate of generator point in group G2

8.23.3.27 CURVE_Pyaa_GOLDILOCKS

`const BIG_448_58 CURVE_Pyaa_GOLDILOCKS`

real part of y-coordinate of generator point in group G2

8.23.3.28 CURVE_Pyaaa_GOLDILOCKS

`const BIG_448_58 CURVE_Pyaaa_GOLDILOCKS`

real part of y-coordinate of generator point in group G2

8.23.3.29 CURVE_Pyaab_GOLDILOCKS

`const BIG_448_58 CURVE_Pyaab_GOLDILOCKS`

imaginary part of y-coordinate of generator point in group G2

8.23.3.30 CURVE_Pyab_GOLDILOCKS

`const BIG_448_58 CURVE_Pyab_GOLDILOCKS`

imaginary part of y-coordinate of generator point in group G2

8.23.3.31 CURVE_Pyaba_GOLDILOCKS

`const BIG_448_58 CURVE_Pyaba_GOLDILOCKS`

real part of y-coordinate of generator point in group G2

8.23.3.32 CURVE_Pyabb_GOLDILOCKS

`const BIG_448_58 CURVE_Pyabb_GOLDILOCKS`

imaginary part of y-coordinate of generator point in group G2

8.23.3.33 CURVE_Pyb_GOLDILOCKS

`const BIG_448_58 CURVE_Pyb_GOLDILOCKS`

imaginary part of y-coordinate of generator point in group G2

8.23.3.34 CURVE_Pyba_GOLDILOCKS

`const BIG_448_58 CURVE_Pyba_GOLDILOCKS`

real part of y-coordinate of generator point in group G2

8.23.3.35 CURVE_Pybaa_GOLDILOCKS

`const BIG_448_58 CURVE_Pybaa_GOLDILOCKS`

real part of y-coordinate of generator point in group G2

8.23.3.36 CURVE_Pybab_GOLDILOCKS

```
const BIG_448_58 CURVE_Pybab_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

8.23.3.37 CURVE_Pybb_GOLDILOCKS

```
const BIG_448_58 CURVE_Pybb_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

8.23.3.38 CURVE_Pybba_GOLDILOCKS

```
const BIG_448_58 CURVE_Pybba_GOLDILOCKS
```

real part of y-coordinate of generator point in group G2

8.23.3.39 CURVE_Pybbb_GOLDILOCKS

```
const BIG_448_58 CURVE_Pybbb_GOLDILOCKS
```

imaginary part of y-coordinate of generator point in group G2

8.23.3.40 CURVE_SB_GOLDILOCKS

```
const BIG_448_58 CURVE_SB_GOLDILOCKS[2][2]
```

BN curve constant for GLV decomposition

8.23.3.41 CURVE_W_GOLDILOCKS

```
const BIG_448_58 CURVE_W_GOLDILOCKS[2]
```

BN curve constant for GLV decomposition

8.23.3.42 CURVE_WB_GOLDILOCKS

```
const BIG_448_58 CURVE_WB_GOLDILOCKS[4]
```

BN curve constant for GS decomposition

8.23.3.43 Fra_GOLDILOCKS

```
const BIG_448_58 Fra_GOLDILOCKS
```

real part of BN curve Frobenius Constant

8.23.3.44 Frb_GOLDILOCKS

```
const BIG_448_58 Frb_GOLDILOCKS
```

imaginary part of BN curve Frobenius Constant

8.24 ecp_NIST256.h File Reference

ECP Header File.

```
#include "fp_NIST256.h"
#include "config_curve_NIST256.h"
```

Data Structures

- struct ECP_NIST256

  ECP structure - Elliptic Curve Point over base field.

Functions

- int ECP_NIST256_isinf (ECP_NIST256 ∗P)

  Tests for ECP point equal to infinity.
- int ECP_NIST256_equals (ECP_NIST256 ∗P, ECP_NIST256 ∗Q)

  Tests for equality of two ECPs.
- void ECP_NIST256_copy (ECP_NIST256 ∗P, ECP_NIST256 ∗Q)

  Copy ECP point to another ECP point.
- void ECP_NIST256_neg (ECP_NIST256 ∗P)

  Negation of an ECP point.
- void ECP_NIST256_inf (ECP_NIST256 ∗P)

  Set ECP to point-at-infinity.
- void ECP_NIST256_rhs (FP_NIST256 ∗r, FP_NIST256 ∗x)

  Calculate Right Hand Side of curve equation y^2=f(x)
- int ECP_NIST256_set (ECP_NIST256 ∗P, BIG_256_56 x, BIG_256_56 y)

  Set ECP to point(x,y) given x and y.
- int ECP_NIST256_get (BIG_256_56 x, BIG_256_56 y, ECP_NIST256 ∗P)

  Extract x and y coordinates of an ECP point P.
- void ECP_NIST256_add (ECP_NIST256 ∗P, ECP_NIST256 ∗Q)

  Adds ECP instance Q to ECP instance P.
- void ECP_NIST256_sub (ECP_NIST256 ∗P, ECP_NIST256 ∗Q)

  Subtracts ECP instance Q from ECP instance P.
- int ECP_NIST256_setx (ECP_NIST256 ∗P, BIG_256_56 x, int s)

  Set ECP to point(x,y) given just x and sign of y.
- void ECP_NIST256_cfp (ECP_NIST256 ∗Q)

  Multiplies Point by curve co-factor.
- void ECP_NIST256_mapit (ECP_NIST256 ∗Q, octet ∗w)

  Maps random BIG to curve point of correct order.
- void ECP_NIST256_affine (ECP_NIST256 ∗P)

Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.
- void ECP_NIST256_outputxyz (ECP_NIST256 ∗P)

  Formats and outputs an ECP point to the console, in projective coordinates.
- void ECP_NIST256_output (ECP_NIST256 ∗P)

  Formats and outputs an ECP point to the console, converted to affine coordinates.
- void ECP_NIST256_rawoutput (ECP_NIST256 ∗P)

  Formats and outputs an ECP point to the console.
- void ECP_NIST256_toOctet (octet ∗S, ECP_NIST256 ∗P, bool c)

  Formats and outputs an ECP point to an octet string The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If c is true, only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd.
- int ECP_NIST256_fromOctet (ECP_NIST256 ∗P, octet ∗S)

  Creates an ECP point from an octet string.
- void ECP_NIST256_dbl (ECP_NIST256 ∗P)

  Doubles an ECP instance P.
- void ECP_NIST256_pinmul (ECP_NIST256 ∗P, int i, int b)

  Multiplies an ECP instance P by a small integer, side-channel resistant.
- void ECP_NIST256_mul (ECP_NIST256 ∗P, BIG_256_56 b)

  Multiplies an ECP instance P by a BIG, side-channel resistant.
- void ECP_NIST256_mul2 (ECP_NIST256 ∗P, ECP_NIST256 ∗Q, BIG_256_56 e, BIG_256_56 f)

  Calculates double multiplication P=e∗P+f∗Q, side-channel resistant.
- void ECP_NIST256_generator (ECP_NIST256 ∗G)

  Get Group Generator from ROM.

Variables

- const int CURVE_A_NIST256
- const int CURVE_Cof_I_NIST256
- const int CURVE_B_I_NIST256
- const BIG_256_56 CURVE_B_NIST256
- const BIG_256_56 CURVE_Order_NIST256
- const BIG_256_56 CURVE_Cof_NIST256
- const BIG_256_56 CURVE_Gx_NIST256
- const BIG_256_56 CURVE_Gy_NIST256
- const BIG_256_56 CURVE_Pxa_NIST256
- const BIG_256_56 CURVE_Pxb_NIST256
- const BIG_256_56 CURVE_Pya_NIST256
- const BIG_256_56 CURVE_Pyb_NIST256
- const BIG_256_56 CURVE_Pxaa_NIST256
- const BIG_256_56 CURVE_Pxab_NIST256
- const BIG_256_56 CURVE_Pxba_NIST256
- const BIG_256_56 CURVE_Pxbb_NIST256
- const BIG_256_56 CURVE_Pyaa_NIST256
- const BIG_256_56 CURVE_Pyab_NIST256
- const BIG_256_56 CURVE_Pyba_NIST256
- const BIG_256_56 CURVE_Pybb_NIST256
- const BIG_256_56 CURVE_Pxaaa_NIST256
- const BIG_256_56 CURVE_Pxaab_NIST256
- const BIG_256_56 CURVE_Pxaba_NIST256
- const BIG_256_56 CURVE_Pxabb_NIST256
- const BIG_256_56 CURVE_Pxbaa_NIST256
- const BIG_256_56 CURVE_Pxbab_NIST256

- const BIG_256_56 CURVE_Pxbba_NIST256
- const BIG_256_56 CURVE_Pxbbb_NIST256
- const BIG_256_56 CURVE_Pyaaa_NIST256
- const BIG_256_56 CURVE_Pyaab_NIST256
- const BIG_256_56 CURVE_Pyaba_NIST256
- const BIG_256_56 CURVE_Pyabb_NIST256
- const BIG_256_56 CURVE_Pybaa_NIST256
- const BIG_256_56 CURVE_Pybab_NIST256
- const BIG_256_56 CURVE_Pybba_NIST256
- const BIG_256_56 CURVE_Pybbb_NIST256
- const BIG_256_56 CURVE_Bnx_NIST256
- const BIG_256_56 CURVE_Cru_NIST256
- const BIG_256_56 Fra_NIST256
- const BIG_256_56 Frb_NIST256
- const BIG_256_56 CURVE_W_NIST256 [2]
- const BIG_256_56 CURVE_SB_NIST256 [2][2]
- const BIG_256_56 CURVE_WB_NIST256 [4]
- const BIG_256_56 CURVE_BB_NIST256 [4][4]

8.24.1 Detailed Description

Author

Mike Scott

8.24.2 Function Documentation

8.24.2.1 ECP_NIST256_add()

```
void ECP_NIST256_add (
            ECP_NIST256 * P,
            ECP_NIST256 * Q )
```

Parameters

P	ECP instance, on exit =P+Q
Q	ECP instance to be added to P

8.24.2.2 ECP_NIST256_affine()

```
void ECP_NIST256_affine (
            ECP_NIST256 * P )
```

Parameters

P	ECP instance to be converted to affine form

8.24.2.3 ECP_NIST256_cfp()

```
void ECP_NIST256_cfp (
            ECP_NIST256 * Q )
```

Parameters

Q	ECP instance

8.24.2.4 ECP_NIST256_copy()

```
void ECP_NIST256_copy (
            ECP_NIST256 * P,
            ECP_NIST256 * Q )
```

Parameters

P	ECP instance, on exit = Q
Q	ECP instance to be copied

8.24.2.5 ECP_NIST256_dbl()

```
void ECP_NIST256_dbl (
            ECP_NIST256 * P )
```

Parameters

P	ECP instance, on exit =2∗P

8.24.2.6 ECP_NIST256_equals()

```
int ECP_NIST256_equals (
            ECP_NIST256 * P,
            ECP_NIST256 * Q )
```

Parameters

P	ECP instance to be compared
Q	ECP instance to be compared

Returns

1 if P=Q, else returns 0

8.24.2.7 ECP_NIST256_fromOctet()

```
int ECP_NIST256_fromOctet (
            ECP_NIST256 * P,
            octet * S )
```

The octet string is normally in the standard form 0x04|x|y Here x (and y) are the x and y coordinates in left justified big-endian base 256 form. For Montgomery curve it is 0x06|x If in compressed form only the x coordinate is provided as in 0x2|x if y is even, or 0x3|x if y is odd

Parameters

P	ECP instance to be created from the octet string
S	input octet string return 1 if octet string corresponds to a point on the curve, else 0

8.24.2.8 ECP_NIST256_generator()

```
void ECP_NIST256_generator (
            ECP_NIST256 * G )
```

Parameters

G	ECP instance

8.24.2.9 ECP_NIST256_get()

```
int ECP_NIST256_get (
            BIG_256_56 x,
            BIG_256_56 y,
            ECP_NIST256 * P )
```

If x=y, returns only x

Parameters

x	BIG on exit = x coordinate of point
y	BIG on exit = y coordinate of point (unless x=y)
P	ECP instance (x,y)

Returns

sign of y, or -1 if P is point-at-infinity

8.24.2.10 ECP_NIST256_inf()

```
void ECP_NIST256_inf (
            ECP_NIST256 * P )
```

Parameters

P	ECP instance to be set to infinity

8.24.2.11 ECP_NIST256_isinf()

```
int ECP_NIST256_isinf (
            ECP_NIST256 * P )
```

Parameters

P	ECP point to be tested

Returns

1 if infinity, else returns 0

8.24.2.12 ECP_NIST256_mapit()

```
void ECP_NIST256_mapit (
            ECP_NIST256 * Q,
            octet * w )
```

Parameters

Q	ECP instance of correct order
w	OCTET byte array to be mapped

8.24.2.13 ECP_NIST256_mul()

```
void ECP_NIST256_mul (
            ECP_NIST256 * P,
            BIG_256_56 b )
```

Uses Montgomery ladder for Montgomery curves, otherwise fixed sized windows.

Parameters

P	ECP instance, on exit =b∗P
b	BIG number multiplier

8.24.2.14 ECP_NIST256_mul2()

```
void ECP_NIST256_mul2 (
            ECP_NIST256 * P,
            ECP_NIST256 * Q,
            BIG_256_56 e,
            BIG_256_56 f )
```

Parameters

P	ECP instance, on exit =e∗P+f∗Q
Q	ECP instance
e	BIG number multiplier
f	BIG number multiplier

8.24.2.15 ECP_NIST256_neg()

```
void ECP_NIST256_neg (
            ECP_NIST256 * P )
```

Parameters

P	ECP instance, on exit = -P

8.24.2.16 ECP_NIST256_output()

```
void ECP_NIST256_output (
```

```
            ECP_NIST256 * P )
```

Parameters

P	ECP instance to be printed

8.24.2.17 ECP_NIST256_outputxyz()

```
void ECP_NIST256_outputxyz (
            ECP_NIST256 * P )
```

Parameters

P	ECP instance to be printed

8.24.2.18 ECP_NIST256_pinmul()

```
void ECP_NIST256_pinmul (
            ECP_NIST256 * P,
            int i,
            int b )
```

Parameters

P	ECP instance, on exit =i∗P
i	small integer multiplier
b	maximum number of bits in multiplier

8.24.2.19 ECP_NIST256_rawoutput()

```
void ECP_NIST256_rawoutput (
            ECP_NIST256 * P )
```

Parameters

P	ECP instance to be printed

8.24.2.20 ECP_NIST256_rhs()

```
void ECP_NIST256_rhs (
            FP_NIST256 * r,
            FP_NIST256 * x )
```

Function f(x) depends on form of elliptic curve, Weierstrass, Edwards or Montgomery. Used internally.

Parameters

r	BIG n-residue value of f(x)
x	BIG n-residue x

8.24.2.21 ECP_NIST256_set()

```
int ECP_NIST256_set (
            ECP_NIST256 * P,
            BIG_256_56 x,
            BIG_256_56 y )
```

Point P set to infinity if no such point on the curve.

Parameters

P	ECP instance to be set (x,y)
x	BIG x coordinate of point
y	BIG y coordinate of point

Returns

1 if point exists, else 0

8.24.2.22 ECP_NIST256_setx()

```
int ECP_NIST256_setx (
            ECP_NIST256 * P,
            BIG_256_56 x,
            int s )
```

Point P set to infinity if no such point on the curve. If x is on the curve then y is calculated from the curve equation. The correct y value (plus or minus) is selected given its sign s.

Parameters

P	ECP instance to be set (x,[y])
x	BIG x coordinate of point
s	an integer representing the "sign" of y, in fact its least significant bit.

8.24.2.23 ECP_NIST256_sub()

```
void ECP_NIST256_sub (
            ECP_NIST256 * P,
            ECP_NIST256 * Q )
```

Parameters

P	ECP instance, on exit =P-Q
Q	ECP instance to be subtracted from P

8.24.2.24 ECP_NIST256_toOctet()

```
void ECP_NIST256_toOctet (
            octet * S,
            ECP_NIST256 * P,
            bool c )
```

Parameters

c	compression required, true or false
S	output octet string
P	ECP instance to be converted to an octet string

8.24.3 Variable Documentation

8.24.3.1 CURVE_A_NIST256

```
const int CURVE_A_NIST256
```

Elliptic curve A parameter

8.24.3.2 CURVE_B_I_NIST256

```
const int CURVE_B_I_NIST256
```

Elliptic curve B_i parameter

8.24.3.3 CURVE_B_NIST256

```
const BIG_256_56 CURVE_B_NIST256
```

Elliptic curve B parameter

8.24.3.4 CURVE_BB_NIST256

```
const BIG_256_56 CURVE_BB_NIST256[4][4]
```

BN curve constant for GS decomposition

8.24.3.5 CURVE_Bnx_NIST256

```
const BIG_256_56 CURVE_Bnx_NIST256
```

BN curve x parameter

8.24.3.6 CURVE_Cof_I_NIST256

```
const int CURVE_Cof_I_NIST256
```

Elliptic curve cofactor

8.24.3.7 CURVE_Cof_NIST256

```
const BIG_256_56 CURVE_Cof_NIST256
```

Elliptic curve cofactor

8.24.3.8 CURVE_Cru_NIST256

```
const BIG_256_56 CURVE_Cru_NIST256
```

BN curve Cube Root of Unity

8.24.3.9 CURVE_Gx_NIST256

```
const BIG_256_56 CURVE_Gx_NIST256
```

x-coordinate of generator point in group G1

8.24.3.10 CURVE_Gy_NIST256

```
const BIG_256_56 CURVE_Gy_NIST256
```

y-coordinate of generator point in group G1

8.24.3.11 CURVE_Order_NIST256

`const BIG_256_56 CURVE_Order_NIST256`

Elliptic curve group order

8.24.3.12 CURVE_Pxa_NIST256

`const BIG_256_56 CURVE_Pxa_NIST256`

real part of x-coordinate of generator point in group G2

8.24.3.13 CURVE_Pxaa_NIST256

`const BIG_256_56 CURVE_Pxaa_NIST256`

real part of x-coordinate of generator point in group G2

8.24.3.14 CURVE_Pxaaa_NIST256

`const BIG_256_56 CURVE_Pxaaa_NIST256`

real part of x-coordinate of generator point in group G2

8.24.3.15 CURVE_Pxaab_NIST256

`const BIG_256_56 CURVE_Pxaab_NIST256`

imaginary part of x-coordinate of generator point in group G2

8.24.3.16 CURVE_Pxab_NIST256

`const BIG_256_56 CURVE_Pxab_NIST256`

imaginary part of x-coordinate of generator point in group G2

8.24.3.17 CURVE_Pxaba_NIST256

`const BIG_256_56 CURVE_Pxaba_NIST256`

real part of x-coordinate of generator point in group G2

8.24.3.18 CURVE_Pxabb_NIST256

`const BIG_256_56 CURVE_Pxabb_NIST256`

imaginary part of x-coordinate of generator point in group G2

8.24.3.19 CURVE_Pxb_NIST256

```
const BIG_256_56 CURVE_Pxb_NIST256
```

imaginary part of x-coordinate of generator point in group G2

8.24.3.20 CURVE_Pxba_NIST256

```
const BIG_256_56 CURVE_Pxba_NIST256
```

real part of x-coordinate of generator point in group G2

8.24.3.21 CURVE_Pxbaa_NIST256

```
const BIG_256_56 CURVE_Pxbaa_NIST256
```

real part of x-coordinate of generator point in group G2

8.24.3.22 CURVE_Pxbab_NIST256

```
const BIG_256_56 CURVE_Pxbab_NIST256
```

imaginary part of x-coordinate of generator point in group G2

8.24.3.23 CURVE_Pxbb_NIST256

```
const BIG_256_56 CURVE_Pxbb_NIST256
```

imaginary part of x-coordinate of generator point in group G2

8.24.3.24 CURVE_Pxbba_NIST256

```
const BIG_256_56 CURVE_Pxbba_NIST256
```

real part of x-coordinate of generator point in group G2

8.24.3.25 CURVE_Pxbbb_NIST256

```
const BIG_256_56 CURVE_Pxbbb_NIST256
```

imaginary part of x-coordinate of generator point in group G2

8.24.3.26 CURVE_Pya_NIST256

```
const BIG_256_56 CURVE_Pya_NIST256
```

real part of y-coordinate of generator point in group G2

8.24.3.27 CURVE_Pyaa_NIST256

```
const BIG_256_56 CURVE_Pyaa_NIST256
```

real part of y-coordinate of generator point in group G2

8.24.3.28 CURVE_Pyaaa_NIST256

```
const BIG_256_56 CURVE_Pyaaa_NIST256
```

real part of y-coordinate of generator point in group G2

8.24.3.29 CURVE_Pyaab_NIST256

```
const BIG_256_56 CURVE_Pyaab_NIST256
```

imaginary part of y-coordinate of generator point in group G2

8.24.3.30 CURVE_Pyab_NIST256

```
const BIG_256_56 CURVE_Pyab_NIST256
```

imaginary part of y-coordinate of generator point in group G2

8.24.3.31 CURVE_Pyaba_NIST256

```
const BIG_256_56 CURVE_Pyaba_NIST256
```

real part of y-coordinate of generator point in group G2

8.24.3.32 CURVE_Pyabb_NIST256

```
const BIG_256_56 CURVE_Pyabb_NIST256
```

imaginary part of y-coordinate of generator point in group G2

8.24.3.33 CURVE_Pyb_NIST256

```
const BIG_256_56 CURVE_Pyb_NIST256
```

imaginary part of y-coordinate of generator point in group G2

8.24.3.34 CURVE_Pyba_NIST256

```
const BIG_256_56 CURVE_Pyba_NIST256
```

real part of y-coordinate of generator point in group G2

8.24.3.35 CURVE_Pybaa_NIST256

```
const BIG_256_56 CURVE_Pybaa_NIST256
```

real part of y-coordinate of generator point in group G2

8.24.3.36 CURVE_Pybab_NIST256

```
const BIG_256_56 CURVE_Pybab_NIST256
```

imaginary part of y-coordinate of generator point in group G2

8.24.3.37 CURVE_Pybb_NIST256

```
const BIG_256_56 CURVE_Pybb_NIST256
```

imaginary part of y-coordinate of generator point in group G2

8.24.3.38 CURVE_Pybba_NIST256

```
const BIG_256_56 CURVE_Pybba_NIST256
```

real part of y-coordinate of generator point in group G2

8.24.3.39 CURVE_Pybbb_NIST256

```
const BIG_256_56 CURVE_Pybbb_NIST256
```

imaginary part of y-coordinate of generator point in group G2

8.24.3.40 CURVE_SB_NIST256

```
const BIG_256_56 CURVE_SB_NIST256[2][2]
```

BN curve constant for GLV decomposition

8.24.3.41 CURVE_W_NIST256

```
const BIG_256_56 CURVE_W_NIST256[2]
```

BN curve constant for GLV decomposition

8.24.3.42 CURVE_WB_NIST256

```
const BIG_256_56 CURVE_WB_NIST256[4]
```

BN curve constant for GS decomposition

8.24.3.43 Fra_NIST256

```
const BIG_256_56 Fra_NIST256
```

real part of BN curve Frobenius Constant

8.24.3.44 Frb_NIST256

```
const BIG_256_56 Frb_NIST256
```

imaginary part of BN curve Frobenius Constant

8.25 ff_2048.h File Reference

FF Header File.

```
#include "big_1024_58.h"
#include "config_ff_2048.h"
```

Macros

- #define HFLEN_2048 (FFLEN_2048/2)
- #define P_MBITS_2048 (MODBYTES_1024_58*8)
- #define P_TBITS_2048 (P_MBITS_2048%BASEBITS_1024_58)
- #define P_EXCESS_2048(a) (((a[NLEN_1024_58-1])>>(P_TBITS_2048))+1)
- #define P_FEXCESS_2048 ((chunk)1<<(BASEBITS_1024_58*NLEN_1024_58-P_MBITS_2048-1))

Functions

- void FF_2048_copy (BIG_1024_58 *x, BIG_1024_58 *y, int n)

  Copy one FF element of given length to another.
- void FF_2048_init (BIG_1024_58 *x, sign32 m, int n)

  Initialize an FF element of given length from a 32-bit integer m.
- void FF_2048_zero (BIG_1024_58 *x, int n)

  Set FF element of given size to zero.
- int FF_2048_iszilch (BIG_1024_58 *x, int n)

