MIRACL Core

4.1

Generated by Doxygen 1.9.1

Chapter 1

Description

Namespaces are simulated to separate different curves.

To this end the BIG type is renamed to BIG_XXX, where XXX can be changed to describe the size and layout of the BIG variable. Similarily the FP type is renamed FP_YYY, where YYY reflects the modulus used. Also the ECP type is renamed ECP_ZZZ, where ZZZ describes the actual curve. Function names are also decorated in the same way.

So for example to support both ED25519 and the NIST P256 curve on a 64-bit processor, we would need to create BIG_256_56, FP_25519, ECP_ED25519 and BIG_256_56, FP_NIST256, ECP_NIST256. Note that both curves could be built on top of BIG_256_56, as both require support for 256-bit numbers using an internal number base of 2^56.

Separate ROM files provide the constants required for each curve. The associated header files (big.h, fp.h and ecp.h) also specify certain constants that must be set for the particular curve.

1.1 Installation and Testing

To build the library and see it in action, copy all of the files in this directory to a fresh directory. Then execute the python3 script config32.py for a 32-bit build, or config64.py for a 64-bit build, and select the curves that you wish to support. Note that support for 16-bit builds is currently somewhat limited - see config16.py. A library is built automatically including all of the modules that you will need.

The configuration files assume the gcc compiler. For clang edit the config32.py and config64.py files and substitute "clang" for "gcc". Note that clang is about 10-15% faster.∗

Make sure to use a 64-bit compiler on a 64-bit architecture.

NOTE: In the file config_curve.h a couple of methods with possible IP issues are commented out. For faster pairing code, edit this file.

To create a 32-bit library

```
python3 config32.py
```

Then select options 1, 3, 7, 28, 30, 36, 37 and 40, which are fixed for the example program. (For a 16-bit build select 1,4 and 6). Select 0 then to exit.
Then compile

```
gcc -O2 -std=c99 testecc.c core.a -o testecc
```

(if using MINGW-W64 in Windows change -o testecc to -o testecc.exe)
The test program exercises 3 different ordinary elliptic curves (for ECDH Key exchange, ECDSA signature and ECIES encryption), plus RSA, all in the one binary
Next compile

```
gcc -O2  -std=c99 testmpin.c core.a -o testmpin
```

This test program exercises 4 different pairing friendly curves using the MPIN authentication protocol.
The correct PIN is 1234
Next compile

```
gcc -O2 -std=c99 testbls.c core.a -o testbls
```

This program implements the pairing-based BLS signature
Next compile

```
gcc -O2 -std=c99 benchtest_all.c core.a -o benchtest_all
```

This program provides some timings.
Finally

```
gcc -O2 -std=c99 testnhs.c core.a -o testnhs
```

Alternatively building and testing can be combined via

```
python3 configXX.py test
```

where XX can be 16, 32 or 64

NEW: support for emerging Hash To Curve standard. See https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/
Create 32 or 64-bit library selecting curves 1, 2, 3, 7, 17 and 31 (ED25519, C25519, NIST256, GOLDILOCKS, SECP256K1 and BLS12381)

```
gcc -O2 -std=c99 testhtp.c core.a -o testhtp
```

Test program runs through test vectors from the draft standard.

NEW: Experimental support for emerging HPKE (Hybrid Public Key Encryption) standard. See https://datatracker.ietf.org/doc/draft-irtf-cfrg-hpke/
New hpke.c/.h api files

- Supports KEM_IDs for X25519, X448, P256 and P521
- Supports HDF_IDs for SHA256/512
- Supports AEAD_IDs for AES-GCM-128/256 only

Create 32 or 64-bit library selecting curves 2 and 10 (X25519 and P521)

```
gcc -O2 -std=c99 testhpke.c core.a -o testhpke
```

Test program runs through test vectors for all modes 0-3.

---

*Using clang on Windows Download latest clang from http://releases.llvm.org/download.html
Choose Clang for Windows (64-bit) (.sig) Install a free version of Microsoft Visual C++ https://www.visualstudio.com/downloads/ Now use "clang" wherever "gcc" was used before.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 core_aes Struct Reference

AES instance.

```
#include <core.h>
```

Public Attributes

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

4.1.1 Detailed Description

AES instance.

4.1.2 Member Data Documentation

4.1.2.1 Nk

`int core_aes::Nk`

AES Key Length

4.1.2.2 Nr

`int core_aes::Nr`

AES Number of rounds

4.1.2.3 mode

`int core_aes::mode`

AES mode of operation

4.1.2.4 fkey

`unsign32 core_aes::fkey[60]`

subkeys for encrypton

4.1.2.5 rkey

`unsign32 core_aes::rkey[60]`

subkeys for decrypton

4.1.2.6 f

`char core_aes::f[16]`

buffer for chaining vector

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

- core.h

4.2 csprng Struct Reference

Cryptographically secure pseudo-random number generator instance.

`#include <core.h>`

Public Attributes

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

4.2.1 Detailed Description

Cryptographically secure pseudo-random number generator instance.

4.2.2 Member Data Documentation

4.2.2.1 ira

`unsign32 csprng::ira[NK]`

random number array

4.2.2.2 rndptr

`int csprng::rndptr`

pointer into array

4.2.2.3 borrow

`unsign32 csprng::borrow`

borrow as a result of subtraction

4.2.2.4 pool_ptr

`int csprng::pool_ptr`

pointer into random pool

4.2.2.5 pool

`char csprng::pool[32]`

random pool

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

- core.h

4.3 ECP2_ZZZ Struct Reference

ECP2 Structure - Elliptic Curve Point over quadratic extension field.

`#include <ecp2.h>`

Public Attributes

- FP2_YYY x
- FP2_YYY y
- FP2_YYY z

4.3.1 Detailed Description

ECP2 Structure - Elliptic Curve Point over quadratic extension field.

4.3.2 Member Data Documentation

4.3.2.1 x

`FP2_YYY ECP2_ZZZ::x`

x-coordinate of point

4.3.2.2 y

`FP2_YYY ECP2_ZZZ::y`

y-coordinate of point

4.3.2.3 z

`FP2_YYY ECP2_ZZZ::z`

z-coordinate of point

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

- ecp2.h

4.4 ECP4_ZZZ Struct Reference

ECP4 Structure - Elliptic Curve Point over quadratic extension field.

`#include <ecp4.h>`

Public Attributes

- FP4_YYY x
- FP4_YYY y
- FP4_YYY z

4.4.1 Detailed Description

ECP4 Structure - Elliptic Curve Point over quadratic extension field.

4.4.2 Member Data Documentation

4.4.2.1 x

`FP4_YYY ECP4_ZZZ::x`

x-coordinate of point

4.4.2.2 y

`FP4_YYY ECP4_ZZZ::y`

y-coordinate of point

4.4.2.3 z

`FP4_YYY ECP4_ZZZ::z`

z-coordinate of point

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

- ecp4.h

4.5 ECP8_ZZZ Struct Reference

ECP8 Structure - Elliptic Curve Point over quadratic extension field.

`#include <ecp8.h>`

Public Attributes

- FP8_YYY x
- FP8_YYY y
- FP8_YYY z

4.5.1 Detailed Description

ECP8 Structure - Elliptic Curve Point over quadratic extension field.

4.5.2 Member Data Documentation

4.5.2.1 x

`FP8_YYY ECP8_ZZZ::x`

x-coordinate of point

4.5.2.2 y

`FP8_YYY ECP8_ZZZ::y`

y-coordinate of point

4.5.2.3 z

`FP8_YYY ECP8_ZZZ::z`

z-coordinate of point

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

- ecp8.h

4.6 ECP_ZZZ Struct Reference

ECP structure - Elliptic Curve Point over base field.

```
#include <ecp.h>
```

Public Attributes

- FP_YYY x
- FP_YYY z

4.6.1 Detailed Description

ECP structure - Elliptic Curve Point over base field.

4.6.2 Member Data Documentation

4.6.2.1 x

```
FP_YYY ECP_ZZZ::x
```

x-coordinate of point

4.6.2.2 z

```
FP_YYY ECP_ZZZ::z
```

z-coordinate of point

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

- ecp.h

4.7 FP12_YYY Struct Reference

FP12 Structure - towered over three FP4.

```
#include <fp12.h>
```

Public Attributes

- FP4_YYY a
- FP4_YYY b
- FP4_YYY c
- int type

4.7.1 Detailed Description

FP12 Structure - towered over three FP4.

4.7.2 Member Data Documentation

4.7.2.1 a

```
FP4_YYY FP12_YYY::a
```

first part of FP12

4.7.2.2 b

`FP4_YYY FP12_YYY::b`

second part of FP12

4.7.2.3 c

`FP4_YYY FP12_YYY::c`

third part of FP12

4.7.2.4 type

`int FP12_YYY::type`

record sparseness

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

- fp12.h

4.8 FP16_YYY Struct Reference

FP16 Structure - towered over two FP8.

`#include <fp16.h>`

Public Attributes

- FP8_YYY a
- FP8_YYY b

4.8.1 Detailed Description

FP16 Structure - towered over two FP8.

4.8.2 Member Data Documentation

4.8.2.1 a

`FP8_YYY FP16_YYY::a`

real part of FP16

4.8.2.2 b

`FP8_YYY FP16_YYY::b`

imaginary part of FP16

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

- fp16.h

4.9 FP24_YYY Struct Reference

FP12 Structure - towered over three FP8.

`#include <fp24.h>`

Public Attributes

- FP8_YYY a
- FP8_YYY b
- FP8_YYY c
- int type

4.9.1 Detailed Description

FP12 Structure - towered over three FP8.

4.9.2 Member Data Documentation

4.9.2.1 a

`FP8_YYY FP24_YYY::a`
first part of FP12

4.9.2.2 b

`FP8_YYY FP24_YYY::b`
second part of FP12

4.9.2.3 c

`FP8_YYY FP24_YYY::c`
third part of FP12

4.9.2.4 type

`int FP24_YYY::type`
record sparseness
The documentation for this struct was generated from the following file:

- fp24.h

4.10 FP2_YYY Struct Reference

FP2 Structure - quadratic extension field.
`#include <fp2.h>`

Public Attributes

- FP_YYY a
- FP_YYY b

4.10.1 Detailed Description

FP2 Structure - quadratic extension field.

4.10.2 Member Data Documentation

4.10.2.1 a

`FP_YYY FP2_YYY::a`
real part of FP2

4.10.2.2 b

`FP_YYY FP2_YYY::b`
imaginary part of FP2
The documentation for this struct was generated from the following file:

- fp2.h

4.11 FP48_YYY Struct Reference

FP12 Structure - towered over three FP16.

```
#include <fp48.h>
```

Public Attributes

- FP16_YYY a
- FP16_YYY b
- FP16_YYY c
- int type

4.11.1 Detailed Description

FP12 Structure - towered over three FP16.

4.11.2 Member Data Documentation

4.11.2.1 a

`FP16_YYY FP48_YYY::a`

first part of FP12

4.11.2.2 b

`FP16_YYY FP48_YYY::b`

second part of FP12

4.11.2.3 c

`FP16_YYY FP48_YYY::c`

third part of FP12

4.11.2.4 type

`int FP48_YYY::type`

record sparseness

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

- fp48.h

4.12 FP4_YYY Struct Reference

FP4 Structure - towered over two FP2.

```
#include <fp4.h>
```

Public Attributes

- FP2_YYY a
- FP2_YYY b

4.12.1 Detailed Description

FP4 Structure - towered over two FP2.

4.12.2 Member Data Documentation

4.12.2.1 a

`FP2_YYY FP4_YYY::a`

real part of FP4

4.12.2.2 b

`FP2_YYY FP4_YYY::b`

imaginary part of FP4

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

- fp4.h

4.13 FP8_YYY Struct Reference

FP8 Structure - towered over two FP4.

`#include <fp8.h>`

Public Attributes

- FP4_YYY a
- FP4_YYY b

4.13.1 Detailed Description

FP8 Structure - towered over two FP4.

4.13.2 Member Data Documentation

4.13.2.1 a

`FP4_YYY FP8_YYY::a`

real part of FP8

4.13.2.2 b

`FP4_YYY FP8_YYY::b`

imaginary part of FP8

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

- fp8.h

4.14 FP_YYY Struct Reference

FP Structure - quadratic extension field.

`#include <fp.h>`

Public Attributes

- BIG_XXX g
- sign32 XES

4.14.1 Detailed Description

FP Structure - quadratic extension field.

4.14.2 Member Data Documentation

4.14.2.1 g

`BIG_XXX FP_YYY::g`

Big representation of field element

4.14.2.2 XES

`sign32 FP_YYY::XES`

Excess

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

- fp.h

4.15 gcm Struct Reference

GCM mode instance, using AES internally.

`#include <core.h>`

Public Attributes

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

4.15.1 Detailed Description

GCM mode instance, using AES internally.

4.15.2 Member Data Documentation

4.15.2.1 table

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

2k byte table

4.15.2.2 stateX

`uchar gcm::stateX[16]`

GCM Internal State

4.15.2.3 Y_0

`uchar gcm::Y_0[16]`

GCM Internal State

4.15.2.4 lenA

`unsign32 gcm::lenA[2]`
GCM 64-bit length of header

4.15.2.5 lenC

`unsign32 gcm::lenC[2]`
GCM 64-bit length of ciphertext

4.15.2.6 status

`int gcm::status`
GCM Status

4.15.2.7 a

`core_aes gcm::a`
Internal Instance of CORE_AES cipher
The documentation for this struct was generated from the following file:

- core.h

4.16 hash256 Struct Reference

SHA256 hash function instance.
`#include <core.h>`

Public Attributes

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

4.16.1 Detailed Description

SHA256 hash function instance.

4.16.2 Member Data Documentation

4.16.2.1 length

`unsign32 hash256::length[2]`
64-bit input length

4.16.2.2 h

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

4.16.2.3 w

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

4.16.2.4 hlen

`int hash256::hlen`

Hash length in bytes

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

- core.h

4.17 hash512 Struct Reference

SHA384-512 hash function instance.

`#include <core.h>`

Public Attributes

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

4.17.1 Detailed Description

SHA384-512 hash function instance.

4.17.2 Member Data Documentation

4.17.2.1 length

`unsign64 hash512::length[2]`

64-bit input length

4.17.2.2 h

`unsign64 hash512::h[8]`

Internal state

4.17.2.3 w

`unsign64 hash512::w[80]`

Internal state

4.17.2.4 hlen

`int hash512::hlen`

Hash length in bytes

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

- core.h

4.18 octet Struct Reference

Portable representation of a big positive number.

`#include <core.h>`

Public Attributes

- int len
- int max
- char ∗ val

4.18.1 Detailed Description

Portable representation of a big positive number.

4.18.2 Member Data Documentation

4.18.2.1 len

`int octet::len`

length in bytes

4.18.2.2 max

`int octet::max`

max length allowed - enforce truncation

4.18.2.3 val

`char* octet::val`

byte array

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

- core.h

4.19 pktype Struct Reference

Public key type.

`#include <x509.h>`

Public Attributes

- int type
- int hash
- int curve

4.19.1 Detailed Description

Public key type.

4.19.2 Member Data Documentation

4.19.2.1 type

`int pktype::type`

signature type (ECC or RSA)

4.19.2.2 hash

```
int pktype::hash
```

hash type

4.19.2.3 curve

```
int pktype::curve
```

elliptic curve used or RSA key length in bits

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

- x509.h

4.20 rsa_private_key_WWW Struct Reference

Integer Factorisation Private Key.

```
#include <rsa.h>
```

Public Attributes

- BIG_XXX p [FFLEN_WWW/2]
- BIG_XXX q [FFLEN_WWW/2]
- BIG_XXX dp [FFLEN_WWW/2]
- BIG_XXX dq [FFLEN_WWW/2]
- BIG_XXX c [FFLEN_WWW/2]

4.20.1 Detailed Description

Integer Factorisation Private Key.

4.20.2 Member Data Documentation

4.20.2.1 p

```
BIG_XXX rsa_private_key_WWW::p[FFLEN_WWW/2]
```

secret prime p

4.20.2.2 q

```
BIG_XXX rsa_private_key_WWW::q[FFLEN_WWW/2]
```

secret prime q

4.20.2.3 dp

```
BIG_XXX rsa_private_key_WWW::dp[FFLEN_WWW/2]
```

decrypting exponent mod (p-1)

4.20.2.4 dq

```
BIG_XXX rsa_private_key_WWW::dq[FFLEN_WWW/2]
```

decrypting exponent mod (q-1)

4.20.2.5 c

`BIG_XXX rsa_private_key_WWW::c[FFLEN_WWW/2]`

1/p mod q

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

- rsa.h

4.21 rsa_public_key_WWW Struct Reference

Integer Factorisation Public Key.

`#include <rsa.h>`

Public Attributes

- sign32 e
- BIG_XXX n [FFLEN_WWW]

4.21.1 Detailed Description

Integer Factorisation Public Key.

4.21.2 Member Data Documentation

4.21.2.1 e

`sign32 rsa_public_key_WWW::e`

RSA exponent (typically 65537)

4.21.2.2 n

`BIG_XXX rsa_public_key_WWW::n[FFLEN_WWW]`

An array of BIGs to store public key

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

- rsa.h

4.22 sha3 Struct Reference

SHA3 hash function instance.

`#include <core.h>`

Public Attributes

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

4.22.1 Detailed Description

SHA3 hash function instance.

4.22.2 Member Data Documentation

4.22.2.1 length

`unsign64 sha3::length`
64-bit input length

4.22.2.2 S

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

4.22.2.3 rate

`int sha3::rate`
TODO

4.22.2.4 len

`int sha3::len`
Hash length in bytes
The documentation for this struct was generated from the following file:

- core.h

4.23 share Struct Reference

Share instance.
`#include <core.h>`

Public Attributes

- int id
- int nsr
- octet * B

4.23.1 Detailed Description

Share instance.

4.23.2 Member Data Documentation

4.23.2.1 id

`int share::id`
Unique Share ID

4.23.2.2 nsr

`int share::nsr`
number of shares required

4.23.2.3 B

`octet* share::B`
share as octet
The documentation for this struct was generated from the following file:

- core.h

Chapter 5

File Documentation

5.1 arch.h File Reference

Architecture Header File.

Macros

- #define CHUNK @WL@
- #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

5.1.1 Detailed Description

Architecture Header File.

Author

Mike Scott

Date

23rd February 2016

Specify Processor Architecture

5.1.2 Macro Definition Documentation

5.1.2.1 CHUNK

```
#define CHUNK @WL@
```

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

5.1.2.2 byte

```
#define byte unsigned char
```

8-bit unsigned integer

5.1.2.3 sign32

```
#define sign32 __int32
```

32-bit signed integer

5.1.2.4 sign8

```
#define sign8 signed char
```

8-bit signed integer

5.1.2.5 sign16

```
#define sign16 short int
```

16-bit signed integer

5.1.2.6 sign64

```
#define sign64 long long
```

64-bit signed integer

5.1.2.7 unsign32

```
#define unsign32 unsigned __int32
```

32-bit unsigned integer

5.1.2.8 unsign64

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

64-bit unsigned integer

5.1.2.9 uchar

```
#define uchar unsigned char
```

Unsigned char

5.2 big.h File Reference

BIG Header File.

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "arch.h"
#include "core.h"
#include "config_big_XXX.h"
```

Macros

- #define UNWOUND
- #define USE_KARATSUBA
- #define BIGBITS_XXX (8∗MODBYTES_XXX)
- #define NLEN_XXX (1+((8∗MODBYTES_XXX-1)/BASEBITS_XXX))
- #define DNLEN_XXX 2∗NLEN_XXX
- #define BMASK_XXX (((chunk)1<<BASEBITS_XXX)-1)
- #define NEXCESS_XXX (1<<(CHUNK-BASEBITS_XXX-1))
- #define HBITS_XXX (BASEBITS_XXX/2)
- #define HMASK_XXX (((chunk)1<<HBITS_XXX)-1)

Typedefs

- typedef chunk BIG_XXX[NLEN_XXX]
- typedef chunk DBIG_XXX[DNLEN_XXX]

Functions

- int BIG_XXX_iszilch (BIG_XXX x)

  Tests for BIG equal to zero (Constant Time)
- int BIG_XXX_isunity (BIG_XXX x)

  Tests for BIG equal to one (Constant Time)
- int BIG_XXX_diszilch (DBIG_XXX x)

  Tests for DBIG equal to zero (Constant Time)
- void BIG_XXX_output (BIG_XXX x)

  Outputs a BIG number to the console (Variable Time)
- void BIG_XXX_rawoutput (BIG_XXX x)

  Outputs a BIG number to the console in raw form (Variable Time for debugging)
- void BIG_XXX_cswap (BIG_XXX x, BIG_XXX y, int s)

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

  Conditional copy of BIG number.
- void BIG_XXX_dcmove (BIG_XXX x, BIG_XXX y, int s)

  Conditional copy of DBIG number.
- void BIG_XXX_toBytes (char ∗a, BIG_XXX x)

  Convert from BIG number to byte array (Constant Time)
- void BIG_XXX_fromBytes (BIG_XXX x, char ∗a)

  Convert to BIG number from byte array (Constant Time)
- void BIG_XXX_fromBytesLen (BIG_XXX x, char ∗a, int s)

  Convert to BIG number from byte array of given length (Variable Time)
- void BIG_XXX_dfromBytesLen (DBIG_XXX x, char ∗a, int s)

  Convert to DBIG number from byte array of given length (Variable Time)
- void BIG_XXX_doutput (DBIG_XXX x)

  Outputs a DBIG number to the console (Variable Time)
- void BIG_XXX_drawoutput (DBIG_XXX x)

  Outputs a DBIG number to the console (Variable Time)
- void BIG_XXX_rcopy (BIG_XXX x, const BIG_XXX y)

  Copy BIG from Read-Only Memory to a BIG (Constant Time)
- void BIG_XXX_copy (BIG_XXX x, BIG_XXX y)

  Copy BIG to another BIG (Constant Time)
- void BIG_XXX_dcopy (DBIG_XXX x, DBIG_XXX y)

  Copy DBIG to another DBIG (Constant Time)
- void BIG_XXX_dsucopy (DBIG_XXX x, BIG_XXX y)

  Copy BIG to upper half of DBIG (Constant Time)
- void BIG_XXX_dscopy (DBIG_XXX x, BIG_XXX y)

  Copy BIG to lower half of DBIG (Constant Time)
- void BIG_XXX_sdcopy (BIG_XXX x, DBIG_XXX y)

  Copy lower half of DBIG to a BIG (Constant Time)
- void BIG_XXX_sducopy (BIG_XXX x, DBIG_XXX y)

  Copy upper half of DBIG to a BIG (Constant Time)
- void BIG_XXX_zero (BIG_XXX x)

  Set BIG to zero (Constant Time)
- void BIG_XXX_dzero (DBIG_XXX x)

Set DBIG to zero (Constant Time)
- void BIG_XXX_one (BIG_XXX x)

  Set BIG to one (unity) (Constant Time)
- void BIG_XXX_invmod2m (BIG_XXX x)

  Set BIG to inverse mod 2^256 (Constant Time)
- void BIG_XXX_add (BIG_XXX x, BIG_XXX y, BIG_XXX z)

  Set BIG to sum of two BIGs - output not normalised (Constant Time)
- void BIG_XXX_or (BIG_XXX x, BIG_XXX y, BIG_XXX z)

  Set BIG to logical or of two BIGs - output normalised (Constant Time)
- void BIG_XXX_inc (BIG_XXX x, int i)

  Increment BIG by a small integer - output not normalised (Constant Time)
- void BIG_XXX_sub (BIG_XXX x, BIG_XXX y, BIG_XXX z)

  Set BIG to difference of two BIGs (Constant Time)
- void BIG_XXX_dec (BIG_XXX x, int i)

  Decrement BIG by a small integer - output not normalised (Constant Time)
- void BIG_XXX_dadd (DBIG_XXX x, DBIG_XXX y, DBIG_XXX z)

  Set DBIG to sum of two DBIGs (Constant Time)
- void BIG_XXX_dsub (DBIG_XXX x, DBIG_XXX y, DBIG_XXX z)

  Set DBIG to difference of two DBIGs (Constant Time)
- void BIG_XXX_imul (BIG_XXX x, BIG_XXX y, int i)

  Multiply BIG by a small integer - output not normalised (Constant Time)
- chunk BIG_XXX_pmul (BIG_XXX x, BIG_XXX y, int i)

  Multiply BIG by not-so-small small integer - output normalised (Constant Time)
- int BIG_XXX_div3 (BIG_XXX x)

  Divide BIG by 3 - output normalised (Constant Time)
- void BIG_XXX_pxmul (DBIG_XXX x, BIG_XXX y, int i)

  Multiply BIG by even bigger small integer resulting in a DBIG - output normalised (Constant Time)
- void BIG_XXX_mul (DBIG_XXX x, BIG_XXX y, BIG_XXX z)

  Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised (Constant Time)
- void BIG_XXX_smul (BIG_XXX x, BIG_XXX y, BIG_XXX z)

  Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised (Constant Time)
- void BIG_XXX_sqr (DBIG_XXX x, BIG_XXX y)

  Square BIG resulting in a DBIG - input normalised and output normalised (Constant Time)
- void BIG_XXX_monty (BIG_XXX a, BIG_XXX md, chunk MC, DBIG_XXX d)

  Montgomery reduction of a DBIG to a BIG - input normalised and output normalised (Constant Time)
- void BIG_XXX_shl (BIG_XXX x, int s)

  Shifts a BIG left by any number of bits - input must be normalised, output normalised (Constant Time)
- int BIG_XXX_fshl (BIG_XXX x, int s)

  Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised (Constant Time)
- void BIG_XXX_dshl (DBIG_XXX x, int s)

  Shifts a DBIG left by any number of bits - input must be normalised, output normalised (Constant Time)
- void BIG_XXX_shr (BIG_XXX x, int s)

  Shifts a BIG right by any number of bits - input must be normalised, output normalised (Constant Time)
- int BIG_XXX_ssn (BIG_XXX r, BIG_XXX a, BIG_XXX m)

  Fast time-critical combined shift by 1 bit, subtract and normalise (Constant Time)
- int BIG_XXX_fshr (BIG_XXX x, int s)

  Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised (Constant Time)
- void BIG_XXX_dshr (DBIG_XXX x, int s)

  Shifts a DBIG right by any number of bits - input must be normalised, output normalised (Constant Time)
- chunk BIG_XXX_split (BIG_XXX x, BIG_XXX y, DBIG_XXX z, int s)

  Splits a DBIG into two BIGs - input must be normalised, outputs normalised (Constant Time as used)

- chunk BIG_XXX_norm (BIG_XXX x)

  Normalizes a BIG number - output normalised (Constant Time)
- void BIG_XXX_dnorm (DBIG_XXX x)

  Normalizes a DBIG number - output normalised (Constant Time)
- int BIG_XXX_comp (BIG_XXX x, BIG_XXX y)

  Compares two BIG numbers. Inputs must be normalised externally (Constant Time)
- int BIG_XXX_dcomp (DBIG_XXX x, DBIG_XXX y)

  Compares two DBIG numbers. Inputs must be normalised externally (Constant Time)
- int BIG_XXX_nbits (BIG_XXX x)

  Calculate number of bits in a BIG - output normalised (Variable Time)
- int BIG_XXX_dnbits (DBIG_XXX x)

  Calculate number of bits in a DBIG - output normalised (Variable Time)
- void BIG_XXX_mod (BIG_XXX x, BIG_XXX n)

  Reduce x mod n - input and output normalised (Variable Time)
- void BIG_XXX_sdiv (BIG_XXX x, BIG_XXX n)

  Divide x by n - output normalised (Variable Time)
- void BIG_XXX_dmod (BIG_XXX x, DBIG_XXX y, BIG_XXX n)

  x=y mod n - output normalised (Variable Time)
- void BIG_XXX_ddiv (BIG_XXX x, DBIG_XXX y, BIG_XXX n)

  x=y/n - output normalised (Variable Time)
- int BIG_XXX_parity (BIG_XXX x)

  return parity of BIG, that is the least significant bit (Constant Time)
- int BIG_XXX_bit (BIG_XXX x, int i)

  return i-th of BIG (Constant Time)
- int BIG_XXX_lastbits (BIG_XXX x, int n)

  return least significant bits of a BIG (Constant Time)
- void BIG_XXX_random (BIG_XXX x, csprng ∗r)

  Create a random BIG from a random number generator (Constant Time)
- void BIG_XXX_randomnum (BIG_XXX x, BIG_XXX n, csprng ∗r)

  Create an unbiased random BIG from a random number generator, reduced with respect to a modulus (Constant Time as used)
- void BIG_XXX_randtrunc (BIG_XXX x, BIG_XXX n, int t, csprng ∗r)

  Create an unbiased random BIG from a random number generator, reduced with respect to a modulus and truncated to max bit length (Constant Time as used)
- void BIG_XXX_modmul (BIG_XXX x, BIG_XXX y, BIG_XXX z, BIG_XXX n)

  Calculate x=y∗z mod n (Variable Time)
- void BIG_XXX_moddiv (BIG_XXX x, BIG_XXX y, BIG_XXX z, BIG_XXX n)

  Calculate x=y/z mod n (Variable Time)
- void BIG_XXX_modsqr (BIG_XXX x, BIG_XXX y, BIG_XXX n)

  Calculate x=y^2 mod n (Variable Time)
- void BIG_XXX_modneg (BIG_XXX x, BIG_XXX y, BIG_XXX n)

  Calculate x=-y mod n (Variable Time)
- void BIG_XXX_modadd (BIG_XXX x, BIG_XXX y, BIG_XXX z, BIG_XXX n)

  Calculate x=y+z mod n (Variable Time)
- int BIG_XXX_jacobi (BIG_XXX x, BIG_XXX y)

  Calculate jacobi Symbol (x/y) (Variable Time)
- void BIG_XXX_invmodp (BIG_XXX x, BIG_XXX y, BIG_XXX n)

  Calculate x=1/y mod n (Variable Time)
- void BIG_XXX_mod2m (BIG_XXX x, int m)

  Calculate x=x mod 2^m (Variable Time)

5.2.1 Detailed Description

BIG Header File.

Author

Mike Scott

5.2.2 Macro Definition Documentation

5.2.2.1 UNWOUND

```
#define UNWOUND
```

Default to unwound code

5.2.2.2 USE_KARATSUBA

```
#define USE_KARATSUBA
```

Default to use Karatsuba method

5.2.2.3 BIGBITS_XXX

```
#define BIGBITS_XXX (8*MODBYTES_XXX)
```

Length in bits

5.2.2.4 NLEN_XXX

```
#define NLEN_XXX (1+((8*MODBYTES_XXX-1)/BASEBITS_XXX))
```

length in bytes

5.2.2.5 DNLEN_XXX

```
#define DNLEN_XXX 2*NLEN_XXX
```

Double length in bytes

5.2.2.6 BMASK_XXX

```
#define BMASK_XXX (((chunk)1<<BASEBITS_XXX)-1)
```

Mask = 2^BASEBITS-1

5.2.2.7 NEXCESS_XXX

```
#define NEXCESS_XXX (1<<(CHUNK-BASEBITS_XXX-1))
```

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

5.2.2.8 HBITS_XXX

```
#define HBITS_XXX (BASEBITS_XXX/2)
```

Number of bits in number base divided by 2

5.2.2.9 HMASK_XXX

```
#define HMASK_XXX (((chunk)1<<HBITS_XXX)-1)
```

Mask = 2^HBITS-1

5.2.3 Typedef Documentation

5.2.3.1 BIG_XXX

`typedef chunk BIG_XXX[NLEN_XXX]`

Define type BIG as array of chunks

5.2.3.2 DBIG_XXX

`typedef chunk DBIG_XXX[DNLEN_XXX]`

Define type DBIG as array of chunks

5.2.4 Function Documentation

5.2.4.1 BIG_XXX_iszilch()

```
int BIG_XXX_iszilch (
        BIG_XXX x )
```

Tests for BIG equal to zero (Constant Time)

Parameters

x	a BIG number

Returns

1 if zero, else returns 0

5.2.4.2 BIG_XXX_isunity()

```
int BIG_XXX_isunity (
        BIG_XXX x )
```

Tests for BIG equal to one (Constant Time)

Parameters

x	a BIG number

Returns

1 if one, else returns 0

5.2.4.3 BIG_XXX_diszilch()

```
int BIG_XXX_diszilch (
        DBIG_XXX x )
```

Tests for DBIG equal to zero (Constant Time)

Parameters

x	a DBIG number

Returns

1 if zero, else returns 0

5.2.4.4 BIG_XXX_output()

```
void BIG_XXX_output (
            BIG_XXX x )
```

Outputs a BIG number to the console (Variable Time)

Parameters

x	a BIG number

5.2.4.5 BIG_XXX_rawoutput()