  Tests for FF element equal to zero.
- int FF_2048_parity (BIG_1024_58 *x)

  return parity of an FF, that is the least significant bit
- int FF_2048_lastbits (BIG_1024_58 *x, int m)

  return least significant m bits of an FF
- void FF_2048_one (BIG_1024_58 *x, int n)

  Set FF element of given size to unity.
- int FF_2048_comp (BIG_1024_58 *x, BIG_1024_58 *y, int n)

  Compares two FF numbers. Inputs must be normalised externally.
- void FF_2048_add (BIG_1024_58 *x, BIG_1024_58 *y, BIG_1024_58 *z, int n)

  addition of two FFs

- void FF_2048_sub (BIG_1024_58 ∗x, BIG_1024_58 ∗y, BIG_1024_58 ∗z, int n)

  subtraction of two FFs
- void FF_2048_inc (BIG_1024_58 ∗x, int m, int n)

  increment an FF by an integer,and normalise
- void FF_2048_dec (BIG_1024_58 ∗x, int m, int n)

  Decrement an FF by an integer,and normalise.
- void FF_2048_norm (BIG_1024_58 ∗x, int n)

  Normalises the components of an FF.
- void FF_2048_shl (BIG_1024_58 ∗x, int n)

  Shift left an FF by 1 bit.
- void FF_2048_shr (BIG_1024_58 ∗x, int n)

  Shift right an FF by 1 bit.
- void FF_2048_output (BIG_1024_58 ∗x, int n)

  Formats and outputs an FF to the console.
- void FF_2048_rawoutput (BIG_1024_58 ∗x, int n)

  Formats and outputs an FF to the console, in raw form.
- void FF_2048_toOctet (octet ∗S, BIG_1024_58 ∗x, int n)

  Formats and outputs an FF instance to an octet string.
- void FF_2048_fromOctet (BIG_1024_58 ∗x, octet ∗S, int n)

  Populates an FF instance from an octet string.
- void FF_2048_mul (BIG_1024_58 ∗x, BIG_1024_58 ∗y, BIG_1024_58 ∗z, int n)

  Multiplication of two FFs.
- void FF_2048_mod (BIG_1024_58 ∗x, BIG_1024_58 ∗m, int n)

  Reduce FF mod a modulus.
- void FF_2048_sqr (BIG_1024_58 ∗x, BIG_1024_58 ∗y, int n)

  Square an FF.
- void FF_2048_dmod (BIG_1024_58 ∗x, BIG_1024_58 ∗y, BIG_1024_58 ∗z, int n)

  Reduces a double-length FF with respect to a given modulus.
- void FF_2048_invmodp (BIG_1024_58 ∗x, BIG_1024_58 ∗y, BIG_1024_58 ∗z, int n)

  Invert an FF mod a prime modulus.
- void FF_2048_random (BIG_1024_58 ∗x, csprng ∗R, int n)

  Create an FF from a random number generator.
- void FF_2048_randomnum (BIG_1024_58 ∗x, BIG_1024_58 ∗y, csprng ∗R, int n)

  Create a random FF less than a given modulus from a random number generator.
- void FF_2048_skpow (BIG_1024_58 ∗r, BIG_1024_58 ∗x, BIG_1024_58 ∗e, BIG_1024_58 ∗m, int n)

  Calculate r=x^e mod m, side channel resistant.
- void FF_2048_skspow (BIG_1024_58 ∗r, BIG_1024_58 ∗x, BIG_1024_58 e, BIG_1024_58 ∗m, int n)

  Calculate r=x^e mod m, side channel resistant.
- void FF_2048_power (BIG_1024_58 ∗r, BIG_1024_58 ∗x, int e, BIG_1024_58 ∗m, int n)

  Calculate r=x^e mod m.
- void FF_2048_pow (BIG_1024_58 ∗r, BIG_1024_58 ∗x, BIG_1024_58 ∗e, BIG_1024_58 ∗m, int n)

  Calculate r=x^e mod m.
- int FF_2048_cfactor (BIG_1024_58 ∗x, sign32 s, int n)

  Test if an FF has factor in common with integer s.
- int FF_2048_prime (BIG_1024_58 ∗x, csprng ∗R, int n)

  Test if an FF is prime.
- void FF_2048_pow2 (BIG_1024_58 ∗r, BIG_1024_58 ∗x, BIG_1024_58 e, BIG_1024_58 ∗y, BIG_1024_58 f, BIG_1024_58 ∗m, int n)

  Calculate r=x^e.y^f mod m.

8.25.1 Detailed Description

Author

Mike Scott

8.25.2 Macro Definition Documentation

8.25.2.1 HFLEN_2048

```
#define HFLEN_2048 (FFLEN_2048/2)
```

Useful for half-size RSA private key operations

8.25.2.2 P_EXCESS_2048

```
#define P_EXCESS_2048(
              a ) (((a[NLEN_1024_58-1])>>(P_TBITS_2048))+1)
```

TODO

8.25.2.3 P_FEXCESS_2048

```
#define P_FEXCESS_2048 ((chunk)1<<(BASEBITS_1024_58*NLEN_1024_58-P_MBITS_2048-1))
```

TODO

8.25.2.4 P_MBITS_2048

```
#define P_MBITS_2048 (MODBYTES_1024_58*8)
```

Number of bits in modulus

8.25.2.5 P_TBITS_2048

```
#define P_TBITS_2048 (P_MBITS_2048%BASEBITS_1024_58)
```

TODO

8.25.3 Function Documentation

8.25.3.1 FF_2048_add()

```
void FF_2048_add (
            BIG_1024_58 * x,
            BIG_1024_58 * y,
            BIG_1024_58 * z,
            int n )
```

Parameters

x	FF instance, on exit = y+z
y	FF instance
z	FF instance
n	size of FF in BIGs

8.25.3.2 FF_2048_cfactor()

```
int FF_2048_cfactor (
            BIG_1024_58 * x,
            sign32 s,
            int n )
```

Parameters

x	FF instance to be tested
s	the supplied integer
n	size of FF in BIGs

Returns

1 if gcd(x,s)!=1, else return 0

8.25.3.3 FF_2048_comp()

```
int FF_2048_comp (
            BIG_1024_58 * x,
            BIG_1024_58 * y,
            int n )
```

Parameters

x	first FF number to be compared
y	second FF number to be compared
n	size of FF in BIGs

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.25.3.4 FF_2048_copy()

```
void FF_2048_copy (
            BIG_1024_58 * x,
            BIG_1024_58 * y,
            int n )
```

Parameters

x	FF instance to be copied to, on exit = y
y	FF instance to be copied from
n	size of FF in BIGs

8.25.3.5 FF_2048_dec()

```
void FF_2048_dec (
            BIG_1024_58 * x,
            int m,
            int n )
```

Parameters

x	FF instance, on exit = x-m
m	an integer to be subtracted from x
n	size of FF in BIGs

8.25.3.6 FF_2048_dmod()

```
void FF_2048_dmod (
            BIG_1024_58 * x,
            BIG_1024_58 * y,
            BIG_1024_58 * z,
            int n )
```

This is slow

Parameters

x	FF instance, on exit = y mod z
y	FF instance, of double length 2∗n
z	FF modulus
n	size of FF in BIGs

8.25.3.7 FF_2048_fromOctet()

```
void FF_2048_fromOctet (
            BIG_1024_58 * x,
            octet * S,
            int n )
```

Creates FF from big-endian base 256 form.

Parameters

x	FF instance to be created from an octet string
S	input octet string
n	size of FF in BIGs

8.25.3.8 FF_2048_inc()

```
void FF_2048_inc (
            BIG_1024_58 * x,
            int m,
            int n )
```

Parameters

x	FF instance, on exit = x+m
m	an integer to be added to x
n	size of FF in BIGs

8.25.3.9 FF_2048_init()

```
void FF_2048_init (
            BIG_1024_58 * x,
            sign32 m,
            int n )
```

Parameters

x	FF instance to be copied to, on exit = m
m	integer
n	size of FF in BIGs

8.25.3.10 FF_2048_invmodp()

```
void FF_2048_invmodp (
            BIG_1024_58 * x,
            BIG_1024_58 * y,
            BIG_1024_58 * z,
            int n )
```

Parameters

x	FF instance, on exit = 1/y mod z
y	FF instance
z	FF prime modulus
n	size of FF in BIGs

8.25.3.11 FF_2048_iszilch()

```
int FF_2048_iszilch (
            BIG_1024_58 * x,
            int n )
```

Parameters

x	FF number to be tested
n	size of FF in BIGs

Returns

1 if zero, else returns 0

8.25.3.12 FF_2048_lastbits()

```
int FF_2048_lastbits (
            BIG_1024_58 * x,
            int m )
```

Parameters

x	FF number
m	number of bits to return. Assumed to be less than BASEBITS.

Returns

least significant n bits as an integer

8.25.3.13 FF_2048_mod()

```
void FF_2048_mod (
            BIG_1024_58 * x,
            BIG_1024_58 * m,
            int n )
```

This is slow

Parameters

x	FF instance to be reduced mod m - on exit = x mod m
m	FF modulus
n	size of FF in BIGs

8.25.3.14 FF_2048_mul()

```
void FF_2048_mul (
            BIG_1024_58 * x,
            BIG_1024_58 * y,
            BIG_1024_58 * z,
            int n )
```

Uses Karatsuba method internally

Parameters

x	FF instance, on exit = y∗z
y	FF instance
z	FF instance
n	size of FF in BIGs

8.25.3.15 FF_2048_norm()

```
void FF_2048_norm (
            BIG_1024_58 * x,
            int n )
```

Parameters

x	FF instance to be normalised
n	size of FF in BIGs

8.25.3.16 FF_2048_one()

```
void FF_2048_one (
            BIG_1024_58 * x,
            int n )
```

Parameters

x	FF instance to be set to unity
n	size of FF in BIGs

8.25.3.17 FF_2048_output()

```
void FF_2048_output (
            BIG_1024_58 * x,
            int n )
```

Parameters

x	FF instance to be printed
n	size of FF in BIGs

8.25.3.18 FF_2048_parity()

```
int FF_2048_parity (
            BIG_1024_58 * x )
```

Parameters

x	FF number

Returns

0 or 1

8.25.3.19 FF_2048_pow()

```
void FF_2048_pow (
            BIG_1024_58 * r,
            BIG_1024_58 * x,
            BIG_1024_58 * e,
            BIG_1024_58 * m,
            int n )
```

Parameters

r	FF instance, on exit = x^e mod p
x	FF instance
e	FF exponent
m	FF modulus
n	size of FF in BIGs

8.25.3.20 FF_2048_pow2()

```
void FF_2048_pow2 (
            BIG_1024_58 * r,
            BIG_1024_58 * x,
            BIG_1024_58 e,
            BIG_1024_58 * y,
            BIG_1024_58 f,
            BIG_1024_58 * m,
            int n )
```

Parameters

r	FF instance, on exit = x^e.y^f mod p
x	FF instance
e	BIG exponent
y	FF instance
f	BIG exponent
m	FF modulus
n	size of FF in BIGs

8.25.3.21 FF_2048_power()

```
void FF_2048_power (
            BIG_1024_58 * r,
            BIG_1024_58 * x,
            int e,
            BIG_1024_58 * m,
            int n )
```

For very short integer exponent

Parameters

r	FF instance, on exit = x^e mod p
x	FF instance
e	integer exponent
m	FF modulus
n	size of FF in BIGs

8.25.3.22 FF_2048_prime()

```
int FF_2048_prime (
            BIG_1024_58 * x,
            csprng * R,
            int n )
```

Uses Miller-Rabin Method

Parameters

x	FF instance to be tested
R	an instance of a Cryptographically Secure Random Number Generator
n	size of FF in BIGs

Returns

1 if x is (almost certainly) prime, else return 0

8.25.3.23 FF_2048_random()

```
void FF_2048_random (
            BIG_1024_58 * x,
            csprng * R,
            int n )
```

Parameters

x	FF instance, on exit x is a random number of length n BIGs with most significant bit a 1
R	an instance of a Cryptographically Secure Random Number Generator
n	size of FF in BIGs

8.25.3.24 FF_2048_randomnum()

```
void FF_2048_randomnum (
            BIG_1024_58 * x,
            BIG_1024_58 * y,
            csprng * R,
            int n )
```

Parameters

x	FF instance, on exit x is a random number $< y$

Parameters

y	FF instance, the modulus
R	an instance of a Cryptographically Secure Random Number Generator
n	size of FF in BIGs

8.25.3.25 FF_2048_rawoutput()

```
void FF_2048_rawoutput (
            BIG_1024_58 * x,
            int n )
```

Parameters

x	FF instance to be printed
n	size of FF in BIGs

8.25.3.26 FF_2048_shl()

```
void FF_2048_shl (
            BIG_1024_58 * x,
            int n )
```

Parameters

x	FF instance to be shifted left
n	size of FF in BIGs

8.25.3.27 FF_2048_shr()

```
void FF_2048_shr (
            BIG_1024_58 * x,
            int n )
```

Parameters

x	FF instance to be shifted right
n	size of FF in BIGs

8.25.3.28 FF_2048_skpow()

```
void FF_2048_skpow (
            BIG_1024_58 * r,
            BIG_1024_58 * x,
            BIG_1024_58 * e,
            BIG_1024_58 * m,
            int n )
```

Parameters

r	FF instance, on exit = x^e mod p
x	FF instance
e	FF exponent
m	FF modulus
n	size of FF in BIGs

8.25.3.29 FF_2048_skspow()

```
void FF_2048_skspow (
            BIG_1024_58 * r,
            BIG_1024_58 * x,
            BIG_1024_58 e,
            BIG_1024_58 * m,
            int n )
```

For short BIG exponent

Parameters

r	FF instance, on exit = x^e mod p
x	FF instance
e	BIG exponent
m	FF modulus
n	size of FF in BIGs

8.25.3.30 FF_2048_sqr()

```
void FF_2048_sqr (
            BIG_1024_58 * x,
            BIG_1024_58 * y,
            int n )
```

Uses Karatsuba method internally

Parameters

x	FF instance, on exit = y^2
y	FF instance to be squared
n	size of FF in BIGs

8.25.3.31 FF_2048_sub()

```
void FF_2048_sub (
            BIG_1024_58 * x,
            BIG_1024_58 * y,
            BIG_1024_58 * z,
            int n )
```

Parameters

x	FF instance, on exit = y-z
y	FF instance
z	FF instance
n	size of FF in BIGs

8.25.3.32 FF_2048_toOctet()

```
void FF_2048_toOctet (
            octet * S,
            BIG_1024_58 * x,
            int n )
```

Converts an FF to big-endian base 256 form.

Parameters

S	output octet string
x	FF instance to be converted to an octet string
n	size of FF in BIGs

8.25.3.33 FF_2048_zero()

```
void FF_2048_zero (
            BIG_1024_58 * x,
            int n )
```

Parameters

x	FF instance to be set to zero
n	size of FF in BIGs

8.26 ff_3072.h File Reference

FF Header File.

```
#include "big_384_56.h"
#include "config_ff_3072.h"
```

Macros

- #define HFLEN_3072 (FFLEN_3072/2)
- #define P_MBITS_3072 (MODBYTES_384_56∗8)
- #define P_TBITS_3072 (P_MBITS_3072%BASEBITS_384_56)
- #define P_EXCESS_3072(a) (((a[NLEN_384_56-1])>>(P_TBITS_3072))+1)
- #define P_FEXCESS_3072 ((chunk)1<<(BASEBITS_384_56∗NLEN_384_56-P_MBITS_3072-1))

Functions

- void FF_3072_copy (BIG_384_56 ∗x, BIG_384_56 ∗y, int n)

  Copy one FF element of given length to another.
- void FF_3072_init (BIG_384_56 ∗x, sign32 m, int n)

  Initialize an FF element of given length from a 32-bit integer m.
- void FF_3072_zero (BIG_384_56 ∗x, int n)

  Set FF element of given size to zero.
- int FF_3072_iszilch (BIG_384_56 ∗x, int n)

  Tests for FF element equal to zero.
- int FF_3072_parity (BIG_384_56 ∗x)

  return parity of an FF, that is the least significant bit
- int FF_3072_lastbits (BIG_384_56 ∗x, int m)

  return least significant m bits of an FF
- void FF_3072_one (BIG_384_56 ∗x, int n)

  Set FF element of given size to unity.
- int FF_3072_comp (BIG_384_56 ∗x, BIG_384_56 ∗y, int n)

  Compares two FF numbers. Inputs must be normalised externally.
- void FF_3072_add (BIG_384_56 ∗x, BIG_384_56 ∗y, BIG_384_56 ∗z, int n)

  addition of two FFs
- void FF_3072_sub (BIG_384_56 ∗x, BIG_384_56 ∗y, BIG_384_56 ∗z, int n)

  subtraction of two FFs
- void FF_3072_inc (BIG_384_56 ∗x, int m, int n)

  increment an FF by an integer,and normalise
- void FF_3072_dec (BIG_384_56 ∗x, int m, int n)

  Decrement an FF by an integer,and normalise.
- void FF_3072_norm (BIG_384_56 ∗x, int n)

Normalises the components of an FF.

- void FF_3072_shl (BIG_384_56 ∗x, int n)

  Shift left an FF by 1 bit.
- void FF_3072_shr (BIG_384_56 ∗x, int n)

  Shift right an FF by 1 bit.
- void FF_3072_output (BIG_384_56 ∗x, int n)

  Formats and outputs an FF to the console.
- void FF_3072_rawoutput (BIG_384_56 ∗x, int n)

  Formats and outputs an FF to the console, in raw form.
- void FF_3072_toOctet (octet ∗S, BIG_384_56 ∗x, int n)

  Formats and outputs an FF instance to an octet string.
- void FF_3072_fromOctet (BIG_384_56 ∗x, octet ∗S, int n)

  Populates an FF instance from an octet string.
- void FF_3072_mul (BIG_384_56 ∗x, BIG_384_56 ∗y, BIG_384_56 ∗z, int n)

  Multiplication of two FFs.
- void FF_3072_mod (BIG_384_56 ∗x, BIG_384_56 ∗m, int n)

  Reduce FF mod a modulus.
- void FF_3072_sqr (BIG_384_56 ∗x, BIG_384_56 ∗y, int n)

  Square an FF.
- void FF_3072_dmod (BIG_384_56 ∗x, BIG_384_56 ∗y, BIG_384_56 ∗z, int n)

  Reduces a double-length FF with respect to a given modulus.
- void FF_3072_invmodp (BIG_384_56 ∗x, BIG_384_56 ∗y, BIG_384_56 ∗z, int n)

  Invert an FF mod a prime modulus.
- void FF_3072_random (BIG_384_56 ∗x, csprng ∗R, int n)

  Create an FF from a random number generator.
- void FF_3072_randomnum (BIG_384_56 ∗x, BIG_384_56 ∗y, csprng ∗R, int n)

  Create a random FF less than a given modulus from a random number generator.
- void FF_3072_skpow (BIG_384_56 ∗r, BIG_384_56 ∗x, BIG_384_56 ∗e, BIG_384_56 ∗m, int n)

  Calculate r=x^e mod m, side channel resistant.
- void FF_3072_skspow (BIG_384_56 ∗r, BIG_384_56 ∗x, BIG_384_56 e, BIG_384_56 ∗m, int n)

  Calculate r=x^e mod m, side channel resistant.
- void FF_3072_power (BIG_384_56 ∗r, BIG_384_56 ∗x, int e, BIG_384_56 ∗m, int n)

  Calculate r=x^e mod m.
- void FF_3072_pow (BIG_384_56 ∗r, BIG_384_56 ∗x, BIG_384_56 ∗e, BIG_384_56 ∗m, int n)

  Calculate r=x^e mod m.
- int FF_3072_cfactor (BIG_384_56 ∗x, sign32 s, int n)

  Test if an FF has factor in common with integer s.
- int FF_3072_prime (BIG_384_56 ∗x, csprng ∗R, int n)

  Test if an FF is prime.
- void FF_3072_pow2 (BIG_384_56 ∗r, BIG_384_56 ∗x, BIG_384_56 e, BIG_384_56 ∗y, BIG_384_56 f, BI↩
G_384_56 ∗m, int n)

  Calculate r=x^e.y^f mod m.

8.26.1 Detailed Description

Author

Mike Scott

8.26.2 Macro Definition Documentation

8.26.2.1 HFLEN_3072

```
#define HFLEN_3072 (FFLEN_3072/2)
```

Useful for half-size RSA private key operations

8.26.2.2 P_EXCESS_3072

```
#define P_EXCESS_3072(
              a ) (((a[NLEN_384_56-1])>>(P_TBITS_3072))+1)
```

TODO

8.26.2.3 P_FEXCESS_3072

```
#define P_FEXCESS_3072 ((chunk)1<<(BASEBITS_384_56*NLEN_384_56-P_MBITS_3072-1))
```

TODO

8.26.2.4 P_MBITS_3072

```
#define P_MBITS_3072 (MODBYTES_384_56*8)
```

Number of bits in modulus

8.26.2.5 P_TBITS_3072

```
#define P_TBITS_3072 (P_MBITS_3072%BASEBITS_384_56)
```

TODO

8.26.3 Function Documentation

8.26.3.1 FF_3072_add()

```
void FF_3072_add (
            BIG_384_56 * x,
            BIG_384_56 * y,
            BIG_384_56 * z,
            int n )
```

Parameters

x	FF instance, on exit = y+z
y	FF instance
z	FF instance
n	size of FF in BIGs

8.26.3.2 FF_3072_cfactor()

```
int FF_3072_cfactor (
            BIG_384_56 * x,
            sign32 s,
            int n )
```

Parameters

x	FF instance to be tested
s	the supplied integer
n	size of FF in BIGs

Returns

1 if gcd(x,s)!=1, else return 0

8.26.3.3 FF_3072_comp()

```
int FF_3072_comp (
            BIG_384_56 * x,
            BIG_384_56 * y,
            int n )
```

Parameters

x	first FF number to be compared
y	second FF number to be compared
n	size of FF in BIGs

Returns

-1 is x<y, 0 if x=y, 1 if x>y

8.26.3.4 FF_3072_copy()

```
void FF_3072_copy (
            BIG_384_56 * x,
            BIG_384_56 * y,
            int n )
```

Parameters

x	FF instance to be copied to, on exit = y
y	FF instance to be copied from
n	size of FF in BIGs

8.26.3.5 FF_3072_dec()

```
void FF_3072_dec (
            BIG_384_56 * x,
            int m,
            int n )
```

Parameters

x	FF instance, on exit = x-m
m	an integer to be subtracted from x
n	size of FF in BIGs

8.26.3.6 FF_3072_dmod()

```
void FF_3072_dmod (
            BIG_384_56 * x,
            BIG_384_56 * y,
            BIG_384_56 * z,
            int n )
```

This is slow

Parameters

x	FF instance, on exit = y mod z
y	FF instance, of double length 2∗n
z	FF modulus
n	size of FF in BIGs

8.26.3.7 FF_3072_fromOctet()

```
void FF_3072_fromOctet (
            BIG_384_56 * x,
            octet * S,
            int n )
```

Creates FF from big-endian base 256 form.

Parameters

x	FF instance to be created from an octet string
S	input octet string
n	size of FF in BIGs

8.26.3.8 FF_3072_inc()

```
void FF_3072_inc (
            BIG_384_56 * x,
            int m,
            int n )
```

Parameters

x	FF instance, on exit = x+m
m	an integer to be added to x
n	size of FF in BIGs

8.26.3.9 FF_3072_init()

```
void FF_3072_init (
            BIG_384_56 * x,
            sign32 m,
            int n )
```

Parameters

x	FF instance to be copied to, on exit = m
m	integer
n	size of FF in BIGs

8.26.3.10 FF_3072_invmodp()

```
void FF_3072_invmodp (
            BIG_384_56 * x,
            BIG_384_56 * y,
            BIG_384_56 * z,
            int n )
```

Parameters

x	FF instance, on exit = 1/y mod z
y	FF instance
z	FF prime modulus
n	size of FF in BIGs

8.26.3.11 FF_3072_iszilch()

```
int FF_3072_iszilch (
            BIG_384_56 * x,
            int n )
```

Parameters

x	FF number to be tested
n	size of FF in BIGs

Returns

1 if zero, else returns 0

8.26.3.12 FF_3072_lastbits()

```
int FF_3072_lastbits (
            BIG_384_56 * x,
            int m )
```

Parameters

x	FF number
m	number of bits to return. Assumed to be less than BASEBITS.