```
void BIG_XXX_rawoutput (
            BIG_XXX x )
```

Outputs a BIG number to the console in raw form (Variable Time for debugging)

Parameters

x	a BIG number

5.2.4.6 BIG_XXX_cswap()

```
void BIG_XXX_cswap (
            BIG_XXX x,
            BIG_XXX y,
            int s )
```

Conditional constant time swap of two BIG numbers.

Conditionally swaps parameters in constant time (Constant Time without branching)

Parameters

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

5.2.4.7 BIG_XXX_cmove()

```
void BIG_XXX_cmove (
            BIG_XXX x,
            BIG_XXX y,
            int s )
```

Conditional copy of BIG number.

Conditionally copies second parameter to the first (Constant Time without branching)

Parameters

x	a BIG number

Parameters

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

5.2.4.8 BIG_XXX_dcmove()

```
void BIG_XXX_dcmove (
            BIG_XXX x,
            BIG_XXX y,
            int s )
```

Conditional copy of DBIG number.

Conditionally copies second parameter to the first (Constant Time without branching)

Parameters

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

5.2.4.9 BIG_XXX_toBytes()

```
void BIG_XXX_toBytes (
            char * a,
            BIG_XXX x )
```

Convert from BIG number to byte array (Constant Time)

Parameters

a	byte array
x	BIG number

5.2.4.10 BIG_XXX_fromBytes()

```
void BIG_XXX_fromBytes (
            BIG_XXX x,
            char * a )
```

Convert to BIG number from byte array (Constant Time)

Parameters

x	BIG number
a	byte array

5.2.4.11 BIG_XXX_fromBytesLen()

```
void BIG_XXX_fromBytesLen (
            BIG_XXX x,
            char * a,
```

```
int s )
```

Convert to BIG number from byte array of given length (Variable Time)

Parameters

x	BIG number
a	byte array
s	byte array length

5.2.4.12 BIG_XXX_dfromBytesLen()

```
void BIG_XXX_dfromBytesLen (
        DBIG_XXX x,
        char * a,
        int s )
```

Convert to DBIG number from byte array of given length (Variable Time)

Parameters

x	DBIG number
a	byte array
s	byte array length

5.2.4.13 BIG_XXX_doutput()

```
void BIG_XXX_doutput (
        DBIG_XXX x )
```

Outputs a DBIG number to the console (Variable Time)

Parameters

x	a DBIG number

5.2.4.14 BIG_XXX_drawoutput()

```
void BIG_XXX_drawoutput (
        DBIG_XXX x )
```

Outputs a DBIG number to the console (Variable Time)

Parameters

x	a DBIG number

5.2.4.15 BIG_XXX_rcopy()

```
void BIG_XXX_rcopy (
        BIG_XXX x,
        const BIG_XXX y )
```

Copy BIG from Read-Only Memory to a BIG (Constant Time)

Parameters

x	BIG number
y	BIG number in ROM

5.2.4.16 BIG_XXX_copy()

```
void BIG_XXX_copy (
            BIG_XXX x,
            BIG_XXX y )
```

Copy BIG to another BIG (Constant Time)

Parameters

x	BIG number
y	BIG number to be copied

5.2.4.17 BIG_XXX_dcopy()

```
void BIG_XXX_dcopy (
            DBIG_XXX x,
            DBIG_XXX y )
```

Copy DBIG to another DBIG (Constant Time)

Parameters

x	DBIG number
y	DBIG number to be copied

5.2.4.18 BIG_XXX_dsucopy()

```
void BIG_XXX_dsucopy (
            DBIG_XXX x,
            BIG_XXX y )
```

Copy BIG to upper half of DBIG (Constant Time)

Parameters

x	DBIG number
y	BIG number to be copied

5.2.4.19 BIG_XXX_dscopy()

```
void BIG_XXX_dscopy (
            DBIG_XXX x,
            BIG_XXX y )
```

Copy BIG to lower half of DBIG (Constant Time)

Parameters

x	DBIG number
y	BIG number to be copied

5.2.4.20 BIG_XXX_sdcopy()

```
void BIG_XXX_sdcopy (
            BIG_XXX x,
            DBIG_XXX y )
```

Copy lower half of DBIG to a BIG (Constant Time)

Parameters

x	BIG number
y	DBIG number to be copied

5.2.4.21 BIG_XXX_sducopy()

```
void BIG_XXX_sducopy (
            BIG_XXX x,
            DBIG_XXX y )
```

Copy upper half of DBIG to a BIG (Constant Time)

Parameters

x	BIG number
y	DBIG number to be copied

5.2.4.22 BIG_XXX_zero()

```
void BIG_XXX_zero (
            BIG_XXX x )
```

Set BIG to zero (Constant Time)

Parameters

x	BIG number to be set to zero

5.2.4.23 BIG_XXX_dzero()

```
void BIG_XXX_dzero (
            DBIG_XXX x )
```

Set DBIG to zero (Constant Time)

Parameters

x	DBIG number to be set to zero

5.2.4.24 BIG_XXX_one()

```
void BIG_XXX_one (
              BIG_XXX x )
```

Set BIG to one (unity) (Constant Time)

Parameters

x	BIG number to be set to one.

5.2.4.25 BIG_XXX_invmod2m()

```
void BIG_XXX_invmod2m (
              BIG_XXX x )
```

Set BIG to inverse mod 2^256 (Constant Time)

Parameters

x	BIG number to be inverted

5.2.4.26 BIG_XXX_add()

```
void BIG_XXX_add (
              BIG_XXX x,
              BIG_XXX y,
              BIG_XXX z )
```

Set BIG to sum of two BIGs - output not normalised (Constant Time)

Parameters

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

5.2.4.27 BIG_XXX_or()

```
void BIG_XXX_or (
              BIG_XXX x,
              BIG_XXX y,
              BIG_XXX z )
```

Set BIG to logical or of two BIGs - output normalised (Constant Time)

Parameters

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

5.2.4.28 BIG_XXX_inc()

```
void BIG_XXX_inc (
        BIG_XXX x,
        int i )
```

Increment BIG by a small integer - output not normalised (Constant Time)

Parameters

x	BIG number to be incremented
i	integer

5.2.4.29 BIG_XXX_sub()

```
void BIG_XXX_sub (
        BIG_XXX x,
        BIG_XXX y,
        BIG_XXX z )
```

Set BIG to difference of two BIGs (Constant Time)

Parameters

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

5.2.4.30 BIG_XXX_dec()

```
void BIG_XXX_dec (
        BIG_XXX x,
        int i )
```

Decrement BIG by a small integer - output not normalised (Constant Time)

Parameters

x	BIG number to be decremented
i	integer

5.2.4.31 BIG_XXX_dadd()

```
void BIG_XXX_dadd (
        DBIG_XXX x,
        DBIG_XXX y,
        DBIG_XXX z )
```

Set DBIG to sum of two DBIGs (Constant Time)

Parameters

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

5.2.4.32 BIG_XXX_dsub()

```
void BIG_XXX_dsub (
            DBIG_XXX x,
            DBIG_XXX y,
            DBIG_XXX z )
```

Set DBIG to difference of two DBIGs (Constant Time)

Parameters

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

5.2.4.33 BIG_XXX_imul()

```
void BIG_XXX_imul (
            BIG_XXX x,
            BIG_XXX y,
            int i )
```

Multiply BIG by a small integer - output not normalised (Constant Time)

Parameters

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

5.2.4.34 BIG_XXX_pmul()

```
chunk BIG_XXX_pmul (
            BIG_XXX x,
            BIG_XXX y,
            int i )
```

Multiply BIG by not-so-small small integer - output normalised (Constant Time)

Parameters

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

Returns

Overflowing bits

5.2.4.35 BIG_XXX_div3()

```
int BIG_XXX_div3 (
            BIG_XXX x )
```

Divide BIG by 3 - output normalised (Constant Time)

Parameters

x	BIG number

Returns

Remainder

5.2.4.36 BIG_XXX_pxmul()

```
void BIG_XXX_pxmul (
            DBIG_XXX x,
            BIG_XXX y,
            int i )
```

Multiply BIG by even bigger small integer resulting in a DBIG - output normalised (Constant Time)

Parameters

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

5.2.4.37 BIG_XXX_mul()

```
void BIG_XXX_mul (
            DBIG_XXX x,
            BIG_XXX y,
            BIG_XXX z )
```

Multiply BIG by another BIG resulting in DBIG - inputs normalised and output normalised (Constant Time)

Parameters

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

5.2.4.38 BIG_XXX_smul()

```
void BIG_XXX_smul (
            BIG_XXX x,
            BIG_XXX y,
            BIG_XXX z )
```

Multiply BIG by another BIG resulting in another BIG - inputs normalised and output normalised (Constant Time) 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

5.2.4.39 BIG_XXX_sqr()

```
void BIG_XXX_sqr (
            DBIG_XXX x,
            BIG_XXX y )
```

Square BIG resulting in a DBIG - input normalised and output normalised (Constant Time)

Parameters

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

5.2.4.40 BIG_XXX_monty()

```
void BIG_XXX_monty (
            BIG_XXX a,
            BIG_XXX md,
            chunk MC,
            DBIG_XXX d )
```

Montgomery reduction of a DBIG to a BIG - input normalised and output normalised (Constant Time)

Parameters

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

5.2.4.41 BIG_XXX_shl()

```
void BIG_XXX_shl (
            BIG_XXX x,
            int s )
```

Shifts a BIG left by any number of bits - input must be normalised, output normalised (Constant Time)

Parameters

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

5.2.4.42 BIG_XXX_fshl()

```
int BIG_XXX_fshl (
            BIG_XXX x,
            int s )
```

Fast shifts a BIG left by a small number of bits - input must be normalised, output will be normalised (Constant Time)

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

5.2.4.43 BIG_XXX_dshl()

```
void BIG_XXX_dshl (
            DBIG_XXX x,
            int s )
```

Shifts a DBIG left by any number of bits - input must be normalised, output normalised (Constant Time)

Parameters

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

5.2.4.44 BIG_XXX_shr()

```
void BIG_XXX_shr (
            BIG_XXX x,
            int s )
```

Shifts a BIG right by any number of bits - input must be normalised, output normalised (Constant Time)

Parameters

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

5.2.4.45 BIG_XXX_ssn()

```
int BIG_XXX_ssn (
            BIG_XXX r,
            BIG_XXX a,
            BIG_XXX m )
```

Fast time-critical combined shift by 1 bit, subtract and normalise (Constant Time)

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

5.2.4.46 BIG_XXX_fshr()

```
int BIG_XXX_fshr (
            BIG_XXX x,
            int s )
```

Fast shifts a BIG right by a small number of bits - input must be normalised, output will be normalised (Constant Time)
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

5.2.4.47 BIG_XXX_dshr()

```
void BIG_XXX_dshr (
            DBIG_XXX x,
            int s )
```

Shifts a DBIG right by any number of bits - input must be normalised, output normalised (Constant Time)

Parameters

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

5.2.4.48 BIG_XXX_split()

```
chunk BIG_XXX_split (
            BIG_XXX x,
            BIG_XXX y,
            DBIG_XXX z,
            int s )
```

Splits a DBIG into two BIGs - input must be normalised, outputs normalised (Constant Time as used)
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

5.2.4.49 BIG_XXX_norm()

```
chunk BIG_XXX_norm (
            BIG_XXX x )
```

Normalizes a BIG number - output normalised (Constant Time)
All digits of the input BIG are reduced mod 2^BASEBITS

Parameters

x	BIG number to be normalised

5.2.4.50 BIG_XXX_dnorm()

```
void BIG_XXX_dnorm (
            DBIG_XXX x )
```

Normalizes a DBIG number - output normalised (Constant Time)
All digits of the input DBIG are reduced mod 2^BASEBITS

Parameters

x	DBIG number to be normalised

5.2.4.51 BIG_XXX_comp()

```
int BIG_XXX_comp (
            BIG_XXX x,
            BIG_XXX y )
```

Compares two BIG numbers. Inputs must be normalised externally (Constant Time)

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

5.2.4.52 BIG_XXX_dcomp()

```
int BIG_XXX_dcomp (
            DBIG_XXX x,
            DBIG_XXX y )
```

Compares two DBIG numbers. Inputs must be normalised externally (Constant Time)

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

5.2.4.53 BIG_XXX_nbits()

```
int BIG_XXX_nbits (
            BIG_XXX x )
```

Calculate number of bits in a BIG - output normalised (Variable Time)

Parameters

x	BIG number

Returns

Number of bits in x

5.2.4.54 BIG_XXX_dnbits()

```
int BIG_XXX_dnbits (
            DBIG_XXX x )
```

Calculate number of bits in a DBIG - output normalised (Variable Time)

Parameters

x	DBIG number

Returns

Number of bits in x

5.2.4.55 BIG_XXX_mod()

```
void BIG_XXX_mod (
            BIG_XXX x,
            BIG_XXX n )
```

Reduce x mod n - input and output normalised (Variable Time)
Slow but rarely used

Parameters

x	BIG number to be reduced mod n
n	The modulus

5.2.4.56 BIG_XXX_sdiv()

```
void BIG_XXX_sdiv (
            BIG_XXX x,
            BIG_XXX n )
```

Divide x by n - output normalised (Variable Time)
Slow but rarely used

Parameters

x	BIG number to be divided by n
n	The Divisor

5.2.4.57 BIG_XXX_dmod()

```
void BIG_XXX_dmod (
            BIG_XXX x,
            DBIG_XXX y,
            BIG_XXX n )
```

x=y mod n - output normalised (Variable Time)
Slow but rarely used. y is destroyed.

Parameters

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

5.2.4.58 BIG_XXX_ddiv()

```
void BIG_XXX_ddiv (
            BIG_XXX x,
            DBIG_XXX y,
            BIG_XXX n )
```

x=y/n - output normalised (Variable Time)
Slow but rarely used. y is destroyed.

Parameters

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

5.2.4.59 BIG_XXX_parity()

```
int BIG_XXX_parity (
            BIG_XXX x )
```

return parity of BIG, that is the least significant bit (Constant Time)

Parameters

x	BIG number

Returns

0 or 1

5.2.4.60 BIG_XXX_bit()

```
int BIG_XXX_bit (
            BIG_XXX x,
            int i )
```

return i-th of BIG (Constant Time)

Parameters

x	BIG number
i	the bit of x to be returned

Returns

0 or 1

5.2.4.61 BIG_XXX_lastbits()

```
int BIG_XXX_lastbits (
            BIG_XXX x,
            int n )
```

return least significant bits of a BIG (Constant Time)

Parameters

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

Returns

least significant n bits as an integer

5.2.4.62 BIG_XXX_random()

```
void BIG_XXX_random (
            BIG_XXX x,
            csprng * r )
```

Create a random BIG from a random number generator (Constant Time)
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

5.2.4.63 BIG_XXX_randomnum()

```
void BIG_XXX_randomnum (
            BIG_XXX x,
            BIG_XXX n,
            csprng * r )
```

Create an unbiased random BIG from a random number generator, reduced with respect to a modulus (Constant Time as used)

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

5.2.4.64 BIG_XXX_randtrunc()

```
void BIG_XXX_randtrunc (
            BIG_XXX x,
            BIG_XXX n,
            int t,
            csprng * r )
```

Create an unbiased random BIG from a random number generator, reduced with respect to a modulus and truncated to max bit length (Constant Time as used)
Assumes that the random number generator has been suitably initialised

Parameters

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

5.2.4.65 BIG_XXX_modmul()

```
void BIG_XXX_modmul (
            BIG_XXX x,
            BIG_XXX y,
            BIG_XXX z,
            BIG_XXX n )
```

Calculate x=y∗z mod n (Variable Time)
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 5
Slow method for modular multiplication

Parameters

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

5.2.4.66 BIG_XXX_moddiv()

```
void BIG_XXX_moddiv (
```

```
              BIG_XXX x,
              BIG_XXX y,
              BIG_XXX z,
              BIG_XXX n )
```

Calculate x=y/z mod n (Variable Time)
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

5.2.4.67 BIG_XXX_modsqr()

```
void BIG_XXX_modsqr (
              BIG_XXX x,
              BIG_XXX y,
              BIG_XXX n )
```

Calculate x=y^2 mod n (Variable Time)
Slow method for modular squaring

Parameters

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

5.2.4.68 BIG_XXX_modneg()

```
void BIG_XXX_modneg (
              BIG_XXX x,
              BIG_XXX y,
              BIG_XXX n )
```

Calculate x=-y mod n (Variable Time)
Modular negation

Parameters

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

5.2.4.69 BIG_XXX_modadd()

```
void BIG_XXX_modadd (
              BIG_XXX x,
              BIG_XXX y,
              BIG_XXX z,
              BIG_XXX n )
```

Calculate x=y+z mod n (Variable Time)
Slow method for modular addition

Parameters

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

5.2.4.70 BIG_XXX_jacobi()

```
int BIG_XXX_jacobi (
            BIG_XXX x,
            BIG_XXX y )
```

Calculate jacobi Symbol (x/y) (Variable Time)

Parameters

x	BIG number
y	BIG number

Returns

Jacobi symbol, -1,0 or 1

5.2.4.71 BIG_XXX_invmodp()

```
void BIG_XXX_invmodp (
            BIG_XXX x,
            BIG_XXX y,
            BIG_XXX n )
```

Calculate x=1/y mod n (Variable Time)
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

5.2.4.72 BIG_XXX_mod2m()

```
void BIG_XXX_mod2m (
            BIG_XXX x,
            int m )
```

Calculate x=x mod 2^m (Variable Time)
Truncation

Parameters

x	BIG number, on reduced mod 2^m

Parameters

m	new truncated size

5.3 bls.h File Reference

BLS Header file.

```
#include "pair_ZZZ.h"
```

Macros

- #define BGS_ZZZ MODBYTES_XXX
- #define BFS_ZZZ MODBYTES_XXX
- #define BLS_OK 0
- #define BLS_FAIL -1

Functions

- int BLS_ZZZ_INIT ()

  Initialise BLS.
- int BLS_ZZZ_KEY_PAIR_GENERATE (octet *IKM, octet *S, octet *W)

  Generate Key Pair.
- int BLS_ZZZ_CORE_SIGN (octet *SIG, octet *M, octet *S)

  Calculate a signature.
- int BLS_ZZZ_CORE_VERIFY (octet *SIG, octet *M, octet *W)

  Verify a signature.

5.3.1 Detailed Description

BLS Header file.

Author

Mike Scott

Date

28th Novemebr 2018

Allows some user configuration defines structures declares functions

5.3.2 Macro Definition Documentation

5.3.2.1 BGS_ZZZ

```
#define BGS_ZZZ MODBYTES_XXX
```

BLS Group Size

5.3.2.2 BFS_ZZZ

```
#define BFS_ZZZ MODBYTES_XXX
```

BLS Field Size

5.3.2.3 BLS_OK

```
#define BLS_OK 0
```

Function completed without error

5.3.2.4 BLS_FAIL

```
#define BLS_FAIL -1
```

Point is NOT on the curve

5.3.3 Function Documentation

5.3.3.1 BLS_ZZZ_INIT()

```
int BLS_ZZZ_INIT ( )
```

Initialise BLS.

Returns

BLS_OK if worked, otherwise BLS_FAIL

5.3.3.2 BLS_ZZZ_KEY_PAIR_GENERATE()

```
int BLS_ZZZ_KEY_PAIR_GENERATE (
            octet * IKM,
            octet * S,
            octet * W )
```

Generate Key Pair.

Parameters

IKM	is an octet containing random Initial Keying Material
S	on output a private key
W	on output a public key = S∗G, where G is fixed generator

Returns

BLS_OK

5.3.3.3 BLS_ZZZ_CORE_SIGN()

```
int BLS_ZZZ_CORE_SIGN (
            octet * SIG,
            octet * M,
            octet * S )
```

Calculate a signature.

Parameters

SIG	the ouput signature
M	is the message to be signed
S	an input private key

Returns

BLS_OK

5.3.3.4 BLS_ZZZ_CORE_VERIFY()

```
int BLS_ZZZ_CORE_VERIFY (
            octet * SIG,
            octet * M,
            octet * W )
```

Verify a signature.

Parameters

SIG	an input signature
M	is the message whose signature is to be verified.
W	an public key

Returns

BLS_OK if verified, otherwise BLS_FAIL

5.4 bls192.h File Reference

BLS Header file.

`#include "pair4_ZZZ.h"`

Macros

- #define BGS_ZZZ MODBYTES_XXX
- #define BFS_ZZZ MODBYTES_XXX
- #define BLS_OK 0
- #define BLS_FAIL -1

Functions

- int BLS_ZZZ_INIT ()

  Initialise BLS.
- int BLS_ZZZ_KEY_PAIR_GENERATE (octet ∗IKM, octet ∗S, octet ∗W)

  Generate Key Pair.
- int BLS_ZZZ_CORE_SIGN (octet ∗SIG, octet ∗M, octet ∗S)

  Calculate a signature.
- int BLS_ZZZ_CORE_VERIFY (octet ∗SIG, octet ∗M, octet ∗W)

  Verify a signature.

5.4.1 Detailed Description

BLS Header file.

Author

Mike Scott

Date

28th Novemebr 2018

Allows some user configuration defines structures declares functions

5.4.2 Macro Definition Documentation

5.4.2.1 BGS_ZZZ

```
#define BGS_ZZZ MODBYTES_XXX
```

BLS Group Size

5.4.2.2 BFS_ZZZ

```
#define BFS_ZZZ MODBYTES_XXX
```

BLS Field Size

5.4.2.3 BLS_OK

```
#define BLS_OK 0
```

Function completed without error

5.4.2.4 BLS_FAIL

```
#define BLS_FAIL -1
```

Point is NOT on the curve

5.4.3 Function Documentation

5.4.3.1 BLS_ZZZ_INIT()

```
int BLS_ZZZ_INIT ( )
```

Initialise BLS.

Returns

BLS_OK if worked, otherwise BLS_FAIL

5.4.3.2 BLS_ZZZ_KEY_PAIR_GENERATE()

```
int BLS_ZZZ_KEY_PAIR_GENERATE (
            octet * IKM,
            octet * S,
            octet * W )
```

Generate Key Pair.

Parameters

IKM	is an octet containing random Initial Keying Material
S	on output a private key
W	on output a public key = S∗G, where G is fixed generator

Returns

BLS_OK

5.4.3.3 BLS_ZZZ_CORE_SIGN()

```
int BLS_ZZZ_CORE_SIGN (
            octet * SIG,
            octet * M,
            octet * S )
```

Calculate a signature.

Parameters

SIG	the ouput signature
M	is the message to be signed
S	an input private key

Returns

BLS_OK

5.4.3.4 BLS_ZZZ_CORE_VERIFY()

```
int BLS_ZZZ_CORE_VERIFY (
            octet * SIG,
            octet * M,
            octet * W )
```

Verify a signature.

Parameters

SIG	an input signature
M	is the message whose signature is to be verified.
W	an public key

Returns

BLS_OK if verified, otherwise BLS_FAIL

5.5 bls256.h File Reference

BLS Header file.

```
#include "pair8_ZZZ.h"
```

Macros

- #define BGS_ZZZ MODBYTES_XXX
- #define BFS_ZZZ MODBYTES_XXX
- #define BLS_OK 0
- #define BLS_FAIL -1

Functions

- int BLS_ZZZ_INIT ()

    Initialise BLS.
- int BLS_ZZZ_KEY_PAIR_GENERATE (octet *IKM, octet *S, octet *W)

    Generate Key Pair.
- int BLS_ZZZ_CORE_SIGN (octet *SIG, octet *M, octet *S)

    Calculate a signature.
- int BLS_ZZZ_CORE_VERIFY (octet *SIG, octet *M, octet *W)

    Verify a signature.

5.5.1 Detailed Description

BLS Header file.

Author

Mike Scott

Date

28th Novemebr 2018

Allows some user configuration defines structures declares functions

5.5.2 Macro Definition Documentation

5.5.2.1 BGS_ZZZ

```
#define BGS_ZZZ MODBYTES_XXX
```
BLS Group Size

5.5.2.2 BFS_ZZZ

```
#define BFS_ZZZ MODBYTES_XXX
```
BLS Field Size

5.5.2.3 BLS_OK

```
#define BLS_OK 0
```
Function completed without error

5.5.2.4 BLS_FAIL

```
#define BLS_FAIL -1
```
Point is NOT on the curve

5.5.3 Function Documentation

5.5.3.1 BLS_ZZZ_INIT()

```
int BLS_ZZZ_INIT ( )
```
Initialise BLS.

Returns

BLS_OK if worked, otherwise BLS_FAIL

5.5.3.2 BLS_ZZZ_KEY_PAIR_GENERATE()

```
int BLS_ZZZ_KEY_PAIR_GENERATE (
            octet * IKM,
            octet * S,
            octet * W )
```

Generate Key Pair.

Parameters

IKM	is an octet containing random Initial Keying Material
S	on output a private key
W	on output a public key = S∗G, where G is fixed generator

Returns

BLS_OK

5.5.3.3 BLS_ZZZ_CORE_SIGN()

```
int BLS_ZZZ_CORE_SIGN (
            octet * SIG,
            octet * M,
            octet * S )
```

Calculate a signature.

Parameters

SIG	the ouput signature
M	is the message to be signed
S	an input private key

Returns

BLS_OK

5.5.3.4 BLS_ZZZ_CORE_VERIFY()

```
int BLS_ZZZ_CORE_VERIFY (
            octet * SIG,
            octet * M,
            octet * W )
```

Verify a signature.

Parameters

SIG	an input signature
M	is the message whose signature is to be verified.
W	an public key

Returns

BLS_OK if verified, otherwise BLS_FAIL

5.6 config_big.h File Reference

Config BIG Header File.

```
#include "core.h"
```

Macros

- #define MODBYTES_XXX @NB@
- #define BASEBITS_XXX @BASE@

5.6.1 Detailed Description

Config BIG Header File.

Author

Mike Scott

5.6.2 Macro Definition Documentation

5.6.2.1 MODBYTES_XXX

```
#define MODBYTES_XXX @NB@
```

Number of bytes in Modulus

5.6.2.2 BASEBITS_XXX

```
#define BASEBITS_XXX @BASE@
```

Numbers represented to base 2∗BASEBITS

5.7 config_curve.h File Reference

Config Curve Header File.

```
#include "core.h"
#include "config_field_YYY.h"
```

Macros

- #define CURVETYPE_ZZZ @CT@
- #define CURVE_A_ZZZ @CA@
- #define PAIRING_FRIENDLY_ZZZ @PF@
- #define CURVE_SECURITY_ZZZ @CS@
- #define HTC_ISO_ZZZ @HC@

5.7.1 Detailed Description

Config Curve Header File.

Author

Mike Scott

5.7.2 Macro Definition Documentation

5.7.2.1 CURVETYPE_ZZZ

```
#define CURVETYPE_ZZZ @CT@
```

Define Curve Type

5.7.2.2 CURVE_A_ZZZ

```
#define CURVE_A_ZZZ @CA@
```

Curve A parameter

5.7.2.3 PAIRING_FRIENDLY_ZZZ

```
#define PAIRING_FRIENDLY_ZZZ @PF@
```

Is curve pairing-friendly

5.7.2.4 CURVE_SECURITY_ZZZ

```
#define CURVE_SECURITY_ZZZ @CS@
```

Curve security level in AES bits

5.7.2.5 HTC_ISO_ZZZ

```
#define HTC_ISO_ZZZ @HC@
```

Use Isogenies for Hash to Curve

5.8 config_ff.h File Reference

Config FF Header File.

```
#include "core.h"
#include "config_big_XXX.h"
```

Macros

- #define FFLEN_WWW @ML@

5.8.1 Detailed Description

Config FF Header File.

Author

Mike Scott

5.8.2 Macro Definition Documentation

5.8.2.1 FFLEN_WWW

```
#define FFLEN_WWW @ML@
```

2^n multiplier of BIGBITS to specify supported Finite Field size, e.g 2048=256*2^3 where BIGBITS=256

5.9 config_field.h File Reference

Config Curve Header File.