Returns

least significant n bits as an integer

8.26.3.13 FF_3072_mod()

```
void FF_3072_mod (
            BIG_384_56 * x,
            BIG_384_56 * m,
            int n )
```

This is slow

Parameters

x	FF instance to be reduced mod m - on exit = x mod m
m	FF modulus
n	size of FF in BIGs

8.26.3.14 FF_3072_mul()

```
void FF_3072_mul (
            BIG_384_56 * x,
            BIG_384_56 * y,
            BIG_384_56 * z,
            int n )
```

Uses Karatsuba method internally

Parameters

x	FF instance, on exit = y∗z
y	FF instance
z	FF instance
n	size of FF in BIGs

8.26.3.15 FF_3072_norm()

```
void FF_3072_norm (
            BIG_384_56 * x,
            int n )
```

Parameters

x	FF instance to be normalised
n	size of FF in BIGs

8.26.3.16 FF_3072_one()

```
void FF_3072_one (
            BIG_384_56 * x,
            int n )
```

Parameters

x	FF instance to be set to unity
n	size of FF in BIGs

8.26.3.17 FF_3072_output()

```
void FF_3072_output (
            BIG_384_56 * x,
            int n )
```

Parameters

x	FF instance to be printed
n	size of FF in BIGs

8.26.3.18 FF_3072_parity()

```
int FF_3072_parity (
            BIG_384_56 * x )
```

Parameters

x	FF number

Returns

0 or 1

8.26.3.19 FF_3072_pow()

```
void FF_3072_pow (
            BIG_384_56 * r,
            BIG_384_56 * x,
            BIG_384_56 * e,
            BIG_384_56 * m,
            int n )
```

Parameters

r	FF instance, on exit = x^e mod p
x	FF instance
e	FF exponent
m	FF modulus
n	size of FF in BIGs

8.26.3.20 FF_3072_pow2()

```
void FF_3072_pow2 (
            BIG_384_56 * r,
            BIG_384_56 * x,
            BIG_384_56 e,
            BIG_384_56 * y,
            BIG_384_56 f,
            BIG_384_56 * m,
            int n )
```

Parameters

r	FF instance, on exit = x^e.y^f mod p
x	FF instance
e	BIG exponent
y	FF instance
f	BIG exponent
m	FF modulus
n	size of FF in BIGs

8.26.3.21 FF_3072_power()

```
void FF_3072_power (
            BIG_384_56 * r,
            BIG_384_56 * x,
            int e,
            BIG_384_56 * m,
            int n )
```

For very short integer exponent

Parameters

r	FF instance, on exit = x^e mod p
x	FF instance
e	integer exponent
m	FF modulus
n	size of FF in BIGs

8.26.3.22 FF_3072_prime()

```
int FF_3072_prime (
            BIG_384_56 * x,
            csprng * R,
            int n )
```

Uses Miller-Rabin Method

Parameters

x	FF instance to be tested
R	an instance of a Cryptographically Secure Random Number Generator
n	size of FF in BIGs

Returns

1 if x is (almost certainly) prime, else return 0

8.26.3.23 FF_3072_random()

```
void FF_3072_random (
            BIG_384_56 * x,
            csprng * R,
            int n )
```

Parameters

x	FF instance, on exit x is a random number of length n BIGs with most significant bit a 1
R	an instance of a Cryptographically Secure Random Number Generator
n	size of FF in BIGs

8.26.3.24 FF_3072_randomnum()

```
void FF_3072_randomnum (
            BIG_384_56 * x,
            BIG_384_56 * y,
            csprng * R,
            int n )
```

Parameters

x	FF instance, on exit x is a random number $< y$

Parameters

y	FF instance, the modulus
R	an instance of a Cryptographically Secure Random Number Generator
n	size of FF in BIGs

8.26.3.25 FF_3072_rawoutput()

```
void FF_3072_rawoutput (
            BIG_384_56 * x,
            int n )
```

Parameters

x	FF instance to be printed
n	size of FF in BIGs

8.26.3.26 FF_3072_shl()

```
void FF_3072_shl (
            BIG_384_56 * x,
            int n )
```

Parameters

x	FF instance to be shifted left
n	size of FF in BIGs

8.26.3.27 FF_3072_shr()

```
void FF_3072_shr (
            BIG_384_56 * x,
            int n )
```

Parameters

x	FF instance to be shifted right
n	size of FF in BIGs

8.26.3.28 FF_3072_skpow()

```
void FF_3072_skpow (
            BIG_384_56 * r,
            BIG_384_56 * x,
            BIG_384_56 * e,
            BIG_384_56 * m,
            int n )
```

Parameters

r	FF instance, on exit = x^e mod p
x	FF instance
e	FF exponent
m	FF modulus
n	size of FF in BIGs

8.26.3.29 FF_3072_skspow()

```
void FF_3072_skspow (
            BIG_384_56 * r,
            BIG_384_56 * x,
            BIG_384_56 e,
            BIG_384_56 * m,
            int n )
```

For short BIG exponent

Parameters

r	FF instance, on exit = x^e mod p
x	FF instance
e	BIG exponent
m	FF modulus
n	size of FF in BIGs

8.26.3.30 FF_3072_sqr()

```
void FF_3072_sqr (
            BIG_384_56 * x,
            BIG_384_56 * y,
            int n )
```

Uses Karatsuba method internally

Parameters

x	FF instance, on exit = y^2
y	FF instance to be squared
n	size of FF in BIGs

8.26.3.31 FF_3072_sub()

```
void FF_3072_sub (
            BIG_384_56 * x,
            BIG_384_56 * y,
            BIG_384_56 * z,
            int n )
```

Parameters

x	FF instance, on exit = y-z
y	FF instance
z	FF instance
n	size of FF in BIGs

8.26.3.32 FF_3072_toOctet()

```
void FF_3072_toOctet (
            octet * S,
            BIG_384_56 * x,
            int n )
```

Converts an FF to big-endian base 256 form.

Parameters

S	output octet string
x	FF instance to be converted to an octet string
n	size of FF in BIGs

8.26.3.33 FF_3072_zero()

```
void FF_3072_zero (
            BIG_384_56 * x,
            int n )
```

Parameters

x	FF instance to be set to zero
n	size of FF in BIGs

8.27 fp12_BLS381.h File Reference

FP12 Header File.

```
#include "fp4_BLS381.h"
```

Data Structures

- struct FP12_BLS381
  FP12 Structure - towered over three FP4.

Functions

- int FP12_BLS381_iszilch (FP12_BLS381 ∗x)
  Tests for FP12 equal to zero.
- int FP12_BLS381_isunity (FP12_BLS381 ∗x)
  Tests for FP12 equal to unity.
- void FP12_BLS381_copy (FP12_BLS381 ∗x, FP12_BLS381 ∗y)
  Copy FP12 to another FP12.
- void FP12_BLS381_one (FP12_BLS381 ∗x)
  Set FP12 to unity.
- void FP12_BLS381_zero (FP12_BLS381 ∗x)
  Set FP12 to zero.
- int FP12_BLS381_equals (FP12_BLS381 ∗x, FP12_BLS381 ∗y)
  Tests for equality of two FP12s.
- void FP12_BLS381_conj (FP12_BLS381 ∗x, FP12_BLS381 ∗y)
  Conjugation of FP12.
- void FP12_BLS381_from_FP4 (FP12_BLS381 ∗x, FP4_BLS381 ∗a)
  Initialise FP12 from single FP4.
- void FP12_BLS381_from_FP4s (FP12_BLS381 ∗x, FP4_BLS381 ∗a, FP4_BLS381 ∗b, FP4_BLS381 ∗c)
  Initialise FP12 from three FP4s.
- void FP12_BLS381_usqr (FP12_BLS381 ∗x, FP12_BLS381 ∗y)
  Fast Squaring of an FP12 in "unitary" form.
- void FP12_BLS381_sqr (FP12_BLS381 ∗x, FP12_BLS381 ∗y)
  Squaring an FP12.
- void FP12_BLS381_smul (FP12_BLS381 ∗x, FP12_BLS381 ∗y)
  Fast multiplication of two sparse FP12s that arises from ATE pairing line functions.
- void FP12_BLS381_ssmul (FP12_BLS381 ∗x, FP12_BLS381 ∗y)
  Fast multiplication of what may be sparse multiplicands.
- void FP12_BLS381_mul (FP12_BLS381 ∗x, FP12_BLS381 ∗y)
  Full unconditional Multiplication of two FP12s.

- void FP12_BLS381_inv (FP12_BLS381 ∗x, FP12_BLS381 ∗y)
  Inverting an FP12.
- void FP12_BLS381_pow (FP12_BLS381 ∗r, FP12_BLS381 ∗x, BIG_384_58 b)
  Raises an FP12 to the power of a BIG.
- void FP12_BLS381_pinpow (FP12_BLS381 ∗x, int i, int b)
  Raises an FP12 instance x to a small integer power, side-channel resistant.
- void FP12_BLS381_compow (FP4_BLS381 ∗c, FP12_BLS381 ∗x, BIG_384_58 e, BIG_384_58 r)
  Raises an FP12 instance x to a BIG power, compressed to FP4.
- void FP12_BLS381_pow4 (FP12_BLS381 ∗r, FP12_BLS381 ∗x, BIG_384_58 ∗b)
  Calculate x[0]^b[0].x[1]^b[1].x[2]^b[2].x[3]^b[3], side-channel resistant.
- void FP12_BLS381_frob (FP12_BLS381 ∗x, FP2_BLS381 ∗f)
  Raises an FP12 to the power of the internal modulus p, using the Frobenius.
- void FP12_BLS381_reduce (FP12_BLS381 ∗x)
  Reduces all components of possibly unreduced FP12 mod Modulus.
- void FP12_BLS381_norm (FP12_BLS381 ∗x)
  Normalises the components of an FP12.
- void FP12_BLS381_output (FP12_BLS381 ∗x)
  Formats and outputs an FP12 to the console.
- void FP12_BLS381_toOctet (octet ∗S, FP12_BLS381 ∗x)
  Formats and outputs an FP12 instance to an octet string.
- void FP12_BLS381_fromOctet (FP12_BLS381 ∗x, octet ∗S)
  Creates an FP12 instance from an octet string.
- void FP12_BLS381_trace (FP4_BLS381 ∗t, FP12_BLS381 ∗x)
  Calculate the trace of an FP12.
- void FP12_BLS381_cmove (FP12_BLS381 ∗x, FP12_BLS381 ∗y, int s)
  Conditional copy of FP12 number.

Variables

- const BIG_384_58 Fra_BLS381
- const BIG_384_58 Frb_BLS381

8.27.1 Detailed Description

Author

Mike Scott

8.27.2 Function Documentation

8.27.2.1 FP12_BLS381_cmove()

```
void FP12_BLS381_cmove (
            FP12_BLS381 * x,
            FP12_BLS381 * y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	FP12 instance, set to y if s!=0
y	another FP12 instance
s	copy only takes place if not equal to 0

8.27.2.2 FP12_BLS381_compow()

```
void FP12_BLS381_compow (
            FP4_BLS381 * c,
            FP12_BLS381 * x,
            BIG_384_58 e,
            BIG_384_58 r )
```

Parameters

c	FP4 instance, on exit = x^(e mod r) as FP4
x	FP12 input
e	BIG exponent
r	BIG group order

8.27.2.3 FP12_BLS381_conj()

```
void FP12_BLS381_conj (
            FP12_BLS381 * x,
            FP12_BLS381 * y )
```

If y=(a,b,c) (where a,b,c are its three FP4 components) on exit x=(conj(a),-conj(b),conj(c))

Parameters

x	FP12 instance, on exit = conj(y)
y	FP12 instance

8.27.2.4 FP12_BLS381_copy()

```
void FP12_BLS381_copy (
            FP12_BLS381 * x,
            FP12_BLS381 * y )
```

Parameters

x	FP12 instance, on exit = y
y	FP12 instance to be copied

8.27.2.5 FP12_BLS381_equals()

```
int FP12_BLS381_equals (
            FP12_BLS381 * x,
            FP12_BLS381 * y )
```

Parameters

x	FP12 instance to be compared
y	FP12 instance to be compared

Returns

1 if x=y, else returns 0

8.27.2.6 FP12_BLS381_frob()

```
void FP12_BLS381_frob (
            FP12_BLS381 * x,
            FP2_BLS381 * f )
```

Parameters

x	FP12 instance, on exit = x^p
f	FP2 precalculated Frobenius constant

8.27.2.7 FP12_BLS381_from_FP4()

```
void FP12_BLS381_from_FP4 (
            FP12_BLS381 * x,
            FP4_BLS381 * a )
```

Sets first FP4 component of an FP12, other components set to zero

Parameters

x	FP12 instance to be initialised
a	FP4 to form first part of FP4

8.27.2.8 FP12_BLS381_from_FP4s()

```
void FP12_BLS381_from_FP4s (
            FP12_BLS381 * x,
            FP4_BLS381 * a,
            FP4_BLS381 * b,
            FP4_BLS381 * c )
```

Parameters

x	FP12 instance to be initialised
a	FP4 to form first part of FP12
b	FP4 to form second part of FP12
c	FP4 to form third part of FP12

8.27.2.9 FP12_BLS381_fromOctet()

```
void FP12_BLS381_fromOctet (
            FP12_BLS381 * x,
            octet * S )
```

De-serializes the components of an FP12 to create an FP12 from big-endian base 256 components.

Parameters

x	FP12 instance to be created from an octet string
S	input octet string

8.27.2.10 FP12_BLS381_inv()

```
void FP12_BLS381_inv (
            FP12_BLS381 * x,
            FP12_BLS381 * y )
```

Parameters

x	FP12 instance, on exit = 1/y
y	FP12 instance

8.27.2.11 FP12_BLS381_isunity()

```
int FP12_BLS381_isunity (
            FP12_BLS381 * x )
```

Parameters

x	FP12 number to be tested

Returns

1 if unity, else returns 0

8.27.2.12 FP12_BLS381_iszilch()

```
int FP12_BLS381_iszilch (
            FP12_BLS381 * x )
```

Parameters

x	FP12 number to be tested

Returns

1 if zero, else returns 0

8.27.2.13 FP12_BLS381_mul()

```
void FP12_BLS381_mul (
            FP12_BLS381 * x,
            FP12_BLS381 * y )
```

Parameters

x	FP12 instance, on exit = x*y
y	FP12 instance, the multiplier

8.27.2.14 FP12_BLS381_norm()

```
void FP12_BLS381_norm (
            FP12_BLS381 * x )
```

Parameters

x	FP12 instance to be normalised

8.27.2.15 FP12_BLS381_one()

```
void FP12_BLS381_one (
            FP12_BLS381 * x )
```

Parameters

x	FP12 instance to be set to one

8.27.2.16 FP12_BLS381_output()

```
void FP12_BLS381_output (
            FP12_BLS381 * x )
```

Parameters

x	FP12 instance to be printed

8.27.2.17 FP12_BLS381_pinpow()

```
void FP12_BLS381_pinpow (
            FP12_BLS381 * x,
            int i,
            int b )
```

Parameters

x	FP12 instance, on exit = x^i
i	small integer exponent
b	maximum number of bits in exponent

8.27.2.18 FP12_BLS381_pow()

```
void FP12_BLS381_pow (
            FP12_BLS381 * r,
            FP12_BLS381 * x,
            BIG_384_58 b )
```

Parameters

r	FP12 instance, on exit = y^b
x	FP12 instance
b	BIG number

8.27.2.19 FP12_BLS381_pow4()

```
void FP12_BLS381_pow4 (
            FP12_BLS381 * r,
            FP12_BLS381 * x,
            BIG_384_58 * b )
```

Parameters

r	FP12 instance, on exit = x[0]^b[0].x[1]^b[1].x[2]^b[2].x[3]^b[3]
x	FP12 array with 4 FP12s
b	BIG array of 4 exponents

8.27.2.20 FP12_BLS381_reduce()

```
void FP12_BLS381_reduce (
            FP12_BLS381 * x )
```

Parameters

x	FP12 instance, on exit reduced mod Modulus

8.27.2.21 FP12_BLS381_smul()

```
void FP12_BLS381_smul (
            FP12_BLS381 * x,
            FP12_BLS381 * y )
```

Parameters

x	FP12 instance, on exit = x∗y
y	FP12 instance, of special form

8.27.2.22 FP12_BLS381_sqr()

```
void FP12_BLS381_sqr (
            FP12_BLS381 * x,
            FP12_BLS381 * y )
```

Parameters

x	FP12 instance, on exit = y^2
y	FP12 instance

8.27.2.23 FP12_BLS381_ssmul()

```
void FP12_BLS381_ssmul (
            FP12_BLS381 * x,
            FP12_BLS381 * y )
```

Parameters

x	FP12 instance, on exit = x∗y
y	FP12 instance, of special form

8.27.2.24 FP12_BLS381_toOctet()

```
void FP12_BLS381_toOctet (
            octet * S,
            FP12_BLS381 * x )
```

Serializes the components of an FP12 to big-endian base 256 form.

Parameters

S	output octet string
x	FP12 instance to be converted to an octet string

8.27.2.25 FP12_BLS381_trace()

```
void FP12_BLS381_trace (
            FP4_BLS381 * t,
            FP12_BLS381 * x )
```

Parameters

t	FP4 trace of x, on exit = tr(x)
x	FP12 instance

8.27.2.26 FP12_BLS381_usqr()

```
void FP12_BLS381_usqr (
            FP12_BLS381 * x,
            FP12_BLS381 * y )
```

Parameters

x	FP12 instance, on exit = y^2
y	FP4 instance, must be unitary

8.27.2.27 FP12_BLS381_zero()

```
void FP12_BLS381_zero (
            FP12_BLS381 * x )
```

Parameters

x	FP12 instance to be set to zero

8.27.3 Variable Documentation

8.27.3.1 Fra_BLS381

```
const BIG_384_58 Fra_BLS381
```

real part of BN curve Frobenius Constant

8.27.3.2 Frb_BLS381

```
const BIG_384_58 Frb_BLS381
```

imaginary part of BN curve Frobenius Constant

8.28 fp2_BLS381.h File Reference

FP2 Header File.

```
#include "fp_BLS381.h"
```

Data Structures

- struct FP2_BLS381

    FP2 Structure - quadratic extension field.

Functions

- int FP2_BLS381_iszilch (FP2_BLS381 ∗x)

    Tests for FP2 equal to zero.
- void FP2_BLS381_cmove (FP2_BLS381 ∗x, FP2_BLS381 ∗y, int s)

    Conditional copy of FP2 number.
- int FP2_BLS381_isunity (FP2_BLS381 ∗x)

    Tests for FP2 equal to one.
- int FP2_BLS381_equals (FP2_BLS381 ∗x, FP2_BLS381 ∗y)

    Tests for equality of two FP2s.
- void FP2_BLS381_from_FPs (FP2_BLS381 ∗x, FP_BLS381 ∗a, FP_BLS381 ∗b)

    Initialise FP2 from two FP numbers.
- void FP2_BLS381_from_BIGs (FP2_BLS381 ∗x, BIG_384_58 a, BIG_384_58 b)

    Initialise FP2 from two BIG integers.
- void FP2_BLS381_from_FP (FP2_BLS381 ∗x, FP_BLS381 ∗a)

    Initialise FP2 from single FP.
- void FP2_BLS381_from_BIG (FP2_BLS381 ∗x, BIG_384_58 a)

    Initialise FP2 from single BIG.
- void FP2_BLS381_copy (FP2_BLS381 ∗x, FP2_BLS381 ∗y)

    Copy FP2 to another FP2.
- void FP2_BLS381_zero (FP2_BLS381 ∗x)

    Set FP2 to zero.
- void FP2_BLS381_one (FP2_BLS381 ∗x)

    Set FP2 to unity.
- void FP2_BLS381_neg (FP2_BLS381 ∗x, FP2_BLS381 ∗y)

    Negation of FP2.
- void FP2_BLS381_conj (FP2_BLS381 ∗x, FP2_BLS381 ∗y)

    Conjugation of FP2.
- void FP2_BLS381_add (FP2_BLS381 ∗x, FP2_BLS381 ∗y, FP2_BLS381 ∗z)

    addition of two FP2s

- void FP2_BLS381_sub (FP2_BLS381 ∗x, FP2_BLS381 ∗y, FP2_BLS381 ∗z)
  subtraction of two FP2s
- void FP2_BLS381_pmul (FP2_BLS381 ∗x, FP2_BLS381 ∗y, FP_BLS381 ∗b)
  Multiplication of an FP2 by an FP.
- void FP2_BLS381_imul (FP2_BLS381 ∗x, FP2_BLS381 ∗y, int i)
  Multiplication of an FP2 by a small integer.
- void FP2_BLS381_sqr (FP2_BLS381 ∗x, FP2_BLS381 ∗y)
  Squaring an FP2.
- void FP2_BLS381_mul (FP2_BLS381 ∗x, FP2_BLS381 ∗y, FP2_BLS381 ∗z)
  Multiplication of two FP2s.
- void FP2_BLS381_output (FP2_BLS381 ∗x)
  Formats and outputs an FP2 to the console.
- void FP2_BLS381_rawoutput (FP2_BLS381 ∗x)
  Formats and outputs an FP2 to the console in raw form (for debugging)
- void FP2_BLS381_inv (FP2_BLS381 ∗x, FP2_BLS381 ∗y)
  Inverting an FP2.
- void FP2_BLS381_div2 (FP2_BLS381 ∗x, FP2_BLS381 ∗y)
  Divide an FP2 by 2.
- void FP2_BLS381_mul_ip (FP2_BLS381 ∗x)
  Multiply an FP2 by (1+sqrt(-1))
- void FP2_BLS381_div_ip2 (FP2_BLS381 ∗x)
  Divide an FP2 by (1+sqrt(-1))/2 -.
- void FP2_BLS381_div_ip (FP2_BLS381 ∗x)
  Divide an FP2 by (1+sqrt(-1))
- void FP2_BLS381_norm (FP2_BLS381 ∗x)
  Normalises the components of an FP2.
- void FP2_BLS381_reduce (FP2_BLS381 ∗x)
  Reduces all components of possibly unreduced FP2 mod Modulus.
- void FP2_BLS381_pow (FP2_BLS381 ∗x, FP2_BLS381 ∗y, BIG_384_58 b)
  Raises an FP2 to the power of a BIG.
- int FP2_BLS381_sqrt (FP2_BLS381 ∗x, FP2_BLS381 ∗y)
  Square root of an FP2.
- void FP2_BLS381_times_i (FP2_BLS381 ∗x)
  Multiply an FP2 by sqrt(-1)

8.28.1 Detailed Description

Author

Mike Scott

8.28.2 Function Documentation

8.28.2.1 FP2_BLS381_add()

```
void FP2_BLS381_add (
            FP2_BLS381 * x,
            FP2_BLS381 * y,
            FP2_BLS381 * z )
```

Parameters

x	FP2 instance, on exit = y+z
y	FP2 instance
z	FP2 instance

8.28.2.2 FP2_BLS381_cmove()

```
void FP2_BLS381_cmove (
            FP2_BLS381 * x,
            FP2_BLS381 * y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	FP2 instance, set to y if s!=0
y	another FP2 instance
s	copy only takes place if not equal to 0

8.28.2.3 FP2_BLS381_conj()

```
void FP2_BLS381_conj (
            FP2_BLS381 * x,
            FP2_BLS381 * y )
```

If y=(a,b) on exit x=(a,-b)

Parameters

x	FP2 instance, on exit = conj(y)
y	FP2 instance

8.28.2.4 FP2_BLS381_copy()

```
void FP2_BLS381_copy (
            FP2_BLS381 * x,
            FP2_BLS381 * y )
```

Parameters

x	FP2 instance, on exit = y
y	FP2 instance to be copied

8.28.2.5 FP2_BLS381_div2()

```
void FP2_BLS381_div2 (
            FP2_BLS381 * x,
            FP2_BLS381 * y )
```

Parameters

x	FP2 instance, on exit = y/2
y	FP2 instance

8.28.2.6 FP2_BLS381_div_ip()

```
void FP2_BLS381_div_ip (
            FP2_BLS381 * x )
```

Note that (1+sqrt(-1)) is irreducible for FP4

Parameters

x	FP2 instance, on exit = x/(1+sqrt(-1))

8.28.2.7 FP2_BLS381_div_ip2()

```
void FP2_BLS381_div_ip2 (
            FP2_BLS381 * x )
```

Note that (1+sqrt(-1)) is irreducible for FP4

Parameters

x	FP2 instance, on exit = 2x/(1+sqrt(-1))

8.28.2.8 FP2_BLS381_equals()

```
int FP2_BLS381_equals (
            FP2_BLS381 * x,
            FP2_BLS381 * y )
```

Parameters

x	FP2 instance to be compared
y	FP2 instance to be compared

Returns

1 if x=y, else returns 0

8.28.2.9 FP2_BLS381_from_BIG()

```
void FP2_BLS381_from_BIG (
            FP2_BLS381 * x,
            BIG_384_58 a )
```