```
#include "core.h"
#include "config_big_XXX.h"
```

Macros

- #define MBITS_YYY @NBT@
- #define PM1D2_YYY @M8@
- #define MODTYPE_YYY @MT@
- #define MAXXES_YYY @SH@
- #define QNRI_YYY @QI@
- #define RIADZ_YYY @RZ@
- #define RIADZG2A_YYY @RZ2A@
- #define RIADZG2B_YYY @RZ2B@
- #define TOWER_YYY @TW@

5.9.1 Detailed Description

Config Curve Header File.

Author

Mike Scott

5.9.2 Macro Definition Documentation

5.9.2.1 MBITS_YYY

```
#define MBITS_YYY @NBT@
```

Modulus bits

5.9.2.2 PM1D2_YYY

```
#define PM1D2_YYY @M8@
```

Largest m such that 2^m|(p-1)

5.9.2.3 MODTYPE_YYY

```
#define MODTYPE_YYY @MT@
```

Modulus type

5.9.2.4 MAXXES_YYY

```
#define MAXXES_YYY @SH@
```

Maximum excess for lazy reduction

5.9.2.5 QNRI_YYY

```
#define QNRI_YYY @QI@
```

Small Quadratic Non-Residue

5.9.2.6 RIADZ_YYY

```
#define RIADZ_YYY @RZ@
```

Z for hash to Curve

5.9.2.7 RIADZG2A_YYY

```
#define RIADZG2A_YYY @RZ2A@
```

real part of Z in G2 for Hash to Curve

5.9.2.8 RIADZG2B_YYY

```
#define RIADZG2B_YYY @RZ2B@
```

imaginary part of Z in G2 for Hash to Curve

5.9.2.9 TOWER_YYY

```
#define TOWER_YYY @TW@
```

Postive or Negative towering

5.10 core.h File Reference

Main Header File.

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

Classes

- struct hash256

    SHA256 hash function instance.
- struct hash512

    SHA384-512 hash function instance.
- struct sha3

    SHA3 hash function instance.
- struct core_aes

    AES instance.
- struct gcm

    GCM mode instance, using AES internally.
- struct csprng

    Cryptographically secure pseudo-random number generator instance.
- struct octet

    Portable representation of a big positive number.
- struct share

    Share instance.

Macros

- #define NOT_SPECIAL 0
- #define PSEUDO_MERSENNE 1
- #define MONTGOMERY_FRIENDLY 3
- #define GENERALISED_MERSENNE 2
- #define WEIERSTRASS 0
- #define EDWARDS 1
- #define MONTGOMERY 2
- #define NOT_PF 0
- #define BN_CURVE 1

- #define BLS12_CURVE 2
- #define BLS24_CURVE 3
- #define BLS48_CURVE 4
- #define D_TYPE 0
- #define M_TYPE 1
- #define FP_ZILCH 0
- #define FP_UNITY 1
- #define FP_SPARSEST 2
- #define FP_SPARSER 3
- #define FP_SPARSE 4
- #define FP_DENSE 5
- #define NEGATOWER 0
- #define POSITOWER 1
- #define MC_SHA2 2
- #define MC_SHA3 3
- #define SHA256 32
- #define SHA384 48
- #define SHA512 64
- #define SHA3_HASH224 28
- #define SHA3_HASH256 32
- #define SHA3_HASH384 48
- #define SHA3_HASH512 64
- #define SHAKE128 16
- #define SHAKE256 32
- #define RLWE_PRIME 0x3001
- #define RLWE_LGN 10
- #define RLWE_ND 0xF7002FFF
- #define RLWE_ONE 0x2AC8
- #define RLWE_R2MODP 0x1620
- #define ECB 0
- #define CBC 1
- #define CFB1 2
- #define CFB2 3
- #define CFB4 5
- #define OFB1 14
- #define OFB2 15
- #define OFB4 17
- #define OFB8 21
- #define OFB16 29
- #define CTR1 30
- #define CTR2 31
- #define CTR4 33
- #define CTR8 37
- #define CTR16 45
- #define uchar unsigned char
- #define GCM_ACCEPTING_HEADER 0
- #define GCM_ACCEPTING_CIPHER 1
- #define GCM_NOT_ACCEPTING_MORE 2
- #define GCM_FINISHED 3
- #define GCM_ENCRYPTING 0
- #define GCM_DECRYPTING 1
- #define NK 21
- #define NJ 6
- #define NV 8

Typedefs

- typedef hash512 hash384

    SHA384 hash function instance.

Functions

- void OCT_output (octet ∗O)

    Formats and outputs an octet to the console in hex.
- void OCT_output_string (octet ∗O)

    Formats and outputs an octet to the console as a character string.
- void OCT_clear (octet ∗O)

    Wipe clean an octet.
- void OCT_reverse (octet ∗O)

    Reverse bytes in an octet.
- int OCT_comp (octet ∗O, octet ∗P)

    Compare two octets.
- int OCT_ncomp (octet ∗O, octet ∗P, int n)

    Compare first n bytes of two octets.
- void OCT_jstring (octet ∗O, char ∗s)

    Join from a C string to end of an octet.
- void OCT_jbytes (octet ∗O, char ∗s, int n)

    Join bytes to end of an octet.
- void OCT_jbyte (octet ∗O, int b, int n)

    Join single byte to end of an octet, repeated n times.
- void OCT_joctet (octet ∗O, octet ∗P)

    Join one octet to the end of another.
- void OCT_xor (octet ∗O, octet ∗P)

    XOR common bytes of a pair of Octets.
- void OCT_empty (octet ∗O)

    reset Octet to zero length
- int OCT_pad (octet ∗O, int n)

    Pad out an Octet to the given length.
- void OCT_tobase64 (char ∗b, octet ∗O)

    Convert an Octet to printable base64 number.
- void OCT_frombase64 (octet ∗O, char ∗b)

    Populate an Octet from base64 number.
- void OCT_copy (octet ∗O, octet ∗P)

    Copy one Octet into another.
- void OCT_xorbyte (octet ∗O, int m)

    XOR every byte of an octet with input m.
- void OCT_chop (octet ∗O, octet ∗P, int n)

    Chops Octet into two, leaving first n bytes in O, moving the rest to P.
- void OCT_jint (octet ∗O, unsigned int m, int n)

    Join n bytes of integer m to end of Octet O (big endian)
- void OCT_rand (octet ∗O, csprng ∗R, int n)

    Create an Octet from bytes taken from a random number generator.
- void OCT_shl (octet ∗O, int n)

    Shifts Octet left by n bytes.
- void OCT_fromHex (octet ∗dst, char ∗src)

    Convert a hex number to an Octet.
- void OCT_toHex (octet ∗src, char ∗dst)

Convert an Octet to printable hex number.
- void OCT_toStr (octet ∗src, char ∗dst)

  Convert an Octet to string.
- void HASH256_init (hash256 ∗H)

  Initialise an instance of SHA256.
- void HASH256_process (hash256 ∗H, int b)

  Add a byte to the hash.
- void HASH256_hash (hash256 ∗H, char ∗h)

  Generate 32-byte final hash.
- void HASH256_continuing_hash (hash256 ∗H, char ∗h)

  Generate 32-byte intermediate hash.
- void HASH384_init (hash384 ∗H)

  Initialise an instance of SHA384.
- void HASH384_process (hash384 ∗H, int b)

  Add a byte to the hash.
- void HASH384_hash (hash384 ∗H, char ∗h)

  Generate 48-byte final hash.
- void HASH384_continuing_hash (hash384 ∗H, char ∗h)

  Generate 48-byte intermediate hash.
- void HASH512_init (hash512 ∗H)

  Initialise an instance of SHA512.
- void HASH512_process (hash512 ∗H, int b)

  Add a byte to the hash.
- void HASH512_hash (hash512 ∗H, char ∗h)

  Generate 64-byte final hash.
- void HASH512_continuing_hash (hash512 ∗H, char ∗h)

  Generate 64-byte intermediate hash.
- void SHA3_init (sha3 ∗H, int t)

  Initialise an instance of SHA3.
- void SHA3_process (sha3 ∗H, int b)

  process a byte for SHA3
- void SHA3_hash (sha3 ∗H, char ∗h)

  create fixed length final hash output of SHA3
- void SHA3_continuing_hash (sha3 ∗H, char ∗h)

  create fixed length intermediate hash output of SHA3
- void SHA3_shake (sha3 ∗H, char ∗h, int len)

  create variable length final hash output of SHA3
- void SHA3_continuing_shake (sha3 ∗H, char ∗h, int len)

  create variable length intermediate hash output of SHA3
- void SHA3_squeeze (sha3 ∗H, char ∗h, int len)

  generate further hash output of SHA3
- void GPhash (int hash, int hlen, octet ∗w, int olen, int pad, octet ∗p, int n, octet ∗x)

  General Purpose Hashing function.
- void SPhash (int hash, int hlen, octet ∗w, octet ∗p)

  Simple purpose Hashing function.
- void HMAC (int hash, int hlen, octet ∗T, int len, octet ∗K, octet ∗M)

  HMAC function.
- void HKDF_Extract (int hash, int hlen, octet ∗K, octet ∗P, octet ∗S)

  HKDF_Extract function.
- void HKDF_Expand (int hash, int hlen, octet ∗E, int olen, octet ∗K, octet ∗I)

  HKDF_Extract function.

- void XOF_Expand (int hlen, octet ∗E, int olen, octet ∗P, octet ∗S)

  XOF_Expand function.
- void XMD_Expand (int hash, int hlen, octet ∗E, int olen, octet ∗P, octet ∗S)

  XOF_Expand function.
- void KDF2 (int hash, int hlen, octet ∗K, int len, octet ∗Z, octet ∗P)

  Key Derivation Function - generates key K from inputs Z and P.
- void PBKDF2 (int hash, int hlen, octet ∗K, int len, octet ∗P, octet ∗S, int rep)

  Password Based Key Derivation Function - generates key K from password, salt and repeat counter.
- int PKCS15 (int h, octet ∗M, octet ∗W)

  PKCS V1.5 padding of a message prior to RSA signature.
- int PSS_ENCODE (int h, octet ∗M, csprng ∗R, octet ∗W)

  PSS padding of a message prior to RSA signature.
- int PSS_VERIFY (int h, octet ∗M, octet ∗W)

  PSS verification.
- 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.
- void AES_reset (core_aes ∗A, int m, char ∗iv)

  Reset AES mode or IV.
- void AES_getreg (core_aes ∗A, char ∗f)

  Extract chaining vector from CORE_AES instance.
- int AES_init (core_aes ∗A, int m, int n, char ∗k, char ∗iv)

  Initialise an instance of CORE_AES and its mode of operation.
- void AES_ecb_encrypt (core_aes ∗A, uchar ∗b)

  Encrypt a single 16 byte block in ECB mode.
- void AES_ecb_decrypt (core_aes ∗A, uchar ∗b)

  Decrypt a single 16 byte block in ECB mode.
- unsign32 AES_encrypt (core_aes ∗A, char ∗b)

  Encrypt a single 16 byte block in active mode.
- unsign32 AES_decrypt (core_aes ∗A, char ∗b)

  Decrypt a single 16 byte block in active mode.
- void AES_end (core_aes ∗A)

  Clean up after application of AES.
- 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.
- void GCM_init (gcm ∗G, int nk, char ∗k, int n, char ∗iv)

  Initialise an instance of AES-GCM mode.
- int GCM_add_header (gcm ∗G, char ∗b, int n)

  Add header (material to be authenticated but not encrypted)
- int GCM_add_plain (gcm ∗G, char ∗c, char ∗p, int n)

  Add plaintext and extract ciphertext.
- int GCM_add_cipher (gcm ∗G, char ∗p, char ∗c, int n)

  Add ciphertext and extract plaintext.
- void GCM_finish (gcm ∗G, char ∗t)

  Finish off and extract authentication tag (HMAC)
- 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.

- share getshare (int id, int nsr, octet *S, octet *M, octet *R)

  Get a share of a message.
- int recover (octet *M, share *S)

  Recover message from shares.
- void RAND_seed (csprng *R, int n, char *b)

  Seed a random number generator from an array of bytes.
- void RAND_clean (csprng *R)

  Delete all internal state of a random number generator.
- int RAND_byte (csprng *R)

  Return a random byte from a random number generator.

5.10.1 Detailed Description

Main Header File.

Author

Mike Scott

5.10.2 Macro Definition Documentation

5.10.2.1 NOT_SPECIAL

```
#define NOT_SPECIAL 0
```

Modulus of no exploitable form

5.10.2.2 PSEUDO_MERSENNE

```
#define PSEUDO_MERSENNE 1
```

Pseudo-mersenne modulus of form 2^n-c

5.10.2.3 MONTGOMERY_FRIENDLY

```
#define MONTGOMERY_FRIENDLY 3
```

Montgomery Friendly modulus of form $2^a(2^b-c)-1$

5.10.2.4 GENERALISED_MERSENNE

```
#define GENERALISED_MERSENNE 2
```

Generalised-mersenne modulus of form 2^n-2^m-1, GOLDILOCKS only

5.10.2.5 WEIERSTRASS

```
#define WEIERSTRASS 0
```

Short Weierstrass form curve

5.10.2.6 EDWARDS

```
#define EDWARDS 1
```

Edwards or Twisted Edwards curve

5.10.2.7 MONTGOMERY

```
#define MONTGOMERY 2
```

Montgomery form curve

5.10.2.8 NOT_PF

```
#define NOT_PF 0
```

Not a pairing friendly curve

5.10.2.9 BN_CURVE

```
#define BN_CURVE 1
```

BN pairing-friendy curve

5.10.2.10 BLS12_CURVE

```
#define BLS12_CURVE 2
```

BLS12 pairing-friendy curve

5.10.2.11 BLS24_CURVE

```
#define BLS24_CURVE 3
```

BLS24 pairing-friendy curve

5.10.2.12 BLS48_CURVE

```
#define BLS48_CURVE 4
```

BLS48 pairing-friendy curve

5.10.2.13 D_TYPE

```
#define D_TYPE 0
```

D-Type pairing-friendy curve

5.10.2.14 M_TYPE

```
#define M_TYPE 1
```

M-Type pairing-friendy curve

5.10.2.15 FP_ZILCH

```
#define FP_ZILCH 0
```

FP extension is zero

5.10.2.16 FP_UNITY

```
#define FP_UNITY 1
```

FP extension is one

5.10.2.17 FP_SPARSEST

```
#define FP_SPARSEST 2
```

FP extension is sparsest

5.10.2.18 FP_SPARSER

```
#define FP_SPARSER 3
```

FP extension is sparser

5.10.2.19 FP_SPARSE

```
#define FP_SPARSE 4
```

FP extension is sparse

5.10.2.20 FP_DENSE

```
#define FP_DENSE 5
```

FP extension is dense

5.10.2.21 NEGATOWER

```
#define NEGATOWER 0
```

Negative towering

5.10.2.22 POSITOWER

```
#define POSITOWER 1
```

Positive towering

5.10.2.23 MC_SHA2

```
#define MC_SHA2 2
```

SHA2 family member

5.10.2.24 MC_SHA3

```
#define MC_SHA3 3
```

SHA3 family member

5.10.2.25 SHA256

```
#define SHA256 32
```

SHA-256 hashing

5.10.2.26 SHA384

```
#define SHA384 48
```

SHA-384 hashing

5.10.2.27 SHA512

```
#define SHA512 64
```

SHA-512 hashing

5.10.2.28 SHA3_HASH224

```
#define SHA3_HASH224 28
```

SHA3 224 bit hash

5.10.2.29 SHA3_HASH256

```
#define SHA3_HASH256 32
```

SHA3 256 bit hash

5.10.2.30 SHA3_HASH384

```
#define SHA3_HASH384 48
```

SHA3 384 bit hash

5.10.2.31 SHA3_HASH512

```
#define SHA3_HASH512 64
```

SHA3 512 bit hash

5.10.2.32 SHAKE128

```
#define SHAKE128 16
```

SHAKE128 hash

5.10.2.33 SHAKE256

```
#define SHAKE256 32
```

SHAKE256 hash

5.10.2.34 RLWE_PRIME

```
#define RLWE_PRIME 0x3001
```

q in Hex

5.10.2.35 RLWE_LGN

```
#define RLWE_LGN 10
```

Degree n=2^LGN

5.10.2.36 RLWE_ND

```
#define RLWE_ND 0xF7002FFF
```

1/(R-q) mod R

5.10.2.37 RLWE_ONE

```
#define RLWE_ONE 0x2AC8
```

R mod q

5.10.2.38 RLWE_R2MODP

```
#define RLWE_R2MODP 0x1620
```

R^2 mod q

5.10.2.39 ECB

```
#define ECB 0
```

Electronic Code Book

5.10.2.40 CBC

```
#define CBC 1
```

Cipher Block Chaining

5.10.2.41 CFB1

```
#define CFB1 2
```

Cipher Feedback - 1 byte

5.10.2.42 CFB2

```
#define CFB2 3
```

Cipher Feedback - 2 bytes

5.10.2.43 CFB4

```
#define CFB4 5
```

Cipher Feedback - 4 bytes

5.10.2.44 OFB1

```
#define OFB1 14
```

Output Feedback - 1 byte

5.10.2.45 OFB2

```
#define OFB2 15
```

Output Feedback - 2 bytes

5.10.2.46 OFB4

```
#define OFB4 17
```

Output Feedback - 4 bytes

5.10.2.47 OFB8

```
#define OFB8 21
```

Output Feedback - 8 bytes

5.10.2.48 OFB16

```
#define OFB16 29
```

Output Feedback - 16 bytes

5.10.2.49 CTR1

```
#define CTR1 30
```

Counter Mode - 1 byte

5.10.2.50 CTR2

```
#define CTR2 31
```

Counter Mode - 2 bytes

5.10.2.51 CTR4

```
#define CTR4 33
```

Counter Mode - 4 bytes

5.10.2.52 CTR8

```
#define CTR8 37
```

Counter Mode - 8 bytes

5.10.2.53 CTR16

```
#define CTR16 45
```

Counter Mode - 16 bytes

5.10.2.54 uchar

```
#define uchar unsigned char
```

Unsigned char

5.10.2.55 GCM_ACCEPTING_HEADER

```
#define GCM_ACCEPTING_HEADER 0
```

GCM status

5.10.2.56 GCM_ACCEPTING_CIPHER

```
#define GCM_ACCEPTING_CIPHER 1
```

GCM status

5.10.2.57 GCM_NOT_ACCEPTING_MORE

```
#define GCM_NOT_ACCEPTING_MORE 2
```

GCM status

5.10.2.58 GCM_FINISHED

```
#define GCM_FINISHED 3
```

GCM status

5.10.2.59 GCM_ENCRYPTING

```
#define GCM_ENCRYPTING 0
```

GCM mode

5.10.2.60 GCM_DECRYPTING

```
#define GCM_DECRYPTING 1
```

GCM mode

5.10.2.61 NK

```
#define NK 21
```

PRNG constant

5.10.2.62 NJ

```
#define NJ 6
```

PRNG constant

5.10.2.63 NV

```
#define NV 8
```

PRNG constant

5.10.3 Function Documentation

5.10.3.1 OCT_output()

```
void OCT_output (
            octet * O )
```

Formats and outputs an octet to the console in hex.

Parameters

O	Octet to be output

5.10.3.2 OCT_output_string()

```
void OCT_output_string (
            octet * O )
```

Formats and outputs an octet to the console as a character string.

Parameters

O	Octet to be output

5.10.3.3 OCT_clear()

```
void OCT_clear (
            octet * O )
```

Wipe clean an octet.

Parameters

O	Octet to be cleaned

5.10.3.4 OCT_reverse()

```
void OCT_reverse (
            octet * O )
```

Reverse bytes in an octet.

Parameters

O	Octet to be reversed

5.10.3.5 OCT_comp()

```
int OCT_comp (
            octet * O,
            octet * P )
```

Compare two octets.

Parameters

O	first Octet to be compared
P	second Octet to be compared

Returns

1 if equal, else 0

5.10.3.6 OCT_ncomp()

```
int OCT_ncomp (
            octet * O,
```

```
octet * P,
int n )
```

Compare first n bytes of two octets.

Parameters

O	first Octet to be compared
P	second Octet to be compared
n	number of bytes to compare

Returns

1 if equal, else 0

5.10.3.7 OCT_jstring()

```
void OCT_jstring (
        octet * O,
        char * s )
```

Join from a C string to end of an octet.
Truncates if there is no room

Parameters

O	Octet to be written to
s	zero terminated string to be joined to octet

5.10.3.8 OCT_jbytes()

```
void OCT_jbytes (
        octet * O,
        char * s,
        int n )
```

Join bytes to end of an octet.
Truncates if there is no room

Parameters

O	Octet to be written to
s	bytes to be joined to end of octet
n	number of bytes to join

5.10.3.9 OCT_jbyte()

```
void OCT_jbyte (
        octet * O,
        int b,
        int n )
```

Join single byte to end of an octet, repeated n times.
Truncates if there is no room

Parameters

O	Octet to be written to
b	byte to be joined to end of octet
n	number of times b is to be joined

5.10.3.10 OCT_joctet()

```
void OCT_joctet (
            octet * O,
            octet * P )
```

Join one octet to the end of another.
Truncates if there is no room

Parameters

O	Octet to be written to
P	Octet to be joined to the end of O

5.10.3.11 OCT_xor()

```
void OCT_xor (
            octet * O,
            octet * P )
```

XOR common bytes of a pair of Octets.

Parameters

O	Octet - on exit = O xor P
P	Octet to be xored into O

5.10.3.12 OCT_empty()

```
void OCT_empty (
            octet * O )
```

reset Octet to zero length

Parameters

O	Octet to be emptied

5.10.3.13 OCT_pad()

```
int OCT_pad (
            octet * O,
            int n )
```

Pad out an Octet to the given length.
Padding is done by inserting leading zeros, so abcd becomes 00abcd

Parameters

O	Octet to be padded
n	new length of Octet

5.10.3.14 OCT_tobase64()

```
void OCT_tobase64 (
            char * b,
            octet * O )
```

Convert an Octet to printable base64 number.

Parameters

b	zero terminated byte array to take base64 conversion
O	Octet to be converted

5.10.3.15 OCT_frombase64()

```
void OCT_frombase64 (
            octet * O,
            char * b )
```

Populate an Octet from base64 number.

Parameters

O	Octet to be populated
b	zero terminated base64 string

5.10.3.16 OCT_copy()

```
void OCT_copy (
            octet * O,
            octet * P )
```

Copy one Octet into another.

Parameters

O	Octet to be copied to
P	Octet to be copied from

5.10.3.17 OCT_xorbyte()

```
void OCT_xorbyte (
            octet * O,
            int m )
```

XOR every byte of an octet with input m.

Parameters

O	Octet
m	byte to be XORed with every byte of O

5.10.3.18 OCT_chop()

```
void OCT_chop (
            octet * O,
            octet * P,
            int n )
```

Chops Octet into two, leaving first n bytes in O, moving the rest to P.

Parameters

O	Octet to be chopped
P	new Octet to be created
n	number of bytes to chop off O

5.10.3.19 OCT_jint()

```
void OCT_jint (
            octet * O,
            unsigned int m,
            int n )
```

Join n bytes of integer m to end of Octet O (big endian)
Typically n is 4 for a 32-bit integer

Parameters

O	Octet to be appended to
m	integer to be appended to O
n	number of bytes in m

5.10.3.20 OCT_rand()

```
void OCT_rand (
            octet * O,
            csprng * R,
            int n )
```

Create an Octet from bytes taken from a random number generator.
Truncates if there is no room

Parameters

O	Octet to be populated
R	an instance of a Cryptographically Secure Random Number Generator
n	number of bytes to extracted from R

5.10.3.21 OCT_shl()

```
void OCT_shl (
        octet * O,
        int n )
```

Shifts Octet left by n bytes.
Leftmost bytes disappear

Parameters

O	Octet to be shifted
n	number of bytes to shift

5.10.3.22 OCT_fromHex()

```
void OCT_fromHex (
        octet * dst,
        char * src )
```

Convert a hex number to an Octet.

Parameters

dst	Octet
src	Hex string to be converted

5.10.3.23 OCT_toHex()

```
void OCT_toHex (
        octet * src,
        char * dst )
```

Convert an Octet to printable hex number.

Parameters

dst	hex value
src	Octet to be converted

5.10.3.24 OCT_toStr()

```
void OCT_toStr (
        octet * src,
        char * dst )
```

Convert an Octet to string.

Parameters

dst	string value
src	Octet to be converted

5.10.3.25 HASH256_init()

```
void HASH256_init (
            hash256 * H )
```

Initialise an instance of SHA256.

Parameters

H	an instance SHA256

5.10.3.26 HASH256_process()

```
void HASH256_process (
            hash256 * H,
            int b )
```

Add a byte to the hash.

Parameters

H	an instance SHA256
b	byte to be included in hash

5.10.3.27 HASH256_hash()

```
void HASH256_hash (
            hash256 * H,
            char * h )
```

Generate 32-byte final hash.

Parameters

H	an instance SHA256
h	is the output 32-byte hash

5.10.3.28 HASH256_continuing_hash()

```
void HASH256_continuing_hash (
            hash256 * H,
            char * h )
```

Generate 32-byte intermediate hash.

Parameters

H	an instance SHA256
h	is the output 32-byte hash

5.10.3.29 HASH384_init()

```
void HASH384_init (
            hash384 * H )
```

Initialise an instance of SHA384.

Parameters

H	an instance SHA384

5.10.3.30 HASH384_process()

```
void HASH384_process (
            hash384 * H,
            int b )
```

Add a byte to the hash.

Parameters

H	an instance SHA384
b	byte to be included in hash

5.10.3.31 HASH384_hash()

```
void HASH384_hash (
            hash384 * H,
            char * h )
```

Generate 48-byte final hash.

Parameters

H	an instance SHA384
h	is the output 48-byte hash

5.10.3.32 HASH384_continuing_hash()

```
void HASH384_continuing_hash (
            hash384 * H,
            char * h )
```

Generate 48-byte intermediate hash.

Parameters

H	an instance SHA384
h	is the output 48-byte hash

5.10.3.33 HASH512_init()

```
void HASH512_init (
            hash512 * H )
```

Initialise an instance of SHA512.

Parameters

H	an instance SHA512

5.10.3.34 HASH512_process()

```
void HASH512_process (
        hash512 * H,
        int b )
```

Add a byte to the hash.

Parameters

H	an instance SHA512
b	byte to be included in hash

5.10.3.35 HASH512_hash()

```
void HASH512_hash (
        hash512 * H,
        char * h )
```

Generate 64-byte final hash.

Parameters

H	an instance SHA512
h	is the output 64-byte hash

5.10.3.36 HASH512_continuing_hash()

```
void HASH512_continuing_hash (
        hash512 * H,
        char * h )
```

Generate 64-byte intermediate hash.

Parameters

H	an instance SHA512
h	is the output 64-byte hash

5.10.3.37 SHA3_init()

```
void SHA3_init (
        sha3 * H,
        int t )
```

Initialise an instance of SHA3.

Parameters

H	an instance SHA3

Parameters

t	the instance type

5.10.3.38 SHA3_process()

```
void SHA3_process (
            sha3 * H,
            int b )
```

process a byte for SHA3

Parameters

H	an instance SHA3
b	a byte of date to be processed

5.10.3.39 SHA3_hash()

```
void SHA3_hash (
            sha3 * H,
            char * h )
```

create fixed length final hash output of SHA3

Parameters

H	an instance SHA3
h	a byte array to take hash

5.10.3.40 SHA3_continuing_hash()

```
void SHA3_continuing_hash (
            sha3 * H,
            char * h )
```

create fixed length intermediate hash output of SHA3

Parameters

H	an instance SHA3
h	a byte array to take hash

5.10.3.41 SHA3_shake()

```
void SHA3_shake (
            sha3 * H,
            char * h,
            int len )
```

create variable length final hash output of SHA3

Parameters

H	an instance SHA3
h	a byte array to take hash
len	is the length of the hash

5.10.3.42 SHA3_continuing_shake()

```
void SHA3_continuing_shake (
            sha3 * H,
            char * h,
            int len )
```

create variable length intermediate hash output of SHA3

Parameters

H	an instance SHA3
h	a byte array to take hash
len	is the length of the hash

5.10.3.43 SHA3_squeeze()

```
void SHA3_squeeze (
            sha3 * H,
            char * h,
            int len )
```

generate further hash output of SHA3

Parameters

H	an instance SHA3
h	a byte array to take hash
len	is the length of the hash

5.10.3.44 GPhash()

```
void GPhash (
            int hash,
            int hlen,
            octet * w,
            int olen,
            int pad,
            octet * p,
            int n,
            octet * x )
```

General Purpose Hashing function.

Parameters

hash	the hash family (SHA2 or SHA3)
hlen	the hash function output length (32,48 or 64)

Parameters

w	an output octet
olen	the output length
pad	zero padding
p	an input octet
n	an input 32-bit integer
x	an optional input octet

5.10.3.45 SPhash()

```
void SPhash (
        int hash,
        int hlen,
        octet * w,
        octet * p )
```

Simple purpose Hashing function.

Parameters

hash	the hash family (SHA2 or SHA3)
hlen	the hash function output length (32,48 or 64)
w	an output octet
p	an input octet

5.10.3.46 HMAC()

```
void HMAC (
        int hash,
        int hlen,
        octet * T,
        int len,
        octet * K,
        octet * M )
```

HMAC function.

Parameters

hash	the hash family (SHA2 or SHA3)
hlen	the hash function output length (32,48 or 64)
T	an output tag
len	the tag length
K	an input key, or salt
M	an input message

5.10.3.47 HKDF_Extract()

```
void HKDF_Extract (
        int hash,
```

```
int hlen,
octet * K,
octet * P,
octet * S )
```

HKDF_Extract function.

Parameters

hash	the hash family (SHA2 or SHA3)
hlen	the hash function output length (32,48 or 64)
K	an output Key
P	public input salt
S	raw secret keying material

5.10.3.48 HKDF_Expand()

```
void HKDF_Expand (
            int hash,
            int hlen,
            octet * E,
            int olen,
            octet * K,
            octet * I )
```

HKDF_Extract function.

Parameters

hash	the hash family (SHA2 or SHA3)
hlen	the hash function output length (32,48 or 64)
E	an expanded output Key
olen	is the desired length of the expanded key
K	is the fixed length input key
I	is public context information

5.10.3.49 XOF_Expand()

```
void XOF_Expand (
            int hlen,
            octet * E,
            int olen,
            octet * P,
            octet * S )
```

XOF_Expand function.

Parameters

hlen	the SHA3 output length (16 or 32)
E	an expanded messsage
olen	is the desired length of the expanded key
P	is Domain Separator
S	input message

5.10.3.50 XMD_Expand()

```
void XMD_Expand (
            int hash,
            int hlen,
            octet * E,
            int olen,
            octet * P,
            octet * S )
```

XOF_Expand function.

Parameters

hash	the hash family (SHA2 or SHA3)
hlen	the SHA3 output length (16 or 32)
E	an expanded messsage
olen	is the desired length of the expanded key
P	is Domain Separator
S	input message

5.10.3.51 KDF2()

```
void KDF2 (
            int hash,
            int hlen,
            octet * K,
            int len,
            octet * Z,
            octet * P )
```

Key Derivation Function - generates key K from inputs Z and P.
IEEE-1363 KDF2 Key Derivation Function.

Parameters

hash	is the hash family (SHA2 or SHA3)
hlen	the hash function output length (32,48 or 64)
Z	input octet
P	input key derivation parameters - can be NULL
len	is output desired length of key
K	is the derived key

5.10.3.52 PBKDF2()

```
void PBKDF2 (
            int hash,
            int hlen,
            octet * K,
            int len,
            octet * P,
            octet * S,
```

```
int rep )
```

Password Based Key Derivation Function - generates key K from password, salt and repeat counter.
PBKDF2 Password Based Key Derivation Function.

Parameters

hash	is the hash family (SHA2 or SHA3)
hlen	the hash function output length (32,48 or 64)
P	input password
S	input salt
rep	Number of times to be iterated.
len	is output desired length
K	is the derived key

5.10.3.53 PKCS15()

```
int PKCS15 (
        int h,
        octet * M,
        octet * W )
```

PKCS V1.5 padding of a message prior to RSA signature.

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

5.10.3.54 PSS_ENCODE()

```
int PSS_ENCODE (
        int h,
        octet * M,
        csprng * R,
        octet * W )
```

PSS padding of a message prior to RSA signature.

Parameters

h	is the hash type
M	is the input message
R	is a pointer to a cryptographically secure random number generator
W	is the output encoding, ready for RSA signature

Returns

1 if OK, else 0

5.10.3.55 PSS_VERIFY()

```
int PSS_VERIFY (
            int h,
            octet * M,
            octet * W )
```

PSS verification.

Parameters

h	is the hash type
M	is the message
W	is the message encoding

Returns

1 if OK, else 0

5.10.3.56 OAEP_ENCODE()

```
int OAEP_ENCODE (
            int h,
            octet * M,
            csprng * R,
            octet * P,
            octet * F )
```

OAEP padding of a message prior to RSA encryption.

Parameters

h	is the hash type
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

1 if OK, else 0

5.10.3.57 OAEP_DECODE()

```
int OAEP_DECODE (
            int h,
            octet * P,
            octet * F )
```

OAEP unpadding of a message after RSA decryption.
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

1 if OK, else 0

5.10.3.58 AES_reset()

```
void AES_reset (
            core_aes * A,
            int m,
            char * iv )
```

Reset AES mode or IV.

Parameters

A	an instance of the CORE_AES
m	is the new active mode of operation (ECB, CBC, OFB, CFB etc)
iv	the new Initialisation Vector

5.10.3.59 AES_getreg()

```
void AES_getreg (
            core_aes * A,
            char * f )
```

Extract chaining vector from CORE_AES instance.

Parameters

A	an instance of the CORE_AES
f	the extracted chaining vector

5.10.3.60 AES_init()

```
int AES_init (
            core_aes * A,
            int m,
            int n,
            char * k,
            char * iv )
```

Initialise an instance of CORE_AES and its mode of operation.

Parameters

A	an instance CORE_AES
m	is the active mode of operation (ECB, CBC, OFB, CFB etc)
n	is the key length in bytes, 16, 24 or 32
k	the AES key as an array of 16 bytes
iv	the Initialisation Vector

Returns

0 for invalid n

5.10.3.61 AES_ecb_encrypt()

```
void AES_ecb_encrypt (
            core_aes * A,
            uchar * b )
```

Encrypt a single 16 byte block in ECB mode.

Parameters

A	an instance of the CORE_AES
b	is an array of 16 plaintext bytes, on exit becomes ciphertext

5.10.3.62 AES_ecb_decrypt()

```
void AES_ecb_decrypt (
            core_aes * A,
            uchar * b )
```

Decrypt a single 16 byte block in ECB mode.

Parameters

A	an instance of the CORE_AES
b	is an array of 16 cipherext bytes, on exit becomes plaintext

5.10.3.63 AES_encrypt()

```
unsign32 AES_encrypt (
            core_aes * A,
            char * b )
```

Encrypt a single 16 byte block in active mode.

Parameters

A	an instance of the CORE_AES
b	is an array of 16 plaintext bytes, on exit becomes ciphertext

Returns

0, or overflow bytes from CFB mode

5.10.3.64 AES_decrypt()

```
unsign32 AES_decrypt (
            core_aes * A,
            char * b )
```

Decrypt a single 16 byte block in active mode.

Parameters

A	an instance of the CORE_AES
b	is an array of 16 ciphertext bytes, on exit becomes plaintext

Returns

0, or overflow bytes from CFB mode

5.10.3.65 AES_end()

```
void AES_end (
            core_aes * A )
```

Clean up after application of AES.

Parameters

A	an instance of the CORE_AES

5.10.3.66 AES_CBC_IV0_ENCRYPT()

```
void AES_CBC_IV0_ENCRYPT (
            octet * K,
            octet * P,
            octet * C )
```

AES encrypts a plaintext to a ciphtertext.
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

5.10.3.67 AES_CBC_IV0_DECRYPT()

```
int AES_CBC_IV0_DECRYPT (
            octet * K,
            octet * C,
            octet * P )
```

AES encrypts a plaintext to a ciphtertext.
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

5.10.3.68 GCM_init()

```
void GCM_init (
            gcm * G,
            int nk,
            char * k,
            int n,
            char * iv )
```

Initialise an instance of AES-GCM mode.

Parameters

G	an instance AES-GCM
nk	is the key length in bytes, 16, 24 or 32
k	the AES key as an array of 16 bytes
n	the number of bytes in the Initialisation Vector (IV)
iv	the IV

5.10.3.69 GCM_add_header()

```
int GCM_add_header (
            gcm * G,
            char * b,
            int n )
```

Add header (material to be authenticated but not encrypted)

Note that this function can be called any number of times with n a multiple of 16, and then one last time with any value for n

Parameters

G	an instance AES-GCM
b	is the header material to be added
n	the number of bytes in the header

5.10.3.70 GCM_add_plain()

```
int GCM_add_plain (
            gcm * G,
            char * c,
            char * p,
            int n )
```

Add plaintext and extract ciphertext.

Note that this function can be called any number of times with n a multiple of 16, and then one last time with any value for n

Parameters

G	an instance AES-GCM
c	is the ciphertext generated

Parameters

p	is the plaintext material to be added
n	the number of bytes in the plaintext

5.10.3.71 GCM_add_cipher()

```
int GCM_add_cipher (
            gcm * G,
            char * p,
            char * c,
            int n )
```

Add ciphertext and extract plaintext.

Note that this function can be called any number of times with n a multiple of 16, and then one last time with any value for n

Parameters

G	an instance AES-GCM
p	is the plaintext generated
c	is the ciphertext material to be added
n	the number of bytes in the ciphertext

5.10.3.72 GCM_finish()

```
void GCM_finish (
            gcm * G,
            char * t )
```

Finish off and extract authentication tag (HMAC)

Parameters

G	is an active instance AES-GCM
t	is the output 16 byte authentication tag

5.10.3.73 AES_GCM_ENCRYPT()

```
void AES_GCM_ENCRYPT (
            octet * K,
            octet * IV,
            octet * H,
            octet * P,
            octet * C,
            octet * T )
```

AES-GCM Encryption.

Parameters

K	AES key
IV	Initialization vector
H	Header

Parameters

P	Plaintext
C	Ciphertext
T	Checksum

5.10.3.74 AES_GCM_DECRYPT()

```
void AES_GCM_DECRYPT (
            octet * K,
            octet * IV,
            octet * H,
            octet * C,
            octet * P,
            octet * T )
```

AES-GCM Decryption.

Parameters

K	AES key
IV	Initialization vector
H	Header
P	Plaintext
C	Ciphertext
T	Checksum

5.10.3.75 getshare()

```
share getshare (
            int id,
            int nsr,
            octet * S,
            octet * M,
            octet * R )
```

Get a share of a message.

Parameters

id	unique share ID
nsr	number of shares needed for message recovery
S	the output share as an octet
M	the Message octet to be shared
R	an octet of random seed bytes

Returns

a share structure

5.10.3.76 recover()

```
int recover (
```

```
            octet * M,
            share * S )
```
Recover message from shares.

Parameters

M	the recovered Message octet
S	an array of sufficient shares

Returns

0 on success else -1

5.10.3.77 RAND_seed()

```
void RAND_seed (
            csprng * R,
            int n,
            char * b )
```
Seed a random number generator from an array of bytes.
The provided seed should be truly random

Parameters

R	an instance of a Cryptographically Secure Random Number Generator
n	the number of seed bytes provided
b	an array of seed bytes

5.10.3.78 RAND_clean()

```
void RAND_clean (
            csprng * R )
```
Delete all internal state of a random number generator.

Parameters

R	an instance of a Cryptographically Secure Random Number Generator

5.10.3.79 RAND_byte()

```
int RAND_byte (
            csprng * R )
```
Return a random byte from a random number generator.

Parameters

R	an instance of a Cryptographically Secure Random Number Generator

Returns

a random byte

5.11 ecdh.h File Reference

ECDH Header file for implementation of standard EC protocols.

```
#include "ecp_ZZZ.h"
```

Macros

- #define EGS_ZZZ MODBYTES_XXX
- #define EFS_ZZZ MODBYTES_XXX
- #define ECDH_OK 0
- #define ECDH_INVALID_PUBLIC_KEY -2
- #define ECDH_ERROR -3

Functions

- int ECP_ZZZ_IN_RANGE (octet *s)

  Test if group element in correct range.
- int ECP_ZZZ_KEY_PAIR_GENERATE (csprng *R, octet *s, octet *W)

  Generate an ECC public/private key pair.
- int ECP_ZZZ_PUBLIC_KEY_VALIDATE (octet *W)

  Validate an ECC public key.
- int ECP_ZZZ_SVDP_DH (octet *s, octet *W, octet *K, int type)

  Generate Diffie-Hellman shared key.
- void ECP_ZZZ_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_ZZZ_ECIES_DECRYPT (int h, octet *P1, octet *P2, octet *V, octet *C, octet *T, octet *U, octet *M)

  ECIES Decryption.
- int ECP_ZZZ_SP_DSA (int h, csprng *R, octet *k, octet *s, octet *M, octet *c, octet *d)

  ECDSA Signature.
- int ECP_ZZZ_VP_DSA (int h, octet *W, octet *M, octet *c, octet *d)

  ECDSA Signature Verification.

5.11.1 Detailed Description

ECDH Header file for implementation of standard EC protocols.

Author

Mike Scott

5.11.2 Macro Definition Documentation

5.11.2.1 EGS_ZZZ

```
#define EGS_ZZZ MODBYTES_XXX
```

ECC Group Size in bytes

5.11.2.2 EFS_ZZZ

```
#define EFS_ZZZ MODBYTES_XXX
```

ECC Field Size in bytes

5.11.2.3 ECDH_OK

```
#define ECDH_OK 0
```

Function completed without error

5.11.2.4 ECDH_INVALID_PUBLIC_KEY

```
#define ECDH_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

5.11.2.5 ECDH_ERROR

```
#define ECDH_ERROR -3
```

ECDH Internal Error

5.11.3 Function Documentation

5.11.3.1 ECP_ZZZ_IN_RANGE()

```
int ECP_ZZZ_IN_RANGE (
            octet * s )
```

Test if group element in correct range.

Parameters

s	is a random number

Returns

1 if 0<s<r where r is group order, else 0

5.11.3.2 ECP_ZZZ_KEY_PAIR_GENERATE()

```
int ECP_ZZZ_KEY_PAIR_GENERATE (
            csprng * R,
            octet * s,
            octet * W )
```

Generate an ECC public/private key pair.

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

5.11.3.3 ECP_ZZZ_PUBLIC_KEY_VALIDATE()

```
int ECP_ZZZ_PUBLIC_KEY_VALIDATE (
            octet * W )
```

Validate an ECC public key.

Parameters

W	the input public key to be validated

Returns

0 if public key is OK, or an error code

5.11.3.4 ECP_ZZZ_SVDP_DH()

```
int ECP_ZZZ_SVDP_DH (
            octet * s,
            octet * W,
            octet * K,
            int type )
```

Generate Diffie-Hellman shared key.
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
type	the output form = 0 for just x, 1 for compressed, 2 for uncompressed

Returns

0 or an error code

5.11.3.5 ECP_ZZZ_ECIES_ENCRYPT()

```
void ECP_ZZZ_ECIES_ENCRYPT (
            int h,
            octet * P1,
            octet * P2,
            csprng * R,
            octet * W,
            octet * M,
            int len,
            octet * V,
            octet * C,
            octet * T )
```

ECIES Encryption.
IEEE-1363 ECIES Encryption

Parameters

h	is the hash type
P1	input Key Derivation parameters

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

5.11.3.6 ECP_ZZZ_ECIES_DECRYPT()

```
int ECP_ZZZ_ECIES_DECRYPT (
            int h,
            octet * P1,
            octet * P2,
            octet * V,
            octet * C,
            octet * T,
            octet * U,
            octet * M )
```

ECIES Decryption.
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

5.11.3.7 ECP_ZZZ_SP_DSA()

```
int ECP_ZZZ_SP_DSA (
            int h,
            csprng * R,
            octet * k,
            octet * s,
            octet * M,
            octet * c,
            octet * d )
```

ECDSA Signature.

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

5.11.3.8 ECP_ZZZ_VP_DSA()

```
int ECP_ZZZ_VP_DSA (
            int h,
            octet * W,
            octet * M,
            octet * c,
            octet * d )
```

ECDSA Signature Verification.
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

5.12 ecp.h File Reference

ECP Header File.
`#include "fp_YYY.h"`
`#include "config_curve_ZZZ.h"`

Classes

- struct ECP_ZZZ

  ECP structure - Elliptic Curve Point over base field.

Functions

- int ECP_ZZZ_isinf (ECP_ZZZ ∗P)

  Tests for ECP point equal to infinity.
- int ECP_ZZZ_equals (ECP_ZZZ ∗P, ECP_ZZZ ∗Q)

  Tests for equality of two ECPs.
- void ECP_ZZZ_copy (ECP_ZZZ ∗P, ECP_ZZZ ∗Q)

Copy ECP point to another ECP point.
- void ECP_ZZZ_neg (ECP_ZZZ ∗P)

  Negation of an ECP point.
- void ECP_ZZZ_inf (ECP_ZZZ ∗P)

  Set ECP to point-at-infinity.
- void ECP_ZZZ_rhs (FP_YYY ∗r, FP_YYY ∗x)

  Calculate Right Hand Side of curve equation y^2=f(x)
- int ECP_ZZZ_set (ECP_ZZZ ∗P, BIG_XXX x)

  Set ECP to point(x,[y]) given x.
- int ECP_ZZZ_get (BIG_XXX x, ECP_ZZZ ∗P)

  Extract x coordinate of an ECP point P.
- void ECP_ZZZ_add (ECP_ZZZ ∗P, ECP_ZZZ ∗Q, ECP_ZZZ ∗D)

  Adds ECP instance Q to ECP instance P, given difference D=P-Q.
- void ECP_ZZZ_cfp (ECP_ZZZ ∗Q)

  Multiplies Point by curve co-factor.
- void ECP_ZZZ_map2point (ECP_ZZZ ∗Q, FP_YYY ∗x)

  Maps random BIG to curve point in constant time.
- void ECP_ZZZ_hap2point (ECP_ZZZ ∗Q, BIG_XXX x)

  Maps random BIG to curve point using hunt-and-peck.
- void ECP_ZZZ_mapit (ECP_ZZZ ∗Q, octet ∗w)

  Maps random octet to curve point of correct order.
- void ECP_ZZZ_affine (ECP_ZZZ ∗P)

  Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.
- void ECP_ZZZ_outputxyz (ECP_ZZZ ∗P)

  Formats and outputs an ECP point to the console, in projective coordinates.
- void ECP_ZZZ_output (ECP_ZZZ ∗P)

  Formats and outputs an ECP point to the console, converted to affine coordinates.
- void ECP_ZZZ_rawoutput (ECP_ZZZ ∗P)

  Formats and outputs an ECP point to the console.
- void ECP_ZZZ_toOctet (octet ∗S, ECP_ZZZ ∗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_ZZZ_fromOctet (ECP_ZZZ ∗P, octet ∗S)

  Creates an ECP point from an octet string.
- void ECP_ZZZ_dbl (ECP_ZZZ ∗P)

  Doubles an ECP instance P.
- void ECP_ZZZ_pinmul (ECP_ZZZ ∗P, int i, int b)

  Multiplies an ECP instance P by a small integer, side-channel resistant.
- void ECP_ZZZ_mul (ECP_ZZZ ∗P, BIG_XXX b)

  Multiplies an ECP instance P by a BIG, side-channel resistant.
- void ECP_ZZZ_mul2 (ECP_ZZZ ∗P, ECP_ZZZ ∗Q, BIG_XXX e, BIG_XXX f)

  Calculates double multiplication P=e∗P+f∗Q, side-channel resistant.
- void ECP_ZZZ_muln (ECP_ZZZ ∗P, int n, ECP_ZZZ X[ ], BIG_XXX e[ ])

  Calculates multi-multiplication P=Sigma e_i∗X_i, side-channel resistant.
- int ECP_ZZZ_generator (ECP_ZZZ ∗G)

  Get Group Generator from ROM.

Variables

- const int CURVE_Cof_I_ZZZ
- const int CURVE_B_I_ZZZ
- const BIG_XXX CURVE_B_ZZZ
- const BIG_XXX CURVE_Order_ZZZ
- const BIG_XXX CURVE_Cof_ZZZ
- const BIG_XXX CURVE_HTPC_ZZZ
- const BIG_XXX CURVE_HTPC2_ZZZ
- const BIG_XXX CURVE_Ad_ZZZ
- const BIG_XXX CURVE_Bd_ZZZ
- const BIG_XXX PC_ZZZ [ ]
- const BIG_XXX CURVE_Adr_ZZZ
- const BIG_XXX CURVE_Adi_ZZZ
- const BIG_XXX CURVE_Bdr_ZZZ
- const BIG_XXX CURVE_Bdi_ZZZ
- const BIG_XXX PCR_ZZZ [ ]
- const BIG_XXX PCI_ZZZ [ ]
- const BIG_XXX CURVE_Gx_ZZZ
- const BIG_XXX CURVE_Gy_ZZZ
- const BIG_XXX CURVE_Pxa_ZZZ
- const BIG_XXX CURVE_Pxb_ZZZ
- const BIG_XXX CURVE_Pya_ZZZ
- const BIG_XXX CURVE_Pyb_ZZZ
- const BIG_XXX CURVE_Pxaa_ZZZ
- const BIG_XXX CURVE_Pxab_ZZZ
- const BIG_XXX CURVE_Pxba_ZZZ
- const BIG_XXX CURVE_Pxbb_ZZZ
- const BIG_XXX CURVE_Pyaa_ZZZ
- const BIG_XXX CURVE_Pyab_ZZZ
- const BIG_XXX CURVE_Pyba_ZZZ
- const BIG_XXX CURVE_Pybb_ZZZ
- const BIG_XXX CURVE_Pxaaa_ZZZ
- const BIG_XXX CURVE_Pxaab_ZZZ
- const BIG_XXX CURVE_Pxaba_ZZZ
- const BIG_XXX CURVE_Pxabb_ZZZ
- const BIG_XXX CURVE_Pxbaa_ZZZ
- const BIG_XXX CURVE_Pxbab_ZZZ
- const BIG_XXX CURVE_Pxbba_ZZZ
- const BIG_XXX CURVE_Pxbbb_ZZZ
- const BIG_XXX CURVE_Pyaaa_ZZZ
- const BIG_XXX CURVE_Pyaab_ZZZ
- const BIG_XXX CURVE_Pyaba_ZZZ
- const BIG_XXX CURVE_Pyabb_ZZZ
- const BIG_XXX CURVE_Pybaa_ZZZ
- const BIG_XXX CURVE_Pybab_ZZZ
- const BIG_XXX CURVE_Pybba_ZZZ
- const BIG_XXX CURVE_Pybbb_ZZZ
- const BIG_XXX CURVE_Bnx_ZZZ
- const BIG_XXX Fra_YYY
- const BIG_XXX Frb_YYY
- const BIG_XXX CURVE_W_ZZZ [2]
- const BIG_XXX CURVE_SB_ZZZ [2][2]
- const BIG_XXX CURVE_WB_ZZZ [4]
- const BIG_XXX CURVE_BB_ZZZ [4][4]

5.12.1 Detailed Description

ECP Header File.

Author

Mike Scott

5.12.2 Function Documentation

5.12.2.1 ECP_ZZZ_isinf()

```
int ECP_ZZZ_isinf (
            ECP_ZZZ * P )
```

Tests for ECP point equal to infinity.

Parameters

P	ECP point to be tested

Returns

1 if infinity, else returns 0

5.12.2.2 ECP_ZZZ_equals()

```
int ECP_ZZZ_equals (
            ECP_ZZZ * P,
            ECP_ZZZ * Q )
```

Tests for equality of two ECPs.

Parameters

P	ECP instance to be compared
Q	ECP instance to be compared

Returns

1 if P=Q, else returns 0

5.12.2.3 ECP_ZZZ_copy()

```
void ECP_ZZZ_copy (
            ECP_ZZZ * P,
            ECP_ZZZ * Q )
```

Copy ECP point to another ECP point.

Parameters

P	ECP instance, on exit = Q
Q	ECP instance to be copied

5.12.2.4 ECP_ZZZ_neg()

```
void ECP_ZZZ_neg (
        ECP_ZZZ * P )
```

Negation of an ECP point.

Parameters

P	ECP instance, on exit = -P

5.12.2.5 ECP_ZZZ_inf()

```
void ECP_ZZZ_inf (
        ECP_ZZZ * P )
```

Set ECP to point-at-infinity.

Parameters

P	ECP instance to be set to infinity

5.12.2.6 ECP_ZZZ_rhs()

```
void ECP_ZZZ_rhs (
        FP_YYY * r,
        FP_YYY * x )
```

Calculate Right Hand Side of curve equation y^2=f(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

5.12.2.7 ECP_ZZZ_set()

```
int ECP_ZZZ_set (
        ECP_ZZZ * P,
        BIG_XXX x )
```

Set ECP to point(x,[y]) given x.

Point P set to infinity if no such point on the curve. Note that y coordinate is not needed.

Parameters

P	ECP instance to be set (x,[y])
x	BIG x coordinate of point

Returns

1 if point exists, else 0

5.12.2.8 ECP_ZZZ_get()

```
int ECP_ZZZ_get (
            BIG_XXX x,
            ECP_ZZZ * P )
```

Extract x coordinate of an ECP point P.

Parameters

x	BIG on exit = x coordinate of point
P	ECP instance (x,[y])

Returns

-1 if P is point-at-infinity, else 0

5.12.2.9 ECP_ZZZ_add()

```
void ECP_ZZZ_add (
            ECP_ZZZ * P,
            ECP_ZZZ * Q,
            ECP_ZZZ * D )
```

Adds ECP instance Q to ECP instance P, given difference D=P-Q.
Differential addition of points on a Montgomery curve

Parameters

P	ECP instance, on exit =P+Q
Q	ECP instance to be added to P
D	Difference between P and Q

5.12.2.10 ECP_ZZZ_cfp()

```
void ECP_ZZZ_cfp (
            ECP_ZZZ * Q )
```

Multiplies Point by curve co-factor.

Parameters

Q	ECP instance

5.12.2.11 ECP_ZZZ_map2point()

```
void ECP_ZZZ_map2point (
            ECP_ZZZ * Q,
            FP_YYY * x )
```

Maps random BIG to curve point in constant time.

Parameters

Q	ECP instance
x	FP derived from hash

5.12.2.12 ECP_ZZZ_hap2point()

```
void ECP_ZZZ_hap2point (
            ECP_ZZZ * Q,
            BIG_XXX x )
```

Maps random BIG to curve point using hunt-and-peck.

Parameters

Q	ECP instance
x	Fp derived from hash

5.12.2.13 ECP_ZZZ_mapit()

```
void ECP_ZZZ_mapit (
            ECP_ZZZ * Q,
            octet * w )
```

Maps random octet to curve point of correct order.

Parameters

Q	ECP instance of correct order
w	OCTET byte array to be mapped

5.12.2.14 ECP_ZZZ_affine()

```
void ECP_ZZZ_affine (
            ECP_ZZZ * P )
```

Converts an ECP point from Projective (x,y,z) coordinates to affine (x,y) coordinates.

Parameters

P	ECP instance to be converted to affine form

5.12.2.15 ECP_ZZZ_outputxyz()

```
void ECP_ZZZ_outputxyz (
            ECP_ZZZ * P )
```

Formats and outputs an ECP point to the console, in projective coordinates.

Parameters

P	ECP instance to be printed

5.12.2.16 ECP_ZZZ_output()

```
void ECP_ZZZ_output (
            ECP_ZZZ * P )
```

Formats and outputs an ECP point to the console, converted to affine coordinates.

Parameters

P	ECP instance to be printed

5.12.2.17 ECP_ZZZ_rawoutput()

```
void ECP_ZZZ_rawoutput (
            ECP_ZZZ * P )
```

Formats and outputs an ECP point to the console.

Parameters

P	ECP instance to be printed

5.12.2.18 ECP_ZZZ_toOctet()

```
void ECP_ZZZ_toOctet (
            octet * S,
            ECP_ZZZ * 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.

Parameters

c	compression required, true or false
S	output octet string
P	ECP instance to be converted to an octet string

5.12.2.19 ECP_ZZZ_fromOctet()

```
int ECP_ZZZ_fromOctet (
            ECP_ZZZ * P,
            octet * S )
```

Creates an ECP point from 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 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

5.12.2.20 ECP_ZZZ_dbl()

```
void ECP_ZZZ_dbl (
            ECP_ZZZ * P )
```

Doubles an ECP instance P.

Parameters

P	ECP instance, on exit =2∗P

5.12.2.21 ECP_ZZZ_pinmul()

```
void ECP_ZZZ_pinmul (
            ECP_ZZZ * P,
            int i,
            int b )
```

Multiplies an ECP instance P by a small integer, side-channel resistant.

Parameters

P	ECP instance, on exit =i∗P
i	small integer multiplier
b	maximum number of bits in multiplier

5.12.2.22 ECP_ZZZ_mul()

```
void ECP_ZZZ_mul (
            ECP_ZZZ * P,
            BIG_XXX b )
```

Multiplies an ECP instance P by a BIG, side-channel resistant.
Uses Montgomery ladder for Montgomery curves, otherwise fixed sized windows.

Parameters

P	ECP instance, on exit =b∗P
b	BIG number multiplier

5.12.2.23 ECP_ZZZ_mul2()

```
void ECP_ZZZ_mul2 (
            ECP_ZZZ * P,
            ECP_ZZZ * Q,
            BIG_XXX e,
            BIG_XXX f )
```

Calculates double multiplication P=e∗P+f∗Q, side-channel resistant.

Parameters

P	ECP instance, on exit =e∗P+f∗Q
Q	ECP instance
e	BIG number multiplier
f	BIG number multiplier

5.12.2.24 ECP_ZZZ_muln()

```
void ECP_ZZZ_muln (
            ECP_ZZZ * P,
            int n,
            ECP_ZZZ X[ ],
            BIG_XXX e[ ] )
```

Calculates multi-multiplication P=Sigma e_i*X_i, side-channel resistant.

Parameters

P	ECP instance, on exit = Sigma e_i*X_i
n	Number of multiplications
X	array of n ECPs
e	array of n BIG multipliers

5.12.2.25 ECP_ZZZ_generator()

```
int ECP_ZZZ_generator (
            ECP_ZZZ * G )
```

Get Group Generator from ROM.

Parameters

G	ECP instance

Returns

success

5.12.3 Variable Documentation

5.12.3.1 CURVE_Cof_I_ZZZ

```
const int CURVE_Cof_I_ZZZ [extern]
```

Elliptic curve cofactor

5.12.3.2 CURVE_B_I_ZZZ

```
const int CURVE_B_I_ZZZ [extern]
```

Elliptic curve B_i parameter

5.12.3.3 CURVE_B_ZZZ

```
const BIG_XXX CURVE_B_ZZZ [extern]
```

Elliptic curve B parameter

5.12.3.4 CURVE_Order_ZZZ

```
const BIG_XXX CURVE_Order_ZZZ [extern]
```

Elliptic curve group order

5.12.3.5 CURVE_Cof_ZZZ

`const BIG_XXX CURVE_Cof_ZZZ [extern]`

Elliptic curve cofactor

5.12.3.6 CURVE_HTPC_ZZZ

`const BIG_XXX CURVE_HTPC_ZZZ [extern]`

Hash to Point precomputation

5.12.3.7 CURVE_HTPC2_ZZZ

`const BIG_XXX CURVE_HTPC2_ZZZ [extern]`

Hash to Point precomputation for G2

5.12.3.8 CURVE_Ad_ZZZ

`const BIG_XXX CURVE_Ad_ZZZ [extern]`

A parameter of isogenous curve

5.12.3.9 CURVE_Bd_ZZZ

`const BIG_XXX CURVE_Bd_ZZZ [extern]`

B parameter of isogenous curve

5.12.3.10 PC_ZZZ

`const BIG_XXX PC_ZZZ[ ] [extern]`

Precomputed isogenies

5.12.3.11 CURVE_Adr_ZZZ

`const BIG_XXX CURVE_Adr_ZZZ [extern]`

Real part of A parameter of isogenous curve in G2

5.12.3.12 CURVE_Adi_ZZZ

`const BIG_XXX CURVE_Adi_ZZZ [extern]`

Imaginary part of A parameter of isogenous curve in G2

5.12.3.13 CURVE_Bdr_ZZZ

`const BIG_XXX CURVE_Bdr_ZZZ [extern]`

Real part of B parameter of isogenous curve in G2

5.12.3.14 CURVE_Bdi_ZZZ

`const BIG_XXX CURVE_Bdi_ZZZ [extern]`

Imaginary part of B parameter of isogenous curve in G2

5.12.3.15 PCR_ZZZ

`const BIG_XXX PCR_ZZZ[ ] [extern]`

Real parts of precomputed isogenies

5.12.3.16 PCI_ZZZ

`const BIG_XXX PCI_ZZZ[ ] [extern]`

Imaginary parts of precomputed isogenies

5.12.3.17 CURVE_Gx_ZZZ

`const BIG_XXX CURVE_Gx_ZZZ [extern]`

x-coordinate of generator point in group G1

5.12.3.18 CURVE_Gy_ZZZ

`const BIG_XXX CURVE_Gy_ZZZ [extern]`

y-coordinate of generator point in group G1

5.12.3.19 CURVE_Pxa_ZZZ

`const BIG_XXX CURVE_Pxa_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.12.3.20 CURVE_Pxb_ZZZ

`const BIG_XXX CURVE_Pxb_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.12.3.21 CURVE_Pya_ZZZ

`const BIG_XXX CURVE_Pya_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.12.3.22 CURVE_Pyb_ZZZ

`const BIG_XXX CURVE_Pyb_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.12.3.23 CURVE_Pxaa_ZZZ

`const BIG_XXX CURVE_Pxaa_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.12.3.24 CURVE_Pxab_ZZZ

`const BIG_XXX CURVE_Pxab_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.12.3.25 CURVE_Pxba_ZZZ

`const BIG_XXX CURVE_Pxba_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.12.3.26 CURVE_Pxbb_ZZZ

`const BIG_XXX CURVE_Pxbb_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.12.3.27 CURVE_Pyaa_ZZZ

`const BIG_XXX CURVE_Pyaa_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.12.3.28 CURVE_Pyab_ZZZ

`const BIG_XXX CURVE_Pyab_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.12.3.29 CURVE_Pyba_ZZZ

`const BIG_XXX CURVE_Pyba_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.12.3.30 CURVE_Pybb_ZZZ

`const BIG_XXX CURVE_Pybb_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.12.3.31 CURVE_Pxaaa_ZZZ

`const BIG_XXX CURVE_Pxaaa_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.12.3.32 CURVE_Pxaab_ZZZ

`const BIG_XXX CURVE_Pxaab_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.12.3.33 CURVE_Pxaba_ZZZ

`const BIG_XXX CURVE_Pxaba_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.12.3.34 CURVE_Pxabb_ZZZ

`const BIG_XXX CURVE_Pxabb_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.12.3.35 CURVE_Pxbaa_ZZZ

`const BIG_XXX CURVE_Pxbaa_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.12.3.36 CURVE_Pxbab_ZZZ

`const BIG_XXX CURVE_Pxbab_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.12.3.37 CURVE_Pxbba_ZZZ

`const BIG_XXX CURVE_Pxbba_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.12.3.38 CURVE_Pxbbb_ZZZ

`const BIG_XXX CURVE_Pxbbb_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.12.3.39 CURVE_Pyaaa_ZZZ

`const BIG_XXX CURVE_Pyaaa_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.12.3.40 CURVE_Pyaab_ZZZ

`const BIG_XXX CURVE_Pyaab_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.12.3.41 CURVE_Pyaba_ZZZ

`const BIG_XXX CURVE_Pyaba_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.12.3.42 CURVE_Pyabb_ZZZ

`const BIG_XXX CURVE_Pyabb_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.12.3.43 CURVE_Pybaa_ZZZ

`const BIG_XXX CURVE_Pybaa_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.12.3.44 CURVE_Pybab_ZZZ

`const BIG_XXX CURVE_Pybab_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.12.3.45 CURVE_Pybba_ZZZ

`const BIG_XXX CURVE_Pybba_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.12.3.46 CURVE_Pybbb_ZZZ

`const BIG_XXX CURVE_Pybbb_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.12.3.47 CURVE_Bnx_ZZZ

`const BIG_XXX CURVE_Bnx_ZZZ [extern]`

BN curve x parameter

5.12.3.48 Fra_YYY

`const BIG_XXX Fra_YYY [extern]`

real part of BN curve Frobenius Constant

5.12.3.49 Frb_YYY

`const BIG_XXX Frb_YYY [extern]`

imaginary part of BN curve Frobenius Constant

5.12.3.50 CURVE_W_ZZZ

`const BIG_XXX CURVE_W_ZZZ[2] [extern]`

BN curve constant for GLV decomposition

5.12.3.51 CURVE_SB_ZZZ

`const BIG_XXX CURVE_SB_ZZZ[2][2] [extern]`

BN curve constant for GLV decomposition

5.12.3.52 CURVE_WB_ZZZ

`const BIG_XXX CURVE_WB_ZZZ[4] [extern]`

BN curve constant for GS decomposition

5.12.3.53 CURVE_BB_ZZZ

```
const BIG_XXX CURVE_BB_ZZZ[4][4] [extern]
```

BN curve constant for GS decomposition

5.13 ecp2.h File Reference

ECP2 Header File.