Imaginary part is set to zero

Parameters

x	FP2 instance to be initialised
a	BIG to form real part of FP2

8.28.2.10 FP2_BLS381_from_BIGs()

```
void FP2_BLS381_from_BIGs (
            FP2_BLS381 * x,
            BIG_384_58 a,
            BIG_384_58 b )
```

Parameters

x	FP2 instance to be initialised
a	BIG to form real part of FP2
b	BIG to form imaginary part of FP2

8.28.2.11 FP2_BLS381_from_FP()

```
void FP2_BLS381_from_FP (
            FP2_BLS381 * x,
            FP_BLS381 * a )
```

Imaginary part is set to zero

Parameters

x	FP2 instance to be initialised
a	FP to form real part of FP2

8.28.2.12 FP2_BLS381_from_FPs()

```
void FP2_BLS381_from_FPs (
            FP2_BLS381 * x,
            FP_BLS381 * a,
            FP_BLS381 * b )
```

Parameters

x	FP2 instance to be initialised
a	FP to form real part of FP2
b	FP to form imaginary part of FP2

8.28.2.13 FP2_BLS381_imul()

```
void FP2_BLS381_imul (
            FP2_BLS381 * x,
            FP2_BLS381 * y,
            int i )
```

Parameters

x	FP2 instance, on exit = y*i
y	FP2 instance
i	an integer

8.28.2.14 FP2_BLS381_inv()

```
void FP2_BLS381_inv (
```

```
FP2_BLS381 * x,
FP2_BLS381 * y )
```

Parameters

x	FP2 instance, on exit = 1/y
y	FP2 instance

8.28.2.15 FP2_BLS381_isunity()

```
int FP2_BLS381_isunity (
            FP2_BLS381 * x )
```

Parameters

x	FP2 instance to be tested

Returns

1 if x=1, else returns 0

8.28.2.16 FP2_BLS381_iszilch()

```
int FP2_BLS381_iszilch (
            FP2_BLS381 * x )
```

Parameters

x	FP2 number to be tested

Returns

1 if zero, else returns 0

8.28.2.17 FP2_BLS381_mul()

```
void FP2_BLS381_mul (
            FP2_BLS381 * x,
            FP2_BLS381 * y,
            FP2_BLS381 * z )
```

Parameters

x	FP2 instance, on exit = y∗z
y	FP2 instance
z	FP2 instance

8.28.2.18 FP2_BLS381_mul_ip()

```
void FP2_BLS381_mul_ip (
            FP2_BLS381 * x )
```

Note that (1+sqrt(-1)) is irreducible for FP4

Parameters

x	FP2 instance, on exit = x∗(1+sqrt(-1))

8.28.2.19 FP2_BLS381_neg()

```
void FP2_BLS381_neg (
            FP2_BLS381 * x,
            FP2_BLS381 * y )
```

Parameters

x	FP2 instance, on exit = -y
y	FP2 instance

8.28.2.20 FP2_BLS381_norm()

```
void FP2_BLS381_norm (
            FP2_BLS381 * x )
```

Parameters

x	FP2 instance to be normalised

8.28.2.21 FP2_BLS381_one()

```
void FP2_BLS381_one (
            FP2_BLS381 * x )
```

Parameters

x	FP2 instance to be set to one

8.28.2.22 FP2_BLS381_output()

```
void FP2_BLS381_output (
            FP2_BLS381 * x )
```

Parameters

x	FP2 instance

8.28.2.23 FP2_BLS381_pmul()

```
void FP2_BLS381_pmul (
            FP2_BLS381 * x,
            FP2_BLS381 * y,
            FP_BLS381 * b )
```

Parameters

x	FP2 instance, on exit = y*b
y	FP2 instance
b	FP residue

8.28.2.24 FP2_BLS381_pow()

```
void FP2_BLS381_pow (
            FP2_BLS381 * x,
            FP2_BLS381 * y,
            BIG_384_58 b )
```

Parameters

x	FP2 instance, on exit = y^b
y	FP2 instance
b	BIG number

8.28.2.25 FP2_BLS381_rawoutput()

```
void FP2_BLS381_rawoutput (
            FP2_BLS381 * x )
```

Parameters

x	FP2 instance

8.28.2.26 FP2_BLS381_reduce()

```
void FP2_BLS381_reduce (
            FP2_BLS381 * x )
```

Parameters

x	FP2 instance, on exit reduced mod Modulus

8.28.2.27 FP2_BLS381_sqr()

```
void FP2_BLS381_sqr (
            FP2_BLS381 * x,
            FP2_BLS381 * y )
```

Parameters

x	FP2 instance, on exit = y^2
y	FP2 instance

8.28.2.28 FP2_BLS381_sqrt()

```
int FP2_BLS381_sqrt (
            FP2_BLS381 * x,
            FP2_BLS381 * y )
```

Parameters

x	FP2 instance, on exit = sqrt(y)
y	FP2 instance

8.28.2.29 FP2_BLS381_sub()

```
void FP2_BLS381_sub (
            FP2_BLS381 * x,
            FP2_BLS381 * y,
            FP2_BLS381 * z )
```

Parameters

x	FP2 instance, on exit = y-z
y	FP2 instance
z	FP2 instance

8.28.2.30 FP2_BLS381_times_i()

```
void FP2_BLS381_times_i (
            FP2_BLS381 * x )
```

Note that -1 is QNR

Parameters

x	FP2 instance, on exit = x∗sqrt(-1)

8.28.2.31 FP2_BLS381_zero()

```
void FP2_BLS381_zero (
            FP2_BLS381 * x )
```

Parameters

x	FP2 instance to be set to zero

8.29 fp4_BLS381.h File Reference

FP4 Header File.

```
#include "fp2_BLS381.h"
#include "config_curve_BLS381.h"
```

Data Structures

- struct FP4_BLS381

    FP4 Structure - towered over two FP2.

Functions

- int FP4_BLS381_iszilch (FP4_BLS381 ∗x)

    Tests for FP4 equal to zero.
- int FP4_BLS381_isunity (FP4_BLS381 ∗x)

    Tests for FP4 equal to unity.
- int FP4_BLS381_equals (FP4_BLS381 ∗x, FP4_BLS381 ∗y)

    Tests for equality of two FP4s.
- int FP4_BLS381_isreal (FP4_BLS381 ∗x)

    Tests for FP4 having only a real part and no imaginary part.
- void FP4_BLS381_from_FP2s (FP4_BLS381 ∗x, FP2_BLS381 ∗a, FP2_BLS381 ∗b)

    Initialise FP4 from two FP2s.
- void FP4_BLS381_from_FP2 (FP4_BLS381 ∗x, FP2_BLS381 ∗a)

    Initialise FP4 from single FP2.
- void FP4_BLS381_from_FP2H (FP4_BLS381 ∗x, FP2_BLS381 ∗a)

    Initialise FP4 from single FP2.
- void FP4_BLS381_copy (FP4_BLS381 ∗x, FP4_BLS381 ∗y)

    Copy FP4 to another FP4.
- void FP4_BLS381_zero (FP4_BLS381 ∗x)

    Set FP4 to zero.
- void FP4_BLS381_one (FP4_BLS381 ∗x)

    Set FP4 to unity.
- void FP4_BLS381_neg (FP4_BLS381 ∗x, FP4_BLS381 ∗y)

    Negation of FP4.
- void FP4_BLS381_conj (FP4_BLS381 ∗x, FP4_BLS381 ∗y)

    Conjugation of FP4.
- void FP4_BLS381_nconj (FP4_BLS381 ∗x, FP4_BLS381 ∗y)

    Negative conjugation of FP4.
- void FP4_BLS381_add (FP4_BLS381 ∗x, FP4_BLS381 ∗y, FP4_BLS381 ∗z)

    addition of two FP4s
- void FP4_BLS381_sub (FP4_BLS381 ∗x, FP4_BLS381 ∗y, FP4_BLS381 ∗z)

    subtraction of two FP4s
- void FP4_BLS381_pmul (FP4_BLS381 ∗x, FP4_BLS381 ∗y, FP2_BLS381 ∗a)

    Multiplication of an FP4 by an FP2.
- void FP4_BLS381_qmul (FP4_BLS381 ∗x, FP4_BLS381 ∗y, FP_BLS381 ∗a)

    Multiplication of an FP4 by an FP.
- void FP4_BLS381_imul (FP4_BLS381 ∗x, FP4_BLS381 ∗y, int i)

    Multiplication of an FP4 by a small integer.
- void FP4_BLS381_sqr (FP4_BLS381 ∗x, FP4_BLS381 ∗y)

    Squaring an FP4.
- void FP4_BLS381_mul (FP4_BLS381 ∗x, FP4_BLS381 ∗y, FP4_BLS381 ∗z)

    Multiplication of two FP4s.
- void FP4_BLS381_inv (FP4_BLS381 ∗x, FP4_BLS381 ∗y)

    Inverting an FP4.
- void FP4_BLS381_output (FP4_BLS381 ∗x)

Formats and outputs an FP4 to the console.
- void FP4_BLS381_rawoutput (FP4_BLS381 ∗x)

    Formats and outputs an FP4 to the console in raw form (for debugging)
- void FP4_BLS381_times_i (FP4_BLS381 ∗x)

    multiplies an FP4 instance by irreducible polynomial sqrt(1+sqrt(-1))
- void FP4_BLS381_norm (FP4_BLS381 ∗x)

    Normalises the components of an FP4.
- void FP4_BLS381_reduce (FP4_BLS381 ∗x)

    Reduces all components of possibly unreduced FP4 mod Modulus.
- void FP4_BLS381_pow (FP4_BLS381 ∗x, FP4_BLS381 ∗y, BIG_384_58 b)

    Raises an FP4 to the power of a BIG.
- void FP4_BLS381_frob (FP4_BLS381 ∗x, FP2_BLS381 ∗f)

    Raises an FP4 to the power of the internal modulus p, using the Frobenius.
- void FP4_BLS381_xtr_A (FP4_BLS381 ∗r, FP4_BLS381 ∗w, FP4_BLS381 ∗x, FP4_BLS381 ∗y, FP4_BL↩ S381 ∗z)

    Calculates the XTR addition function r=w∗x-conj(x)∗y+z.
- void FP4_BLS381_xtr_D (FP4_BLS381 ∗r, FP4_BLS381 ∗x)

    Calculates the XTR doubling function r=x^2-2∗conj(x)
- void FP4_BLS381_xtr_pow (FP4_BLS381 ∗r, FP4_BLS381 ∗x, BIG_384_58 b)

    Calculates FP4 trace of an FP12 raised to the power of a BIG number.
- void FP4_BLS381_xtr_pow2 (FP4_BLS381 ∗r, FP4_BLS381 ∗c, FP4_BLS381 ∗d, FP4_BLS381 ∗e, FP4_↩ BLS381 ∗f, BIG_384_58 a, BIG_384_58 b)

    Calculates FP4 trace of c^a.d^b, where c and d are derived from FP4 traces of FP12s.
- void FP4_BLS381_cmove (FP4_BLS381 ∗x, FP4_BLS381 ∗y, int s)

    Conditional copy of FP4 number.
- int FP4_BLS381_sqrt (FP4_BLS381 ∗r, FP4_BLS381 ∗x)

    Calculate square root of an FP4.
- void FP4_BLS381_div_i (FP4_BLS381 ∗x)

    Divide FP4 number by QNR.
- void FP4_BLS381_div_2i (FP4_BLS381 ∗x)

    Divide an FP4 by QNR/2.
- void FP4_BLS381_div2 (FP4_BLS381 ∗x, FP4_BLS381 ∗y)

    Divide an FP4 by 2.

8.29.1 Detailed Description

Author

Mike Scott

8.29.2 Function Documentation

8.29.2.1 FP4_BLS381_add()

```
void FP4_BLS381_add (
            FP4_BLS381 * x,
            FP4_BLS381 * y,
            FP4_BLS381 * z )
```

Parameters

x	FP4 instance, on exit = y+z
y	FP4 instance
z	FP4 instance

8.29.2.2 FP4_BLS381_cmove()

```
void FP4_BLS381_cmove (
            FP4_BLS381 * x,
            FP4_BLS381 * y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	FP4 instance, set to y if s!=0
y	another FP4 instance
s	copy only takes place if not equal to 0

8.29.2.3 FP4_BLS381_conj()

```
void FP4_BLS381_conj (
            FP4_BLS381 * x,
            FP4_BLS381 * y )
```

If y=(a,b) on exit x=(a,-b)

Parameters

x	FP4 instance, on exit = conj(y)
y	FP4 instance

8.29.2.4 FP4_BLS381_copy()

```
void FP4_BLS381_copy (
            FP4_BLS381 * x,
            FP4_BLS381 * y )
```

Parameters

x	FP4 instance, on exit = y
y	FP4 instance to be copied

8.29.2.5 FP4_BLS381_div2()

```
void FP4_BLS381_div2 (
            FP4_BLS381 * x,
            FP4_BLS381 * y )
```

Parameters

x	FP4 instance, on exit = y/2
y	FP4 instance

8.29.2.6 FP4_BLS381_div_2i()

```
void FP4_BLS381_div_2i (
            FP4_BLS381 * x )
```

Divide FP4 by the QNR/2

Parameters

x	FP4 instance

8.29.2.7 FP4_BLS381_div_i()

```
void FP4_BLS381_div_i (
            FP4_BLS381 * x )
```

Divide FP4 by the QNR

Parameters

x	FP4 instance

8.29.2.8 FP4_BLS381_equals()

```
int FP4_BLS381_equals (
            FP4_BLS381 * x,
            FP4_BLS381 * y )
```

Parameters

x	FP4 instance to be compared
y	FP4 instance to be compared

Returns

1 if x=y, else returns 0

8.29.2.9 FP4_BLS381_frob()

```
void FP4_BLS381_frob (
            FP4_BLS381 * x,
            FP2_BLS381 * f )
```

Parameters

x	FP4 instance, on exit = x^p
f	FP2 precalculated Frobenius constant

8.29.2.10 FP4_BLS381_from_FP2()

```
void FP4_BLS381_from_FP2 (
            FP4_BLS381 * x,
            FP2_BLS381 * a )
```

Imaginary part is set to zero

Parameters

x	FP4 instance to be initialised
a	FP2 to form real part of FP4

8.29.2.11 FP4_BLS381_from_FP2H()

```
void FP4_BLS381_from_FP2H (
```

```
             FP4_BLS381 * x,
             FP2_BLS381 * a )
```

real part is set to zero

Parameters

x	FP4 instance to be initialised
a	FP2 to form imaginary part of FP4

8.29.2.12 FP4_BLS381_from_FP2s()

```
void FP4_BLS381_from_FP2s (
             FP4_BLS381 * x,
             FP2_BLS381 * a,
             FP2_BLS381 * b )
```

Parameters

x	FP4 instance to be initialised
a	FP2 to form real part of FP4
b	FP2 to form imaginary part of FP4

8.29.2.13 FP4_BLS381_imul()

```
void FP4_BLS381_imul (
             FP4_BLS381 * x,
             FP4_BLS381 * y,
             int i )
```

Parameters

x	FP4 instance, on exit = y*i
y	FP4 instance
i	an integer

8.29.2.14 FP4_BLS381_inv()

```
void FP4_BLS381_inv (
             FP4_BLS381 * x,
             FP4_BLS381 * y )
```

Parameters

x	FP4 instance, on exit = 1/y
y	FP4 instance

8.29.2.15 FP4_BLS381_isreal()

```
int FP4_BLS381_isreal (
            FP4_BLS381 * x )
```

Parameters

x	FP4 number to be tested

Returns

1 if real, else returns 0

8.29.2.16 FP4_BLS381_isunity()

```
int FP4_BLS381_isunity (
            FP4_BLS381 * x )
```

Parameters

x	FP4 number to be tested

Returns

1 if unity, else returns 0

8.29.2.17 FP4_BLS381_iszilch()

```
int FP4_BLS381_iszilch (
            FP4_BLS381 * x )
```

Parameters

x	FP4 number to be tested

Returns

1 if zero, else returns 0

8.29.2.18 FP4_BLS381_mul()

```
void FP4_BLS381_mul (
            FP4_BLS381 * x,
            FP4_BLS381 * y,
            FP4_BLS381 * z )
```

Parameters

x	FP4 instance, on exit = y∗z
y	FP4 instance
z	FP4 instance

8.29.2.19 FP4_BLS381_nconj()

```
void FP4_BLS381_nconj (
            FP4_BLS381 * x,
            FP4_BLS381 * y )
```

If y=(a,b) on exit x=(-a,b)

Parameters

x	FP4 instance, on exit = -conj(y)
y	FP4 instance

8.29.2.20 FP4_BLS381_neg()

```
void FP4_BLS381_neg (
            FP4_BLS381 * x,
            FP4_BLS381 * y )
```

Parameters

x	FP4 instance, on exit = -y
y	FP4 instance

8.29.2.21 FP4_BLS381_norm()

```
void FP4_BLS381_norm (
            FP4_BLS381 * x )
```

Parameters

x	FP4 instance to be normalised

8.29.2.22 FP4_BLS381_one()

```
void FP4_BLS381_one (
            FP4_BLS381 * x )
```

Parameters

x	FP4 instance to be set to one

8.29.2.23 FP4_BLS381_output()

```
void FP4_BLS381_output (
            FP4_BLS381 * x )
```

Parameters

x	FP4 instance to be printed

8.29.2.24 FP4_BLS381_pmul()

```
void FP4_BLS381_pmul (
            FP4_BLS381 * x,
            FP4_BLS381 * y,
            FP2_BLS381 * a )
```

Parameters

x	FP4 instance, on exit = y*a
y	FP4 instance
a	FP2 multiplier

8.29.2.25 FP4_BLS381_pow()

```
void FP4_BLS381_pow (
            FP4_BLS381 * x,
            FP4_BLS381 * y,
            BIG_384_58 b )
```

Parameters

x	FP4 instance, on exit = y^b
y	FP4 instance
b	BIG number

8.29.2.26 FP4_BLS381_qmul()

```
void FP4_BLS381_qmul (
            FP4_BLS381 * x,
            FP4_BLS381 * y,
            FP_BLS381 * a )
```

Parameters

x	FP4 instance, on exit = y*a
y	FP4 instance
a	FP multiplier

8.29.2.27 FP4_BLS381_rawoutput()

```
void FP4_BLS381_rawoutput (
            FP4_BLS381 * x )
```

Parameters

x	FP4 instance to be printed

8.29.2.28 FP4_BLS381_reduce()

```
void FP4_BLS381_reduce (
            FP4_BLS381 * x )
```

Parameters

x	FP4 instance, on exit reduced mod Modulus

8.29.2.29 FP4_BLS381_sqr()

```
void FP4_BLS381_sqr (
            FP4_BLS381 * x,
            FP4_BLS381 * y )
```

Parameters

x	FP4 instance, on exit = y^2
y	FP4 instance

8.29.2.30 FP4_BLS381_sqrt()

```
int FP4_BLS381_sqrt (
            FP4_BLS381 * r,
            FP4_BLS381 * x )
```

Square root

Parameters

r	FP4 instance, on exit = sqrt(x)
x	FP4 instance

Returns

1 x is a QR, otherwise 0

8.29.2.31 FP4_BLS381_sub()

```
void FP4_BLS381_sub (
            FP4_BLS381 * x,
            FP4_BLS381 * y,
            FP4_BLS381 * z )
```

Parameters

x	FP4 instance, on exit = y-z
y	FP4 instance
z	FP4 instance

8.29.2.32 FP4_BLS381_times_i()

```
void FP4_BLS381_times_i (
            FP4_BLS381 * x )
```

Parameters

x	FP4 instance, on exit = sqrt(1+sqrt(-1)*x

8.29.2.33 FP4_BLS381_xtr_A()

```
void FP4_BLS381_xtr_A (
            FP4_BLS381 * r,
            FP4_BLS381 * w,
            FP4_BLS381 * x,
            FP4_BLS381 * y,
            FP4_BLS381 * z )
```

Parameters

r	FP4 instance, on exit = w*x-conj(x)*y+z
w	FP4 instance
x	FP4 instance
y	FP4 instance
z	FP4 instance

8.29.2.34 FP4_BLS381_xtr_D()

```
void FP4_BLS381_xtr_D (
            FP4_BLS381 * r,
            FP4_BLS381 * x )
```

Parameters

r	FP4 instance, on exit = x^2-2*conj(x)
x	FP4 instance

8.29.2.35 FP4_BLS381_xtr_pow()

```
void FP4_BLS381_xtr_pow (
```

```
            FP4_BLS381 * r,
            FP4_BLS381 * x,
            BIG_384_58 b )
```

XTR single exponentiation

Parameters

r	FP4 instance, on exit = trace(w^b)
x	FP4 instance, trace of an FP12 w
b	BIG number

8.29.2.36 FP4_BLS381_xtr_pow2()

```
void FP4_BLS381_xtr_pow2 (
            FP4_BLS381 * r,
            FP4_BLS381 * c,
            FP4_BLS381 * d,
            FP4_BLS381 * e,
            FP4_BLS381 * f,
            BIG_384_58 a,
            BIG_384_58 b )
```

XTR double exponentiation Assumes c=tr(x^m), d=tr(x^n), e=tr(x^(m-n)), f=tr(x^(m-2n))

Parameters

r	FP4 instance, on exit = trace(c^a.d^b)
c	FP4 instance, trace of an FP12
d	FP4 instance, trace of an FP12
e	FP4 instance, trace of an FP12
f	FP4 instance, trace of an FP12
a	BIG number
b	BIG number

8.29.2.37 FP4_BLS381_zero()

```
void FP4_BLS381_zero (
            FP4_BLS381 * x )
```

Parameters

x	FP4 instance to be set to zero

8.30 fp_25519.h File Reference

FP Header File.