```
#include "fp2_YYY.h"
#include "config_curve_ZZZ.h"
```

Classes

- struct ECP2_ZZZ

  ECP2 Structure - Elliptic Curve Point over quadratic extension field.

Functions

- int ECP2_ZZZ_isinf (ECP2_ZZZ ∗P)

  Tests for ECP2 point equal to infinity.
- void ECP2_ZZZ_copy (ECP2_ZZZ ∗P, ECP2_ZZZ ∗Q)

  Copy ECP2 point to another ECP2 point.
- void ECP2_ZZZ_inf (ECP2_ZZZ ∗P)

  Set ECP2 to point-at-infinity.
- int ECP2_ZZZ_equals (ECP2_ZZZ ∗P, ECP2_ZZZ ∗Q)

  Tests for equality of two ECP2s.
- void ECP2_ZZZ_affine (ECP2_ZZZ ∗P)

  Converts an ECP2 point from Projective (x,y,z) coordinates to affine (x,y) coordinates.
- int ECP2_ZZZ_get (FP2_YYY ∗x, FP2_YYY ∗y, ECP2_ZZZ ∗P)

  Extract x and y coordinates of an ECP2 point P.
- void ECP2_ZZZ_output (ECP2_ZZZ ∗P)

  Formats and outputs an ECP2 point to the console, converted to affine coordinates.
- void ECP2_ZZZ_outputxyz (ECP2_ZZZ ∗P)

  Formats and outputs an ECP2 point to the console, in projective coordinates.
- void ECP2_ZZZ_toOctet (octet ∗S, ECP2_ZZZ ∗P, bool c)

  Formats and outputs an ECP2 point to an octet string.
- int ECP2_ZZZ_fromOctet (ECP2_ZZZ ∗P, octet ∗S)

  Creates an ECP2 point from an octet string.
- void ECP2_ZZZ_rhs (FP2_YYY ∗r, FP2_YYY ∗x)

  Calculate Right Hand Side of curve equation y^2=f(x)
- int ECP2_ZZZ_set (ECP2_ZZZ ∗P, FP2_YYY ∗x, FP2_YYY ∗y)

  Set ECP2 to point(x,y) given x and y.
- int ECP2_ZZZ_setx (ECP2_ZZZ ∗P, FP2_YYY ∗x, int s)

  Set ECP to point(x,[y]) given x and sign of y.
- void ECP2_ZZZ_neg (ECP2_ZZZ ∗P)

  Negation of an ECP2 point.
- int ECP2_ZZZ_dbl (ECP2_ZZZ ∗P)

  Doubles an ECP2 instance P.
- int ECP2_ZZZ_add (ECP2_ZZZ ∗P, ECP2_ZZZ ∗Q)

  Adds ECP2 instance Q to ECP2 instance P.
- void ECP2_ZZZ_sub (ECP2_ZZZ ∗P, ECP2_ZZZ ∗Q)

  Subtracts ECP instance Q from ECP2 instance P.

- void ECP2_ZZZ_mul (ECP2_ZZZ *P, BIG_XXX b)

  Multiplies an ECP2 instance P by a BIG, side-channel resistant.
- void ECP2_ZZZ_frob (ECP2_ZZZ *P, FP2_YYY *f)

  Multiplies an ECP2 instance P by the internal modulus p, using precalculated Frobenius constant f.
- void ECP2_ZZZ_mul4 (ECP2_ZZZ *P, ECP2_ZZZ *Q, BIG_XXX *b)

  *Calculates P=b[0]*Q[0]+b[1]*Q[1]+b[2]*Q[2]+b[3]*Q[3].*
- void ECP2_ZZZ_cfp (ECP2_ZZZ *Q)

  Multiplies random point by co-factor.
- void ECP2_ZZZ_map2point (ECP2_ZZZ *Q, FP2_YYY *x)

  Maps random BIG to curve point in constant time.
- void ECP2_ZZZ_hap2point (ECP2_ZZZ *Q, BIG_XXX x)

  Maps random BIG to curve point using hunt-and-peck.
- void ECP2_ZZZ_mapit (ECP2_ZZZ *P, octet *w)

  Maps random BIG to curve point of correct order.
- int ECP2_ZZZ_generator (ECP2_ZZZ *G)

  Get Group Generator from ROM.

Variables

- const int CURVE_B_I_ZZZ
- const BIG_XXX CURVE_B_ZZZ
- const BIG_XXX CURVE_Order_ZZZ
- const BIG_XXX CURVE_Cof_ZZZ
- const BIG_XXX CURVE_Bnx_ZZZ
- const BIG_XXX CURVE_HTPC_ZZZ
- const BIG_XXX Fra_YYY
- const BIG_XXX Frb_YYY
- const BIG_XXX CURVE_Gx_ZZZ
- const BIG_XXX CURVE_Gy_ZZZ
- const BIG_XXX CURVE_Pxa_ZZZ
- const BIG_XXX CURVE_Pxb_ZZZ
- const BIG_XXX CURVE_Pya_ZZZ
- const BIG_XXX CURVE_Pyb_ZZZ

5.13.1 Detailed Description

ECP2 Header File.

Author

Mike Scott

5.13.2 Function Documentation

5.13.2.1 ECP2_ZZZ_isinf()

```
int ECP2_ZZZ_isinf (
                ECP2_ZZZ * P )
```

Tests for ECP2 point equal to infinity.

Parameters

P	ECP2 point to be tested

Returns

1 if infinity, else returns 0

5.13.2.2 ECP2_ZZZ_copy()

```
void ECP2_ZZZ_copy (
            ECP2_ZZZ * P,
            ECP2_ZZZ * Q )
```

Copy ECP2 point to another ECP2 point.

Parameters

P	ECP2 instance, on exit = Q
Q	ECP2 instance to be copied

5.13.2.3 ECP2_ZZZ_inf()

```
void ECP2_ZZZ_inf (
            ECP2_ZZZ * P )
```

Set ECP2 to point-at-infinity.

Parameters

P	ECP2 instance to be set to infinity

5.13.2.4 ECP2_ZZZ_equals()

```
int ECP2_ZZZ_equals (
            ECP2_ZZZ * P,
            ECP2_ZZZ * Q )
```

Tests for equality of two ECP2s.

Parameters

P	ECP2 instance to be compared
Q	ECP2 instance to be compared

Returns

1 if P=Q, else returns 0

5.13.2.5 ECP2_ZZZ_affine()

```
void ECP2_ZZZ_affine (
            ECP2_ZZZ * P )
```

Converts an ECP2 point from Projective (x,y,z) coordinates to affine (x,y) coordinates.

Parameters

P	ECP2 instance to be converted to affine form

5.13.2.6 ECP2_ZZZ_get()

```
int ECP2_ZZZ_get (
            FP2_YYY * x,
            FP2_YYY * y,
            ECP2_ZZZ * P )
```

Extract x and y coordinates of an ECP2 point 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

5.13.2.7 ECP2_ZZZ_output()

```
void ECP2_ZZZ_output (
            ECP2_ZZZ * P )
```

Formats and outputs an ECP2 point to the console, converted to affine coordinates.

Parameters

P	ECP2 instance to be printed

5.13.2.8 ECP2_ZZZ_outputxyz()

```
void ECP2_ZZZ_outputxyz (
            ECP2_ZZZ * P )
```

Formats and outputs an ECP2 point to the console, in projective coordinates.

Parameters

P	ECP2 instance to be printed

5.13.2.9 ECP2_ZZZ_toOctet()

```
void ECP2_ZZZ_toOctet (
            octet * S,
            ECP2_ZZZ * P,
            bool c )
```

Formats and outputs an ECP2 point to an octet string.
The octet string is created in the form x|y or just x if compressed 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
c	true for compression

5.13.2.10 ECP2_ZZZ_fromOctet()

```
int ECP2_ZZZ_fromOctet (
            ECP2_ZZZ * P,
            octet * S )
```

Creates an ECP2 point from an octet string.

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. 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	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

5.13.2.11 ECP2_ZZZ_rhs()

```
void ECP2_ZZZ_rhs (
            FP2_YYY * r,
            FP2_YYY * x )
```

Calculate Right Hand Side of curve equation y^2=f(x)

Function f(x)=x^3+Ax+B Used internally.

Parameters

r	FP2 value of f(x)
x	FP2 instance

5.13.2.12 ECP2_ZZZ_set()

```
int ECP2_ZZZ_set (
            ECP2_ZZZ * P,
            FP2_YYY * x,
            FP2_YYY * y )
```

Set ECP2 to point(x,y) given x and 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

5.13.2.13 ECP2_ZZZ_setx()

```
int ECP2_ZZZ_setx (
            ECP2_ZZZ * P,
            FP2_YYY * x,
            int s )
```

Set ECP to point(x,[y]) given x and sign of y.
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
s	sign of y

Returns

1 if point exists, else 0

5.13.2.14 ECP2_ZZZ_neg()

```
void ECP2_ZZZ_neg (
            ECP2_ZZZ * P )
```

Negation of an ECP2 point.

Parameters

P	ECP2 instance, on exit = -P

5.13.2.15 ECP2_ZZZ_dbl()

```
int ECP2_ZZZ_dbl (
            ECP2_ZZZ * P )
```

Doubles an ECP2 instance P.

Parameters

P	ECP2 instance, on exit =2∗P

5.13.2.16 ECP2_ZZZ_add()

```
int ECP2_ZZZ_add (
            ECP2_ZZZ * P,
            ECP2_ZZZ * Q )
```

Adds ECP2 instance Q to ECP2 instance P.

Parameters

P	ECP2 instance, on exit =P+Q
Q	ECP2 instance to be added to P

5.13.2.17 ECP2_ZZZ_sub()

```
void ECP2_ZZZ_sub (
            ECP2_ZZZ * P,
            ECP2_ZZZ * Q )
```

Subtracts ECP instance Q from ECP2 instance P.

Parameters

P	ECP2 instance, on exit =P-Q
Q	ECP2 instance to be subtracted from P

5.13.2.18 ECP2_ZZZ_mul()

```
void ECP2_ZZZ_mul (
            ECP2_ZZZ * P,
            BIG_XXX b )
```

Multiplies an ECP2 instance P by a BIG, side-channel resistant.
Uses fixed sized windows.

Parameters

P	ECP2 instance, on exit =b∗P
b	BIG number multiplier

5.13.2.19 ECP2_ZZZ_frob()

```
void ECP2_ZZZ_frob (
            ECP2_ZZZ * P,
            FP2_YYY * f )
```

Multiplies an ECP2 instance P by the internal modulus p, using precalculated Frobenius constant f.
Fast point multiplication using Frobenius

Parameters

P	ECP2 instance, on exit = p∗P
f	FP2 precalculated Frobenius constant

5.13.2.20 ECP2_ZZZ_mul4()

```
void ECP2_ZZZ_mul4 (
            ECP2_ZZZ * P,
            ECP2_ZZZ * Q,
            BIG_XXX * b )
```

Calculates P=b[0]∗Q[0]+b[1]∗Q[1]+b[2]∗Q[2]+b[3]∗Q[3].

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

5.13.2.21 ECP2_ZZZ_cfp()

```
void ECP2_ZZZ_cfp (
            ECP2_ZZZ * Q )
```

Multiplies random point by co-factor.

Parameters

Q	ECP2 multiplied by co-factor

5.13.2.22 ECP2_ZZZ_map2point()

```
void ECP2_ZZZ_map2point (
            ECP2_ZZZ * Q,
            FP2_YYY * x )
```

Maps random BIG to curve point in constant time.

Parameters

Q	ECP2 instance
x	FP2 derived from hash

5.13.2.23 ECP2_ZZZ_hap2point()

```
void ECP2_ZZZ_hap2point (
            ECP2_ZZZ * Q,
            BIG_XXX x )
```

Maps random BIG to curve point using hunt-and-peck.

Parameters

Q	ECP2 instance
x	Fp derived from hash

5.13.2.24 ECP2_ZZZ_mapit()

```
void ECP2_ZZZ_mapit (
            ECP2_ZZZ * P,
            octet * w )
```

Maps random BIG to curve point of correct order.

Parameters

P	ECP2 instance of correct order
w	OCTET byte array to be mapped

5.13.2.25 ECP2_ZZZ_generator()

```
int ECP2_ZZZ_generator (
                ECP2_ZZZ * G )
```

Get Group Generator from ROM.

Parameters

G	ECP2 instance

Returns

1 if point exists, else 0

5.13.3 Variable Documentation

5.13.3.1 CURVE_B_I_ZZZ

```
const int CURVE_B_I_ZZZ [extern]
```

Elliptic curve B parameter

5.13.3.2 CURVE_B_ZZZ

```
const BIG_XXX CURVE_B_ZZZ [extern]
```

Elliptic curve B parameter

5.13.3.3 CURVE_Order_ZZZ

```
const BIG_XXX CURVE_Order_ZZZ [extern]
```

Elliptic curve group order

5.13.3.4 CURVE_Cof_ZZZ

```
const BIG_XXX CURVE_Cof_ZZZ [extern]
```

Elliptic curve cofactor

5.13.3.5 CURVE_Bnx_ZZZ

```
const BIG_XXX CURVE_Bnx_ZZZ [extern]
```

Elliptic curve parameter

5.13.3.6 CURVE_HTPC_ZZZ

```
const BIG_XXX CURVE_HTPC_ZZZ [extern]
```

Hash to Point precomputation

5.13.3.7 Fra_YYY

```
const BIG_XXX Fra_YYY [extern]
```

real part of BN curve Frobenius Constant

5.13.3.8 Frb_YYY

`const BIG_XXX Frb_YYY [extern]`

imaginary part of BN curve Frobenius Constant

5.13.3.9 CURVE_Gx_ZZZ

`const BIG_XXX CURVE_Gx_ZZZ [extern]`

x-coordinate of generator point in group G1

5.13.3.10 CURVE_Gy_ZZZ

`const BIG_XXX CURVE_Gy_ZZZ [extern]`

y-coordinate of generator point in group G1

5.13.3.11 CURVE_Pxa_ZZZ

`const BIG_XXX CURVE_Pxa_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.13.3.12 CURVE_Pxb_ZZZ

`const BIG_XXX CURVE_Pxb_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.13.3.13 CURVE_Pya_ZZZ

`const BIG_XXX CURVE_Pya_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.13.3.14 CURVE_Pyb_ZZZ

`const BIG_XXX CURVE_Pyb_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.14 ecp4.h File Reference

ECP2 Header File.

```
#include "fp4_YYY.h"
#include "config_curve_ZZZ.h"
```

Classes

- struct ECP4_ZZZ

  ECP4 Structure - Elliptic Curve Point over quadratic extension field.

Functions

- int ECP4_ZZZ_isinf (ECP4_ZZZ ∗P)

  Tests for ECP4 point equal to infinity.
- void ECP4_ZZZ_copy (ECP4_ZZZ ∗P, ECP4_ZZZ ∗Q)

  Copy ECP4 point to another ECP4 point.
- void ECP4_ZZZ_inf (ECP4_ZZZ ∗P)

  Set ECP4 to point-at-infinity.
- int ECP4_ZZZ_equals (ECP4_ZZZ ∗P, ECP4_ZZZ ∗Q)

Tests for equality of two ECP4s.

- void ECP4_ZZZ_affine (ECP4_ZZZ ∗P)

  Converts an ECP4 point from Projective (x,y,z) coordinates to affine (x,y) coordinates.
- int ECP4_ZZZ_get (FP4_YYY ∗x, FP4_YYY ∗y, ECP4_ZZZ ∗P)

  Extract x and y coordinates of an ECP4 point P.
- void ECP4_ZZZ_output (ECP4_ZZZ ∗P)

  Formats and outputs an ECP4 point to the console, converted to affine coordinates.
- void ECP4_ZZZ_toOctet (octet ∗S, ECP4_ZZZ ∗P, bool c)

  Formats and outputs an ECP4 point to an octet string.
- int ECP4_ZZZ_fromOctet (ECP4_ZZZ ∗P, octet ∗S)

  Creates an ECP4 point from an octet string.
- void ECP4_ZZZ_rhs (FP4_YYY ∗r, FP4_YYY ∗x)

  Calculate Right Hand Side of curve equation y^2=f(x)
- int ECP4_ZZZ_set (ECP4_ZZZ ∗P, FP4_YYY ∗x, FP4_YYY ∗y)

  Set ECP4 to point(x,y) given x and y.
- int ECP4_ZZZ_setx (ECP4_ZZZ ∗P, FP4_YYY ∗x, int s)

  Set ECP to point(x,[y]) given x.
- void ECP4_ZZZ_neg (ECP4_ZZZ ∗P)

  Negation of an ECP4 point.
- void ECP4_ZZZ_reduce (ECP4_ZZZ ∗P)

  Reduction of an ECP4 point.
- int ECP4_ZZZ_dbl (ECP4_ZZZ ∗P)

  Doubles an ECP4 instance P.
- int ECP4_ZZZ_add (ECP4_ZZZ ∗P, ECP4_ZZZ ∗Q)

  Adds ECP4 instance Q to ECP4 instance P.
- void ECP4_ZZZ_sub (ECP4_ZZZ ∗P, ECP4_ZZZ ∗Q)

  Subtracts ECP instance Q from ECP4 instance P.
- void ECP4_ZZZ_mul (ECP4_ZZZ ∗P, BIG_XXX b)

  Multiplies an ECP4 instance P by a BIG, side-channel resistant.
- void ECP4_ZZZ_frob_constants (FP2_YYY F[3])

  Calculates required Frobenius constants.
- void ECP4_ZZZ_frob (ECP4_ZZZ ∗P, FP2_YYY F[3], int n)

  Multiplies an ECP4 instance P by the internal modulus p^n, using precalculated Frobenius constants.
- void ECP4_ZZZ_mul8 (ECP4_ZZZ ∗P, ECP4_ZZZ ∗Q, BIG_XXX ∗b)

  Calculates P=Sigma b[i]∗Q[i] for i=0 to 7.
- void ECP4_ZZZ_cfp (ECP4_ZZZ ∗Q)

  Multiplies random point by co-factor.
- void ECP4_ZZZ_map2point (ECP4_ZZZ ∗Q, FP4_YYY ∗x)

  Maps random BIG to curve point in constant time.
- void ECP4_ZZZ_hap2point (ECP4_ZZZ ∗Q, BIG_XXX x)

  Maps random BIG to curve point using hunt-and-peck.
- void ECP4_ZZZ_mapit (ECP4_ZZZ ∗P, octet ∗W)

  Maps random BIG to curve point of correct order.
- int ECP4_ZZZ_generator (ECP4_ZZZ ∗G)

  Get Group Generator from ROM.

Variables

- const int CURVE_B_I_ZZZ
- const BIG_XXX CURVE_B_ZZZ
- const BIG_XXX CURVE_Order_ZZZ
- const BIG_XXX CURVE_Cof_ZZZ
- const BIG_XXX CURVE_Bnx_ZZZ
- const BIG_XXX CURVE_HTPC_ZZZ
- const BIG_XXX Fra_YYY
- const BIG_XXX Frb_YYY
- const BIG_XXX CURVE_Gx_ZZZ
- const BIG_XXX CURVE_Gy_ZZZ
- const BIG_XXX CURVE_Pxaa_ZZZ
- const BIG_XXX CURVE_Pxab_ZZZ
- const BIG_XXX CURVE_Pxba_ZZZ
- const BIG_XXX CURVE_Pxbb_ZZZ
- const BIG_XXX CURVE_Pyaa_ZZZ
- const BIG_XXX CURVE_Pyab_ZZZ
- const BIG_XXX CURVE_Pyba_ZZZ
- const BIG_XXX CURVE_Pybb_ZZZ

5.14.1 Detailed Description

ECP2 Header File.

Author

Mike Scott

5.14.2 Function Documentation

5.14.2.1 ECP4_ZZZ_isinf()

```
int ECP4_ZZZ_isinf (
            ECP4_ZZZ * P )
```

Tests for ECP4 point equal to infinity.

Parameters

P	ECP4 point to be tested

Returns

1 if infinity, else returns 0

5.14.2.2 ECP4_ZZZ_copy()

```
void ECP4_ZZZ_copy (
            ECP4_ZZZ * P,
            ECP4_ZZZ * Q )
```

Copy ECP4 point to another ECP4 point.

Parameters

P	ECP4 instance, on exit = Q
Q	ECP4 instance to be copied

5.14.2.3 ECP4_ZZZ_inf()

```
void ECP4_ZZZ_inf (
        ECP4_ZZZ * P )
```

Set ECP4 to point-at-infinity.

Parameters

P	ECP4 instance to be set to infinity

5.14.2.4 ECP4_ZZZ_equals()

```
int ECP4_ZZZ_equals (
        ECP4_ZZZ * P,
        ECP4_ZZZ * Q )
```

Tests for equality of two ECP4s.

Parameters

P	ECP4 instance to be compared
Q	ECP4 instance to be compared

Returns

1 if P=Q, else returns 0

5.14.2.5 ECP4_ZZZ_affine()

```
void ECP4_ZZZ_affine (
        ECP4_ZZZ * P )
```

Converts an ECP4 point from Projective (x,y,z) coordinates to affine (x,y) coordinates.

Parameters

P	ECP4 instance to be converted to affine form

5.14.2.6 ECP4_ZZZ_get()

```
int ECP4_ZZZ_get (
        FP4_YYY * x,
        FP4_YYY * y,
        ECP4_ZZZ * P )
```

Extract x and y coordinates of an ECP4 point P.
If x=y, returns only x

Parameters

x	FP4 on exit = x coordinate of point
y	FP4 on exit = y coordinate of point (unless x=y)
P	ECP4 instance (x,y)

Returns

-1 if P is point-at-infinity, else 0

5.14.2.7 ECP4_ZZZ_output()

```
void ECP4_ZZZ_output (
            ECP4_ZZZ * P )
```

Formats and outputs an ECP4 point to the console, converted to affine coordinates.

Parameters

P	ECP4 instance to be printed

5.14.2.8 ECP4_ZZZ_toOctet()

```
void ECP4_ZZZ_toOctet (
            octet * S,
            ECP4_ZZZ * P,
            bool c )
```

Formats and outputs an ECP4 point to an octet string.

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	ECP4 instance to be converted to an octet string
c	true for compression

5.14.2.9 ECP4_ZZZ_fromOctet()

```
int ECP4_ZZZ_fromOctet (
            ECP4_ZZZ * P,
            octet * S )
```

Creates an ECP4 point from an octet string.

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	ECP4 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

5.14.2.10 ECP4_ZZZ_rhs()

```
void ECP4_ZZZ_rhs (
            FP4_YYY * r,
            FP4_YYY * x )
```

Calculate Right Hand Side of curve equation y^2=f(x)

Function f(x)=x^3+Ax+B Used internally.

Parameters

r	FP4 value of f(x)
x	FP4 instance

5.14.2.11 ECP4_ZZZ_set()

```
int ECP4_ZZZ_set (
            ECP4_ZZZ * P,
            FP4_YYY * x,
            FP4_YYY * y )
```

Set ECP4 to point(x,y) given x and y.
Point P set to infinity if no such point on the curve.

Parameters

P	ECP4 instance to be set (x,y)
x	FP4 x coordinate of point
y	FP4 y coordinate of point

Returns

1 if point exists, else 0

5.14.2.12 ECP4_ZZZ_setx()

```
int ECP4_ZZZ_setx (
            ECP4_ZZZ * P,
            FP4_YYY * x,
            int s )
```

Set ECP to point(x,[y]) given 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
s	sign of y

Returns

1 if point exists, else 0

5.14.2.13 ECP4_ZZZ_neg()

```
void ECP4_ZZZ_neg (
            ECP4_ZZZ * P )
```

Negation of an ECP4 point.

Parameters

P	ECP4 instance, on exit = -P

5.14.2.14 ECP4_ZZZ_reduce()

```
void ECP4_ZZZ_reduce (
        ECP4_ZZZ * P )
```

Reduction of an ECP4 point.

Parameters

P	ECP4 instance, on exit (x,y) are reduced wrt the modulus

5.14.2.15 ECP4_ZZZ_dbl()

```
int ECP4_ZZZ_dbl (
        ECP4_ZZZ * P )
```

Doubles an ECP4 instance P.

Parameters

P	ECP4 instance, on exit =2∗P

5.14.2.16 ECP4_ZZZ_add()

```
int ECP4_ZZZ_add (
        ECP4_ZZZ * P,
        ECP4_ZZZ * Q )
```

Adds ECP4 instance Q to ECP4 instance P.

Parameters

P	ECP4 instance, on exit =P+Q
Q	ECP4 instance to be added to P

5.14.2.17 ECP4_ZZZ_sub()

```
void ECP4_ZZZ_sub (
        ECP4_ZZZ * P,
        ECP4_ZZZ * Q )
```

Subtracts ECP instance Q from ECP4 instance P.

Parameters

P	ECP4 instance, on exit =P-Q
Q	ECP4 instance to be subtracted from P

5.14.2.18 ECP4_ZZZ_mul()

```
void ECP4_ZZZ_mul (
            ECP4_ZZZ * P,
            BIG_XXX b )
```

Multiplies an ECP4 instance P by a BIG, side-channel resistant.
Uses fixed sized windows.

Parameters

P	ECP4 instance, on exit =b∗P
b	BIG number multiplier

5.14.2.19 ECP4_ZZZ_frob_constants()

```
void ECP4_ZZZ_frob_constants (
            FP2_YYY F[3] )
```

Calculates required Frobenius constants.
Calculate Frobenius constants

Parameters

F	array of FP2 precalculated constants

5.14.2.20 ECP4_ZZZ_frob()

```
void ECP4_ZZZ_frob (
            ECP4_ZZZ * P,
            FP2_YYY F[3],
            int n )
```

Multiplies an ECP4 instance P by the internal modulus p^n, using precalculated Frobenius constants.
Fast point multiplication using Frobenius

Parameters

P	ECP4 instance, on exit = p^n∗P
F	array of FP2 precalculated Frobenius constant
n	power of prime

5.14.2.21 ECP4_ZZZ_mul8()

```
void ECP4_ZZZ_mul8 (
            ECP4_ZZZ * P,
            ECP4_ZZZ * Q,
            BIG_XXX * b )
```

Calculates P=Sigma b[i]∗Q[i] for i=0 to 7.

Parameters

P	ECP4 instance, on exit = Sigma b[i]∗Q[i] for i=0 to 7
Q	ECP4 array of 4 points
b	BIG array of 4 multipliers

5.14.2.22 ECP4_ZZZ_cfp()

```
void ECP4_ZZZ_cfp (
            ECP4_ZZZ * Q )
```

Multiplies random point by co-factor.

Parameters

Q	ECP4 multiplied by co-factor

5.14.2.23 ECP4_ZZZ_map2point()

```
void ECP4_ZZZ_map2point (
            ECP4_ZZZ * Q,
            FP4_YYY * x )
```

Maps random BIG to curve point in constant time.

Parameters

Q	ECP4 instance
x	FP4 derived from hash

5.14.2.24 ECP4_ZZZ_hap2point()

```
void ECP4_ZZZ_hap2point (
            ECP4_ZZZ * Q,
            BIG_XXX x )
```

Maps random BIG to curve point using hunt-and-peck.

Parameters

Q	ECP4 instance
x	Fp derived from hash

5.14.2.25 ECP4_ZZZ_mapit()

```
void ECP4_ZZZ_mapit (
            ECP4_ZZZ * P,
            octet * W )
```

Maps random BIG to curve point of correct order.

Parameters

P	ECP4 instance of correct order
W	OCTET byte array to be mapped

5.14.2.26 ECP4_ZZZ_generator()

```
int ECP4_ZZZ_generator (
            ECP4_ZZZ * G )
```

Get Group Generator from ROM.

Parameters

G	ECP4 instance

Returns

1 if point exists, else 0

5.14.3 Variable Documentation

5.14.3.1 CURVE_B_I_ZZZ

```
const int CURVE_B_I_ZZZ [extern]
```

Elliptic curve B parameter

5.14.3.2 CURVE_B_ZZZ

```
const BIG_XXX CURVE_B_ZZZ [extern]
```

Elliptic curve B parameter

5.14.3.3 CURVE_Order_ZZZ

```
const BIG_XXX CURVE_Order_ZZZ [extern]
```

Elliptic curve group order

5.14.3.4 CURVE_Cof_ZZZ

```
const BIG_XXX CURVE_Cof_ZZZ [extern]
```

Elliptic curve cofactor

5.14.3.5 CURVE_Bnx_ZZZ

```
const BIG_XXX CURVE_Bnx_ZZZ [extern]
```

Elliptic curve parameter

5.14.3.6 CURVE_HTPC_ZZZ

```
const BIG_XXX CURVE_HTPC_ZZZ [extern]
```

Hash to Point precomputation

5.14.3.7 Fra_YYY

```
const BIG_XXX Fra_YYY [extern]
```

real part of curve Frobenius Constant

5.14.3.8 Frb_YYY

```
const BIG_XXX Frb_YYY [extern]
```

imaginary part of curve Frobenius Constant

5.14.3.9 CURVE_Gx_ZZZ

`const BIG_XXX CURVE_Gx_ZZZ [extern]`

x-coordinate of generator point in group G1

5.14.3.10 CURVE_Gy_ZZZ

`const BIG_XXX CURVE_Gy_ZZZ [extern]`

y-coordinate of generator point in group G1

5.14.3.11 CURVE_Pxaa_ZZZ

`const BIG_XXX CURVE_Pxaa_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.14.3.12 CURVE_Pxab_ZZZ

`const BIG_XXX CURVE_Pxab_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.14.3.13 CURVE_Pxba_ZZZ

`const BIG_XXX CURVE_Pxba_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.14.3.14 CURVE_Pxbb_ZZZ

`const BIG_XXX CURVE_Pxbb_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.14.3.15 CURVE_Pyaa_ZZZ

`const BIG_XXX CURVE_Pyaa_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.14.3.16 CURVE_Pyab_ZZZ

`const BIG_XXX CURVE_Pyab_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.14.3.17 CURVE_Pyba_ZZZ

`const BIG_XXX CURVE_Pyba_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.14.3.18 CURVE_Pybb_ZZZ

`const BIG_XXX CURVE_Pybb_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.15 ecp8.h File Reference

ECP2 Header File.

```
#include "fp8_YYY.h"
#include "config_curve_ZZZ.h"
```

Classes

- struct ECP8_ZZZ

  ECP8 Structure - Elliptic Curve Point over quadratic extension field.

Functions

- int ECP8_ZZZ_isinf (ECP8_ZZZ *P)

  Tests for ECP8 point equal to infinity.
- void ECP8_ZZZ_copy (ECP8_ZZZ *P, ECP8_ZZZ *Q)

  Copy ECP8 point to another ECP8 point.
- void ECP8_ZZZ_inf (ECP8_ZZZ *P)

  Set ECP8 to point-at-infinity.
- int ECP8_ZZZ_equals (ECP8_ZZZ *P, ECP8_ZZZ *Q)

  Tests for equality of two ECP8s.
- void ECP8_ZZZ_affine (ECP8_ZZZ *P)

  Converts an ECP8 point from Projective (x,y,z) coordinates to affine (x,y) coordinates.
- int ECP8_ZZZ_get (FP8_YYY *x, FP8_YYY *y, ECP8_ZZZ *P)

  Extract x and y coordinates of an ECP8 point P.
- void ECP8_ZZZ_output (ECP8_ZZZ *P)

  Formats and outputs an ECP8 point to the console, converted to affine coordinates.
- void ECP8_ZZZ_toOctet (octet *S, ECP8_ZZZ *P, bool c)

  Formats and outputs an ECP8 point to an octet string.
- int ECP8_ZZZ_fromOctet (ECP8_ZZZ *P, octet *S)

  Creates an ECP8 point from an octet string.
- void ECP8_ZZZ_rhs (FP8_YYY *r, FP8_YYY *x)

  Calculate Right Hand Side of curve equation y^2=f(x)
- int ECP8_ZZZ_set (ECP8_ZZZ *P, FP8_YYY *x, FP8_YYY *y)

  Set ECP8 to point(x,y) given x and y.
- int ECP8_ZZZ_setx (ECP8_ZZZ *P, FP8_YYY *x, int s)

  Set ECP to point(x,[y]) given x.
- void ECP8_ZZZ_neg (ECP8_ZZZ *P)

  Negation of an ECP8 point.
- void ECP8_ZZZ_reduce (ECP8_ZZZ *P)

  Reduction of an ECP8 point.
- int ECP8_ZZZ_dbl (ECP8_ZZZ *P)

  Doubles an ECP8 instance P.
- int ECP8_ZZZ_add (ECP8_ZZZ *P, ECP8_ZZZ *Q)

  Adds ECP8 instance Q to ECP8 instance P.
- void ECP8_ZZZ_sub (ECP8_ZZZ *P, ECP8_ZZZ *Q)

  Subtracts ECP instance Q from ECP8 instance P.
- void ECP8_ZZZ_mul (ECP8_ZZZ *P, BIG_XXX b)

  Multiplies an ECP8 instance P by a BIG, side-channel resistant.
- void ECP8_ZZZ_frob_constants (FP2_YYY F[3])

  Calculates required Frobenius constants.
- void ECP8_ZZZ_frob (ECP8_ZZZ *P, FP2_YYY F[3], int n)

  Multiplies an ECP8 instance P by the internal modulus p^n, using precalculated Frobenius constants.
- void ECP8_ZZZ_mul16 (ECP8_ZZZ *P, ECP8_ZZZ *Q, BIG_XXX *b)

  *Calculates P=Sigma b[i]*Q[i] for i=0 to 7.*
- void ECP8_ZZZ_cfp (ECP8_ZZZ *Q)

  Multiplies random point by co-factor.
- void ECP8_ZZZ_hap2point (ECP8_ZZZ *Q, BIG_XXX x)

Hashes random BIG to curve point using hunt-and-peck.

- void ECP8_ZZZ_map2point (ECP8_ZZZ ∗Q, FP8_YYY ∗x)

  Hashes random BIG to curve point in constant time.

- void ECP8_ZZZ_mapit (ECP8_ZZZ ∗P, octet ∗W)

  Maps random BIG to curve point of correct order.

- int ECP8_ZZZ_generator (ECP8_ZZZ ∗G)

  Get Group Generator from ROM.

Variables

- const BIG_XXX Fra_YYY
- const BIG_XXX Frb_YYY
- const int CURVE_B_I_ZZZ
- const BIG_XXX CURVE_B_ZZZ
- const BIG_XXX CURVE_Order_ZZZ
- const BIG_XXX CURVE_Cof_ZZZ
- const BIG_XXX CURVE_Bnx_ZZZ
- const BIG_XXX CURVE_HTPC_ZZZ
- const BIG_XXX CURVE_Gx
- const BIG_XXX CURVE_Gy
- const BIG_XXX CURVE_Pxaaa_ZZZ
- const BIG_XXX CURVE_Pxaab_ZZZ
- const BIG_XXX CURVE_Pxaba_ZZZ
- const BIG_XXX CURVE_Pxabb_ZZZ
- const BIG_XXX CURVE_Pxbaa_ZZZ
- const BIG_XXX CURVE_Pxbab_ZZZ
- const BIG_XXX CURVE_Pxbba_ZZZ
- const BIG_XXX CURVE_Pxbbb_ZZZ
- const BIG_XXX CURVE_Pyaaa_ZZZ
- const BIG_XXX CURVE_Pyaab_ZZZ
- const BIG_XXX CURVE_Pyaba_ZZZ
- const BIG_XXX CURVE_Pyabb_ZZZ
- const BIG_XXX CURVE_Pybaa_ZZZ
- const BIG_XXX CURVE_Pybab_ZZZ
- const BIG_XXX CURVE_Pybba_ZZZ
- const BIG_XXX CURVE_Pybbb_ZZZ

5.15.1 Detailed Description

ECP2 Header File.

Author

Mike Scott

5.15.2 Function Documentation

5.15.2.1 ECP8_ZZZ_isinf()

```
int ECP8_ZZZ_isinf (
                ECP8_ZZZ * P )
```

Tests for ECP8 point equal to infinity.

Parameters

P	ECP8 point to be tested

Returns

1 if infinity, else returns 0

5.15.2.2 ECP8_ZZZ_copy()

```
void ECP8_ZZZ_copy (
            ECP8_ZZZ * P,
            ECP8_ZZZ * Q )
```

Copy ECP8 point to another ECP8 point.

Parameters

P	ECP8 instance, on exit = Q
Q	ECP8 instance to be copied

5.15.2.3 ECP8_ZZZ_inf()

```
void ECP8_ZZZ_inf (
            ECP8_ZZZ * P )
```

Set ECP8 to point-at-infinity.

Parameters

P	ECP8 instance to be set to infinity

5.15.2.4 ECP8_ZZZ_equals()

```
int ECP8_ZZZ_equals (
            ECP8_ZZZ * P,
            ECP8_ZZZ * Q )
```

Tests for equality of two ECP8s.

Parameters

P	ECP8 instance to be compared
Q	ECP8 instance to be compared

Returns

1 if P=Q, else returns 0

5.15.2.5 ECP8_ZZZ_affine()

```
void ECP8_ZZZ_affine (
            ECP8_ZZZ * P )
```

Converts an ECP8 point from Projective (x,y,z) coordinates to affine (x,y) coordinates.

Parameters

P	ECP8 instance to be converted to affine form

5.15.2.6 ECP8_ZZZ_get()

```
int ECP8_ZZZ_get (
            FP8_YYY * x,
            FP8_YYY * y,
            ECP8_ZZZ * P )
```

Extract x and y coordinates of an ECP8 point P.
If x=y, returns only x

Parameters

x	FP8 on exit = x coordinate of point
y	FP8 on exit = y coordinate of point (unless x=y)
P	ECP8 instance (x,y)

Returns

-1 if P is point-at-infinity, else 0

5.15.2.7 ECP8_ZZZ_output()

```
void ECP8_ZZZ_output (
            ECP8_ZZZ * P )
```

Formats and outputs an ECP8 point to the console, converted to affine coordinates.

Parameters

P	ECP8 instance to be printed

5.15.2.8 ECP8_ZZZ_toOctet()

```
void ECP8_ZZZ_toOctet (
            octet * S,
            ECP8_ZZZ * P,
            bool c )
```

Formats and outputs an ECP8 point to an octet string.
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	ECP8 instance to be converted to an octet string
c	true for compression

5.15.2.9 ECP8_ZZZ_fromOctet()

```
int ECP8_ZZZ_fromOctet (
            ECP8_ZZZ * P,
            octet * S )
```

Creates an ECP8 point from an octet string.

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	ECP8 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

5.15.2.10 ECP8_ZZZ_rhs()

```
void ECP8_ZZZ_rhs (
            FP8_YYY * r,
            FP8_YYY * x )
```

Calculate Right Hand Side of curve equation y^2=f(x)

Function f(x)=x^3+Ax+B Used internally.

Parameters

r	FP8 value of f(x)
x	FP8 instance

5.15.2.11 ECP8_ZZZ_set()

```
int ECP8_ZZZ_set (
            ECP8_ZZZ * P,
            FP8_YYY * x,
            FP8_YYY * y )
```

Set ECP8 to point(x,y) given x and y.

Point P set to infinity if no such point on the curve.

Parameters

P	ECP8 instance to be set (x,y)
x	FP8 x coordinate of point
y	FP8 y coordinate of point

Returns

1 if point exists, else 0

5.15.2.12 ECP8_ZZZ_setx()

```
int ECP8_ZZZ_setx (
            ECP8_ZZZ * P,
            FP8_YYY * x,
            int s )
```

Set ECP to point(x,[y]) given 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
s	sign of y

Returns

1 if point exists, else 0

5.15.2.13 ECP8_ZZZ_neg()

```
void ECP8_ZZZ_neg (
        ECP8_ZZZ * P )
```

Negation of an ECP8 point.

Parameters

P	ECP8 instance, on exit = -P

5.15.2.14 ECP8_ZZZ_reduce()

```
void ECP8_ZZZ_reduce (
        ECP8_ZZZ * P )
```

Reduction of an ECP8 point.

Parameters

P	ECP8 instance, on exit (x,y) are reduced wrt the modulus

5.15.2.15 ECP8_ZZZ_dbl()

```
int ECP8_ZZZ_dbl (
        ECP8_ZZZ * P )
```

Doubles an ECP8 instance P.

Parameters

P	ECP8 instance, on exit =2∗P

5.15.2.16 ECP8_ZZZ_add()

```
int ECP8_ZZZ_add (
        ECP8_ZZZ * P,
        ECP8_ZZZ * Q )
```

Adds ECP8 instance Q to ECP8 instance P.

Parameters

P	ECP8 instance, on exit =P+Q
Q	ECP8 instance to be added to P

5.15.2.17 ECP8_ZZZ_sub()

```
void ECP8_ZZZ_sub (
            ECP8_ZZZ * P,
            ECP8_ZZZ * Q )
```

Subtracts ECP instance Q from ECP8 instance P.

Parameters

P	ECP8 instance, on exit =P-Q
Q	ECP8 instance to be subtracted from P

5.15.2.18 ECP8_ZZZ_mul()

```
void ECP8_ZZZ_mul (
            ECP8_ZZZ * P,
            BIG_XXX b )
```

Multiplies an ECP8 instance P by a BIG, side-channel resistant.
Uses fixed sized windows.

Parameters

P	ECP8 instance, on exit =b∗P
b	BIG number multiplier

5.15.2.19 ECP8_ZZZ_frob_constants()

```
void ECP8_ZZZ_frob_constants (
            FP2_YYY F[3] )
```

Calculates required Frobenius constants.
Calculate Frobenius constants

Parameters

F	array of FP2 precalculated constants

5.15.2.20 ECP8_ZZZ_frob()

```
void ECP8_ZZZ_frob (
            ECP8_ZZZ * P,
            FP2_YYY F[3],
            int n )
```

Multiplies an ECP8 instance P by the internal modulus p^n, using precalculated Frobenius constants.
Fast point multiplication using Frobenius

Parameters

P	ECP8 instance, on exit = p^n∗P
F	array of FP2 precalculated Frobenius constant
n	power of prime

5.15.2.21 ECP8_ZZZ_mul16()

```
void ECP8_ZZZ_mul16 (
            ECP8_ZZZ * P,
            ECP8_ZZZ * Q,
            BIG_XXX * b )
```

Calculates P=Sigma b[i]∗Q[i] for i=0 to 7.

Parameters

P	ECP8 instance, on exit = Sigma b[i]∗Q[i] for i=0 to 7
Q	ECP8 array of 4 points
b	BIG array of 4 multipliers

5.15.2.22 ECP8_ZZZ_cfp()

```
void ECP8_ZZZ_cfp (
            ECP8_ZZZ * Q )
```

Multiplies random point by co-factor.

Parameters

Q	ECP8 multiplied by co-factor

5.15.2.23 ECP8_ZZZ_hap2point()

```
void ECP8_ZZZ_hap2point (
            ECP8_ZZZ * Q,
            BIG_XXX x )
```

Hashes random BIG to curve point using hunt-and-peck.

Parameters

Q	ECP8 instance
x	Fp derived from hash

5.15.2.24 ECP8_ZZZ_map2point()

```
void ECP8_ZZZ_map2point (
            ECP8_ZZZ * Q,
            FP8_YYY * x )
```

Hashes random BIG to curve point in constant time.

Parameters

Q	ECP8 instance
x	FP8 derived from hash

5.15.2.25 ECP8_ZZZ_mapit()

```
void ECP8_ZZZ_mapit (
            ECP8_ZZZ * P,
            octet * W )
```

Maps random BIG to curve point of correct order.

Parameters

P	ECP8 instance of correct order
W	OCTET byte array to be mapped

5.15.2.26 ECP8_ZZZ_generator()

```
int ECP8_ZZZ_generator (
            ECP8_ZZZ * G )
```

Get Group Generator from ROM.

Parameters

G	ECP8 instance

Returns

1 if point exists, else 0

5.15.3 Variable Documentation

5.15.3.1 Fra_YYY

```
const BIG_XXX Fra_YYY [extern]
```

real part of BN curve Frobenius Constant

5.15.3.2 Frb_YYY

```
const BIG_XXX Frb_YYY [extern]
```

imaginary part of BN curve Frobenius Constant

5.15.3.3 CURVE_B_I_ZZZ

```
const int CURVE_B_I_ZZZ [extern]
```

Elliptic curve B parameter

5.15.3.4 CURVE_B_ZZZ

```
const BIG_XXX CURVE_B_ZZZ [extern]
```

Elliptic curve B parameter

5.15.3.5 CURVE_Order_ZZZ

`const BIG_XXX CURVE_Order_ZZZ [extern]`

Elliptic curve group order

5.15.3.6 CURVE_Cof_ZZZ

`const BIG_XXX CURVE_Cof_ZZZ [extern]`

Elliptic curve cofactor

5.15.3.7 CURVE_Bnx_ZZZ

`const BIG_XXX CURVE_Bnx_ZZZ [extern]`

Elliptic curve parameter

5.15.3.8 CURVE_HTPC_ZZZ

`const BIG_XXX CURVE_HTPC_ZZZ [extern]`

Hash to Point precomputation

5.15.3.9 CURVE_Gx

`const BIG_XXX CURVE_Gx [extern]`

x-coordinate of generator point in group G1

5.15.3.10 CURVE_Gy

`const BIG_XXX CURVE_Gy [extern]`

y-coordinate of generator point in group G1

5.15.3.11 CURVE_Pxaaa_ZZZ

`const BIG_XXX CURVE_Pxaaa_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.15.3.12 CURVE_Pxaab_ZZZ

`const BIG_XXX CURVE_Pxaab_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.15.3.13 CURVE_Pxaba_ZZZ

`const BIG_XXX CURVE_Pxaba_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.15.3.14 CURVE_Pxabb_ZZZ

`const BIG_XXX CURVE_Pxabb_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.15.3.15 CURVE_Pxbaa_ZZZ

`const BIG_XXX CURVE_Pxbaa_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.15.3.16 CURVE_Pxbab_ZZZ

`const BIG_XXX CURVE_Pxbab_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.15.3.17 CURVE_Pxbba_ZZZ

`const BIG_XXX CURVE_Pxbba_ZZZ [extern]`

real part of x-coordinate of generator point in group G2

5.15.3.18 CURVE_Pxbbb_ZZZ

`const BIG_XXX CURVE_Pxbbb_ZZZ [extern]`

imaginary part of x-coordinate of generator point in group G2

5.15.3.19 CURVE_Pyaaa_ZZZ

`const BIG_XXX CURVE_Pyaaa_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.15.3.20 CURVE_Pyaab_ZZZ

`const BIG_XXX CURVE_Pyaab_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.15.3.21 CURVE_Pyaba_ZZZ

`const BIG_XXX CURVE_Pyaba_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.15.3.22 CURVE_Pyabb_ZZZ

`const BIG_XXX CURVE_Pyabb_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.15.3.23 CURVE_Pybaa_ZZZ

`const BIG_XXX CURVE_Pybaa_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.15.3.24 CURVE_Pybab_ZZZ

`const BIG_XXX CURVE_Pybab_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.15.3.25 CURVE_Pybba_ZZZ

`const BIG_XXX CURVE_Pybba_ZZZ [extern]`

real part of y-coordinate of generator point in group G2

5.15.3.26 CURVE_Pybbb_ZZZ

`const BIG_XXX CURVE_Pybbb_ZZZ [extern]`

imaginary part of y-coordinate of generator point in group G2

5.16 ff.h File Reference

FF Header File.

```
#include "big_XXX.h"
#include "config_ff_WWW.h"
```

Macros

- #define HFLEN_WWW (FFLEN_WWW/2)
- #define P_MBITS_WWW (MODBYTES_XXX*8)
- #define P_TBITS_WWW (P_MBITS_WWW%BASEBITS_XXX)
- #define P_EXCESS_WWW(a) (((a[NLEN_XXX-1])>>(P_TBITS_WWW))+1)
- #define P_FEXCESS_WWW ((chunk)1<<(BASEBITS_XXX*NLEN_XXX-P_MBITS_WWW-1))

Functions

- void FF_WWW_copy (BIG_XXX *x, BIG_XXX *y, int n)

  Copy one FF element of given length to another.
- void FF_WWW_init (BIG_XXX *x, sign32 m, int n)

  Initialize an FF element of given length from a 32-bit integer m.
- void FF_WWW_zero (BIG_XXX *x, int n)

  Set FF element of given size to zero.
- int FF_WWW_iszilch (BIG_XXX *x, int n)

  Tests for FF element equal to zero.
- int FF_WWW_parity (BIG_XXX *x)

  return parity of an FF, that is the least significant bit
- int FF_WWW_lastbits (BIG_XXX *x, int m)

  return least significant m bits of an FF
- void FF_WWW_one (BIG_XXX *x, int n)

  Set FF element of given size to unity.
- int FF_WWW_comp (BIG_XXX *x, BIG_XXX *y, int n)

  Compares two FF numbers. Inputs must be normalised externally.
- void FF_WWW_add (BIG_XXX *x, BIG_XXX *y, BIG_XXX *z, int n)

  addition of two FFs
- void FF_WWW_sub (BIG_XXX *x, BIG_XXX *y, BIG_XXX *z, int n)

  subtraction of two FFs
- void FF_WWW_inc (BIG_XXX *x, int m, int n)

  increment an FF by an integer,and normalise
- void FF_WWW_dec (BIG_XXX *x, int m, int n)

  Decrement an FF by an integer,and normalise.
- void FF_WWW_norm (BIG_XXX *x, int n)

  Normalises the components of an FF.
- void FF_WWW_shl (BIG_XXX *x, int n)

  Shift left an FF by 1 bit.
- void FF_WWW_shr (BIG_XXX *x, int n)

  Shift right an FF by 1 bit.
- void FF_WWW_output (BIG_XXX *x, int n)

  Formats and outputs an FF to the console.
- void FF_WWW_rawoutput (BIG_XXX *x, int n)

  Formats and outputs an FF to the console, in raw form.
- void FF_WWW_toOctet (octet *S, BIG_XXX *x, int n)

  Formats and outputs an FF instance to an octet string.
- void FF_WWW_fromOctet (BIG_XXX *x, octet *S, int n)

  Populates an FF instance from an octet string.
- void FF_WWW_mul (BIG_XXX *x, BIG_XXX *y, BIG_XXX *z, int n)

  Multiplication of two FFs.
- void FF_WWW_mod (BIG_XXX *x, BIG_XXX *m, int n)

  Reduce FF mod a modulus.

- void FF_WWW_sqr (BIG_XXX ∗x, BIG_XXX ∗y, int n)

    Square an FF.
- void FF_WWW_dmod (BIG_XXX ∗x, BIG_XXX ∗y, BIG_XXX ∗z, int n)

    Reduces a double-length FF with respect to a given modulus.
- void FF_WWW_invmodp (BIG_XXX ∗x, BIG_XXX ∗y, BIG_XXX ∗z, int n)

    Invert an FF mod a prime modulus.
- void FF_WWW_random (BIG_XXX ∗x, csprng ∗R, int n)

    Create an FF from a random number generator.
- void FF_WWW_randomnum (BIG_XXX ∗x, BIG_XXX ∗y, csprng ∗R, int n)

    Create a random FF less than a given modulus from a random number generator.
- void FF_WWW_skpow (BIG_XXX ∗r, BIG_XXX ∗x, BIG_XXX ∗e, BIG_XXX ∗m, int n)

    Calculate r=x^e mod m, side channel resistant.
- void FF_WWW_skspow (BIG_XXX ∗r, BIG_XXX ∗x, BIG_XXX e, BIG_XXX ∗m, int n)

    Calculate r=x^e mod m, side channel resistant.
- void FF_WWW_power (BIG_XXX ∗r, BIG_XXX ∗x, int e, BIG_XXX ∗m, int n)

    Calculate r=x^e mod m.
- void FF_WWW_pow (BIG_XXX ∗r, BIG_XXX ∗x, BIG_XXX ∗e, BIG_XXX ∗m, int n)

    Calculate r=x^e mod m.
- int FF_WWW_cfactor (BIG_XXX ∗x, sign32 s, int n)

    Test if an FF has factor in common with integer s.
- int FF_WWW_prime (BIG_XXX ∗x, csprng ∗R, int n)

    Test if an FF is prime.
- void FF_WWW_pow2 (BIG_XXX ∗r, BIG_XXX ∗x, BIG_XXX e, BIG_XXX ∗y, BIG_XXX f, BIG_XXX ∗m, int n)

    Calculate r=x^e.y^f mod m.

5.16.1 Detailed Description

FF Header File.

Author

Mike Scott

5.16.2 Macro Definition Documentation

5.16.2.1 HFLEN_WWW

```
#define HFLEN_WWW (FFLEN_WWW/2)
```

Useful for half-size RSA private key operations

5.16.2.2 P_MBITS_WWW

```
#define P_MBITS_WWW (MODBYTES_XXX*8)
```

Number of bits in modulus

5.16.2.3 P_TBITS_WWW

```
#define P_TBITS_WWW (P_MBITS_WWW%BASEBITS_XXX)
```

TODO

5.16.2.4 P_EXCESS_WWW

```
#define P_EXCESS_WWW(
              a ) (((a[NLEN_XXX-1])>>(P_TBITS_WWW))+1)
```

TODO

5.16.2.5 P_FEXCESS_WWW

```
#define P_FEXCESS_WWW ((chunk)1<<(BASEBITS_XXX*NLEN_XXX-P_MBITS_WWW-1))
```

TODO

5.16.3 Function Documentation

5.16.3.1 FF_WWW_copy()

```
void FF_WWW_copy (
            BIG_XXX * x,
            BIG_XXX * y,
            int n )
```

Copy one FF element of given length to another.

Parameters

x	FF instance to be copied to, on exit = y
y	FF instance to be copied from
n	size of FF in BIGs

5.16.3.2 FF_WWW_init()

```
void FF_WWW_init (
            BIG_XXX * x,
            sign32 m,
            int n )
```

Initialize an FF element of given length from a 32-bit integer m.

Parameters

x	FF instance to be copied to, on exit = m
m	integer
n	size of FF in BIGs

5.16.3.3 FF_WWW_zero()

```
void FF_WWW_zero (
            BIG_XXX * x,
            int n )
```

Set FF element of given size to zero.

Parameters

x	FF instance to be set to zero
n	size of FF in BIGs

5.16.3.4 FF_WWW_iszilch()

```
int FF_WWW_iszilch (
```

```
BIG_XXX * x,
int n )
```

Tests for FF element equal to zero.

Parameters

x	FF number to be tested
n	size of FF in BIGs

Returns

1 if zero, else returns 0

5.16.3.5 FF_WWW_parity()

```
int FF_WWW_parity (
    BIG_XXX * x )
```

return parity of an FF, that is the least significant bit

Parameters

x	FF number

Returns

0 or 1

5.16.3.6 FF_WWW_lastbits()

```
int FF_WWW_lastbits (
    BIG_XXX * x,
    int m )
```

return least significant m bits of an FF

Parameters

x	FF number
m	number of bits to return. Assumed to be less than BASEBITS.

Returns

least significant n bits as an integer

5.16.3.7 FF_WWW_one()

```
void FF_WWW_one (
    BIG_XXX * x,
    int n )
```

Set FF element of given size to unity.

Parameters

x	FF instance to be set to unity
n	size of FF in BIGs

5.16.3.8 FF_WWW_comp()

```
int FF_WWW_comp (
            BIG_XXX * x,
            BIG_XXX * y,
            int n )
```

Compares two FF numbers. Inputs must be normalised externally.

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

5.16.3.9 FF_WWW_add()

```
void FF_WWW_add (
            BIG_XXX * x,
            BIG_XXX * y,
            BIG_XXX * z,
            int n )
```

addition of two FFs

Parameters

x	FF instance, on exit = y+z
y	FF instance
z	FF instance
n	size of FF in BIGs

5.16.3.10 FF_WWW_sub()

```
void FF_WWW_sub (
            BIG_XXX * x,
            BIG_XXX * y,
            BIG_XXX * z,
            int n )
```

subtraction of two FFs

Parameters

x	FF instance, on exit = y-z
y	FF instance
z	FF instance
n	size of FF in BIGs

5.16.3.11 FF_WWW_inc()

```
void FF_WWW_inc (
            BIG_XXX * x,
            int m,
            int n )
```

increment an FF by an integer,and normalise

Parameters

x	FF instance, on exit = x+m
m	an integer to be added to x
n	size of FF in BIGs

5.16.3.12 FF_WWW_dec()

```
void FF_WWW_dec (
            BIG_XXX * x,
            int m,
            int n )
```

Decrement an FF by an integer,and normalise.

Parameters

x	FF instance, on exit = x-m
m	an integer to be subtracted from x
n	size of FF in BIGs

5.16.3.13 FF_WWW_norm()

```
void FF_WWW_norm (
            BIG_XXX * x,
            int n )
```

Normalises the components of an FF.

Parameters

x	FF instance to be normalised
n	size of FF in BIGs

5.16.3.14 FF_WWW_shl()

```
void FF_WWW_shl (
            BIG_XXX * x,
            int n )
```

Shift left an FF by 1 bit.

Parameters

x	FF instance to be shifted left
n	size of FF in BIGs

5.16.3.15 FF_WWW_shr()

```
void FF_WWW_shr (
            BIG_XXX * x,
            int n )
```

Shift right an FF by 1 bit.

Parameters

x	FF instance to be shifted right
n	size of FF in BIGs

5.16.3.16 FF_WWW_output()

```
void FF_WWW_output (
            BIG_XXX * x,
            int n )
```

Formats and outputs an FF to the console.

Parameters

x	FF instance to be printed
n	size of FF in BIGs

5.16.3.17 FF_WWW_rawoutput()

```
void FF_WWW_rawoutput (
            BIG_XXX * x,
            int n )
```

Formats and outputs an FF to the console, in raw form.

Parameters

x	FF instance to be printed
n	size of FF in BIGs

5.16.3.18 FF_WWW_toOctet()

```
void FF_WWW_toOctet (
            octet * S,
            BIG_XXX * x,
            int n )
```

Formats and outputs an FF instance to an octet string.
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

5.16.3.19 FF_WWW_fromOctet()

```
void FF_WWW_fromOctet (
            BIG_XXX * x,
            octet * S,
            int n )
```

Populates an FF instance from an octet string.
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

5.16.3.20 FF_WWW_mul()

```
void FF_WWW_mul (
            BIG_XXX * x,
            BIG_XXX * y,
            BIG_XXX * z,
            int n )
```

Multiplication of two FFs.
Uses Karatsuba method internally

Parameters

x	FF instance, on exit = y*z
y	FF instance
z	FF instance
n	size of FF in BIGs

5.16.3.21 FF_WWW_mod()

```
void FF_WWW_mod (
            BIG_XXX * x,
            BIG_XXX * m,
            int n )
```

Reduce FF mod a modulus.
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

5.16.3.22 FF_WWW_sqr()

```
void FF_WWW_sqr (
            BIG_XXX * x,
            BIG_XXX * y,
            int n )
```

Square an FF.
Uses Karatsuba method internally

Parameters

x	FF instance, on exit = y^2
y	FF instance to be squared
n	size of FF in BIGs

5.16.3.23 FF_WWW_dmod()

```
void FF_WWW_dmod (
            BIG_XXX * x,
            BIG_XXX * y,
            BIG_XXX * z,
            int n )
```

Reduces a double-length FF with respect to a given modulus.
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

5.16.3.24 FF_WWW_invmodp()

```
void FF_WWW_invmodp (
            BIG_XXX * x,
            BIG_XXX * y,
            BIG_XXX * z,
            int n )
```

Invert an FF mod a prime modulus.

Parameters

x	FF instance, on exit = 1/y mod z
y	FF instance
z	FF prime modulus
n	size of FF in BIGs

5.16.3.25 FF_WWW_random()

```
void FF_WWW_random (
```

```
BIG_XXX * x,
csprng * R,
int n )
```

Create an FF from a random number generator.

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

5.16.3.26 FF_WWW_randomnum()

```
void FF_WWW_randomnum (
            BIG_XXX * x,
            BIG_XXX * y,
            csprng * R,
            int n )
```

Create a random FF less than a given modulus from a random number generator.

Parameters

x	FF instance, on exit x is a random number $<$ y
y	FF instance, the modulus
R	an instance of a Cryptographically Secure Random Number Generator
n	size of FF in BIGs

5.16.3.27 FF_WWW_skpow()

```
void FF_WWW_skpow (
            BIG_XXX * r,
            BIG_XXX * x,
            BIG_XXX * e,
            BIG_XXX * m,
            int n )
```

Calculate r=x^e mod m, side channel resistant.

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

5.16.3.28 FF_WWW_skspow()

```
void FF_WWW_skspow (
            BIG_XXX * r,
            BIG_XXX * x,
```

```
              BIG_XXX e,
              BIG_XXX * m,
              int n )
```

Calculate r=x^e mod m, side channel resistant.
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

5.16.3.29 FF_WWW_power()

```
void FF_WWW_power (
              BIG_XXX * r,
              BIG_XXX * x,
              int e,
              BIG_XXX * m,
              int n )
```

Calculate r=x^e mod m.
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

5.16.3.30 FF_WWW_pow()

```
void FF_WWW_pow (
              BIG_XXX * r,
              BIG_XXX * x,
              BIG_XXX * e,
              BIG_XXX * m,
              int n )
```

Calculate r=x^e mod m.

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

5.16.3.31 FF_WWW_cfactor()

```
int FF_WWW_cfactor (
            BIG_XXX * x,
            sign32 s,
            int n )
```

Test if an FF has factor in common with integer s.

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

5.16.3.32 FF_WWW_prime()

```
int FF_WWW_prime (
            BIG_XXX * x,
            csprng * R,
            int n )
```

Test if an FF is prime.
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

5.16.3.33 FF_WWW_pow2()

```
void FF_WWW_pow2 (
            BIG_XXX * r,
            BIG_XXX * x,
            BIG_XXX e,
            BIG_XXX * y,
            BIG_XXX f,
            BIG_XXX * m,
            int n )
```

Calculate r=x^e.y^f mod m.

Parameters

r	FF instance, on exit = x^e.y^f mod p
x	FF instance
e	BIG exponent

Parameters

y	FF instance
f	BIG exponent
m	FF modulus
n	size of FF in BIGs

5.17 fp.h File Reference

FP Header File.

```
#include "big_XXX.h"
#include "config_field_YYY.h"
```

Classes

- struct FP_YYY

  FP Structure - quadratic extension field.

Macros

- #define MODBITS_YYY MBITS_YYY
- #define TBITS_YYY (MBITS_YYY%BASEBITS_XXX)
- #define TMASK_YYY (((chunk)1<<TBITS_YYY)-1)
- #define FEXCESS_YYY (((sign32)1<<MAXXES_YYY)-1)
- #define OMASK_YYY (-((chunk)(1)<<TBITS_YYY))

Functions

- void FP_YYY_from_int (FP_YYY ∗x, int a)

  Create FP from integer.
- int FP_YYY_iszilch (FP_YYY ∗x)

  Tests for FP equal to zero mod Modulus.
- int FP_YYY_islarger (FP_YYY ∗x)

  Tests for lexically largest.
- void FP_YYY_toBytes (char ∗b, FP_YYY ∗x)

  Serialize out FP
- void FP_YYY_fromBytes (FP_YYY ∗x, char ∗b)

  Serialize in FP
- int FP_YYY_isunity (FP_YYY ∗x)

  Tests for FP equal to one mod Modulus.
- void FP_YYY_zero (FP_YYY ∗x)

  Set FP to zero.
- void FP_YYY_copy (FP_YYY ∗y, FP_YYY ∗x)

  Copy an FP.
- void FP_YYY_rcopy (FP_YYY ∗y, const BIG_XXX x)

  Copy from ROM to an FP.
- int FP_YYY_equals (FP_YYY ∗x, FP_YYY ∗y)

  Compares two FPs.
- void FP_YYY_cswap (FP_YYY ∗x, FP_YYY ∗y, int s)

Conditional constant time swap of two FP numbers.
- void FP_YYY_cmove (FP_YYY ∗x, FP_YYY ∗y, int s)

  Conditional copy of FP number.
- void FP_YYY_nres (FP_YYY ∗y, BIG_XXX x)

  Converts from BIG integer to residue form mod Modulus.
- void FP_YYY_redc (BIG_XXX x, FP_YYY ∗y)

  Converts from residue form back to BIG integer form.
- void FP_YYY_one (FP_YYY ∗x)

  Sets FP to representation of unity in residue form.
- int FP_YYY_sign (FP_YYY ∗x)

  returns "sign" of an FP
- void FP_YYY_mod (BIG_XXX r, DBIG_XXX d)

  Reduces DBIG to BIG exploiting special form of the modulus.
- void FP_YYY_mul (FP_YYY ∗x, FP_YYY ∗y, FP_YYY ∗z)

  Fast Modular multiplication of two FPs, mod Modulus.
- void FP_YYY_imul (FP_YYY ∗x, FP_YYY ∗y, int i)

  Fast Modular multiplication of an FP, by a small integer, mod Modulus.
- void FP_YYY_sqr (FP_YYY ∗x, FP_YYY ∗y)

  Fast Modular squaring of an FP, mod Modulus.
- void FP_YYY_add (FP_YYY ∗x, FP_YYY ∗y, FP_YYY ∗z)

  Modular addition of two FPs, mod Modulus.
- void FP_YYY_sub (FP_YYY ∗x, FP_YYY ∗y, FP_YYY ∗z)

  Modular subtraction of two FPs, mod Modulus.
- void FP_YYY_div2 (FP_YYY ∗x, FP_YYY ∗y)

  Modular division by 2 of an FP, mod Modulus.
- void FP_YYY_pow (FP_YYY ∗x, FP_YYY ∗y, BIG_XXX z)

  Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.
- void FP_YYY_progen (FP_YYY ∗r, FP_YYY ∗x)

  Inverse square root precalculation.
- void FP_YYY_sqrt (FP_YYY ∗x, FP_YYY ∗y, FP_YYY ∗h)

  Fast Modular square root of a an FP, mod Modulus.
- void FP_YYY_neg (FP_YYY ∗x, FP_YYY ∗y)

  Modular negation of a an FP, mod Modulus.
- void FP_YYY_output (FP_YYY ∗x)

  Outputs an FP number to the console.
- void FP_YYY_rawoutput (FP_YYY ∗x)

  Outputs an FP number to the console, in raw form.
- void FP_YYY_reduce (FP_YYY ∗x)

  Reduces possibly unreduced FP mod Modulus.
- void FP_YYY_norm (FP_YYY ∗x)

  normalizes FP
- int FP_YYY_qr (FP_YYY ∗x, FP_YYY ∗h)

  Tests for FP a quadratic residue mod Modulus.
- int FP_YYY_invsqrt (FP_YYY ∗i, FP_YYY ∗s, FP_YYY ∗x)

  Simultaneous Inverse and Square root.
- int FP_YYY_tpo (FP_YYY ∗i, FP_YYY ∗s)

  Simultaneous Inverse and Square root of different numbers.
- void FP_YYY_inv (FP_YYY ∗x, FP_YYY ∗y, FP_YYY ∗h)

  Modular inverse of a an FP, mod Modulus.
- void FP_YYY_rand (FP_YYY ∗x, csprng ∗rng)

  Generate random FP.

Variables

- const BIG_XXX Modulus_YYY
- const BIG_XXX ROI_YYY
- const BIG_XXX R2modp_YYY
- const BIG_XXX CRu_YYY
- const BIG_XXX SQRTm3_YYY
- const BIG_XXX TWK_YYY
- const chunk MConst_YYY

5.17.1 Detailed Description

FP Header File.

Author

Mike Scott

5.17.2 Macro Definition Documentation

5.17.2.1 MODBITS_YYY

```
#define MODBITS_YYY MBITS_YYY
```

Number of bits in Modulus for selected curve

5.17.2.2 TBITS_YYY

```
#define TBITS_YYY (MBITS_YYY%BASEBITS_XXX)
```

Number of active bits in top word

5.17.2.3 TMASK_YYY

```
#define TMASK_YYY (((chunk)1<<TBITS_YYY)-1)
```

Mask for active bits in top word

5.17.2.4 FEXCESS_YYY

```
#define FEXCESS_YYY (((sign32)1<<MAXXES_YYY)-1)
```

2^(BASEBITS∗NLEN-MODBITS)-1 - normalised BIG can be multiplied by less than this before reduction

5.17.2.5 OMASK_YYY

```
#define OMASK_YYY (-((chunk)(1)<<TBITS_YYY))
```

for masking out overflow bits

5.17.3 Function Documentation

5.17.3.1 FP_YYY_from_int()

```
void FP_YYY_from_int (
            FP_YYY * x,
            int a )
```

Create FP from integer.

Parameters

x	FP to be initialised
a	integer

5.17.3.2 FP_YYY_iszilch()

```
int FP_YYY_iszilch (
            FP_YYY * x )
```

Tests for FP equal to zero mod Modulus.

Parameters

x	FP number to be tested

Returns

1 if zero, else returns 0

5.17.3.3 FP_YYY_islarger()

```
int FP_YYY_islarger (
            FP_YYY * x )
```

Tests for lexically largest.

Parameters

x	FP number to be tested if larger than -x

Returns

1 if larger, else returns 0

5.17.3.4 FP_YYY_toBytes()

```
void FP_YYY_toBytes (
            char * b,
            FP_YYY * x )
```

Serialize out FP

Parameters

b	buffer for output
x	FP number to be serialized

5.17.3.5 FP_YYY_fromBytes()

```
void FP_YYY_fromBytes (
            FP_YYY * x,
            char * b )
```

Serialize in FP

Parameters

x	FP number to be serialized
b	buffer for input

5.17.3.6 FP_YYY_isunity()