```
#include "big_256_56.h"
#include "config_field_25519.h"
```

Data Structures

- struct FP_25519

  FP Structure - quadratic extension field.

Macros

- #define MODBITS_25519 MBITS_25519
- #define TBITS_25519 (MBITS_25519%BASEBITS_256_56)
- #define TMASK_25519 (((chunk)1<<TBITS_25519)-1)
- #define FEXCESS_25519 (((sign32)1<<MAXXES_25519)-1)
- #define OMASK_25519 (-((chunk)(1)<<TBITS_25519))

Functions

- int FP_25519_iszilch (FP_25519 ∗x)

  Tests for FP equal to zero mod Modulus.
- void FP_25519_zero (FP_25519 ∗x)

  Set FP to zero.
- void FP_25519_copy (FP_25519 ∗y, FP_25519 ∗x)

  Copy an FP.
- void FP_25519_rcopy (FP_25519 ∗y, const BIG_256_56 x)

  Copy from ROM to an FP.
- int FP_25519_equals (FP_25519 ∗x, FP_25519 ∗y)

  Compares two FPs.
- void FP_25519_cswap (FP_25519 ∗x, FP_25519 ∗y, int s)

  Conditional constant time swap of two FP numbers.
- void FP_25519_cmove (FP_25519 ∗x, FP_25519 ∗y, int s)

  Conditional copy of FP number.
- void FP_25519_nres (FP_25519 ∗y, BIG_256_56 x)

  Converts from BIG integer to residue form mod Modulus.
- void FP_25519_redc (BIG_256_56 x, FP_25519 ∗y)

  Converts from residue form back to BIG integer form.
- void FP_25519_one (FP_25519 ∗x)

  Sets FP to representation of unity in residue form.
- void FP_25519_mod (BIG_256_56 r, DBIG_256_56 d)

  Reduces DBIG to BIG exploiting special form of the modulus.
- void FP_25519_mul (FP_25519 ∗x, FP_25519 ∗y, FP_25519 ∗z)

  Fast Modular multiplication of two FPs, mod Modulus.
- void FP_25519_imul (FP_25519 ∗x, FP_25519 ∗y, int i)

  Fast Modular multiplication of an FP, by a small integer, mod Modulus.

- void FP_25519_sqr (FP_25519 ∗x, FP_25519 ∗y)

  Fast Modular squaring of an FP, mod Modulus.
- void FP_25519_add (FP_25519 ∗x, FP_25519 ∗y, FP_25519 ∗z)

  Modular addition of two FPs, mod Modulus.
- void FP_25519_sub (FP_25519 ∗x, FP_25519 ∗y, FP_25519 ∗z)

  Modular subtraction of two FPs, mod Modulus.
- void FP_25519_div2 (FP_25519 ∗x, FP_25519 ∗y)

  Modular division by 2 of an FP, mod Modulus.
- void FP_25519_pow (FP_25519 ∗x, FP_25519 ∗y, BIG_256_56 z)

  Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.
- void FP_25519_sqrt (FP_25519 ∗x, FP_25519 ∗y)

  Fast Modular square root of a an FP, mod Modulus.
- void FP_25519_neg (FP_25519 ∗x, FP_25519 ∗y)

  Modular negation of a an FP, mod Modulus.
- void FP_25519_output (FP_25519 ∗x)

  Outputs an FP number to the console.
- void FP_25519_rawoutput (FP_25519 ∗x)

  Outputs an FP number to the console, in raw form.
- void FP_25519_reduce (FP_25519 ∗x)

  Reduces possibly unreduced FP mod Modulus.
- void FP_25519_norm (FP_25519 ∗x)

  normalizes FP
- int FP_25519_qr (FP_25519 ∗x)

  Tests for FP a quadratic residue mod Modulus.
- void FP_25519_inv (FP_25519 ∗x, FP_25519 ∗y)

  Modular inverse of a an FP, mod Modulus.

Variables

- const BIG_256_56 Modulus_25519
- const BIG_256_56 R2modp_25519
- const chunk MConst_25519

8.30.1 Detailed Description

Author

Mike Scott

8.30.2 Macro Definition Documentation

8.30.2.1 FEXCESS_25519

```
#define FEXCESS_25519 (((sign32)1<<MAXXES_25519)-1)
```

2^(BASEBITS∗NLEN-MODBITS)-1 - normalised BIG can be multiplied by less than this before reduction

8.30.2.2 MODBITS_25519

```
#define MODBITS_25519 MBITS_25519
```

Number of bits in Modulus for selected curve

8.30.2.3 OMASK_25519

```
#define OMASK_25519 (-((chunk)(1)<<TBITS_25519))
```

for masking out overflow bits

8.30.2.4 TBITS_25519

```
#define TBITS_25519 (MBITS_25519%BASEBITS_256_56)
```

Number of active bits in top word

8.30.2.5 TMASK_25519

```
#define TMASK_25519 (((chunk)1<<TBITS_25519)-1)
```

Mask for active bits in top word

8.30.3 Function Documentation

8.30.3.1 FP_25519_add()

```
void FP_25519_add (
            FP_25519 * x,
            FP_25519 * y,
            FP_25519 * z )
```

Parameters

x	FP number, on exit the modular sum = y+z mod Modulus
y	FP number
z	FP number

8.30.3.2 FP_25519_cmove()

```
void FP_25519_cmove (
```

```
              FP_25519 * x,
              FP_25519 * y,
              int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	an FP number
y	another FP number
s	copy takes place if not equal to 0

8.30.3.3 FP_25519_copy()

```
void FP_25519_copy (
              FP_25519 * y,
              FP_25519 * x )
```

Parameters

y	FP number to be copied to
x	FP to be copied from

8.30.3.4 FP_25519_cswap()

```
void FP_25519_cswap (
              FP_25519 * x,
              FP_25519 * y,
              int s )
```

Conditionally swaps parameters in constant time (without branching)

Parameters

x	an FP number
y	another FP number
s	swap takes place if not equal to 0

8.30.3.5 FP_25519_div2()

```
void FP_25519_div2 (
              FP_25519 * x,
              FP_25519 * y )
```

Parameters

x	FP number, on exit =y/2 mod Modulus
y	FP number

8.30.3.6 FP_25519_equals()

```
int FP_25519_equals (
            FP_25519 * x,
            FP_25519 * y )
```

Parameters

x	FP number
y	FP number

Returns

1 if equal, else returns 0

8.30.3.7 FP_25519_imul()

```
void FP_25519_imul (
            FP_25519 * x,
            FP_25519 * y,
            int i )
```

Parameters

x	FP number, on exit the modular product = y∗i mod Modulus
y	FP number, the multiplicand
i	a small number, the multiplier

8.30.3.8 FP_25519_inv()

```
void FP_25519_inv (
            FP_25519 * x,
            FP_25519 * y )
```

Parameters

x	FP number, on exit = 1/y mod Modulus
y	FP number

8.30.3.9 FP_25519_iszilch()

```
int FP_25519_iszilch (
            FP_25519 * x )
```

Parameters

x	BIG number to be tested

Returns

1 if zero, else returns 0

8.30.3.10 FP_25519_mod()

```
void FP_25519_mod (
            BIG_256_56 r,
            DBIG_256_56 d )
```

This function comes in different flavours depending on the form of Modulus that is currently in use.

Parameters

r	BIG number, on exit = d mod Modulus
d	DBIG number to be reduced

8.30.3.11 FP_25519_mul()

```
void FP_25519_mul (
            FP_25519 * x,
            FP_25519 * y,
            FP_25519 * z )
```

Uses appropriate fast modular reduction method

Parameters

x	FP number, on exit the modular product = y∗z mod Modulus
y	FP number, the multiplicand
z	FP number, the multiplier

8.30.3.12 FP_25519_neg()

```
void FP_25519_neg (
            FP_25519 * x,
            FP_25519 * y )
```

Parameters

x	FP number, on exit = -y mod Modulus
y	FP number

8.30.3.13 FP_25519_norm()

```
void FP_25519_norm (
            FP_25519 * x )
```

Parameters

x	FP number, on exit normalized

8.30.3.14 FP_25519_nres()

```
void FP_25519_nres (
            FP_25519 * y,
            BIG_256_56 x )
```

Parameters

x	BIG number to be converted
y	FP result

8.30.3.15 FP_25519_one()

```
void FP_25519_one (
            FP_25519 * x )
```

Parameters

x	FP number to be set equal to unity.

8.30.3.16 FP_25519_output()

```
void FP_25519_output (
            FP_25519 * x )
```

Converts from residue form before output

Parameters

x	an FP number

8.30.3.17 FP_25519_pow()

```
void FP_25519_pow (
            FP_25519 * x,
            FP_25519 * y,
            BIG_256_56 z )
```

Parameters

x	FP number, on exit = y^z mod Modulus
y	FP number
z	BIG number exponent

8.30.3.18 FP_25519_qr()

```
int FP_25519_qr (
            FP_25519 * x )
```

Parameters

x	FP number to be tested

Returns

1 if quadratic residue, else returns 0 if quadratic non-residue

8.30.3.19 FP_25519_rawoutput()

```
void FP_25519_rawoutput (
            FP_25519 * x )
```

Parameters

x	a BIG number

8.30.3.20 FP_25519_rcopy()

```
void FP_25519_rcopy (
            FP_25519 * y,
            const BIG_256_56 x )
```

Parameters

y	FP number to be copied to
x	BIG to be copied from ROM

8.30.3.21 FP_25519_redc()

```
void FP_25519_redc (
            BIG_256_56 x,
            FP_25519 * y )
```

Parameters

y	FP number to be converted to BIG
x	BIG result

8.30.3.22 FP_25519_reduce()

```
void FP_25519_reduce (
            FP_25519 * x )
```

Parameters

x	FP number, on exit reduced mod Modulus

8.30.3.23 FP_25519_sqr()

```
void FP_25519_sqr (
            FP_25519 * x,
            FP_25519 * y )
```

Uses appropriate fast modular reduction method

Parameters

x	FP number, on exit the modular product = y^2 mod Modulus
y	FP number, the number to be squared

8.30.3.24 FP_25519_sqrt()

```
void FP_25519_sqrt (
            FP_25519 * x,
            FP_25519 * y )
```

Parameters

x	FP number, on exit = sqrt(y) mod Modulus
y	FP number, the number whose square root is calculated

8.30.3.25 FP_25519_sub()

```
void FP_25519_sub (
            FP_25519 * x,
            FP_25519 * y,
            FP_25519 * z )
```

Parameters

x	FP number, on exit the modular difference = y-z mod Modulus
y	FP number
z	FP number

8.30.3.26 FP_25519_zero()

```
void FP_25519_zero (
            FP_25519 * x )
```

Parameters

x	FP number to be set to 0

8.30.4 Variable Documentation

8.30.4.1 MConst_25519

```
const chunk MConst_25519
```

Constant associated with Modulus - for Montgomery = 1/p mod 2^BASEBITS

8.30.4.2 Modulus_25519

```
const BIG_256_56 Modulus_25519
```

Actual Modulus set in romf_yyy.c

8.30.4.3 R2modp_25519

```
const BIG_256_56 R2modp_25519
```

Montgomery constant

8.31 fp_BLS381.h File Reference

FP Header File.

```
#include "big_384_58.h"
#include "config_field_BLS381.h"
```

Data Structures

- struct FP_BLS381

    FP Structure - quadratic extension field.

Macros

- #define MODBITS_BLS381 MBITS_BLS381
- #define TBITS_BLS381 (MBITS_BLS381%BASEBITS_384_58)
- #define TMASK_BLS381 (((chunk)1<<TBITS_BLS381)-1)
- #define FEXCESS_BLS381 (((sign32)1<<MAXXES_BLS381)-1)
- #define OMASK_BLS381 (-((chunk)(1)<<TBITS_BLS381))

Functions

- int FP_BLS381_iszilch (FP_BLS381 ∗x)

  Tests for FP equal to zero mod Modulus.
- void FP_BLS381_zero (FP_BLS381 ∗x)

  Set FP to zero.
- void FP_BLS381_copy (FP_BLS381 ∗y, FP_BLS381 ∗x)

  Copy an FP.
- void FP_BLS381_rcopy (FP_BLS381 ∗y, const BIG_384_58 x)

  Copy from ROM to an FP.
- int FP_BLS381_equals (FP_BLS381 ∗x, FP_BLS381 ∗y)

  Compares two FPs.
- void FP_BLS381_cswap (FP_BLS381 ∗x, FP_BLS381 ∗y, int s)

  Conditional constant time swap of two FP numbers.
- void FP_BLS381_cmove (FP_BLS381 ∗x, FP_BLS381 ∗y, int s)

  Conditional copy of FP number.
- void FP_BLS381_nres (FP_BLS381 ∗y, BIG_384_58 x)

  Converts from BIG integer to residue form mod Modulus.
- void FP_BLS381_redc (BIG_384_58 x, FP_BLS381 ∗y)

  Converts from residue form back to BIG integer form.
- void FP_BLS381_one (FP_BLS381 ∗x)

  Sets FP to representation of unity in residue form.
- void FP_BLS381_mod (BIG_384_58 r, DBIG_384_58 d)

  Reduces DBIG to BIG exploiting special form of the modulus.
- void FP_BLS381_mul (FP_BLS381 ∗x, FP_BLS381 ∗y, FP_BLS381 ∗z)

  Fast Modular multiplication of two FPs, mod Modulus.
- void FP_BLS381_imul (FP_BLS381 ∗x, FP_BLS381 ∗y, int i)

  Fast Modular multiplication of an FP, by a small integer, mod Modulus.
- void FP_BLS381_sqr (FP_BLS381 ∗x, FP_BLS381 ∗y)

  Fast Modular squaring of an FP, mod Modulus.
- void FP_BLS381_add (FP_BLS381 ∗x, FP_BLS381 ∗y, FP_BLS381 ∗z)

  Modular addition of two FPs, mod Modulus.
- void FP_BLS381_sub (FP_BLS381 ∗x, FP_BLS381 ∗y, FP_BLS381 ∗z)

  Modular subtraction of two FPs, mod Modulus.
- void FP_BLS381_div2 (FP_BLS381 ∗x, FP_BLS381 ∗y)

  Modular division by 2 of an FP, mod Modulus.
- void FP_BLS381_pow (FP_BLS381 ∗x, FP_BLS381 ∗y, BIG_384_58 z)

  Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.
- void FP_BLS381_sqrt (FP_BLS381 ∗x, FP_BLS381 ∗y)

  Fast Modular square root of a an FP, mod Modulus.
- void FP_BLS381_neg (FP_BLS381 ∗x, FP_BLS381 ∗y)

  Modular negation of a an FP, mod Modulus.
- void FP_BLS381_output (FP_BLS381 ∗x)

Outputs an FP number to the console.

- void FP_BLS381_rawoutput (FP_BLS381 ∗x)

    Outputs an FP number to the console, in raw form.

- void FP_BLS381_reduce (FP_BLS381 ∗x)

    Reduces possibly unreduced FP mod Modulus.

- void FP_BLS381_norm (FP_BLS381 ∗x)

    normalizes FP

- int FP_BLS381_qr (FP_BLS381 ∗x)

    Tests for FP a quadratic residue mod Modulus.

- void FP_BLS381_inv (FP_BLS381 ∗x, FP_BLS381 ∗y)

    Modular inverse of a an FP, mod Modulus.

Variables

- const BIG_384_58 Modulus_BLS381
- const BIG_384_58 R2modp_BLS381
- const chunk MConst_BLS381

8.31.1 Detailed Description

Author

Mike Scott

8.31.2 Macro Definition Documentation

8.31.2.1 FEXCESS_BLS381

```
#define FEXCESS_BLS381 (((sign32)1<<MAXXES_BLS381)-1)
```

2^(BASEBITS∗NLEN-MODBITS)-1 - normalised BIG can be multiplied by less than this before reduction

8.31.2.2 MODBITS_BLS381

```
#define MODBITS_BLS381 MBITS_BLS381
```

Number of bits in Modulus for selected curve

8.31.2.3 OMASK_BLS381

```
#define OMASK_BLS381 (-((chunk)(1)<<TBITS_BLS381))
```

for masking out overflow bits

8.31.2.4 TBITS_BLS381

```
#define TBITS_BLS381 (MBITS_BLS381%BASEBITS_384_58)
```

Number of active bits in top word

8.31.2.5 TMASK_BLS381

```
#define TMASK_BLS381 (((chunk)1<<TBITS_BLS381)-1)
```

Mask for active bits in top word

8.31.3 Function Documentation

8.31.3.1 FP_BLS381_add()

```
void FP_BLS381_add (
            FP_BLS381 * x,
            FP_BLS381 * y,
            FP_BLS381 * z )
```

Parameters

x	FP number, on exit the modular sum = y+z mod Modulus
y	FP number
z	FP number

8.31.3.2 FP_BLS381_cmove()

```
void FP_BLS381_cmove (
            FP_BLS381 * x,
            FP_BLS381 * y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	an FP number
y	another FP number
s	copy takes place if not equal to 0

8.31.3.3 FP_BLS381_copy()

```
void FP_BLS381_copy (
            FP_BLS381 * y,
            FP_BLS381 * x )
```

Parameters

y	FP number to be copied to
x	FP to be copied from

8.31.3.4 FP_BLS381_cswap()

```
void FP_BLS381_cswap (
            FP_BLS381 * x,
            FP_BLS381 * y,
            int s )
```

Conditionally swaps parameters in constant time (without branching)

Parameters

x	an FP number
y	another FP number
s	swap takes place if not equal to 0

8.31.3.5 FP_BLS381_div2()

```
void FP_BLS381_div2 (
            FP_BLS381 * x,
            FP_BLS381 * y )
```

Parameters

x	FP number, on exit =y/2 mod Modulus
y	FP number

8.31.3.6 FP_BLS381_equals()

```
int FP_BLS381_equals (
            FP_BLS381 * x,
            FP_BLS381 * y )
```

Parameters

x	FP number
y	FP number

Returns

1 if equal, else returns 0

8.31.3.7 FP_BLS381_imul()

```
void FP_BLS381_imul (
            FP_BLS381 * x,
            FP_BLS381 * y,
            int i )
```

Parameters

x	FP number, on exit the modular product = y∗i mod Modulus
y	FP number, the multiplicand
i	a small number, the multiplier

8.31.3.8 FP_BLS381_inv()

```
void FP_BLS381_inv (
            FP_BLS381 * x,
            FP_BLS381 * y )
```

Parameters

x	FP number, on exit = 1/y mod Modulus
y	FP number

8.31.3.9 FP_BLS381_iszilch()

```
int FP_BLS381_iszilch (
            FP_BLS381 * x )
```

Parameters

x	BIG number to be tested

Returns

1 if zero, else returns 0

8.31.3.10 FP_BLS381_mod()

```
void FP_BLS381_mod (
            BIG_384_58 r,
            DBIG_384_58 d )
```

This function comes in different flavours depending on the form of Modulus that is currently in use.

Parameters

r	BIG number, on exit = d mod Modulus
d	DBIG number to be reduced

8.31.3.11 FP_BLS381_mul()

```
void FP_BLS381_mul (
            FP_BLS381 * x,
            FP_BLS381 * y,
            FP_BLS381 * z )
```

Uses appropriate fast modular reduction method

Parameters

x	FP number, on exit the modular product = y*z mod Modulus
y	FP number, the multiplicand
z	FP number, the multiplier

8.31.3.12 FP_BLS381_neg()

```
void FP_BLS381_neg (
            FP_BLS381 * x,
            FP_BLS381 * y )
```

Parameters

x	FP number, on exit = -y mod Modulus
y	FP number

8.31.3.13 FP_BLS381_norm()

```
void FP_BLS381_norm (
            FP_BLS381 * x )
```

Parameters

x	FP number, on exit normalized

8.31.3.14 FP_BLS381_nres()

```
void FP_BLS381_nres (
            FP_BLS381 * y,
            BIG_384_58 x )
```

Parameters

x	BIG number to be converted
y	FP result

8.31.3.15 FP_BLS381_one()

```
void FP_BLS381_one (
            FP_BLS381 * x )
```

Parameters

x	FP number to be set equal to unity.

8.31.3.16 FP_BLS381_output()

```
void FP_BLS381_output (
            FP_BLS381 * x )
```

Converts from residue form before output

Parameters

x	an FP number

8.31.3.17 FP_BLS381_pow()

```
void FP_BLS381_pow (
            FP_BLS381 * x,
            FP_BLS381 * y,
            BIG_384_58 z )
```

Parameters

x	FP number, on exit = y^z mod Modulus
y	FP number
z	BIG number exponent

8.31.3.18 FP_BLS381_qr()

```
int FP_BLS381_qr (
            FP_BLS381 * x )
```

Parameters

x	FP number to be tested

Returns

1 if quadratic residue, else returns 0 if quadratic non-residue

8.31.3.19 FP_BLS381_rawoutput()

```
void FP_BLS381_rawoutput (
            FP_BLS381 * x )
```

Parameters

x	a BIG number

8.31.3.20 FP_BLS381_rcopy()

```
void FP_BLS381_rcopy (
```

```
            FP_BLS381 * y,
            const BIG_384_58 x )
```

Parameters

y	FP number to be copied to
x	BIG to be copied from ROM

8.31.3.21 FP_BLS381_redc()

```
void FP_BLS381_redc (
            BIG_384_58 x,
            FP_BLS381 * y )
```

Parameters

y	FP number to be converted to BIG
x	BIG result

8.31.3.22 FP_BLS381_reduce()

```
void FP_BLS381_reduce (
            FP_BLS381 * x )
```

Parameters

x	FP number, on exit reduced mod Modulus

8.31.3.23 FP_BLS381_sqr()

```
void FP_BLS381_sqr (
            FP_BLS381 * x,
            FP_BLS381 * y )
```

Uses appropriate fast modular reduction method

Parameters

x	FP number, on exit the modular product = y^2 mod Modulus
y	FP number, the number to be squared

8.31.3.24 FP_BLS381_sqrt()

```
void FP_BLS381_sqrt (
            FP_BLS381 * x,
            FP_BLS381 * y )
```

Parameters

x	FP number, on exit = sqrt(y) mod Modulus
y	FP number, the number whose square root is calculated

8.31.3.25 FP_BLS381_sub()

```
void FP_BLS381_sub (
            FP_BLS381 * x,
            FP_BLS381 * y,
            FP_BLS381 * z )
```

Parameters

x	FP number, on exit the modular difference = y-z mod Modulus
y	FP number
z	FP number

8.31.3.26 FP_BLS381_zero()

```
void FP_BLS381_zero (
            FP_BLS381 * x )
```

Parameters

x	FP number to be set to 0

8.31.4 Variable Documentation

8.31.4.1 MConst_BLS381

```
const chunk MConst_BLS381
```

Constant associated with Modulus - for Montgomery = 1/p mod 2^BASEBITS

8.31.4.2 Modulus_BLS381

```
const BIG_384_58 Modulus_BLS381
```

Actual Modulus set in romf_yyy.c

8.31.4.3 R2modp_BLS381

```
const BIG_384_58 R2modp_BLS381
```

Montgomery constant

8.32 fp_GOLDILOCKS.h File Reference

FP Header File.

```
#include "big_448_58.h"
#include "config_field_GOLDILOCKS.h"
```

Data Structures

- struct FP_GOLDILOCKS

  FP Structure - quadratic extension field.

Macros

- #define MODBITS_GOLDILOCKS MBITS_GOLDILOCKS
- #define TBITS_GOLDILOCKS (MBITS_GOLDILOCKS%BASEBITS_448_58)
- #define TMASK_GOLDILOCKS (((chunk)1<<TBITS_GOLDILOCKS)-1)
- #define FEXCESS_GOLDILOCKS (((sign32)1<<MAXXES_GOLDILOCKS)-1)
- #define OMASK_GOLDILOCKS (-((chunk)(1)<<TBITS_GOLDILOCKS))

Functions

- int FP_GOLDILOCKS_iszilch (FP_GOLDILOCKS ∗x)

  Tests for FP equal to zero mod Modulus.
- void FP_GOLDILOCKS_zero (FP_GOLDILOCKS ∗x)

  Set FP to zero.
- void FP_GOLDILOCKS_copy (FP_GOLDILOCKS ∗y, FP_GOLDILOCKS ∗x)

  Copy an FP.
- void FP_GOLDILOCKS_rcopy (FP_GOLDILOCKS ∗y, const BIG_448_58 x)

  Copy from ROM to an FP.
- int FP_GOLDILOCKS_equals (FP_GOLDILOCKS ∗x, FP_GOLDILOCKS ∗y)

  Compares two FPs.
- void FP_GOLDILOCKS_cswap (FP_GOLDILOCKS ∗x, FP_GOLDILOCKS ∗y, int s)

  Conditional constant time swap of two FP numbers.

- void FP_GOLDILOCKS_cmove (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y, int s)
  Conditional copy of FP number.
- void FP_GOLDILOCKS_nres (FP_GOLDILOCKS *y, BIG_448_58 x)
  Converts from BIG integer to residue form mod Modulus.
- void FP_GOLDILOCKS_redc (BIG_448_58 x, FP_GOLDILOCKS *y)
  Converts from residue form back to BIG integer form.
- void FP_GOLDILOCKS_one (FP_GOLDILOCKS *x)
  Sets FP to representation of unity in residue form.
- void FP_GOLDILOCKS_mod (BIG_448_58 r, DBIG_448_58 d)
  Reduces DBIG to BIG exploiting special form of the modulus.
- void FP_GOLDILOCKS_mul (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y, FP_GOLDILOCKS *z)
  Fast Modular multiplication of two FPs, mod Modulus.
- void FP_GOLDILOCKS_imul (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y, int i)
  Fast Modular multiplication of an FP, by a small integer, mod Modulus.
- void FP_GOLDILOCKS_sqr (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y)
  Fast Modular squaring of an FP, mod Modulus.
- void FP_GOLDILOCKS_add (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y, FP_GOLDILOCKS *z)
  Modular addition of two FPs, mod Modulus.
- void FP_GOLDILOCKS_sub (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y, FP_GOLDILOCKS *z)
  Modular subtraction of two FPs, mod Modulus.
- void FP_GOLDILOCKS_div2 (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y)
  Modular division by 2 of an FP, mod Modulus.
- void FP_GOLDILOCKS_pow (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y, BIG_448_58 z)
  Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.
- void FP_GOLDILOCKS_sqrt (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y)
  Fast Modular square root of a an FP, mod Modulus.
- void FP_GOLDILOCKS_neg (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y)
  Modular negation of a an FP, mod Modulus.
- void FP_GOLDILOCKS_output (FP_GOLDILOCKS *x)
  Outputs an FP number to the console.
- void FP_GOLDILOCKS_rawoutput (FP_GOLDILOCKS *x)
  Outputs an FP number to the console, in raw form.
- void FP_GOLDILOCKS_reduce (FP_GOLDILOCKS *x)
  Reduces possibly unreduced FP mod Modulus.
- void FP_GOLDILOCKS_norm (FP_GOLDILOCKS *x)
  normalizes FP
- int FP_GOLDILOCKS_qr (FP_GOLDILOCKS *x)
  Tests for FP a quadratic residue mod Modulus.
- void FP_GOLDILOCKS_inv (FP_GOLDILOCKS *x, FP_GOLDILOCKS *y)
  Modular inverse of a an FP, mod Modulus.

Variables

- const BIG_448_58 Modulus_GOLDILOCKS
- const BIG_448_58 R2modp_GOLDILOCKS
- const chunk MConst_GOLDILOCKS

8.32.1 Detailed Description

Author

Mike Scott

8.32.2 Macro Definition Documentation

8.32.2.1 FEXCESS_GOLDILOCKS