```
int FP_YYY_isunity (
            FP_YYY * x )
```

Tests for FP equal to one mod Modulus.

Parameters

x	FP number to be tested

Returns

1 if one, else returns 0

5.17.3.7 FP_YYY_zero()

```
void FP_YYY_zero (
            FP_YYY * x )
```

Set FP to zero.

Parameters

x	FP number to be set to 0

5.17.3.8 FP_YYY_copy()

```
void FP_YYY_copy (
            FP_YYY * y,
            FP_YYY * x )
```

Copy an FP.

Parameters

y	FP number to be copied to
x	FP to be copied from

5.17.3.9 FP_YYY_rcopy()

```
void FP_YYY_rcopy (
            FP_YYY * y,
            const BIG_XXX x )
```

Copy from ROM to an FP.

Parameters

Parameters

y	FP number to be copied to
x	BIG to be copied from ROM

5.17.3.10 FP_YYY_equals()

```
int FP_YYY_equals (
            FP_YYY * x,
            FP_YYY * y )
```

Compares two FPs.

Parameters

x	FP number
y	FP number

Returns

1 if equal, else returns 0

5.17.3.11 FP_YYY_cswap()

```
void FP_YYY_cswap (
            FP_YYY * x,
            FP_YYY * y,
            int s )
```

Conditional constant time swap of two FP numbers.
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

5.17.3.12 FP_YYY_cmove()

```
void FP_YYY_cmove (
            FP_YYY * x,
            FP_YYY * y,
            int s )
```

Conditional copy of FP number.
Conditionally copies second parameter to the first (without branching)

Parameters

x	an FP number
y	another FP number

Parameters

s	copy takes place if not equal to 0

5.17.3.13 FP_YYY_nres()

```
void FP_YYY_nres (
            FP_YYY * y,
            BIG_XXX x )
```

Converts from BIG integer to residue form mod Modulus.

Parameters

x	BIG number to be converted
y	FP result

5.17.3.14 FP_YYY_redc()

```
void FP_YYY_redc (
            BIG_XXX x,
            FP_YYY * y )
```

Converts from residue form back to BIG integer form.

Parameters

y	FP number to be converted to BIG
x	BIG result

5.17.3.15 FP_YYY_one()

```
void FP_YYY_one (
            FP_YYY * x )
```

Sets FP to representation of unity in residue form.

Parameters

x	FP number to be set equal to unity.

5.17.3.16 FP_YYY_sign()

```
int FP_YYY_sign (
            FP_YYY * x )
```

returns "sign" of an FP

Parameters

x	FP number

Returns

0 for positive, 1 for negative

5.17.3.17 FP_YYY_mod()

```
void FP_YYY_mod (
            BIG_XXX r,
            DBIG_XXX d )
```

Reduces DBIG to BIG exploiting special form of the modulus.
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

5.17.3.18 FP_YYY_mul()

```
void FP_YYY_mul (
            FP_YYY * x,
            FP_YYY * y,
            FP_YYY * z )
```

Fast Modular multiplication of two FPs, mod Modulus.
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

5.17.3.19 FP_YYY_imul()

```
void FP_YYY_imul (
            FP_YYY * x,
            FP_YYY * y,
            int i )
```

Fast Modular multiplication of an FP, by a small integer, mod Modulus.

Parameters

x	FP number, on exit the modular product = y∗i mod Modulus
y	FP number, the multiplicand
i	a small number, the multiplier

5.17.3.20 FP_YYY_sqr()

```
void FP_YYY_sqr (
            FP_YYY * x,
```

```
FP_YYY * y )
```

Fast Modular squaring of an FP, mod Modulus.
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

5.17.3.21 FP_YYY_add()

```
void FP_YYY_add (
        FP_YYY * x,
        FP_YYY * y,
        FP_YYY * z )
```

Modular addition of two FPs, mod Modulus.

Parameters

x	FP number, on exit the modular sum = y+z mod Modulus
y	FP number
z	FP number

5.17.3.22 FP_YYY_sub()

```
void FP_YYY_sub (
        FP_YYY * x,
        FP_YYY * y,
        FP_YYY * z )
```

Modular subtraction of two FPs, mod Modulus.

Parameters

x	FP number, on exit the modular difference = y-z mod Modulus
y	FP number
z	FP number

5.17.3.23 FP_YYY_div2()

```
void FP_YYY_div2 (
        FP_YYY * x,
        FP_YYY * y )
```

Modular division by 2 of an FP, mod Modulus.

Parameters

x	FP number, on exit =y/2 mod Modulus
y	FP number

5.17.3.24 FP_YYY_pow()

```
void FP_YYY_pow (
            FP_YYY * x,
            FP_YYY * y,
            BIG_XXX z )
```

Fast Modular exponentiation of an FP, to the power of a BIG, mod Modulus.

Parameters

x	FP number, on exit = y^z mod Modulus
y	FP number
z	BIG number exponent

5.17.3.25 FP_YYY_progen()

```
void FP_YYY_progen (
            FP_YYY * r,
            FP_YYY * x )
```

Inverse square root precalculation.

Parameters

r	FP number, on exit = x^(p-2*e-1)/2^(e+1) mod Modulus
x	FP number

5.17.3.26 FP_YYY_sqrt()

```
void FP_YYY_sqrt (
            FP_YYY * x,
            FP_YYY * y,
            FP_YYY * h )
```

Fast Modular square root of a an FP, mod Modulus.

Parameters

x	FP number, on exit = sqrt(y) mod Modulus
y	FP number, the number whose square root is calculated
h	an optional precalculation

5.17.3.27 FP_YYY_neg()

```
void FP_YYY_neg (
            FP_YYY * x,
            FP_YYY * y )
```

Modular negation of a an FP, mod Modulus.

Parameters

x	FP number, on exit = -y mod Modulus
y	FP number

5.17.3.28 FP_YYY_output()

```
void FP_YYY_output (
        FP_YYY * x )
```

Outputs an FP number to the console.
Converts from residue form before output

Parameters

x	an FP number

5.17.3.29 FP_YYY_rawoutput()

```
void FP_YYY_rawoutput (
        FP_YYY * x )
```

Outputs an FP number to the console, in raw form.

Parameters

x	a BIG number

5.17.3.30 FP_YYY_reduce()

```
void FP_YYY_reduce (
        FP_YYY * x )
```

Reduces possibly unreduced FP mod Modulus.

Parameters

x	FP number, on exit reduced mod Modulus

5.17.3.31 FP_YYY_norm()

```
void FP_YYY_norm (
        FP_YYY * x )
```

normalizes FP

Parameters

x	FP number, on exit normalized

5.17.3.32 FP_YYY_qr()

```
int FP_YYY_qr (
        FP_YYY * x,
        FP_YYY * h )
```

Tests for FP a quadratic residue mod Modulus.

Parameters

x	FP number to be tested
h	an optional precalculation

Returns

1 if quadratic residue, else returns 0 if quadratic non-residue

5.17.3.33 FP_YYY_invsqrt()

```
int FP_YYY_invsqrt (
            FP_YYY * i,
            FP_YYY * s,
            FP_YYY * x )
```

Simultaneous Inverse and Square root.

Parameters

i	FP number, on exit = 1/x mod Modulus
s	FP number, on exit = sqrt(x) mod Modulus
x	FP number

Returns

1 if quadratic residue, else returns 0 if quadratic non-residue

5.17.3.34 FP_YYY_tpo()

```
int FP_YYY_tpo (
            FP_YYY * i,
            FP_YYY * s )
```

Simultaneous Inverse and Square root of different numbers.

Parameters

i	FP number, on exit = 1/i mod Modulus
s	FP number, on exit = sqrt(s) mod Modulus

Returns

1 if quadratic residue, else returns 0 if quadratic non-residue

5.17.3.35 FP_YYY_inv()

```
void FP_YYY_inv (
            FP_YYY * x,
            FP_YYY * y,
            FP_YYY * h )
```

Modular inverse of a an FP, mod Modulus.

Parameters

x	FP number, on exit = 1/y mod Modulus
y	FP number
h	an optional input precalculation

5.17.3.36 FP_YYY_rand()

```
void FP_YYY_rand (
            FP_YYY * x,
            csprng * rng )
```

Generate random FP.

Parameters

x	random FP number
rng	random number generator

5.17.4 Variable Documentation

5.17.4.1 Modulus_YYY

```
const BIG_XXX Modulus_YYY [extern]
```

Actual Modulus set in rom_field_yyy.c

5.17.4.2 ROI_YYY

```
const BIG_XXX ROI_YYY [extern]
```

Root of unity set in rom_field_yyy.c

5.17.4.3 R2modp_YYY

```
const BIG_XXX R2modp_YYY [extern]
```

Montgomery constant

5.17.4.4 CRu_YYY

```
const BIG_XXX CRu_YYY [extern]
```

Cube Root of Unity

5.17.4.5 SQRTm3_YYY

```
const BIG_XXX SQRTm3_YYY [extern]
```

Square root of -3

5.17.4.6 TWK_YYY

```
const BIG_XXX TWK_YYY [extern]
```

Tweak for square roots, pre-calculated from field norm

5.17.4.7 MConst_YYY

```
const chunk MConst_YYY [extern]
```

Constant associated with Modulus - for Montgomery = 1/p mod 2^BASEBITS

5.18 fp12.h File Reference

FP12 Header File.

```
#include "fp4_YYY.h"
```

Classes

- struct FP12_YYY

  FP12 Structure - towered over three FP4.

Functions

- int FP12_YYY_iszilch (FP12_YYY ∗x)

  Tests for FP12 equal to zero.
- int FP12_YYY_isunity (FP12_YYY ∗x)

  Tests for FP12 equal to unity.
- void FP12_YYY_copy (FP12_YYY ∗x, FP12_YYY ∗y)

  Copy FP12 to another FP12.
- void FP12_YYY_one (FP12_YYY ∗x)

  Set FP12 to unity.
- void FP12_YYY_zero (FP12_YYY ∗x)

  Set FP12 to zero.
- int FP12_YYY_equals (FP12_YYY ∗x, FP12_YYY ∗y)

  Tests for equality of two FP12s.
- void FP12_YYY_conj (FP12_YYY ∗x, FP12_YYY ∗y)

  Conjugation of FP12.
- void FP12_YYY_from_FP4 (FP12_YYY ∗x, FP4_YYY ∗a)

  Initialise FP12 from single FP4.
- void FP12_YYY_from_FP4s (FP12_YYY ∗x, FP4_YYY ∗a, FP4_YYY ∗b, FP4_YYY ∗c)

  Initialise FP12 from three FP4s.
- void FP12_YYY_usqr (FP12_YYY ∗x, FP12_YYY ∗y)

  Fast Squaring of an FP12 in "unitary" form.
- void FP12_YYY_sqr (FP12_YYY ∗x, FP12_YYY ∗y)

  Squaring an FP12.
- void FP12_YYY_smul (FP12_YYY ∗x, FP12_YYY ∗y)

  Fast multiplication of two sparse FP12s that arises from ATE pairing line functions.
- void FP12_YYY_ssmul (FP12_YYY ∗x, FP12_YYY ∗y)

  Fast multiplication of what may be sparse multiplicands.
- void FP12_YYY_mul (FP12_YYY ∗x, FP12_YYY ∗y)

  Full unconditional Multiplication of two FP12s.
- void FP12_YYY_inv (FP12_YYY ∗x, FP12_YYY ∗y)

  Inverting an FP12.
- void FP12_YYY_pow (FP12_YYY ∗r, FP12_YYY ∗x, BIG_XXX b)

  Raises an FP12 to the power of a BIG.
- void FP12_YYY_pinpow (FP12_YYY ∗x, int i, int b)

  Raises an FP12 instance x to a small integer power, side-channel resistant.
- void FP12_YYY_compow (FP4_YYY ∗c, FP12_YYY ∗x, BIG_XXX e, BIG_XXX r)

  Raises an FP12 instance x to a BIG power, compressed to FP4.
- void FP12_YYY_pow4 (FP12_YYY ∗r, FP12_YYY ∗x, BIG_XXX ∗b)

  Calculate x[0]^b[0].x[1]^b[1].x[2]^b[2].x[3]^b[3], side-channel resistant.
- void FP12_YYY_frob (FP12_YYY ∗x, FP2_YYY ∗f)

Raises an FP12 to the power of the internal modulus p, using the Frobenius.

- void FP12_YYY_reduce (FP12_YYY *x)

  Reduces all components of possibly unreduced FP12 mod Modulus.
- void FP12_YYY_norm (FP12_YYY *x)

  Normalises the components of an FP12.
- void FP12_YYY_output (FP12_YYY *x)

  Formats and outputs an FP12 to the console.
- void FP12_YYY_toOctet (octet *S, FP12_YYY *x)

  Formats and outputs an FP12 instance to an octet string.
- void FP12_YYY_fromOctet (FP12_YYY *x, octet *S)

  Creates an FP12 instance from an octet string.
- void FP12_YYY_trace (FP4_YYY *t, FP12_YYY *x)

  Calculate the trace of an FP12.
- void FP12_YYY_cmove (FP12_YYY *x, FP12_YYY *y, int s)

  Conditional copy of FP12 number.

Variables

- const BIG_XXX Fra_YYY
- const BIG_XXX Frb_YYY

5.18.1 Detailed Description

FP12 Header File.

Author

Mike Scott

5.18.2 Function Documentation

5.18.2.1 FP12_YYY_iszilch()

```
int FP12_YYY_iszilch (
              FP12_YYY * x )
```

Tests for FP12 equal to zero.

Parameters

x	FP12 number to be tested

Returns

1 if zero, else returns 0

5.18.2.2 FP12_YYY_isunity()

```
int FP12_YYY_isunity (
              FP12_YYY * x )
```

Tests for FP12 equal to unity.

Parameters

x	FP12 number to be tested

Returns

1 if unity, else returns 0

5.18.2.3 FP12_YYY_copy()

```
void FP12_YYY_copy (
            FP12_YYY * x,
            FP12_YYY * y )
```

Copy FP12 to another FP12.

Parameters

x	FP12 instance, on exit = y
y	FP12 instance to be copied

5.18.2.4 FP12_YYY_one()

```
void FP12_YYY_one (
            FP12_YYY * x )
```

Set FP12 to unity.

Parameters

x	FP12 instance to be set to one

5.18.2.5 FP12_YYY_zero()

```
void FP12_YYY_zero (
            FP12_YYY * x )
```

Set FP12 to zero.

Parameters

x	FP12 instance to be set to zero

5.18.2.6 FP12_YYY_equals()

```
int FP12_YYY_equals (
            FP12_YYY * x,
            FP12_YYY * y )
```

Tests for equality of two FP12s.

Parameters

x	FP12 instance to be compared
y	FP12 instance to be compared

Returns

1 if x=y, else returns 0

5.18.2.7 FP12_YYY_conj()

```
void FP12_YYY_conj (
            FP12_YYY * x,
            FP12_YYY * y )
```

Conjugation of FP12.

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

5.18.2.8 FP12_YYY_from_FP4()

```
void FP12_YYY_from_FP4 (
            FP12_YYY * x,
            FP4_YYY * a )
```

Initialise FP12 from single FP4.

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

5.18.2.9 FP12_YYY_from_FP4s()

```
void FP12_YYY_from_FP4s (
            FP12_YYY * x,
            FP4_YYY * a,
            FP4_YYY * b,
            FP4_YYY * c )
```

Initialise FP12 from three FP4s.

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

5.18.2.10 FP12_YYY_usqr()

```
void FP12_YYY_usqr (
            FP12_YYY * x,
```

```
            FP12_YYY * y )
```

Fast Squaring of an FP12 in "unitary" form.

Parameters

x	FP12 instance, on exit = y^2
y	FP4 instance, must be unitary

5.18.2.11 FP12_YYY_sqr()

```
void FP12_YYY_sqr (
            FP12_YYY * x,
            FP12_YYY * y )
```

Squaring an FP12.

Parameters

x	FP12 instance, on exit = y^2
y	FP12 instance

5.18.2.12 FP12_YYY_smul()

```
void FP12_YYY_smul (
            FP12_YYY * x,
            FP12_YYY * y )
```

Fast multiplication of two sparse FP12s that arises from ATE pairing line functions.

Parameters

x	FP12 instance, on exit = x*y
y	FP12 instance, of special form

5.18.2.13 FP12_YYY_ssmul()

```
void FP12_YYY_ssmul (
            FP12_YYY * x,
            FP12_YYY * y )
```

Fast multiplication of what may be sparse multiplicands.

Parameters

x	FP12 instance, on exit = x*y
y	FP12 instance, of special form

5.18.2.14 FP12_YYY_mul()

```
void FP12_YYY_mul (
            FP12_YYY * x,
            FP12_YYY * y )
```

Full unconditional Multiplication of two FP12s.

Parameters

x	FP12 instance, on exit = x∗y
y	FP12 instance, the multiplier

5.18.2.15 FP12_YYY_inv()

```
void FP12_YYY_inv (
            FP12_YYY * x,
            FP12_YYY * y )
```

Inverting an FP12.

Parameters

x	FP12 instance, on exit = 1/y
y	FP12 instance

5.18.2.16 FP12_YYY_pow()

```
void FP12_YYY_pow (
            FP12_YYY * r,
            FP12_YYY * x,
            BIG_XXX b )
```

Raises an FP12 to the power of a BIG.

Parameters

r	FP12 instance, on exit = y^b
x	FP12 instance
b	BIG number

5.18.2.17 FP12_YYY_pinpow()

```
void FP12_YYY_pinpow (
            FP12_YYY * x,
            int i,
            int b )
```

Raises an FP12 instance x to a small integer power, side-channel resistant.

Parameters

x	FP12 instance, on exit = x^i
i	small integer exponent
b	maximum number of bits in exponent

5.18.2.18 FP12_YYY_compow()

```
void FP12_YYY_compow (
            FP4_YYY * c,
            FP12_YYY * x,
            BIG_XXX e,
            BIG_XXX r )
```

Raises an FP12 instance x to a BIG power, compressed to FP4.

Parameters

c	FP4 instance, on exit = x^(e mod r) as FP4
x	FP12 input
e	BIG exponent
r	BIG group order

5.18.2.19 FP12_YYY_pow4()

```
void FP12_YYY_pow4 (
            FP12_YYY * r,
            FP12_YYY * x,
            BIG_XXX * b )
```

Calculate x[0]^b[0].x[1]^b[1].x[2]^b[2].x[3]^b[3], side-channel resistant.

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

5.18.2.20 FP12_YYY_frob()

```
void FP12_YYY_frob (
            FP12_YYY * x,
            FP2_YYY * f )
```

Raises an FP12 to the power of the internal modulus p, using the Frobenius.

Parameters

x	FP12 instance, on exit = x^p
f	FP2 precalculated Frobenius constant

5.18.2.21 FP12_YYY_reduce()

```
void FP12_YYY_reduce (
            FP12_YYY * x )
```

Reduces all components of possibly unreduced FP12 mod Modulus.

Parameters

x	FP12 instance, on exit reduced mod Modulus

5.18.2.22 FP12_YYY_norm()

```
void FP12_YYY_norm (
            FP12_YYY * x )
```

Normalises the components of an FP12.

Parameters

x	FP12 instance to be normalised

5.18.2.23 FP12_YYY_output()

```
void FP12_YYY_output (
            FP12_YYY * x )
```

Formats and outputs an FP12 to the console.

Parameters

x	FP12 instance to be printed

5.18.2.24 FP12_YYY_toOctet()

```
void FP12_YYY_toOctet (
            octet * S,
            FP12_YYY * x )
```

Formats and outputs an FP12 instance to an octet string.
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

5.18.2.25 FP12_YYY_fromOctet()

```
void FP12_YYY_fromOctet (
            FP12_YYY * x,
            octet * S )
```

Creates an FP12 instance from an octet string.
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

5.18.2.26 FP12_YYY_trace()

```
void FP12_YYY_trace (
            FP4_YYY * t,
            FP12_YYY * x )
```

Calculate the trace of an FP12.

Parameters

t	FP4 trace of x, on exit = tr(x)
x	FP12 instance

5.18.2.27 FP12_YYY_cmove()

```
void FP12_YYY_cmove (
            FP12_YYY * x,
            FP12_YYY * y,
            int s )
```

Conditional copy of FP12 number.
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

5.18.3 Variable Documentation

5.18.3.1 Fra_YYY

```
const BIG_XXX Fra_YYY [extern]
```

real part of BN curve Frobenius Constant

5.18.3.2 Frb_YYY

```
const BIG_XXX Frb_YYY [extern]
```

imaginary part of BN curve Frobenius Constant

5.19 fp16.h File Reference

FP16 Header File.

```
#include "fp8_YYY.h"
#include "config_curve_ZZZ.h"
```

Classes

- struct FP16_YYY

    FP16 Structure - towered over two FP8.

Functions

- int FP16_YYY_iszilch (FP16_YYY *x)

  Tests for FP16 equal to zero.
- void FP16_YYY_toBytes (char *b, FP16_YYY *x)

  Serialize in FP16
- void FP16_YYY_fromBytes (FP16_YYY *x, char *b)

  Serialize out FP16
- int FP16_YYY_isunity (FP16_YYY *x)

  Tests for FP16 equal to unity.
- int FP16_YYY_equals (FP16_YYY *x, FP16_YYY *y)

  Tests for equality of two FP16s.
- int FP16_YYY_isreal (FP16_YYY *x)

  Tests for FP16 having only a real part and no imaginary part.
- void FP16_YYY_from_FP8s (FP16_YYY *x, FP8_YYY *a, FP8_YYY *b)

  Initialise FP16 from two FP8s.
- void FP16_YYY_from_FP8 (FP16_YYY *x, FP8_YYY *a)

  Initialise FP16 from single FP8.
- void FP16_YYY_from_FP8H (FP16_YYY *x, FP8_YYY *a)

  Initialise FP16 from single FP8.
- void FP16_YYY_copy (FP16_YYY *x, FP16_YYY *y)

  Copy FP16 to another FP16.
- void FP16_YYY_zero (FP16_YYY *x)

  Set FP16 to zero.
- void FP16_YYY_one (FP16_YYY *x)

  Set FP16 to unity.
- void FP16_YYY_neg (FP16_YYY *x, FP16_YYY *y)

  Negation of FP16.
- void FP16_YYY_conj (FP16_YYY *x, FP16_YYY *y)

  Conjugation of FP16.
- void FP16_YYY_nconj (FP16_YYY *x, FP16_YYY *y)

  Negative conjugation of FP16.
- void FP16_YYY_add (FP16_YYY *x, FP16_YYY *y, FP16_YYY *z)

  addition of two FP16s
- void FP16_YYY_sub (FP16_YYY *x, FP16_YYY *y, FP16_YYY *z)

  subtraction of two FP16s
- void FP16_YYY_pmul (FP16_YYY *x, FP16_YYY *y, FP8_YYY *a)

  Multiplication of an FP16 by an FP8.
- void FP16_YYY_qmul (FP16_YYY *x, FP16_YYY *y, FP2_YYY *a)

  Multiplication of an FP16 by an FP2.
- void FP16_YYY_tmul (FP16_YYY *x, FP16_YYY *y, FP_YYY *a)

  Multiplication of an FP16 by an FP.
- void FP16_YYY_imul (FP16_YYY *x, FP16_YYY *y, int i)

  Multiplication of an FP16 by a small integer.
- void FP16_YYY_sqr (FP16_YYY *x, FP16_YYY *y)

  Squaring an FP16.
- void FP16_YYY_mul (FP16_YYY *x, FP16_YYY *y, FP16_YYY *z)

  Multiplication of two FP16s.
- void FP16_YYY_inv (FP16_YYY *x, FP16_YYY *y)

  Inverting an FP16.

- void FP16_YYY_output (FP16_YYY *x)

  Formats and outputs an FP16 to the console.
- void FP16_YYY_rawoutput (FP16_YYY *x)

  Formats and outputs an FP16 to the console in raw form (for debugging)
- void FP16_YYY_times_i (FP16_YYY *x)

  multiplies an FP16 instance by irreducible polynomial sqrt(1+sqrt(-1))
- void FP16_YYY_times_i2 (FP16_YYY *x)

  multiplies an FP16 instance by irreducible polynomial (1+sqrt(-1))
- void FP16_YYY_times_i4 (FP16_YYY *x)

  multiplies an FP16 instance by irreducible polynomial (1+sqrt(-1))
- void FP16_YYY_norm (FP16_YYY *x)

  Normalises the components of an FP16.
- void FP16_YYY_reduce (FP16_YYY *x)

  Reduces all components of possibly unreduced FP16 mod Modulus.
- void FP16_YYY_pow (FP16_YYY *x, FP16_YYY *y, BIG_XXX b)

  Raises an FP16 to the power of a BIG.
- void FP16_YYY_frob (FP16_YYY *x, FP2_YYY *f)

  Raises an FP16 to the power of the internal modulus p, using the Frobenius.
- void FP16_YYY_xtr_A (FP16_YYY *r, FP16_YYY *w, FP16_YYY *x, FP16_YYY *y, FP16_YYY *z)

  *Calculates the XTR addition function r=w*x-conj(x)*y+z.*
- void FP16_YYY_xtr_D (FP16_YYY *r, FP16_YYY *x)

  *Calculates the XTR doubling function r=x^2-2*conj(x)*
- void FP16_YYY_xtr_pow (FP16_YYY *r, FP16_YYY *x, BIG_XXX b)

  Calculates FP16 trace of an FP12 raised to the power of a BIG number.
- void FP16_YYY_xtr_pow2 (FP16_YYY *r, FP16_YYY *c, FP16_YYY *d, FP16_YYY *e, FP16_YYY *f, BIG_XXX a, BIG_XXX b)

  Calculates FP16 trace of c^a.d^b, where c and d are derived from FP16 traces of FP12s.
- void FP16_YYY_cmove (FP16_YYY *x, FP16_YYY *y, int s)

  Conditional copy of FP16 number.

5.19.1 Detailed Description

FP16 Header File.

Author

Mike Scott

5.19.2 Function Documentation

5.19.2.1 FP16_YYY_iszilch()

```
int FP16_YYY_iszilch (
                FP16_YYY * x )
```

Tests for FP16 equal to zero.

Parameters

x	FP16 number to be tested

Returns

1 if zero, else returns 0

5.19.2.2 FP16_YYY_toBytes()

```
void FP16_YYY_toBytes (
            char * b,
            FP16_YYY * x )
```

Serialize in FP16

Parameters

b	buffer for output
x	FP16 number to be serialized

5.19.2.3 FP16_YYY_fromBytes()

```
void FP16_YYY_fromBytes (
            FP16_YYY * x,
            char * b )
```

Serialize out FP16

Parameters

x	FP16 number to be serialized
b	buffer for input

5.19.2.4 FP16_YYY_isunity()

```
int FP16_YYY_isunity (
            FP16_YYY * x )
```

Tests for FP16 equal to unity.

Parameters

x	FP16 number to be tested

Returns

1 if unity, else returns 0

5.19.2.5 FP16_YYY_equals()

```
int FP16_YYY_equals (
            FP16_YYY * x,
            FP16_YYY * y )
```

Tests for equality of two FP16s.

Parameters

x	FP16 instance to be compared
y	FP16 instance to be compared

Returns

1 if x=y, else returns 0

5.19.2.6 FP16_YYY_isreal()

```
int FP16_YYY_isreal (
            FP16_YYY * x )
```

Tests for FP16 having only a real part and no imaginary part.

Parameters

x	FP16 number to be tested

Returns

1 if real, else returns 0

5.19.2.7 FP16_YYY_from_FP8s()

```
void FP16_YYY_from_FP8s (
            FP16_YYY * x,
            FP8_YYY * a,
            FP8_YYY * b )
```

Initialise FP16 from two FP8s.

Parameters

x	FP16 instance to be initialised
a	FP8 to form real part of FP16
b	FP8 to form imaginary part of FP16

5.19.2.8 FP16_YYY_from_FP8()

```
void FP16_YYY_from_FP8 (
            FP16_YYY * x,
            FP8_YYY * a )
```

Initialise FP16 from single FP8.
Imaginary part is set to zero

Parameters

x	FP16 instance to be initialised
a	FP8 to form real part of FP16

5.19.2.9 FP16_YYY_from_FP8H()

```
void FP16_YYY_from_FP8H (
            FP16_YYY * x,
            FP8_YYY * a )
```

Initialise FP16 from single FP8.
real part is set to zero

Parameters

x	FP16 instance to be initialised
a	FP8 to form imaginary part of FP16

5.19.2.10 FP16_YYY_copy()

```
void FP16_YYY_copy (
            FP16_YYY * x,
            FP16_YYY * y )
```

Copy FP16 to another FP16.

Parameters

x	FP16 instance, on exit = y
y	FP16 instance to be copied

5.19.2.11 FP16_YYY_zero()

```
void FP16_YYY_zero (
            FP16_YYY * x )
```

Set FP16 to zero.

Parameters

x	FP16 instance to be set to zero

5.19.2.12 FP16_YYY_one()

```
void FP16_YYY_one (
            FP16_YYY * x )
```

Set FP16 to unity.

Parameters

x	FP16 instance to be set to one

5.19.2.13 FP16_YYY_neg()

```
void FP16_YYY_neg (
            FP16_YYY * x,
            FP16_YYY * y )
```

Negation of FP16.

Parameters

x	FP16 instance, on exit = -y
y	FP16 instance

5.19.2.14 FP16_YYY_conj()

```
void FP16_YYY_conj (
            FP16_YYY * x,
            FP16_YYY * y )
```

Conjugation of FP16.
If y=(a,b) on exit x=(a,-b)

Parameters

x	FP16 instance, on exit = conj(y)
y	FP16 instance

5.19.2.15 FP16_YYY_nconj()

```
void FP16_YYY_nconj (
            FP16_YYY * x,
            FP16_YYY * y )
```

Negative conjugation of FP16.
If y=(a,b) on exit x=(-a,b)

Parameters

x	FP16 instance, on exit = -conj(y)
y	FP16 instance

5.19.2.16 FP16_YYY_add()

```
void FP16_YYY_add (
            FP16_YYY * x,
            FP16_YYY * y,
            FP16_YYY * z )
```

addition of two FP16s

Parameters

x	FP16 instance, on exit = y+z
y	FP16 instance
z	FP16 instance

5.19.2.17 FP16_YYY_sub()

```
void FP16_YYY_sub (
        FP16_YYY * x,
        FP16_YYY * y,
        FP16_YYY * z )
```

subtraction of two FP16s

Parameters

x	FP16 instance, on exit = y-z
y	FP16 instance
z	FP16 instance

5.19.2.18 FP16_YYY_pmul()

```
void FP16_YYY_pmul (
        FP16_YYY * x,
        FP16_YYY * y,
        FP8_YYY * a )
```

Multiplication of an FP16 by an FP8.

Parameters

x	FP16 instance, on exit = y∗a
y	FP16 instance
a	FP8 multiplier

5.19.2.19 FP16_YYY_qmul()

```
void FP16_YYY_qmul (
        FP16_YYY * x,
        FP16_YYY * y,
        FP2_YYY * a )
```

Multiplication of an FP16 by an FP2.

Parameters

x	FP16 instance, on exit = y∗a
y	FP16 instance
a	FP2 multiplier

5.19.2.20 FP16_YYY_tmul()

```
void FP16_YYY_tmul (
        FP16_YYY * x,
        FP16_YYY * y,
        FP_YYY * a )
```

Multiplication of an FP16 by an FP.

Parameters

x	FP16 instance, on exit = y∗a
y	FP16 instance
a	FP multiplier

5.19.2.21 FP16_YYY_imul()

```
void FP16_YYY_imul (
            FP16_YYY * x,
            FP16_YYY * y,
            int i )
```

Multiplication of an FP16 by a small integer.

Parameters

x	FP16 instance, on exit = y∗i
y	FP16 instance
i	an integer

5.19.2.22 FP16_YYY_sqr()

```
void FP16_YYY_sqr (
            FP16_YYY * x,
            FP16_YYY * y )
```

Squaring an FP16.

Parameters

x	FP16 instance, on exit = y^2
y	FP16 instance

5.19.2.23 FP16_YYY_mul()

```
void FP16_YYY_mul (
            FP16_YYY * x,
            FP16_YYY * y,
            FP16_YYY * z )
```

Multiplication of two FP16s.

Parameters

x	FP16 instance, on exit = y∗z
y	FP16 instance
z	FP16 instance

5.19.2.24 FP16_YYY_inv()

```
void FP16_YYY_inv (
        FP16_YYY * x,
        FP16_YYY * y )
```

Inverting an FP16.

Parameters

x	FP16 instance, on exit = 1/y
y	FP16 instance

5.19.2.25 FP16_YYY_output()

```
void FP16_YYY_output (
        FP16_YYY * x )
```

Formats and outputs an FP16 to the console.

Parameters

x	FP16 instance to be printed

5.19.2.26 FP16_YYY_rawoutput()

```
void FP16_YYY_rawoutput (
        FP16_YYY * x )
```

Formats and outputs an FP16 to the console in raw form (for debugging)

Parameters

x	FP16 instance to be printed

5.19.2.27 FP16_YYY_times_i()

```
void FP16_YYY_times_i (
        FP16_YYY * x )
```

multiplies an FP16 instance by irreducible polynomial sqrt(1+sqrt(-1))

Parameters

x	FP16 instance, on exit = sqrt(1+sqrt(-1)*x

5.19.2.28 FP16_YYY_times_i2()

```
void FP16_YYY_times_i2 (
        FP16_YYY * x )
```

multiplies an FP16 instance by irreducible polynomial (1+sqrt(-1))

Parameters

x	FP16 instance, on exit = sqrt(1+sqrt(-1))^2*x

5.19.2.29 FP16_YYY_times_i4()

```
void FP16_YYY_times_i4 (
        FP16_YYY * x )
```

multiplies an FP16 instance by irreducible polynomial (1+sqrt(-1))

Parameters

x	FP16 instance, on exit = sqrt(1+sqrt(-1))^4*x

5.19.2.30 FP16_YYY_norm()

```
void FP16_YYY_norm (
        FP16_YYY * x )
```

Normalises the components of an FP16.

Parameters

x	FP16 instance to be normalised

5.19.2.31 FP16_YYY_reduce()

```
void FP16_YYY_reduce (
        FP16_YYY * x )
```

Reduces all components of possibly unreduced FP16 mod Modulus.

Parameters

x	FP16 instance, on exit reduced mod Modulus

5.19.2.32 FP16_YYY_pow()

```
void FP16_YYY_pow (
        FP16_YYY * x,
        FP16_YYY * y,
        BIG_XXX b )
```

Raises an FP16 to the power of a BIG.

Parameters

x	FP16 instance, on exit = y^b
y	FP16 instance
b	BIG number

5.19.2.33 FP16_YYY_frob()

```
void FP16_YYY_frob (
            FP16_YYY * x,
            FP2_YYY * f )
```

Raises an FP16 to the power of the internal modulus p, using the Frobenius.

Parameters

x	FP16 instance, on exit = x^p
f	FP2 precalculated Frobenius constant

5.19.2.34 FP16_YYY_xtr_A()

```
void FP16_YYY_xtr_A (
            FP16_YYY * r,
            FP16_YYY * w,
            FP16_YYY * x,
            FP16_YYY * y,
            FP16_YYY * z )
```

Calculates the XTR addition function r=w∗x-conj(x)∗y+z.

Parameters

r	FP16 instance, on exit = w∗x-conj(x)∗y+z
w	FP16 instance
x	FP16 instance
y	FP16 instance
z	FP16 instance

5.19.2.35 FP16_YYY_xtr_D()

```
void FP16_YYY_xtr_D (
            FP16_YYY * r,
            FP16_YYY * x )
```

Calculates the XTR doubling function r=x^2-2∗conj(x)

Parameters

r	FP16 instance, on exit = x^2-2∗conj(x)
x	FP16 instance

5.19.2.36 FP16_YYY_xtr_pow()

```
void FP16_YYY_xtr_pow (
            FP16_YYY * r,
            FP16_YYY * x,
            BIG_XXX b )
```

Calculates FP16 trace of an FP12 raised to the power of a BIG number.
XTR single exponentiation

Parameters

r	FP16 instance, on exit = trace(w^b)
x	FP16 instance, trace of an FP12 w
b	BIG number

5.19.2.37 FP16_YYY_xtr_pow2()

```
void FP16_YYY_xtr_pow2 (
            FP16_YYY * r,
            FP16_YYY * c,
            FP16_YYY * d,
            FP16_YYY * e,
            FP16_YYY * f,
            BIG_XXX a,
            BIG_XXX b )
```

Calculates FP16 trace of c^a.d^b, where c and d are derived from FP16 traces of FP12s.
XTR double exponentiation Assumes c=tr(x^m), d=tr(x^n), e=tr(x^(m-n)), f=tr(x^(m-2n))

Parameters

r	FP16 instance, on exit = trace(c^a.d^b)
c	FP16 instance, trace of an FP12
d	FP16 instance, trace of an FP12
e	FP16 instance, trace of an FP12
f	FP16 instance, trace of an FP12
a	BIG number
b	BIG number

5.19.2.38 FP16_YYY_cmove()

```
void FP16_YYY_cmove (
            FP16_YYY * x,
            FP16_YYY * y,
            int s )
```

Conditional copy of FP16 number.
Conditionally copies second parameter to the first (without branching)

Parameters

x	FP16 instance, set to y if s!=0
y	another FP16 instance
s	copy only takes place if not equal to 0

5.20 fp2.h File Reference

FP2 Header File.