```
#define FEXCESS_GOLDILOCKS (((sign32)1<<MAXXES_GOLDILOCKS)-1)
```

2^(BASEBITS*NLEN-MODBITS)-1 - normalised BIG can be multiplied by less than this before reduction

8.32.2.2 MODBITS_GOLDILOCKS

```
#define MODBITS_GOLDILOCKS MBITS_GOLDILOCKS
```

Number of bits in Modulus for selected curve

8.32.2.3 OMASK_GOLDILOCKS

```
#define OMASK_GOLDILOCKS (-((chunk)(1)<<TBITS_GOLDILOCKS))
```

for masking out overflow bits

8.32.2.4 TBITS_GOLDILOCKS

```
#define TBITS_GOLDILOCKS (MBITS_GOLDILOCKS%BASEBITS_448_58)
```

Number of active bits in top word

8.32.2.5 TMASK_GOLDILOCKS

```
#define TMASK_GOLDILOCKS (((chunk)1<<TBITS_GOLDILOCKS)-1)
```

Mask for active bits in top word

8.32.3 Function Documentation

8.32.3.1 FP_GOLDILOCKS_add()

```
void FP_GOLDILOCKS_add (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y,
            FP_GOLDILOCKS * z )
```

Parameters

x	FP number, on exit the modular sum = y+z mod Modulus
y	FP number
z	FP number

8.32.3.2 FP_GOLDILOCKS_cmove()

```
void FP_GOLDILOCKS_cmove (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	an FP number
y	another FP number
s	copy takes place if not equal to 0

8.32.3.3 FP_GOLDILOCKS_copy()

```
void FP_GOLDILOCKS_copy (
            FP_GOLDILOCKS * y,
            FP_GOLDILOCKS * x )
```

Parameters

y	FP number to be copied to
x	FP to be copied from

8.32.3.4 FP_GOLDILOCKS_cswap()

```
void FP_GOLDILOCKS_cswap (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y,
            int s )
```

Conditionally swaps parameters in constant time (without branching)

Parameters

x	an FP number
y	another FP number
s	swap takes place if not equal to 0

8.32.3.5 FP_GOLDILOCKS_div2()

```
void FP_GOLDILOCKS_div2 (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y )
```

Parameters

x	FP number, on exit =y/2 mod Modulus
y	FP number

8.32.3.6 FP_GOLDILOCKS_equals()

```
int FP_GOLDILOCKS_equals (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y )
```

Parameters

x	FP number
y	FP number

Returns

1 if equal, else returns 0

8.32.3.7 FP_GOLDILOCKS_imul()

```
void FP_GOLDILOCKS_imul (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y,
            int i )
```

Parameters

x	FP number, on exit the modular product = y*i mod Modulus
y	FP number, the multiplicand
i	a small number, the multiplier

8.32.3.8 FP_GOLDILOCKS_inv()

```
void FP_GOLDILOCKS_inv (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y )
```

Parameters

x	FP number, on exit = 1/y mod Modulus
y	FP number

8.32.3.9 FP_GOLDILOCKS_iszilch()

```
int FP_GOLDILOCKS_iszilch (
            FP_GOLDILOCKS * x )
```

Parameters

x	BIG number to be tested

Returns

1 if zero, else returns 0

8.32.3.10 FP_GOLDILOCKS_mod()

```
void FP_GOLDILOCKS_mod (
            BIG_448_58 r,
            DBIG_448_58 d )
```

This function comes in different flavours depending on the form of Modulus that is currently in use.

Parameters

r	BIG number, on exit = d mod Modulus
d	DBIG number to be reduced

8.32.3.11 FP_GOLDILOCKS_mul()

```
void FP_GOLDILOCKS_mul (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y,
            FP_GOLDILOCKS * z )
```

Uses appropriate fast modular reduction method

Parameters

x	FP number, on exit the modular product = y∗z mod Modulus
y	FP number, the multiplicand
z	FP number, the multiplier

8.32.3.12 FP_GOLDILOCKS_neg()

```
void FP_GOLDILOCKS_neg (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y )
```

Parameters

x	FP number, on exit = -y mod Modulus
y	FP number

8.32.3.13 FP_GOLDILOCKS_norm()

```
void FP_GOLDILOCKS_norm (
            FP_GOLDILOCKS * x )
```

Parameters

x	FP number, on exit normalized

8.32.3.14 FP_GOLDILOCKS_nres()

```
void FP_GOLDILOCKS_nres (
            FP_GOLDILOCKS * y,
            BIG_448_58 x )
```

Parameters

x	BIG number to be converted
y	FP result

8.32.3.15 FP_GOLDILOCKS_one()

```
void FP_GOLDILOCKS_one (
            FP_GOLDILOCKS * x )
```

Parameters

x	FP number to be set equal to unity.

8.32.3.16 FP_GOLDILOCKS_output()

```
void FP_GOLDILOCKS_output (
            FP_GOLDILOCKS * x )
```

Converts from residue form before output

Parameters

x	an FP number

8.32.3.17 FP_GOLDILOCKS_pow()

```
void FP_GOLDILOCKS_pow (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y,
            BIG_448_58 z )
```

Parameters

x	FP number, on exit = y^z mod Modulus
y	FP number
z	BIG number exponent

8.32.3.18 FP_GOLDILOCKS_qr()

```
int FP_GOLDILOCKS_qr (
            FP_GOLDILOCKS * x )
```

Parameters

x	FP number to be tested

Returns

1 if quadratic residue, else returns 0 if quadratic non-residue

8.32.3.19 FP_GOLDILOCKS_rawoutput()

```
void FP_GOLDILOCKS_rawoutput (
            FP_GOLDILOCKS * x )
```

Parameters

x	a BIG number

8.32.3.20 FP_GOLDILOCKS_rcopy()

```
void FP_GOLDILOCKS_rcopy (
            FP_GOLDILOCKS * y,
            const BIG_448_58 x )
```

Parameters

y	FP number to be copied to
x	BIG to be copied from ROM

8.32.3.21 FP_GOLDILOCKS_redc()

```
void FP_GOLDILOCKS_redc (
            BIG_448_58 x,
            FP_GOLDILOCKS * y )
```

Parameters

y	FP number to be converted to BIG
x	BIG result

8.32.3.22 FP_GOLDILOCKS_reduce()

```
void FP_GOLDILOCKS_reduce (
            FP_GOLDILOCKS * x )
```

Parameters

x	FP number, on exit reduced mod Modulus

8.32.3.23 FP_GOLDILOCKS_sqr()

```
void FP_GOLDILOCKS_sqr (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y )
```

Uses appropriate fast modular reduction method

Parameters

x	FP number, on exit the modular product = y^2 mod Modulus
y	FP number, the number to be squared

8.32.3.24 FP_GOLDILOCKS_sqrt()

```
void FP_GOLDILOCKS_sqrt (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y )
```

Parameters

x	FP number, on exit = sqrt(y) mod Modulus
y	FP number, the number whose square root is calculated

8.32.3.25 FP_GOLDILOCKS_sub()

```
void FP_GOLDILOCKS_sub (
            FP_GOLDILOCKS * x,
            FP_GOLDILOCKS * y,
            FP_GOLDILOCKS * z )
```

Parameters

x	FP number, on exit the modular difference = y-z mod Modulus
y	FP number
z	FP number

8.32.3.26 FP_GOLDILOCKS_zero()

```
void FP_GOLDILOCKS_zero (
            FP_GOLDILOCKS * x )
```

Parameters

x	FP number to be set to 0

8.32.4 Variable Documentation

8.32.4.1 MConst_GOLDILOCKS

```
const chunk MConst_GOLDILOCKS
```

Constant associated with Modulus - for Montgomery = 1/p mod 2^BASEBITS

8.32.4.2 Modulus_GOLDILOCKS

```
const BIG_448_58 Modulus_GOLDILOCKS
```

Actual Modulus set in romf_yyy.c

8.32.4.3 R2modp_GOLDILOCKS

```
const BIG_448_58 R2modp_GOLDILOCKS
```

Montgomery constant

8.33 fp_NIST256.h File Reference

FP Header File.

```
#include "big_256_56.h"
#include "config_field_NIST256.h"
```

Data Structures

- struct FP_NIST256

    FP Structure - quadratic extension field.

Macros

- #define MODBITS_NIST256 MBITS_NIST256
- #define TBITS_NIST256 (MBITS_NIST256%BASEBITS_256_56)
- #define TMASK_NIST256 (((chunk)1<<TBITS_NIST256)-1)
- #define FEXCESS_NIST256 (((sign32)1<<MAXXES_NIST256)-1)
- #define OMASK_NIST256 (-((chunk)(1)<<TBITS_NIST256))

Functions

- int FP_NIST256_iszilch (FP_NIST256 *x)

    Tests for FP equal to zero mod Modulus.
- void FP_NIST256_zero (FP_NIST256 *x)

    Set FP to zero.
- void FP_NIST256_copy (FP_NIST256 *y, FP_NIST256 *x)

    Copy an FP.
- void FP_NIST256_rcopy (FP_NIST256 *y, const BIG_256_56 x)

    Copy from ROM to an FP.
- int FP_NIST256_equals (FP_NIST256 *x, FP_NIST256 *y)

    Compares two FPs.
- void FP_NIST256_cswap (FP_NIST256 *x, FP_NIST256 *y, int s)

    Conditional constant time swap of two FP numbers.
- void FP_NIST256_cmove (FP_NIST256 *x, FP_NIST256 *y, int s)

    Conditional copy of FP number.
- void FP_NIST256_nres (FP_NIST256 *y, BIG_256_56 x)

    Converts from BIG integer to residue form mod Modulus.
- void FP_NIST256_redc (BIG_256_56 x, FP_NIST256 *y)

    Converts from residue form back to BIG integer form.
- void FP_NIST256_one (FP_NIST256 *x)

    Sets FP to representation of unity in residue form.
- void FP_NIST256_mod (BIG_256_56 r, DBIG_256_56 d)

    Reduces DBIG to BIG exploiting special form of the modulus.
- void FP_NIST256_mul (FP_NIST256 *x, FP_NIST256 *y, FP_NIST256 *z)

    Fast Modular multiplication of two FPs, mod Modulus.
- void FP_NIST256_imul (FP_NIST256 *x, FP_NIST256 *y, int i)

    Fast Modular multiplication of an FP, by a small integer, mod Modulus.
- void FP_NIST256_sqr (FP_NIST256 *x, FP_NIST256 *y)

    Fast Modular squaring of an FP, mod Modulus.
- void FP_NIST256_add (FP_NIST256 *x, FP_NIST256 *y, FP_NIST256 *z)

    Modular addition of two FPs, mod Modulus.
- void FP_NIST256_sub (FP_NIST256 *x, FP_NIST256 *y, FP_NIST256 *z)

    Modular subtraction of two FPs, mod Modulus.
- void FP_NIST256_div2 (FP_NIST256 *x, FP_NIST256 *y)

    Modular division by 2 of an FP, mod Modulus.
- void FP_NIST256_pow (FP_NIST256 *x, FP_NIST256 *y, BIG_256_56 z)

Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.
- void FP_NIST256_sqrt (FP_NIST256 ∗x, FP_NIST256 ∗y)

  Fast Modular square root of a an FP, mod Modulus.
- void FP_NIST256_neg (FP_NIST256 ∗x, FP_NIST256 ∗y)

  Modular negation of a an FP, mod Modulus.
- void FP_NIST256_output (FP_NIST256 ∗x)

  Outputs an FP number to the console.
- void FP_NIST256_rawoutput (FP_NIST256 ∗x)

  Outputs an FP number to the console, in raw form.
- void FP_NIST256_reduce (FP_NIST256 ∗x)

  Reduces possibly unreduced FP mod Modulus.
- void FP_NIST256_norm (FP_NIST256 ∗x)

  normalizes FP
- int FP_NIST256_qr (FP_NIST256 ∗x)

  Tests for FP a quadratic residue mod Modulus.
- void FP_NIST256_inv (FP_NIST256 ∗x, FP_NIST256 ∗y)

  Modular inverse of a an FP, mod Modulus.

Variables

- const BIG_256_56 Modulus_NIST256
- const BIG_256_56 R2modp_NIST256
- const chunk MConst_NIST256

8.33.1 Detailed Description

Author

Mike Scott

8.33.2 Macro Definition Documentation

8.33.2.1 FEXCESS_NIST256

```
#define FEXCESS_NIST256 (((sign32)1<<MAXXES_NIST256)-1)
```

2^(BASEBITS∗NLEN-MODBITS)-1 - normalised BIG can be multiplied by less than this before reduction

8.33.2.2 MODBITS_NIST256

```
#define MODBITS_NIST256 MBITS_NIST256
```

Number of bits in Modulus for selected curve

8.33.2.3 OMASK_NIST256

```
#define OMASK_NIST256 (-((chunk)(1)<<TBITS_NIST256))
```

for masking out overflow bits

8.33.2.4 TBITS_NIST256

```
#define TBITS_NIST256 (MBITS_NIST256%BASEBITS_256_56)
```

Number of active bits in top word

8.33.2.5 TMASK_NIST256

```
#define TMASK_NIST256 (((chunk)1<<TBITS_NIST256)-1)
```

Mask for active bits in top word

8.33.3 Function Documentation

8.33.3.1 FP_NIST256_add()

```
void FP_NIST256_add (
            FP_NIST256 * x,
            FP_NIST256 * y,
            FP_NIST256 * z )
```

Parameters

x	FP number, on exit the modular sum = y+z mod Modulus
y	FP number
z	FP number

8.33.3.2 FP_NIST256_cmove()

```
void FP_NIST256_cmove (
            FP_NIST256 * x,
            FP_NIST256 * y,
            int s )
```

Conditionally copies second parameter to the first (without branching)

Parameters

x	an FP number
y	another FP number
s	copy takes place if not equal to 0

8.33.3.3 FP_NIST256_copy()

```
void FP_NIST256_copy (
            FP_NIST256 * y,
            FP_NIST256 * x )
```

Parameters

y	FP number to be copied to
x	FP to be copied from

8.33.3.4 FP_NIST256_cswap()

```
void FP_NIST256_cswap (
            FP_NIST256 * x,
            FP_NIST256 * y,
            int s )
```

Conditionally swaps parameters in constant time (without branching)

Parameters

x	an FP number
y	another FP number
s	swap takes place if not equal to 0

8.33.3.5 FP_NIST256_div2()

```
void FP_NIST256_div2 (
            FP_NIST256 * x,
            FP_NIST256 * y )
```

Parameters

x	FP number, on exit =y/2 mod Modulus
y	FP number

8.33.3.6 FP_NIST256_equals()

```
int FP_NIST256_equals (
            FP_NIST256 * x,
            FP_NIST256 * y )
```

Parameters

x	FP number
y	FP number

Returns

1 if equal, else returns 0

8.33.3.7 FP_NIST256_imul()

```
void FP_NIST256_imul (
            FP_NIST256 * x,
            FP_NIST256 * y,
            int i )
```

Parameters

x	FP number, on exit the modular product = y∗i mod Modulus
y	FP number, the multiplicand
i	a small number, the multiplier

8.33.3.8 FP_NIST256_inv()

```
void FP_NIST256_inv (
            FP_NIST256 * x,
            FP_NIST256 * y )
```

Parameters

x	FP number, on exit = 1/y mod Modulus
y	FP number

8.33.3.9 FP_NIST256_iszilch()

```
int FP_NIST256_iszilch (
            FP_NIST256 * x )
```

Parameters

x	BIG number to be tested

Returns

1 if zero, else returns 0

8.33.3.10 FP_NIST256_mod()

```
void FP_NIST256_mod (
            BIG_256_56 r,
            DBIG_256_56 d )
```

This function comes in different flavours depending on the form of Modulus that is currently in use.

Parameters

r	BIG number, on exit = d mod Modulus
d	DBIG number to be reduced

8.33.3.11 FP_NIST256_mul()

```
void FP_NIST256_mul (
            FP_NIST256 * x,
            FP_NIST256 * y,
            FP_NIST256 * z )
```

Uses appropriate fast modular reduction method

Parameters

x	FP number, on exit the modular product = y∗z mod Modulus
y	FP number, the multiplicand
z	FP number, the multiplier

8.33.3.12 FP_NIST256_neg()

```
void FP_NIST256_neg (
            FP_NIST256 * x,
            FP_NIST256 * y )
```

Parameters

x	FP number, on exit = -y mod Modulus
y	FP number

8.33.3.13 FP_NIST256_norm()

```
void FP_NIST256_norm (
            FP_NIST256 * x )
```

Parameters

x	FP number, on exit normalized

8.33.3.14 FP_NIST256_nres()

```
void FP_NIST256_nres (
            FP_NIST256 * y,
            BIG_256_56 x )
```

Parameters

x	BIG number to be converted
y	FP result

8.33.3.15 FP_NIST256_one()

```
void FP_NIST256_one (
            FP_NIST256 * x )
```

Parameters

x	FP number to be set equal to unity.

8.33.3.16 FP_NIST256_output()

```
void FP_NIST256_output (
            FP_NIST256 * x )
```

Converts from residue form before output

Parameters

x	an FP number

8.33.3.17 FP_NIST256_pow()

```
void FP_NIST256_pow (
            FP_NIST256 * x,
            FP_NIST256 * y,
            BIG_256_56 z )
```

Parameters

x	FP number, on exit = y^z mod Modulus
y	FP number
z	BIG number exponent

8.33.3.18 FP_NIST256_qr()

```
int FP_NIST256_qr (
            FP_NIST256 * x )
```

Parameters

x	FP number to be tested

Returns

1 if quadratic residue, else returns 0 if quadratic non-residue

8.33.3.19 FP_NIST256_rawoutput()

```
void FP_NIST256_rawoutput (
            FP_NIST256 * x )
```

Parameters

x	a BIG number

8.33.3.20 FP_NIST256_rcopy()

```
void FP_NIST256_rcopy (
            FP_NIST256 * y,
            const BIG_256_56 x )
```

Parameters

y	FP number to be copied to
x	BIG to be copied from ROM

8.33.3.21 FP_NIST256_redc()

```
void FP_NIST256_redc (
            BIG_256_56 x,
            FP_NIST256 * y )
```

Parameters

y	FP number to be converted to BIG
x	BIG result

8.33.3.22 FP_NIST256_reduce()

```
void FP_NIST256_reduce (
            FP_NIST256 * x )
```

Parameters

x	FP number, on exit reduced mod Modulus

8.33.3.23 FP_NIST256_sqr()

```
void FP_NIST256_sqr (
```

```
FP_NIST256 * x,
FP_NIST256 * y )
```

Uses appropriate fast modular reduction method

Parameters

x	FP number, on exit the modular product = y^2 mod Modulus
y	FP number, the number to be squared

8.33.3.24 FP_NIST256_sqrt()

```
void FP_NIST256_sqrt (
            FP_NIST256 * x,
            FP_NIST256 * y )
```

Parameters

x	FP number, on exit = sqrt(y) mod Modulus
y	FP number, the number whose square root is calculated

8.33.3.25 FP_NIST256_sub()

```
void FP_NIST256_sub (
            FP_NIST256 * x,
            FP_NIST256 * y,
            FP_NIST256 * z )
```

Parameters

x	FP number, on exit the modular difference = y-z mod Modulus
y	FP number
z	FP number

8.33.3.26 FP_NIST256_zero()

```
void FP_NIST256_zero (
            FP_NIST256 * x )
```

Parameters

x	FP number to be set to 0

8.33.4 Variable Documentation

8.33.4.1 MConst_NIST256

```
const chunk MConst_NIST256
```

Constant associated with Modulus - for Montgomery = 1/p mod 2^BASEBITS

8.33.4.2 Modulus_NIST256

```
const BIG_256_56 Modulus_NIST256
```

Actual Modulus set in romf_yyy.c

8.33.4.3 R2modp_NIST256

```
const BIG_256_56 R2modp_NIST256
```

Montgomery constant

8.34 mpin_BLS381.h File Reference

M-Pin Header file.

```
#include "pair_BLS381.h"
#include "pbc_support.h"
```

Macros

- #define PGS_BLS381 MODBYTES_384_58
- #define PFS_BLS381 MODBYTES_384_58
- #define MPIN_OK 0
- #define MPIN_INVALID_POINT -14
- #define MPIN_BAD_PIN -19
- #define MPIN_PAS 16
- #define MAXPIN 10000
- #define PBLEN 14
- #define MESSAGE_SIZE 256
- #define M_SIZE_BLS381 (MESSAGE_SIZE+2*PFS_BLS381+1)

Functions

- void MPIN_BLS381_GET_Y (int h, int t, octet ∗O, octet ∗Y)

  Generate Y=H(s,O), where s is epoch time, O is an octet, and H(.) is a hash function.
- int MPIN_BLS381_EXTRACT_FACTOR (int h, octet ∗ID, int factor, int facbits, octet ∗CS)

  Extract a PIN number from a client secret.
- int MPIN_BLS381_RESTORE_FACTOR (int h, octet ∗ID, int factor, int facbits, octet ∗CS)

  Extract a PIN number from a client secret.
- int MPIN_BLS381_EXTRACT_PIN (int h, octet ∗ID, int pin, octet ∗CS)

  Extract a PIN number from a client secret.
- int MPIN_BLS381_CLIENT (int h, int d, octet ∗ID, csprng ∗R, octet ∗x, int pin, octet ∗T, octet ∗V, octet ∗U, octet ∗UT, octet ∗TP, octet ∗MESSAGE, int t, octet ∗y)

  Perform client side of the one-pass version of the M-Pin protocol.
- int MPIN_BLS381_CLIENT_1 (int h, int d, octet ∗ID, csprng ∗R, octet ∗x, int pin, octet ∗T, octet ∗S, octet ∗U, octet ∗UT, octet ∗TP)

  Perform first pass of the client side of the 3-pass version of the M-Pin protocol.
- int MPIN_BLS381_RANDOM_GENERATE (csprng ∗R, octet ∗S)

  Generate a random group element.
- int MPIN_BLS381_CLIENT_2 (octet ∗x, octet ∗y, octet ∗V)

  Perform second pass of the client side of the 3-pass version of the M-Pin protocol.
- int MPIN_BLS381_SERVER (int h, int d, octet ∗HID, octet ∗HTID, octet ∗y, octet ∗SS, octet ∗U, octet ∗UT, octet ∗V, octet ∗E, octet ∗F, octet ∗ID, octet ∗MESSAGE, int t, octet ∗Pa)

  Perform server side of the one-pass version of the M-Pin protocol.
- void MPIN_BLS381_SERVER_1 (int h, int d, octet ∗ID, octet ∗HID, octet ∗HTID)

  Perform first pass of the server side of the 3-pass version of the M-Pin protocol.
- int MPIN_BLS381_SERVER_2 (int d, octet ∗HID, octet ∗HTID, octet ∗y, octet ∗SS, octet ∗U, octet ∗UT, octet ∗V, octet ∗E, octet ∗F, octet ∗Pa)

  Perform third pass on the server side of the 3-pass version of the M-Pin protocol.
- int MPIN_BLS381_RECOMBINE_G1 (octet ∗Q1, octet ∗Q2, octet ∗Q)

  Add two members from the group G1.
- int MPIN_BLS381_RECOMBINE_G2 (octet ∗P1, octet ∗P2, octet ∗P)

  Add two members from the group G2.
- int MPIN_BLS381_KANGAROO (octet ∗E, octet ∗F)

  Use Kangaroos to find PIN error.
- int MPIN_BLS381_ENCODING (csprng ∗R, octet ∗TP)

  Encoding of a Time Permit to make it indistinguishable from a random string.
- int MPIN_BLS381_DECODING (octet ∗TP)

  Encoding of an obfuscated Time Permit.
- int MPIN_BLS381_GET_G1_MULTIPLE (csprng ∗R, int type, octet ∗x, octet ∗G, octet ∗W)

  Find a random multiple of a point in G1.
- int MPIN_BLS381_GET_G2_MULTIPLE (csprng ∗R, int type, octet ∗x, octet ∗G, octet ∗W)

  Find a random multiple of a point in G1.
- int MPIN_BLS381_GET_CLIENT_SECRET (octet ∗S, octet ∗ID, octet ∗CS)

  Create a client secret in G1 from a master secret and the client ID.
- int MPIN_BLS381_GET_CLIENT_PERMIT (int h, int d, octet ∗S, octet ∗ID, octet ∗TP)

  Create a Time Permit in G1 from a master secret and the client ID.
- int MPIN_BLS381_GET_SERVER_SECRET (octet ∗S, octet ∗SS)

  Create a server secret in G2 from a master secret.
- int MPIN_BLS381_PRECOMPUTE (octet ∗T, octet ∗ID, octet ∗CP, octet ∗g1, octet ∗g2)

  Precompute values for use by the client side of M-Pin Full.
- int MPIN_BLS381_SERVER_KEY (int h, octet ∗Z, octet ∗SS, octet ∗w, octet ∗p, octet ∗I, octet ∗U, octet ∗UT, octet ∗K)

Calculate Key on Server side for M-Pin Full.

- int MPIN_BLS381_CLIENT_KEY (int h, octet ∗g1, octet ∗g2, int pin, octet ∗r, octet ∗x, octet ∗p, octet ∗T, octet ∗K)

  Calculate Key on Client side for M-Pin Full.

- int MPIN_BLS381_GET_DVS_KEYPAIR (csprng ∗R, octet ∗Z, octet ∗Pa)

  Generates a random public key for the client z.Q.

8.34.1 Detailed Description

Author

Mike Scott

8.34.2 Macro Definition Documentation

8.34.2.1 M_SIZE_BLS381

```
#define M_SIZE_BLS381 (MESSAGE_SIZE+2*PFS_BLS381+1)
```

Signature message size and G1 size

8.34.2.2 MAXPIN

```
#define MAXPIN 10000
```

max PIN

8.34.2.3 MESSAGE_SIZE

```
#define MESSAGE_SIZE 256
```

Signature message size

8.34.2.4 MPIN_BAD_PIN

```
#define MPIN_BAD_PIN -19
```

Bad PIN number entered

8.34.2.5 MPIN_INVALID_POINT

```
#define MPIN_INVALID_POINT -14
```

Point is NOT on the curve

8.34.2.6 MPIN_OK

```
#define MPIN_OK 0
```

Function completed without error

8.34.2.7 MPIN_PAS

```
#define MPIN_PAS 16
```

MPIN Symmetric Key Size

8.34.2.8 PBLEN

```
#define PBLEN 14
```

max length of PIN in bits

8.34.2.9 PFS_BLS381

```
#define PFS_BLS381 MODBYTES_384_58
```

MPIN Field Size

8.34.2.10 PGS_BLS381

```
#define PGS_BLS381 MODBYTES_384_58
```

MPIN Group Size

8.34.3 Function Documentation

8.34.3.1 MPIN_BLS381_CLIENT()

```
int MPIN_BLS381_CLIENT (
            int h,
            int d,
            octet * ID,
            csprng * R,
            octet * x,
            int pin,
            octet * T,
            octet * V,
            octet * U,
            octet * UT,
            octet * TP,
            octet * MESSAGE,
            int t,
            octet * y )
```

If Time Permits are disabled, set d = 0, and UT is not generated and can be set to NULL. If Time Permits are enabled, and PIN error detection is OFF, U is not generated and can be set to NULL. If Time Permits are enabled, and PIN error detection is ON, U and UT are both generated.

Parameters

h	is the hash type
d	is input date, in days since the epoch. Set to 0 if Time permits disabled
ID	is the input client identity
R	is a pointer to a cryptographically secure random number generator
x	an output internally randomly generated if R!=NULL, otherwise must be provided as an input
pin	is the input PIN number
T	is the input M-Pin token (the client secret with PIN portion removed)
V	is output = -(x+y)(CS+TP), where CS is the reconstructed client secret, and TP is the time permit
U	is output = x.H(ID)
UT	is output = x.(H(ID)+H(d|H(ID)))
TP	is the input time permit
MESSAGE	is the message to be signed
t	is input epoch time in seconds - a timestamp
y	is output H(t|U) or H(t|UT) if Time Permits enabled

Returns

0 or an error code

8.34.3.2 MPIN_BLS381_CLIENT_1()

```
int MPIN_BLS381_CLIENT_1 (
            int h,
            int d,
            octet * ID,
            csprng * R,
            octet * x,
            int pin,
            octet * T,
            octet * S,
            octet * U,
            octet * UT,
            octet * TP )
```

If Time Permits are disabled, set d = 0, and UT is not generated and can be set to NULL. If Time Permits are enabled, and PIN error detection is OFF, U is not generated and can be set to NULL. If Time Permits are enabled, and PIN error detection is ON, U and UT are both generated.

Parameters

h	is the hash type
d	is input date, in days since the epoch. Set to 0 if Time permits disabled
ID	is the input client identity
R	is a pointer to a cryptographically secure random number generator
x	an output internally randomly generated if R!=NULL, otherwise must be provided as an input
pin	is the input PIN number
T	is the input M-Pin token (the client secret with PIN portion removed)

Parameters

S	is output = CS+TP, where CS=is the reconstructed client secret, and TP is the time permit
U	is output = x.H(ID)
UT	is output = x.(H(ID)+H(d|H(ID)))
TP	is the input time permit

Returns

0 or an error code

8.34.3.3 MPIN_BLS381_CLIENT_2()

```
int MPIN_BLS381_CLIENT_2 (
            octet * x,
            octet * y,
            octet * V )
```

Parameters

x	an input, a locally generated random number
y	an input random challenge from the server
V	on output = -(x+y).V

Returns

0 or an error code

8.34.3.4 MPIN_BLS381_CLIENT_KEY()

```
int MPIN_BLS381_CLIENT_KEY (
            int h,
            octet * g1,
            octet * g2,
            int pin,
            octet * r,
            octet * x,
            octet * p,
            octet * T,
            octet * K )
```

Parameters

h	is the hash type
g1	precomputed input

Parameters

g2	precomputed input
pin	is the input PIN number
r	is an input, a locally generated random number
x	is an input, a locally generated random number
p	is an input, hash of the protocol transcript
T	is the input Server-side Diffie-Hellman component
K	is the output calculated shared key

Returns

0 or an error code

8.34.3.5 MPIN_BLS381_DECODING()

```
int MPIN_BLS381_DECODING (
            octet * TP )
```

Parameters

TP	is the input obfuscated time permit, restored on output

Returns

0 or an error code

8.34.3.6 MPIN_BLS381_ENCODING()

```
int MPIN_BLS381_ENCODING (
            csprng * R,
            octet * TP )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
TP	is the input time permit, obfuscated on output

Returns

0 or an error code

8.34.3.7 MPIN_BLS381_EXTRACT_FACTOR()

```
int MPIN_BLS381_EXTRACT_FACTOR (
            int h,
            octet * ID,
            int factor,
            int facbits,
            octet * CS )
```

Parameters

h	is the hash type
ID	is the input client identity
factor	is an input factor
facbits	is the number of bits in the factor
CS	is the client secret from which the factor is to be extracted

Returns

0 or an error code

8.34.3.8 MPIN_BLS381_EXTRACT_PIN()

```
int MPIN_BLS381_EXTRACT_PIN (
            int h,
            octet * ID,
            int pin,
            octet * CS )
```

Parameters

h	is the hash type
ID	is the input client identity
pin	is an input PIN number
CS	is the client secret from which the PIN is to be extracted

Returns

0 or an error code

8.34.3.9 MPIN_BLS381_GET_CLIENT_PERMIT()

```
int MPIN_BLS381_GET_CLIENT_PERMIT (
            int h,
            int d,
```

```
octet * S,
octet * ID,
octet * TP )
```

Parameters

h	is the hash type
d	is input date, in days since the epoch.
S	is an input master secret
ID	is the input client identity
TP	is a Time Permit for the given date = s.H(d|H(ID))

Returns

0 or an error code

8.34.3.10 MPIN_BLS381_GET_CLIENT_SECRET()

```
int MPIN_BLS381_GET_CLIENT_SECRET (
            octet * S,
            octet * ID,
            octet * CS )
```

Parameters

S	is an input master secret
ID	is the input client identity
CS	is the full client secret = s.H(ID)

Returns

0 or an error code

8.34.3.11 MPIN_BLS381_GET_DVS_KEYPAIR()

```
int MPIN_BLS381_GET_DVS_KEYPAIR (
            csprng * R,
            octet * Z,
            octet * Pa )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
Z	an output internally randomly generated if R!=NULL, otherwise it must be provided as an input
Pa	the output public key for the client

8.34.3.12 MPIN_BLS381_GET_G1_MULTIPLE()

```
int MPIN_BLS381_GET_G1_MULTIPLE (
            csprng * R,
            int type,
            octet * x,
            octet * G,
            octet * W )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
type	determines type of action to be taken
x	an output internally randomly generated if R!=NULL, otherwise must be provided as an input
G	if type=0 a point in G1, else an octet to be mapped to G1
W	the output =x.G or x.M(G), where M(.) is a mapping

Returns

0 or an error code

8.34.3.13 MPIN_BLS381_GET_G2_MULTIPLE()

```
int MPIN_BLS381_GET_G2_MULTIPLE (
            csprng * R,
            int type,
            octet * x,
            octet * G,
            octet * W )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
type	determines type of action to betaken
x	an output internally randomly generated if R!=NULL, otherwise must be provided as an input
G	a point in G2
W	the output =x.G or (1/x).G

Returns

0 or an error code

8.34.3.14 MPIN_BLS381_GET_SERVER_SECRET()

```
int MPIN_BLS381_GET_SERVER_SECRET (
            octet * S,
            octet * SS )
```

Parameters

S	is an input master secret
SS	is the server secret = s.Q where Q is a fixed generator of G2

Returns

0 or an error code

8.34.3.15 MPIN_BLS381_GET_Y()

```
void MPIN_BLS381_GET_Y (
            int h,
            int t,
            octet * O,
            octet * Y )
```

Parameters

h	is the hash type
t	is epoch time in seconds
O	is an input octet
Y	is the output octet

8.34.3.16 MPIN_BLS381_KANGAROO()

```
int MPIN_BLS381_KANGAROO (
            octet * E,
            octet * F )
```

Parameters

E	a member of the group GT
F	a member of the group GT = E^e

Returns

0 if Kangaroos failed, or the PIN error e

8.34.3.17 MPIN_BLS381_PRECOMPUTE()

```
int MPIN_BLS381_PRECOMPUTE (
            octet * T,
```

```
octet * ID,
octet * CP,
octet * g1,
octet * g2 )
```

Parameters

T	is the input M-Pin token (the client secret with PIN portion removed)
ID	is the input client identity
CP	is Public Key (or NULL)
g1	precomputed output
g2	precomputed output

Returns

0 or an error code

8.34.3.18 MPIN_BLS381_RANDOM_GENERATE()

```
int MPIN_BLS381_RANDOM_GENERATE (
            csprng * R,
            octet * S )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
S	is the output random octet

Returns

0 or an error code

8.34.3.19 MPIN_BLS381_RECOMBINE_G1()

```
int MPIN_BLS381_RECOMBINE_G1 (
            octet * Q1,
            octet * Q2,
            octet * Q )
```

Parameters

Q1	an input member of G1
Q2	an input member of G1
Q	an output member of G1 = Q1+Q2

Returns

0 or an error code

8.34.3.20 MPIN_BLS381_RECOMBINE_G2()

```
int MPIN_BLS381_RECOMBINE_G2 (
            octet * P1,
            octet * P2,
            octet * P )
```

Parameters

P1	an input member of G2
P2	an input member of G2
P	an output member of G2 = P1+P2

Returns

0 or an error code

8.34.3.21 MPIN_BLS381_RESTORE_FACTOR()

```
int MPIN_BLS381_RESTORE_FACTOR (
            int h,
            octet * ID,
            int factor,
            int facbits,
            octet * CS )
```

Parameters

h	is the hash type
ID	is the input client identity
factor	is an input factor
facbits	is the number of bits in the factor
CS	is the client secret to which the factor is to be added

Returns

0 or an error code

8.34.3.22 MPIN_BLS381_SERVER()

```
int MPIN_BLS381_SERVER (
            int h,
            int d,
            octet * HID,
            octet * HTID,
            octet * y,
            octet * SS,
            octet * U,
            octet * UT,
            octet * V,
            octet * E,
            octet * F,
            octet * ID,
            octet * MESSAGE,
            int t,
            octet * Pa )
```

If Time Permits are disabled, set d = 0, and UT and HTID are not generated and can be set to NULL. If Time Permits are enabled, and PIN error detection is OFF, U and HID are not needed and can be set to NULL. If Time Permits are enabled, and PIN error detection is ON, U, UT, HID and HTID are all required.

Parameters

h	is the hash type
d	is input date, in days since the epoch. Set to 0 if Time permits disabled
HID	is output H(ID), a hash of the client ID
HTID	is output H(ID)+H(d|H(ID))
y	is output H(t|U) or H(t|UT) if Time Permits enabled
SS	is the input server secret
U	is input from the client = x.H(ID)
UT	is input from the client= x.(H(ID)+H(d|H(ID)))
V	is an input from the client
E	is an output to help the Kangaroos to find the PIN error, or NULL if not required
F	is an output to help the Kangaroos to find the PIN error, or NULL if not required
ID	is the input claimed client identity
MESSAGE	is the message to be signed
t	is input epoch time in seconds - a timestamp
Pa	is input from the client z.Q or NULL if the key-escrow less scheme is not used

Returns

0 or an error code

8.34.3.23 MPIN_BLS381_SERVER_1()

```
void MPIN_BLS381_SERVER_1 (
            int h,
```

```
int d,
octet * ID,
octet * HID,
octet * HTID )
```

Parameters

h	is the hash type
d	is input date, in days since the epoch. Set to 0 if Time permits disabled
ID	is the input claimed client identity
HID	is output H(ID), a hash of the client ID
HTID	is output H(ID)+H(d|H(ID))

Returns

0 or an error code

8.34.3.24 MPIN_BLS381_SERVER_2()

```
int MPIN_BLS381_SERVER_2 (
        int d,
        octet * HID,
        octet * HTID,
        octet * y,
        octet * SS,
        octet * U,
        octet * UT,
        octet * V,
        octet * E,
        octet * F,
        octet * Pa )
```

If Time Permits are disabled, set d = 0, and UT and HTID are not needed and can be set to NULL. If Time Permits are enabled, and PIN error detection is OFF, U and HID are not needed and can be set to NULL. If Time Permits are enabled, and PIN error detection is ON, U, UT, HID and HTID are all required.

Parameters

d	is input date, in days since the epoch. Set to 0 if Time permits disabled
HID	is input H(ID), a hash of the client ID
HTID	is input H(ID)+H(d|H(ID))
y	is the input server's randomly generated challenge
SS	is the input server secret
U	is input from the client = x.H(ID)
UT	is input from the client= x.(H(ID)+H(d|H(ID)))
V	is an input from the client
E	is an output to help the Kangaroos to find the PIN error, or NULL if not required
F	is an output to help the Kangaroos to find the PIN error, or NULL if not required
Pa	is the input public key from the client, z.Q or NULL if the client uses regular mpin

Returns

0 or an error code

8.34.3.25 MPIN_BLS381_SERVER_KEY()

```
int MPIN_BLS381_SERVER_KEY (
            int h,
            octet * Z,
            octet * SS,
            octet * w,
            octet * p,
            octet * I,
            octet * U,
            octet * UT,
            octet * K )
```

Uses UT internally for the key calculation, unless not available in which case U is used

Parameters

h	is the hash type
Z	is the input Client-side Diffie-Hellman component
SS	is the input server secret
w	is an input random number generated by the server
p	is an input, hash of the protocol transcript
I	is the hashed input client ID = H(ID)
U	is input from the client = x.H(ID)
UT	is input from the client= x.(H(ID)+H(d|H(ID)))
K	is the output calculated shared key

Returns

0 or an error code

8.35 pair_BLS381.h File Reference

PAIR Header File.

```
#include "fp12_BLS381.h"
#include "ecp2_BLS381.h"
#include "ecp_BLS381.h"
```

Functions

- void PAIR_BLS381_another (FP12_BLS381 r[ ], ECP2_BLS381 ∗PV, ECP_BLS381 ∗QV)

    Precompute line functions for n-pairing.
- void PAIR_BLS381_ate (FP12_BLS381 ∗r, ECP2_BLS381 ∗P, ECP_BLS381 ∗Q)

    Calculate Miller loop for Optimal ATE pairing e(P,Q)
- void PAIR_BLS381_double_ate (FP12_BLS381 ∗r, ECP2_BLS381 ∗P, ECP_BLS381 ∗Q, ECP2_BLS381 ∗R, ECP_BLS381 ∗S)

    Calculate Miller loop for Optimal ATE double-pairing e(P,Q).e(R,S)
- void PAIR_BLS381_fexp (FP12_BLS381 ∗x)

    Final exponentiation of pairing, converts output of Miller loop to element in GT.
- void PAIR_BLS381_G1mul (ECP_BLS381 ∗Q, BIG_384_58 b)

    Fast point multiplication of a member of the group G1 by a BIG number.
- void PAIR_BLS381_G2mul (ECP2_BLS381 ∗P, BIG_384_58 b)

    Fast point multiplication of a member of the group G2 by a BIG number.
- void PAIR_BLS381_GTpow (FP12_BLS381 ∗x, BIG_384_58 b)

    Fast raising of a member of GT to a BIG power.
- int PAIR_BLS381_GTmember (FP12_BLS381 ∗x)

    Tests FP12 for membership of GT.
- int PAIR_BLS381_nbits (BIG_384_58 n3, BIG_384_58 n)

    Prepare Ate parameter.
- void PAIR_BLS381_initmp (FP12_BLS381 r[ ])

    Initialise structure for multi-pairing.
- void PAIR_BLS381_miller (FP12_BLS381 ∗res, FP12_BLS381 r[ ])

    Miller loop.

Variables

- const BIG_384_58 CURVE_Bnx_BLS381
- const BIG_384_58 CURVE_Cru_BLS381
- const BIG_384_58 CURVE_W_BLS381 [2]
- const BIG_384_58 CURVE_SB_BLS381 [2][2]
- const BIG_384_58 CURVE_WB_BLS381 [4]
- const BIG_384_58 CURVE_BB_BLS381 [4][4]

8.35.1 Detailed Description

Author

Mike Scott

8.35.2 Function Documentation

8.35.2.1 PAIR_BLS381_another()

```
void PAIR_BLS381_another (
            FP12_BLS381 r[ ],
            ECP2_BLS381 * PV,
            ECP_BLS381 * QV )
```

Parameters

r	array of precomputed FP12 products of line functions
PV	ECP2 instance, an element of G2
QV	ECP instance, an element of G1

8.35.2.2 PAIR_BLS381_ate()

```
void PAIR_BLS381_ate (
            FP12_BLS381 * r,
            ECP2_BLS381 * P,
            ECP_BLS381 * Q )
```

Parameters

r	FP12 result of the pairing calculation e(P,Q)
P	ECP2 instance, an element of G2
Q	ECP instance, an element of G1

8.35.2.3 PAIR_BLS381_double_ate()

```
void PAIR_BLS381_double_ate (
            FP12_BLS381 * r,
            ECP2_BLS381 * P,
            ECP_BLS381 * Q,
            ECP2_BLS381 * R,
            ECP_BLS381 * S )
```

Faster than calculating two separate pairings

Parameters

r	FP12 result of the pairing calculation e(P,Q).e(R,S), an element of GT
P	ECP2 instance, an element of G2
Q	ECP instance, an element of G1
R	ECP2 instance, an element of G2
S	ECP instance, an element of G1

8.35.2.4 PAIR_BLS381_fexp()

```
void PAIR_BLS381_fexp (
            FP12_BLS381 * x )
```

Here p is the internal modulus, and r is the group order

Parameters

x	FP12, on exit = x^((p^12-1)/r)

8.35.2.5 PAIR_BLS381_G1mul()

```
void PAIR_BLS381_G1mul (
            ECP_BLS381 * Q,
            BIG_384_58 b )
```

May exploit endomorphism for speed.

Parameters

Q	ECP member of G1.
b	BIG multiplier

8.35.2.6 PAIR_BLS381_G2mul()

```
void PAIR_BLS381_G2mul (
            ECP2_BLS381 * P,
            BIG_384_58 b )
```

May exploit endomorphism for speed.

Parameters

P	ECP2 member of G1.
b	BIG multiplier

8.35.2.7 PAIR_BLS381_GTmember()

```
int PAIR_BLS381_GTmember (
            FP12_BLS381 * x )
```

Parameters

x	FP12 instance

Returns

1 if x is in GT, else return 0

8.35.2.8 PAIR_BLS381_GTpow()

```
void PAIR_BLS381_GTpow (
            FP12_BLS381 * x,
            BIG_384_58 b )
```

May exploit endomorphism for speed.

Parameters

x	FP12 member of GT.
b	BIG exponent

8.35.2.9 PAIR_BLS381_initmp()

```
void PAIR_BLS381_initmp (
            FP12_BLS381 r[] )
```

Parameters

r	FP12 array, to be initialised to 1

8.35.2.10 PAIR_BLS381_miller()

```
void PAIR_BLS381_miller (
            FP12_BLS381 * res,
            FP12_BLS381 r[] )
```

Parameters

res	FP12 result
r	FP12 precomputed array of accumulated line functions

8.35.2.11 PAIR_BLS381_nbits()

```
int PAIR_BLS381_nbits (
```

```
BIG_384_58 n3,
BIG_384_58 n )
```

Parameters

n	BIG parameter
n3	BIG paramter = 3∗n

Returns

number of nits in n3

8.35.3 Variable Documentation

8.35.3.1 CURVE_BB_BLS381

```
const BIG_384_58 CURVE_BB_BLS381[4][4]
```

BN curve constant for GS decomposition

8.35.3.2 CURVE_Bnx_BLS381

```
const BIG_384_58 CURVE_Bnx_BLS381
```

BN curve x parameter

8.35.3.3 CURVE_Cru_BLS381

```
const BIG_384_58 CURVE_Cru_BLS381
```

BN curve Cube Root of Unity

8.35.3.4 CURVE_SB_BLS381

```
const BIG_384_58 CURVE_SB_BLS381[2][2]
```

BN curve constant for GLV decomposition

8.35.3.5 CURVE_W_BLS381

```
const BIG_384_58 CURVE_W_BLS381[2]
```

BN curve constant for GLV decomposition

8.35.3.6 CURVE_WB_BLS381

```
const BIG_384_58 CURVE_WB_BLS381[4]
```

BN curve constant for GS decomposition

8.36 pbc_support.h File Reference

Auxiliary functions for Pairing-based protocols.

```
#include "amcl.h"
```

Macros

- #define TIME_SLOT_MINUTES 1440

Functions

- void mhashit (int sha, int n, octet ∗x, octet ∗w)
  general purpose hash function w=hash(n|x)
- unsign32 today (void)
  Supply today's date as days from the epoch.
- void HASH_ALL (int h, octet ∗I, octet ∗U, octet ∗CU, octet ∗Y, octet ∗V, octet ∗R, octet ∗W, octet ∗H)
  Hash the session transcript.
- void HASH_ID (int h, octet ∗ID, octet ∗HID)
  Hash an M-Pin Identity to an octet string.
- unsign32 GET_TIME (void)
  Get epoch time as unsigned integer.
- void AES_GCM_ENCRYPT (octet ∗K, octet ∗IV, octet ∗H, octet ∗P, octet ∗C, octet ∗T)
  AES-GCM Encryption.
- void AES_GCM_DECRYPT (octet ∗K, octet ∗IV, octet ∗H, octet ∗C, octet ∗P, octet ∗T)
  AES-GCM Decryption.

8.36.1 Detailed Description

Author

Mike Scott

8.36.2 Macro Definition Documentation

8.36.2.1 TIME_SLOT_MINUTES

```
#define TIME_SLOT_MINUTES 1440
```

Time Slot = 1 day

8.36.3 Function Documentation

8.36.3.1 AES_GCM_DECRYPT()

```
void AES_GCM_DECRYPT (
            octet * K,
            octet * IV,
            octet * H,
            octet * C,
            octet * P,
            octet * T )
```

Parameters

K	AES key
IV	Initialization vector
H	Header
P	Plaintext
C	Ciphertext
T	Checksum

8.36.3.2 AES_GCM_ENCRYPT()

```
void AES_GCM_ENCRYPT (
            octet * K,
            octet * IV,
            octet * H,
            octet * P,
            octet * C,
            octet * T )
```

Parameters

K	AES key
IV	Initialization vector
H	Header
P	Plaintext
C	Ciphertext
T	Checksum

8.36.3.3 GET_TIME()

```
unsign32 GET_TIME (
            void )
```

Returns

current epoch time in seconds

8.36.3.4 HASH_ALL()

```
void HASH_ALL (
            int h,
            octet * I,
            octet * U,
            octet * CU,
            octet * Y,
            octet * V,
            octet * R,
            octet * W,
            octet * H )
```

Parameters

h	is the hash type
I	is the hashed input client ID = H(ID)
U	is the client output = x.H(ID)
CU	is the client output = x.(H(ID)+H(T|H(ID)))
Y	is the server challenge
V	is the client part response
R	is the client part response
W	is the server part response
H	the output is the hash of all of the above that apply

8.36.3.5 HASH_ID()

```
void HASH_ID (
            int h,
            octet * ID,
            octet * HID )
```

Parameters

h	is the hash type
ID	an octet containing the identity
HID	an octet containing the hashed identity

8.36.3.6 mhashit()

```
void mhashit (
            int sha,
            int n,
            octet * x,
            octet * w )
```

Parameters

sha	is the hash type
n	integer involved in the hash
x	octect involved in the h ash
w	output

8.36.3.7 today()

```
unsign32 today (
            void )
```

Returns

today's date, as number of days elapsed since the epoch

8.37 randapi.h File Reference

PRNG API File.

```
#include "amcl.h"
```

Functions

- void CREATE_CSPRNG (csprng ∗R, octet ∗S)

  Initialise a random number generator.
- void KILL_CSPRNG (csprng ∗R)

  Kill a random number generator.

8.37.1 Detailed Description

Author

Mike Scott

8.37.2 Function Documentation

8.37.2.1 CREATE_CSPRNG()

```
void CREATE_CSPRNG (
            csprng * R,
            octet * S )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
S	is an input truly random seed value

8.37.2.2 KILL_CSPRNG()

```
void KILL_CSPRNG (
            csprng * R )
```

Deletes all internal state

Parameters

R	is a pointer to a cryptographically secure random number generator

8.38 rsa_2048.h File Reference

RSA Header file for implementation of RSA protocol.

```
#include "ff_2048.h"
#include "rsa_support.h"
```

Data Structures

- struct rsa_public_key_2048

  Integer Factorisation Public Key.
- struct rsa_private_key_2048

  Integer Factorisation Private Key.

Macros

- #define HASH_TYPE_RSA_2048 SHA256
- #define RFS_2048 MODBYTES_1024_58*FFLEN_2048

Functions

- void RSA_2048_KEY_PAIR (csprng *R, sign32 e, rsa_private_key_2048 *PRIV, rsa_public_key_2048 *PUB, octet *P, octet *Q)

  RSA Key Pair Generator.
- void RSA_2048_ENCRYPT (rsa_public_key_2048 *PUB, octet *F, octet *G)

  RSA encryption of suitably padded plaintext.
- void RSA_2048_DECRYPT (rsa_private_key_2048 *PRIV, octet *G, octet *F)

  RSA decryption of ciphertext.
- void RSA_2048_PRIVATE_KEY_KILL (rsa_private_key_2048 *PRIV)

  Destroy an RSA private Key.
- void RSA_2048_fromOctet (BIG_1024_58 *x, octet *S)

  Populates an RSA public key from an octet string.

8.38.1 Detailed Description

Author

Mike Scott declares functions

8.38.2 Macro Definition Documentation

8.38.2.1 HASH_TYPE_RSA_2048

```
#define HASH_TYPE_RSA_2048 SHA256
```

Chosen Hash algorithm

8.38.2.2 RFS_2048

```
#define RFS_2048 MODBYTES_1024_58*FFLEN_2048
```

RSA Public Key Size in bytes

8.38.3 Function Documentation

8.38.3.1 RSA_2048_DECRYPT()

```
void RSA_2048_DECRYPT (
            rsa_private_key_2048 * PRIV,
            octet * G,
            octet * F )
```

Parameters

P↩ RIV	the input RSA private key
G	is the input ciphertext
F	is output plaintext (requires unpadding)

8.38.3.2 RSA_2048_ENCRYPT()

```
void RSA_2048_ENCRYPT (
            rsa_public_key_2048 * PUB,
            octet * F,
            octet * G )
```

Parameters

PUB	the input RSA public key
F	is input padded message
G	is the output ciphertext

8.38.3.3 RSA_2048_fromOctet()

```
void RSA_2048_fromOctet (
            BIG_1024_58 * x,
            octet * S )
```

Creates RSA public key from big-endian base 256 form.

Parameters

x	FF instance to be created from an octet string
S	input octet string

8.38.3.4 RSA_2048_KEY_PAIR()

```
void RSA_2048_KEY_PAIR (
            csprng * R,
            sign32 e,
            rsa_private_key_2048 * PRIV,
            rsa_public_key_2048 * PUB,
            octet * P,
            octet * Q )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
e	the encryption exponent
P↩ *RIV*	the output RSA private key
PUB	the output RSA public key
P	Input prime number. Used when R is equal to NULL for testing
Q	Inpuy prime number. Used when R is equal to NULL for testing

8.38.3.5 RSA_2048_PRIVATE_KEY_KILL()

```
void RSA_2048_PRIVATE_KEY_KILL (
            rsa_private_key_2048 * PRIV )
```

Parameters

P↩ *RIV*	the input RSA private key. Destroyed on output.

8.39 rsa_3072.h File Reference

RSA Header file for implementation of RSA protocol.

```
#include "ff_3072.h"
#include "rsa_support.h"
```

Data Structures

- struct rsa_public_key_3072

    Integer Factorisation Public Key.
- struct rsa_private_key_3072

    Integer Factorisation Private Key.

Macros

- #define HASH_TYPE_RSA_3072 SHA256
- #define RFS_3072 MODBYTES_384_56∗FFLEN_3072

Functions

- void RSA_3072_KEY_PAIR (csprng *R, sign32 e, rsa_private_key_3072 *PRIV, rsa_public_key_3072 *P←
UB, octet *P, octet *Q)

    RSA Key Pair Generator.
- void RSA_3072_ENCRYPT (rsa_public_key_3072 *PUB, octet *F, octet *G)

    RSA encryption of suitably padded plaintext.
- void RSA_3072_DECRYPT (rsa_private_key_3072 *PRIV, octet *G, octet *F)

    RSA decryption of ciphertext.
- void RSA_3072_PRIVATE_KEY_KILL (rsa_private_key_3072 *PRIV)

    Destroy an RSA private Key.
- void RSA_3072_fromOctet (BIG_384_56 *x, octet *S)

    Populates an RSA public key from an octet string.

8.39.1 Detailed Description

Author

Mike Scott declares functions

8.39.2 Macro Definition Documentation

8.39.2.1 HASH_TYPE_RSA_3072

```
#define HASH_TYPE_RSA_3072 SHA256
```

Chosen Hash algorithm

8.39.2.2 RFS_3072

```
#define RFS_3072 MODBYTES_384_56*FFLEN_3072
```

RSA Public Key Size in bytes

8.39.3 Function Documentation

8.39.3.1 RSA_3072_DECRYPT()

```
void RSA_3072_DECRYPT (
            rsa_private_key_3072 * PRIV,
            octet * G,
            octet * F )
```

Parameters

P↩ RIV	the input RSA private key
G	is the input ciphertext
F	is output plaintext (requires unpadding)

8.39.3.2 RSA_3072_ENCRYPT()

```
void RSA_3072_ENCRYPT (
            rsa_public_key_3072 * PUB,
            octet * F,
            octet * G )
```

Parameters

PUB	the input RSA public key
F	is input padded message
G	is the output ciphertext

8.39.3.3 RSA_3072_fromOctet()

```
void RSA_3072_fromOctet (
            BIG_384_56 * x,
            octet * S )
```

Creates RSA public key from big-endian base 256 form.

Parameters

x	FF instance to be created from an octet string
S	input octet string

8.39.3.4 RSA_3072_KEY_PAIR()

```
void RSA_3072_KEY_PAIR (
            csprng * R,
            sign32 e,
            rsa_private_key_3072 * PRIV,
            rsa_public_key_3072 * PUB,
            octet * P,
            octet * Q )
```

Parameters

R	is a pointer to a cryptographically secure random number generator
e	the encryption exponent
P←→ RIV	the output RSA private key
PUB	the output RSA public key
P	Input prime number. Used when R is equal to NULL for testing
Q	Inpuy prime number. Used when R is equal to NULL for testing

8.39.3.5 RSA_3072_PRIVATE_KEY_KILL()

```
void RSA_3072_PRIVATE_KEY_KILL (
            rsa_private_key_3072 * PRIV )
```

Parameters

P←→ RIV	the input RSA private key. Destroyed on output.

8.40 rsa_support.h File Reference

RSA Support Header File.

```
#include "amcl.h"
```

Macros

- #define MAX_RSA_BYTES 512

Functions

- int PKCS15 (int h, octet ∗M, octet ∗W)

  PKCS V1.5 padding of a message prior to RSA signature.
- int OAEP_ENCODE (int h, octet ∗M, csprng ∗R, octet ∗P, octet ∗F)

  OAEP padding of a message prior to RSA encryption.
- int OAEP_DECODE (int h, octet ∗P, octet ∗F)

  OAEP unpadding of a message after RSA decryption.

8.40.1 Detailed Description

Author

Mike Scott

8.40.2 Macro Definition Documentation

8.40.2.1 MAX_RSA_BYTES

```
#define MAX_RSA_BYTES 512
```

Maximum of 4096

8.40.3 Function Documentation

8.40.3.1 OAEP_DECODE()

```
int OAEP_DECODE (
            int h,
            octet * P,
            octet * F )
```

Unpadding is done in-place

Parameters

h	is the hash type
P	are input encoding parameter string (could be NULL)
F	is input padded message, unpadded on output

Returns

0 if OK, else 1

8.40.3.2 OAEP_ENCODE()

```
int OAEP_ENCODE (
            int h,
            octet * M,
            csprng * R,
            octet * P,
            octet * F )
```

Parameters

h	is the hash type

Parameters

M	is the input message
R	is a pointer to a cryptographically secure random number generator
P	are input encoding parameter string (could be NULL)
F	is the output encoding, ready for RSA encryption

Returns

0 if OK, else 1

8.40.3.3 PKCS15()

```
int PKCS15 (
            int h,
            octet * M,
            octet * W )
```

Parameters

h	is the hash type
M	is the input message
W	is the output encoding, ready for RSA signature

Returns

1 if OK, else 0

8.41 utils.c File Reference

AMCL Support functions for M-Pin servers.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "amcl.h"
#include "utils.h"
```

Functions

- void amcl_hex2bin (const char ∗src, char ∗dst, int src_len)

  Decode hex value.
- void amcl_bin2hex (char ∗src, char ∗dst, int src_len)

  Encode binary string.

- void amcl_print_hex (char *src, int src_len)

    Print encoded binary string in hex.
- int generateOTP (csprng *RNG)

    Generate a random six digit one time password.
- void generateRandom (csprng *RNG, octet *randomValue)

    Generate a random Octet.

8.41.1 Detailed Description

Author

Mike Scott
Kealan McCusker

Date

28th July 2016 LICENSE

8.41.2 Function Documentation

8.41.2.1 amcl_bin2hex()

```
void amcl_bin2hex (
            char * src,
            char * dst,
            int src_len )
```

Encode binary string.

Parameters

src	Binary string
dst	Hex encoded string
src_len	length binary string

8.41.2.2 amcl_hex2bin()

```
void amcl_hex2bin (
            const char * src,
            char * dst,
            int src_len )
```

Decode hex value.

Parameters

src	Hex encoded string
dst	Binary string
src_len	length Hex encoded string

8.41.2.3 amcl_print_hex()

```
void amcl_print_hex (
            char * src,
            int src_len )
```

Print encoded binary string in hex.

Parameters

src	Binary string
src_len	length binary string

8.41.2.4 generateOTP()

```
int generateOTP (
            csprng * RNG )
```

Generates a random six digit one time password.

Parameters

RNG	random number generator

Returns

OTP One Time Password

8.41.2.5 generateRandom()

```
void generateRandom (
            csprng * RNG,
            octet * randomValue )
```

Generate a random Octet.

Parameters

RNG	random number generator
randomValue	random Octet

8.42 utils.h File Reference

Utility functions Header File.

```
#include "amcl.h"
```

Functions

- void amcl_hex2bin (const char *src, char *dst, int src_len)

  Decode hex value.
- void amcl_bin2hex (char *src, char *dst, int src_len)

  Encode binary string.
- void amcl_print_hex (char *src, int src_len)

  Print encoded binary string in hex.
- void generateRandom (csprng *RNG, octet *randomValue)

  Generate a random Octet.
- int generateOTP (csprng *RNG)

  Generate a random six digit one time password.

8.42.1 Detailed Description

Author

Kealan McCusker

8.42.2 Function Documentation

8.42.2.1 amcl_bin2hex()

```
void amcl_bin2hex (
            char * src,
            char * dst,
            int src_len )
```

Encode binary string.

Parameters

src	Binary string
dst	Hex encoded string
src_len	length binary string

8.42.2.2 amcl_hex2bin()

```
void amcl_hex2bin (
             const char * src,
             char * dst,
             int src_len )
```

Decode hex value.

Parameters

src	Hex encoded string
dst	Binary string
src_len	length Hex encoded string

8.42.2.3 amcl_print_hex()

```
void amcl_print_hex (
             char * src,
             int src_len )
```

Print encoded binary string in hex.

Parameters

src	Binary string
src_len	length binary string

8.42.2.4 generateOTP()

```
int generateOTP (
             csprng * RNG )
```

Generates a random six digit one time password.

Parameters

RNG	random number generator

Returns

OTP One Time Password

8.42.2.5 generateRandom()

```
void generateRandom (
            csprng * RNG,
            octet * randomValue )
```

Generate a random Octet.

Parameters

RNG	random number generator
randomValue	random Octet

8.43 version.c File Reference

AMCL version support function.

```
#include "version.h"
```

Functions

- void amcl_version (void)

    Print version number and information about the build.

8.43.1 Detailed Description

Author

Mike Scott
Kealan McCusker

Date

28th April 2016 LICENSE

8.43.2 Function Documentation

8.43.2.1 amcl_version()

```
void amcl_version (
            void )
```

Print version number and information about the build.

8.44 wcc_BLS381.h File Reference

WCC Header File.

```
#include "pair_BLS381.h"
#include "pbc_support.h"
```

Macros

- #define WCC_PGS_BLS381 MODBYTES_384_58
- #define WCC_PFS_BLS381 MODBYTES_384_58
- #define WCC_OK 0
- #define WCC_INVALID_POINT -51
- #define TIME_SLOT_MINUTES 1440
- #define PIV 12
- #define PTAG 16

Functions

- int WCC_BLS381_RANDOM_GENERATE (csprng ∗RNG, octet ∗S)

    Generate a random integer.
- void WCC_BLS381_Hq (int sha, octet ∗A, octet ∗B, octet ∗C, octet ∗D, octet ∗h)

    Hash EC Points and Id to an integer.
- int WCC_BLS381_GET_G2_MULTIPLE (octet ∗S, octet ∗HID, octet ∗VG2)

    Calculate value in G2 multiplied by an integer.
- int WCC_BLS381_GET_G1_MULTIPLE (octet ∗S, octet ∗HID, octet ∗VG1)

    Calculate value in G1 multiplied by an integer.
- int WCC_BLS381_SENDER_KEY (int sha, octet ∗xOct, octet ∗piaOct, octet ∗pibOct, octet ∗PbG2Oct, octet ∗PgG1Oct, octet ∗AKeyG1Oct, octet ∗IdBOct, octet ∗AESKeyOct)

    Calculate the sender AES key.
- int WCC_BLS381_RECEIVER_KEY (int sha, octet ∗yOct, octet ∗wOct, octet ∗piaOct, octet ∗pibOct, octet ∗PaG1Oct, octet ∗PgG1Oct, octet ∗BKeyG2Oct, octet ∗IdAOct, octet ∗AESKeyOct)

    Calculate the receiver AES key.
- int WCC_BLS381_RECOMBINE_G1 (octet ∗R1, octet ∗R2, octet ∗R)

    Add two members from the group G1.
- int WCC_BLS381_RECOMBINE_G2 (octet ∗W1, octet ∗W2, octet ∗W)

    Add two members from the group G2.

8.44.1 Detailed Description

Author

Mike Scott
Kealan McCusker

8.44.2 Macro Definition Documentation

8.44.2.1 PIV

```
#define PIV 12
```

AES-GCM Initialization Vector Size

8.44.2.2 PTAG

```
#define PTAG 16
```

AES-GCM MAC Size

8.44.2.3 TIME_SLOT_MINUTES

```
#define TIME_SLOT_MINUTES 1440
```

Time Slot = 1 day

8.44.2.4 WCC_INVALID_POINT

```
#define WCC_INVALID_POINT -51
```

Point is NOT on the curve

8.44.2.5 WCC_OK

```
#define WCC_OK 0
```

Function completed without error

8.44.2.6 WCC_PFS_BLS381

```
#define WCC_PFS_BLS381 MODBYTES_384_58
```

WCC Field Size

8.44.2.7 WCC_PGS_BLS381

```
#define WCC_PGS_BLS381 MODBYTES_384_58
```

WCC Group Size

8.44.3 Function Documentation

8.44.3.1 WCC_BLS381_GET_G1_MULTIPLE()

```
int WCC_BLS381_GET_G1_MULTIPLE (
            octet * S,
            octet * HID,
            octet * VG1 )
```

Calculate a value in G1. VG1 = s∗H1(ID) where ID is the identity.

1. VG1 = s∗H1(ID)

Parameters

S	integer modulus curve order
HID	Hash of ID padded with zeros to the field size
VG1	EC point VG1 = s∗H1(ID)

Returns

rtn Returns 0 if successful or else an error code

8.44.3.2 WCC_BLS381_GET_G2_MULTIPLE()

```
int WCC_BLS381_GET_G2_MULTIPLE (
            octet * S,
            octet * HID,
            octet * VG2 )
```

Calculate a value in G2. VG2 = s∗H2(ID) where ID is the identity.

1. VG2 = s∗H2(ID)

Parameters

S	integer modulus curve order
HID	Hash of ID padded with zeros to the field size
VG2	EC Point VG2 = s∗H2(ID)

Returns

rtn Returns 0 if successful or else an error code

8.44.3.3 WCC_BLS381_Hq()

```
void WCC_BLS381_Hq (
            int sha,
            octet * A,
            octet * B,
            octet * C,
            octet * D,
            octet * h )
```

Perform sha256 of EC Points and Id. Map to an integer modulo the curve order.

1. x = toInteger(sha256(A,B,C,D))
2. h = x % q where q is the curve order

Parameters

sha	Hash type
A	EC Point
B	EC Point
C	EC Point
D	Identity
h	Integer result

8.44.3.4 WCC_BLS381_RANDOM_GENERATE()

```
int WCC_BLS381_RANDOM_GENERATE (
            csprng * RNG,
            octet * S )
```

Generate a random number modulus the group order.

Parameters

RNG	cryptographically secure random number generator
S	Returned random integer modulus the group order

8.44.3.5 WCC_BLS381_RECEIVER_KEY()

```
int WCC_BLS381_RECEIVER_KEY (
            int sha,
            octet * yOct,
            octet * wOct,
            octet * piaOct,
            octet * pibOct,
            octet * PaG1Oct,
            octet * PgG1Oct,
            octet * BKeyG2Oct,
            octet * IdAOct,
            octet * AESKeyOct )
```

Calculate the receiver AES key.

1. j=e(pia.AG1+PaG1,(y+pib).BKeyG2)
2. K=H(j,w.PaG1)

Parameters

sha	Hash type
yOct	Random y $<$ q where q is the curve order
wOct	Random w $<$ q where q is the curve order
piaOct	Hq(PaG1,PbG2,PgG1)
pibOct	Hq(PbG2,PaG1,PgG1)
PaG1Oct	x.AG1 where x $<$ q
PgG1Oct	w.AG1 where w $<$ q
BKeyG2Oct	Receiver key
IdAOct	Sender identity
AESKeyOct	AES key returned

Returns

rtn Returns 0 if successful or else an error code

8.44.3.6 WCC_BLS381_RECOMBINE_G1()

```
int WCC_BLS381_RECOMBINE_G1 (
            octet * R1,
            octet * R2,
            octet * R )
```

Add two members from the group G1.

Parameters

R1	member of G1
R2	member of G1
R	returns member of G1 = R1+R2

Returns

Returns 0 if successful or else an error code

8.44.3.7 WCC_BLS381_RECOMBINE_G2()

```
int WCC_BLS381_RECOMBINE_G2 (
            octet * W1,
            octet * W2,
            octet * W )
```

Add two members from the group G2.

Parameters

W1	member of G2
W2	member of G2
W	returns member of G2 = W1+W2

Returns

Returns 0 if successful or else an error code

8.44.3.8 WCC_BLS381_SENDER_KEY()

```
int WCC_BLS381_SENDER_KEY (
            int sha,
            octet * xOct,
            octet * piaOct,
            octet * pibOct,
            octet * PbG2Oct,
            octet * PgG1Oct,
            octet * AKeyG1Oct,
            octet * IdBOct,
            octet * AESKeyOct )
```

Calculate the sender AES Key.

1. j=e((x+pia).AKeyG1,pib.BG2+PbG2)
2. K=H(j,x.PgG1)

Parameters

sha	Hash type
xOct	Random x $<$ q where q is the curve order
piaOct	Hq(PaG1,PbG2,PgG1)
pibOct	Hq(PbG2,PaG1,PgG1)
PbG2Oct	y.BG2 where y $<$ q
PgG1Oct	w.AG1 where w $<$ q
AKeyG1Oct	Sender key
IdBOct	Receiver identity
AESKeyOct	Returned AES key

Returns

rtn Returns 0 if successful or else an error code

8.45 x509.h File Reference

X509 function Header File.

Data Structures

- struct pktype

  Public key type.

Functions

- pktype X509_extract_cert_sig (octet ∗c, octet ∗s)

    Extract certificate signature.
- int X509_extract_cert (octet ∗sc, octet ∗c)
- pktype X509_extract_public_key (octet ∗c, octet ∗k)
- int X509_find_issuer (octet ∗c)
- int X509_find_validity (octet ∗c)
- int X509_find_subject (octet ∗c)
- int X509_find_entity_property (octet ∗c, octet ∗S, int s, int ∗f)
- int X509_find_start_date (octet ∗c, int s)
- int X509_find_expiry_date (octet ∗c, int s)

8.45.1 Detailed Description

Author

Mike Scott

8.45.2 Function Documentation

8.45.2.1 X509_extract_cert()

```
int X509_extract_cert (
            octet * sc,
            octet * c )
```

Parameters

sc	a signed certificate
c	the extracted certificate

Returns

0 on failure

8.45.2.2 X509_extract_cert_sig()

```
pktype X509_extract_cert_sig (
            octet * c,
            octet * s )
```

Parameters

c	an X.509 certificate
s	the extracted signature

Returns

0 on failure, or indicator of signature type (ECC or RSA)

8.45.2.3 X509_extract_public_key()

```
pktype X509_extract_public_key (
            octet * c,
            octet * k )
```

Parameters

c	an X.509 certificate
k	the extracted key

Returns

0 on failure, or indicator of public key type (ECC or RSA)

8.45.2.4 X509_find_entity_property()

```
int X509_find_entity_property (
            octet * c,
            octet * S,
            int s,
            int * f )
```

Parameters

c	an X.509 certificate
S	is OID of property we are looking for
s	is a pointer to the section of interest in the cert
f	is pointer to the length of the property

Returns

0 on failure, or pointer to the property

8.45.2.5 X509_find_expiry_date()

```
int X509_find_expiry_date (
            octet * c,
            int s )
```

Parameters

c	an X.509 certificate
s	is a pointer to the start of the validity field

Returns

0 on failure, or pointer to the expiry date

8.45.2.6 X509_find_issuer()

```
int X509_find_issuer (
            octet * c )
```

Parameters

c	an X.509 certificate

Returns

0 on failure, or pointer to issuer field in cert

8.45.2.7 X509_find_start_date()

```
int X509_find_start_date (
            octet * c,
            int s )
```

Parameters

c	an X.509 certificate
s	is a pointer to the start of the validity field

Returns

0 on failure, or pointer to the start date

8.45.2.8 X509_find_subject()

```
int X509_find_subject (
            octet * c )
```

Parameters

c	an X.509 certificate

Returns

0 on failure, or pointer to subject field in cert

8.45.2.9 X509_find_validity()

```
int X509_find_validity (
            octet * c )
```

Parameters

c	an X.509 certificate

Returns

0 on failure, or pointer to validity field in cert

Index