```
#include "fp_YYY.h"
```

Classes

- struct FP2_YYY

  FP2 Structure - quadratic extension field.

Functions

- int FP2_YYY_iszilch (FP2_YYY ∗x)

  Tests for FP2 equal to zero.
- int FP2_YYY_islarger (FP2_YYY ∗x)

  Tests for lexically larger.
- void FP2_YYY_toBytes (char ∗b, FP2_YYY ∗x)

  Serialize out FP2
- void FP2_YYY_fromBytes (FP2_YYY ∗x, char ∗b)

  Serialize in FP2
- void FP2_YYY_cmove (FP2_YYY ∗x, FP2_YYY ∗y, int s)

  Conditional copy of FP2 number.
- int FP2_YYY_isunity (FP2_YYY ∗x)

  Tests for FP2 equal to one.
- int FP2_YYY_equals (FP2_YYY ∗x, FP2_YYY ∗y)

  Tests for equality of two FP2s.
- void FP2_YYY_from_FPs (FP2_YYY ∗x, FP_YYY ∗a, FP_YYY ∗b)

  Initialise FP2 from two FP numbers.
- void FP2_YYY_from_BIGs (FP2_YYY ∗x, BIG_XXX a, BIG_XXX b)

  Initialise FP2 from two BIG integers.
- void FP2_YYY_from_ints (FP2_YYY ∗x, int a, int b)

  Initialise FP2 from two integers.
- void FP2_YYY_from_FP (FP2_YYY ∗x, FP_YYY ∗a)

  Initialise FP2 from single FP.
- void FP2_YYY_from_BIG (FP2_YYY ∗x, BIG_XXX a)

  Initialise FP2 from single BIG.
- void FP2_YYY_copy (FP2_YYY ∗x, FP2_YYY ∗y)

  Copy FP2 to another FP2.
- void FP2_YYY_zero (FP2_YYY ∗x)

  Set FP2 to zero.
- void FP2_YYY_one (FP2_YYY ∗x)

  Set FP2 to unity.
- void FP2_YYY_rcopy (FP2_YYY ∗w, const BIG_XXX a, const BIG_XXX b)

  Copy from ROM to an FP2.
- int FP2_YYY_sign (FP2_YYY ∗x)

  Sign of FP2.
- void FP2_YYY_neg (FP2_YYY ∗x, FP2_YYY ∗y)

  Negation of FP2.
- void FP2_YYY_conj (FP2_YYY ∗x, FP2_YYY ∗y)

  Conjugation of FP2.
- void FP2_YYY_add (FP2_YYY ∗x, FP2_YYY ∗y, FP2_YYY ∗z)

  addition of two FP2s
- void FP2_YYY_sub (FP2_YYY ∗x, FP2_YYY ∗y, FP2_YYY ∗z)

  subtraction of two FP2s
- void FP2_YYY_pmul (FP2_YYY ∗x, FP2_YYY ∗y, FP_YYY ∗b)

Multiplication of an FP2 by an FP.
- void FP2_YYY_imul (FP2_YYY ∗x, FP2_YYY ∗y, int i)

  Multiplication of an FP2 by a small integer.
- void FP2_YYY_sqr (FP2_YYY ∗x, FP2_YYY ∗y)

  Squaring an FP2.
- void FP2_YYY_mul (FP2_YYY ∗x, FP2_YYY ∗y, FP2_YYY ∗z)

  Multiplication of two FP2s.
- void FP2_YYY_output (FP2_YYY ∗x)

  Formats and outputs an FP2 to the console.
- void FP2_YYY_rawoutput (FP2_YYY ∗x)

  Formats and outputs an FP2 to the console in raw form (for debugging)
- void FP2_YYY_inv (FP2_YYY ∗x, FP2_YYY ∗y, FP_YYY ∗h)

  Inverting an FP2.
- void FP2_YYY_div2 (FP2_YYY ∗x, FP2_YYY ∗y)

  Divide an FP2 by 2.
- void FP2_YYY_mul_ip (FP2_YYY ∗x)

  Multiply an FP2 by (1+sqrt(-1))
- void FP2_YYY_div_ip2 (FP2_YYY ∗x)

  Divide an FP2 by (1+sqrt(-1))/2 -.
- void FP2_YYY_div_ip (FP2_YYY ∗x)

  Divide an FP2 by (1+sqrt(-1))
- void FP2_YYY_norm (FP2_YYY ∗x)

  Normalises the components of an FP2.
- void FP2_YYY_reduce (FP2_YYY ∗x)

  Reduces all components of possibly unreduced FP2 mod Modulus.
- void FP2_YYY_pow (FP2_YYY ∗x, FP2_YYY ∗y, BIG_XXX b)

  Raises an FP2 to the power of a BIG.
- int FP2_YYY_qr (FP2_YYY ∗x, FP_YYY ∗h)

  Test FP2 for QR.
- void FP2_YYY_sqrt (FP2_YYY ∗x, FP2_YYY ∗y, FP_YYY ∗h)

  Square root of an FP2.
- void FP2_YYY_times_i (FP2_YYY ∗x)

  Multiply an FP2 by sqrt(-1)
- void FP2_YYY_rand (FP2_YYY ∗x, csprng ∗rng)

  Generate random FP2.

5.20.1 Detailed Description

FP2 Header File.

Author

Mike Scott

5.20.2 Function Documentation

5.20.2.1 FP2_YYY_iszilch()

```
int FP2_YYY_iszilch (
        FP2_YYY * x )
```

Tests for FP2 equal to zero.

Parameters

x	FP2 number to be tested

Returns

1 if zero, else returns 0

5.20.2.2 FP2_YYY_islarger()

```
int FP2_YYY_islarger (
            FP2_YYY * x )
```

Tests for lexically larger.

Parameters

x	FP2 number to be tested if larger than -x

Returns

1 if larger, else returns 0

5.20.2.3 FP2_YYY_toBytes()

```
void FP2_YYY_toBytes (
            char * b,
            FP2_YYY * x )
```

Serialize out FP2

Parameters

b	buffer for output
x	FP2 number to be serialized

5.20.2.4 FP2_YYY_fromBytes()

```
void FP2_YYY_fromBytes (
            FP2_YYY * x,
            char * b )
```

Serialize in FP2

Parameters

x	FP2 number to be serialized
b	buffer for input

5.20.2.5 FP2_YYY_cmove()

```
void FP2_YYY_cmove (
            FP2_YYY * x,
            FP2_YYY * y,
            int s )
```

Conditional copy of FP2 number.
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

5.20.2.6 FP2_YYY_isunity()

```
int FP2_YYY_isunity (
            FP2_YYY * x )
```

Tests for FP2 equal to one.

Parameters

x	FP2 instance to be tested

Returns

1 if x=1, else returns 0

5.20.2.7 FP2_YYY_equals()

```
int FP2_YYY_equals (
            FP2_YYY * x,
            FP2_YYY * y )
```

Tests for equality of two FP2s.

Parameters

x	FP2 instance to be compared
y	FP2 instance to be compared

Returns

1 if x=y, else returns 0

5.20.2.8 FP2_YYY_from_FPs()

```
void FP2_YYY_from_FPs (
            FP2_YYY * x,
            FP_YYY * a,
            FP_YYY * b )
```

Initialise FP2 from two FP numbers.

Parameters

x	FP2 instance to be initialised
a	FP to form real part of FP2
b	FP to form imaginary part of FP2

5.20.2.9 FP2_YYY_from_BIGs()

```
void FP2_YYY_from_BIGs (
            FP2_YYY * x,
            BIG_XXX a,
            BIG_XXX b )
```

Initialise FP2 from two BIG integers.

Parameters

x	FP2 instance to be initialised
a	BIG to form real part of FP2
b	BIG to form imaginary part of FP2

5.20.2.10 FP2_YYY_from_ints()

```
void FP2_YYY_from_ints (
            FP2_YYY * x,
            int a,
            int b )
```

Initialise FP2 from two integers.

Parameters

x	FP2 instance to be initialised
a	int to form real part of FP2
b	int to form imaginary part of FP2

5.20.2.11 FP2_YYY_from_FP()

```
void FP2_YYY_from_FP (
            FP2_YYY * x,
            FP_YYY * a )
```

Initialise FP2 from single FP.
Imaginary part is set to zero

Parameters

x	FP2 instance to be initialised
a	FP to form real part of FP2

5.20.2.12 FP2_YYY_from_BIG()

```
void FP2_YYY_from_BIG (
            FP2_YYY * x,
            BIG_XXX a )
```

Initialise FP2 from single BIG.
Imaginary part is set to zero

Parameters

x	FP2 instance to be initialised
a	BIG to form real part of FP2

5.20.2.13 FP2_YYY_copy()

```
void FP2_YYY_copy (
            FP2_YYY * x,
            FP2_YYY * y )
```

Copy FP2 to another FP2.

Parameters

x	FP2 instance, on exit = y
y	FP2 instance to be copied

5.20.2.14 FP2_YYY_zero()

```
void FP2_YYY_zero (
            FP2_YYY * x )
```

Set FP2 to zero.

Parameters

x	FP2 instance to be set to zero

5.20.2.15 FP2_YYY_one()

```
void FP2_YYY_one (
            FP2_YYY * x )
```

Set FP2 to unity.

Parameters

x	FP2 instance to be set to one

5.20.2.16 FP2_YYY_rcopy()

```
void FP2_YYY_rcopy (
            FP2_YYY * w,
            const BIG_XXX a,
```

```
const BIG_XXX b )
```

Copy from ROM to an FP2.

Parameters

w	FP2 number to be copied to
a	BIG real part to be copied from ROM
b	BIG imag part to be copied from ROM

5.20.2.17 FP2_YYY_sign()

```
int FP2_YYY_sign (
        FP2_YYY * x )
```

Sign of FP2.

Parameters

x	FP2 instance

Returns

"sign" of FP2

5.20.2.18 FP2_YYY_neg()

```
void FP2_YYY_neg (
        FP2_YYY * x,
        FP2_YYY * y )
```

Negation of FP2.

Parameters

x	FP2 instance, on exit = -y
y	FP2 instance

5.20.2.19 FP2_YYY_conj()

```
void FP2_YYY_conj (
        FP2_YYY * x,
        FP2_YYY * y )
```

Conjugation of FP2.
If y=(a,b) on exit x=(a,-b)

Parameters

x	FP2 instance, on exit = conj(y)
y	FP2 instance

5.20.2.20 FP2_YYY_add()

```
void FP2_YYY_add (
            FP2_YYY * x,
            FP2_YYY * y,
            FP2_YYY * z )
```

addition of two FP2s

Parameters

x	FP2 instance, on exit = y+z
y	FP2 instance
z	FP2 instance

5.20.2.21 FP2_YYY_sub()

```
void FP2_YYY_sub (
            FP2_YYY * x,
            FP2_YYY * y,
            FP2_YYY * z )
```

subtraction of two FP2s

Parameters

x	FP2 instance, on exit = y-z
y	FP2 instance
z	FP2 instance

5.20.2.22 FP2_YYY_pmul()

```
void FP2_YYY_pmul (
            FP2_YYY * x,
            FP2_YYY * y,
            FP_YYY * b )
```

Multiplication of an FP2 by an FP.

Parameters

x	FP2 instance, on exit = y*b
y	FP2 instance
b	FP residue

5.20.2.23 FP2_YYY_imul()

```
void FP2_YYY_imul (
            FP2_YYY * x,
            FP2_YYY * y,
            int i )
```

Multiplication of an FP2 by a small integer.

Parameters

x	FP2 instance, on exit = y∗i
y	FP2 instance
i	an integer

5.20.2.24 FP2_YYY_sqr()

```
void FP2_YYY_sqr (
            FP2_YYY * x,
            FP2_YYY * y )
```

Squaring an FP2.

Parameters

x	FP2 instance, on exit = y^2
y	FP2 instance

5.20.2.25 FP2_YYY_mul()

```
void FP2_YYY_mul (
            FP2_YYY * x,
            FP2_YYY * y,
            FP2_YYY * z )
```

Multiplication of two FP2s.

Parameters

x	FP2 instance, on exit = y∗z
y	FP2 instance
z	FP2 instance

5.20.2.26 FP2_YYY_output()

```
void FP2_YYY_output (
            FP2_YYY * x )
```

Formats and outputs an FP2 to the console.

Parameters

x	FP2 instance

5.20.2.27 FP2_YYY_rawoutput()

```
void FP2_YYY_rawoutput (
            FP2_YYY * x )
```

Formats and outputs an FP2 to the console in raw form (for debugging)

Parameters

x	FP2 instance

5.20.2.28 FP2_YYY_inv()

```
void FP2_YYY_inv (
        FP2_YYY * x,
        FP2_YYY * y,
        FP_YYY * h )
```

Inverting an FP2.

Parameters

x	FP2 instance, on exit = 1/y
y	FP2 instance
h	optional input hint

5.20.2.29 FP2_YYY_div2()

```
void FP2_YYY_div2 (
        FP2_YYY * x,
        FP2_YYY * y )
```

Divide an FP2 by 2.

Parameters

x	FP2 instance, on exit = y/2
y	FP2 instance

5.20.2.30 FP2_YYY_mul_ip()

```
void FP2_YYY_mul_ip (
        FP2_YYY * x )
```

Multiply an FP2 by (1+sqrt(-1))
Note that (1+sqrt(-1)) is irreducible for FP4

Parameters

x	FP2 instance, on exit = x∗(1+sqrt(-1))

5.20.2.31 FP2_YYY_div_ip2()

```
void FP2_YYY_div_ip2 (
        FP2_YYY * x )
```

Divide an FP2 by (1+sqrt(-1))/2 -.
Note that (1+sqrt(-1)) is irreducible for FP4

Parameters

x	FP2 instance, on exit = 2x/(1+sqrt(-1))

5.20.2.32 FP2_YYY_div_ip()

```
void FP2_YYY_div_ip (
            FP2_YYY * x )
```

Divide an FP2 by (1+sqrt(-1))
Note that (1+sqrt(-1)) is irreducible for FP4

Parameters

x	FP2 instance, on exit = x/(1+sqrt(-1))

5.20.2.33 FP2_YYY_norm()

```
void FP2_YYY_norm (
            FP2_YYY * x )
```

Normalises the components of an FP2.

Parameters

x	FP2 instance to be normalised

5.20.2.34 FP2_YYY_reduce()

```
void FP2_YYY_reduce (
            FP2_YYY * x )
```

Reduces all components of possibly unreduced FP2 mod Modulus.

Parameters

x	FP2 instance, on exit reduced mod Modulus

5.20.2.35 FP2_YYY_pow()

```
void FP2_YYY_pow (
            FP2_YYY * x,
            FP2_YYY * y,
            BIG_XXX b )
```

Raises an FP2 to the power of a BIG.

Parameters

x	FP2 instance, on exit = y^b
y	FP2 instance
b	BIG number

5.20.2.36 FP2_YYY_qr()

```
int FP2_YYY_qr (
            FP2_YYY * x,
            FP_YYY * h )
```

Test FP2 for QR.

Parameters

x	FP2 instance
h	optional generated hint

Returns

true or false

5.20.2.37 FP2_YYY_sqrt()

```
void FP2_YYY_sqrt (
            FP2_YYY * x,
            FP2_YYY * y,
            FP_YYY * h )
```

Square root of an FP2.

Parameters

x	FP2 instance, on exit = sqrt(y)
y	FP2 instance
h	optional input hint

5.20.2.38 FP2_YYY_times_i()

```
void FP2_YYY_times_i (
            FP2_YYY * x )
```

Multiply an FP2 by sqrt(-1)
Note that -1 is QNR

Parameters

x	FP2 instance, on exit = x*sqrt(-1)

5.20.2.39 FP2_YYY_rand()

```
void FP2_YYY_rand (
            FP2_YYY * x,
            csprng * rng )
```

Generate random FP2.

Parameters

x	random FP2 number

Parameters

rng	random number generator

5.21 fp24.h File Reference

FP24 Header File.
`#include "fp8_YYY.h"`

Classes

- struct FP24_YYY

  FP12 Structure - towered over three FP8.

Functions

- int FP24_YYY_iszilch (FP24_YYY ∗x)

  Tests for FP24 equal to zero.
- int FP24_YYY_isunity (FP24_YYY ∗x)

  Tests for FP24 equal to unity.
- void FP24_YYY_copy (FP24_YYY ∗x, FP24_YYY ∗y)

  Copy FP24 to another FP24.
- void FP24_YYY_one (FP24_YYY ∗x)

  Set FP24 to unity.
- void FP24_YYY_zero (FP24_YYY ∗x)

  Set FP24 to zero.
- int FP24_YYY_equals (FP24_YYY ∗x, FP24_YYY ∗y)

  Tests for equality of two FP24s.
- void FP24_YYY_conj (FP24_YYY ∗x, FP24_YYY ∗y)

  Conjugation of FP24.
- void FP24_YYY_from_FP8 (FP24_YYY ∗x, FP8_YYY ∗a)

  Initialise FP24 from single FP8.
- void FP24_YYY_from_FP8s (FP24_YYY ∗x, FP8_YYY ∗a, FP8_YYY ∗b, FP8_YYY ∗c)

  Initialise FP24 from three FP8s.
- void FP24_YYY_usqr (FP24_YYY ∗x, FP24_YYY ∗y)

  Fast Squaring of an FP24 in "unitary" form.
- void FP24_YYY_sqr (FP24_YYY ∗x, FP24_YYY ∗y)

  Squaring an FP24.
- void FP24_YYY_smul (FP24_YYY ∗x, FP24_YYY ∗y)

  Fast multiplication of two sparse FP24s that arises from ATE pairing line functions.
- void FP24_YYY_ssmul (FP24_YYY ∗x, FP24_YYY ∗y)

  Fast multiplication of what may be sparse multiplicands.
- void FP24_YYY_mul (FP24_YYY ∗x, FP24_YYY ∗y)

  Full unconditional Multiplication of two FP24s.
- void FP24_YYY_inv (FP24_YYY ∗x, FP24_YYY ∗y)

  Inverting an FP24.
- void FP24_YYY_pow (FP24_YYY ∗r, FP24_YYY ∗x, BIG_XXX b)

  Raises an FP24 to the power of a BIG.
- void FP24_YYY_pinpow (FP24_YYY ∗x, int i, int b)

  Raises an FP24 instance x to a small integer power, side-channel resistant.

- void FP24_YYY_compow (FP8_YYY ∗c, FP24_YYY ∗x, BIG_XXX e, BIG_XXX r)

  Raises an FP24 instance x to a BIG power, compressed to FP8.
- void FP24_YYY_pow8 (FP24_YYY ∗r, FP24_YYY ∗x, BIG_XXX ∗b)

  Calculate Pi x[i]^ b[i] for i=0 to 7, side-channel resistant.
- void FP24_YYY_frob (FP24_YYY ∗x, FP2_YYY ∗f, int n)

  Raises an FP24 to the power of the internal modulus p, using the Frobenius.
- void FP24_YYY_reduce (FP24_YYY ∗x)

  Reduces all components of possibly unreduced FP24 mod Modulus.
- void FP24_YYY_norm (FP24_YYY ∗x)

  Normalises the components of an FP24.
- void FP24_YYY_output (FP24_YYY ∗x)

  Formats and outputs an FP24 to the console.
- void FP24_YYY_toOctet (octet ∗S, FP24_YYY ∗x)

  Formats and outputs an FP24 instance to an octet string.
- void FP24_YYY_fromOctet (FP24_YYY ∗x, octet ∗S)

  Creates an FP24 instance from an octet string.
- void FP24_YYY_trace (FP8_YYY ∗t, FP24_YYY ∗x)

  Calculate the trace of an FP24.
- void FP24_YYY_cmove (FP24_YYY ∗x, FP24_YYY ∗y, int s)

  Conditional copy of FP24_YYY number.

Variables

- const BIG_XXX Fra_YYY
- const BIG_XXX Frb_YYY

5.21.1 Detailed Description

FP24 Header File.

Author

Mike Scott

5.21.2 Function Documentation

5.21.2.1 FP24_YYY_iszilch()

```
int FP24_YYY_iszilch (
            FP24_YYY * x )
```

Tests for FP24 equal to zero.

Parameters

x	FP24 number to be tested

Returns

1 if zero, else returns 0

5.21.2.2 FP24_YYY_isunity()

```
int FP24_YYY_isunity (
```

```
FP24_YYY * x )
```

Tests for FP24 equal to unity.

Parameters

x	FP24 number to be tested

Returns

1 if unity, else returns 0

5.21.2.3 FP24_YYY_copy()

```
void FP24_YYY_copy (
        FP24_YYY * x,
        FP24_YYY * y )
```

Copy FP24 to another FP24.

Parameters

x	FP24 instance, on exit = y
y	FP24 instance to be copied

5.21.2.4 FP24_YYY_one()

```
void FP24_YYY_one (
        FP24_YYY * x )
```

Set FP24 to unity.

Parameters

x	FP24 instance to be set to one

5.21.2.5 FP24_YYY_zero()

```
void FP24_YYY_zero (
        FP24_YYY * x )
```

Set FP24 to zero.

Parameters

x	FP24 instance to be set to zero

5.21.2.6 FP24_YYY_equals()

```
int FP24_YYY_equals (
        FP24_YYY * x,
        FP24_YYY * y )
```

Tests for equality of two FP24s.

Parameters

x	FP24 instance to be compared
y	FP24 instance to be compared

Returns

1 if x=y, else returns 0

5.21.2.7 FP24_YYY_conj()

```
void FP24_YYY_conj (
            FP24_YYY * x,
            FP24_YYY * y )
```

Conjugation of FP24.

If y=(a,b,c) (where a,b,c are its three FP8 components) on exit x=(conj(a),-conj(b),conj(c))

Parameters

x	FP24 instance, on exit = conj(y)
y	FP24 instance

5.21.2.8 FP24_YYY_from_FP8()

```
void FP24_YYY_from_FP8 (
            FP24_YYY * x,
            FP8_YYY * a )
```

Initialise FP24 from single FP8.

Sets first FP8 component of an FP24, other components set to zero

Parameters

x	FP24 instance to be initialised
a	FP8 to form first part of FP8

5.21.2.9 FP24_YYY_from_FP8s()

```
void FP24_YYY_from_FP8s (
            FP24_YYY * x,
            FP8_YYY * a,
            FP8_YYY * b,
            FP8_YYY * c )
```

Initialise FP24 from three FP8s.

Parameters

x	FP24 instance to be initialised
a	FP8 to form first part of FP24
b	FP8 to form second part of FP24
c	FP8 to form third part of FP24

5.21.2.10 FP24_YYY_usqr()

```
void FP24_YYY_usqr (
            FP24_YYY * x,
            FP24_YYY * y )
```

Fast Squaring of an FP24 in "unitary" form.

Parameters

x	FP24 instance, on exit = y^2
y	FP8 instance, must be unitary

5.21.2.11 FP24_YYY_sqr()

```
void FP24_YYY_sqr (
            FP24_YYY * x,
            FP24_YYY * y )
```

Squaring an FP24.

Parameters

x	FP24 instance, on exit = y^2
y	FP24 instance

5.21.2.12 FP24_YYY_smul()

```
void FP24_YYY_smul (
            FP24_YYY * x,
            FP24_YYY * y )
```

Fast multiplication of two sparse FP24s that arises from ATE pairing line functions.

Parameters

x	FP24 instance, on exit = x*y
y	FP24 instance, of special form

5.21.2.13 FP24_YYY_ssmul()

```
void FP24_YYY_ssmul (
            FP24_YYY * x,
            FP24_YYY * y )
```

Fast multiplication of what may be sparse multiplicands.

Parameters

x	FP24 instance, on exit = x*y
y	FP24 instance, of special form

5.21.2.14 FP24_YYY_mul()

```
void FP24_YYY_mul (
            FP24_YYY * x,
            FP24_YYY * y )
```

Full unconditional Multiplication of two FP24s.

Parameters

x	FP24 instance, on exit = x*y
y	FP24 instance, the multiplier

5.21.2.15 FP24_YYY_inv()

```
void FP24_YYY_inv (
            FP24_YYY * x,
            FP24_YYY * y )
```

Inverting an FP24.

Parameters

x	FP24 instance, on exit = 1/y
y	FP24 instance

5.21.2.16 FP24_YYY_pow()

```
void FP24_YYY_pow (
            FP24_YYY * r,
            FP24_YYY * x,
            BIG_XXX b )
```

Raises an FP24 to the power of a BIG.

Parameters

r	FP24 instance, on exit = y^b
x	FP24 instance
b	BIG number

5.21.2.17 FP24_YYY_pinpow()

```
void FP24_YYY_pinpow (
            FP24_YYY * x,
            int i,
            int b )
```

Raises an FP24 instance x to a small integer power, side-channel resistant.

Parameters

x	FP24 instance, on exit = x^i
i	small integer exponent
b	maximum number of bits in exponent

5.21.2.18 FP24_YYY_compow()

```
void FP24_YYY_compow (
            FP8_YYY * c,
            FP24_YYY * x,
            BIG_XXX e,
            BIG_XXX r )
```

Raises an FP24 instance x to a BIG power, compressed to FP8.

Parameters

c	FP8 instance, on exit = x^(e mod r) as FP8
x	FP24 input
e	BIG exponent
r	BIG group order

5.21.2.19 FP24_YYY_pow8()

```
void FP24_YYY_pow8 (
            FP24_YYY * r,
            FP24_YYY * x,
            BIG_XXX * b )
```

Calculate Pi x[i]^b[i] for i=0 to 7, side-channel resistant.

Parameters

r	FP24 instance, on exit = Pi x[i]^b[i] for i=0 to 7
x	FP24 array with 4 FP24s
b	BIG array of 4 exponents

5.21.2.20 FP24_YYY_frob()

```
void FP24_YYY_frob (
            FP24_YYY * x,
            FP2_YYY * f,
            int n )
```

Raises an FP24 to the power of the internal modulus p, using the Frobenius.

Parameters

x	FP24 instance, on exit = x^p^n
f	FP2 precalculated Frobenius constant
n	power of p

5.21.2.21 FP24_YYY_reduce()

```
void FP24_YYY_reduce (
            FP24_YYY * x )
```

Reduces all components of possibly unreduced FP24 mod Modulus.

Parameters

x	FP24 instance, on exit reduced mod Modulus

5.21.2.22 FP24_YYY_norm()

```
void FP24_YYY_norm (
            FP24_YYY * x )
```

Normalises the components of an FP24.

Parameters

x	FP24 instance to be normalised

5.21.2.23 FP24_YYY_output()

```
void FP24_YYY_output (
            FP24_YYY * x )
```

Formats and outputs an FP24 to the console.

Parameters

x	FP24 instance to be printed

5.21.2.24 FP24_YYY_toOctet()

```
void FP24_YYY_toOctet (
            octet * S,
            FP24_YYY * x )
```

Formats and outputs an FP24 instance to an octet string.
Serializes the components of an FP24 to big-endian base 256 form.

Parameters

S	output octet string
x	FP24 instance to be converted to an octet string

5.21.2.25 FP24_YYY_fromOctet()

```
void FP24_YYY_fromOctet (
            FP24_YYY * x,
            octet * S )
```

Creates an FP24 instance from an octet string.
De-serializes the components of an FP24 to create an FP24 from big-endian base 256 components.

Parameters

x	FP24 instance to be created from an octet string

Parameters

S	input octet string

5.21.2.26 FP24_YYY_trace()

```
void FP24_YYY_trace (
            FP8_YYY * t,
            FP24_YYY * x )
```

Calculate the trace of an FP24.

Parameters

t	FP8 trace of x, on exit = tr(x)
x	FP24 instance

5.21.2.27 FP24_YYY_cmove()

```
void FP24_YYY_cmove (
            FP24_YYY * x,
            FP24_YYY * y,
            int s )
```

Conditional copy of FP24_YYY number.
Conditionally copies second parameter to the first (without branching)

Parameters

x	FP24_YYY instance, set to y if s!=0
y	another FP24_YYY instance
s	copy only takes place if not equal to 0

5.21.3 Variable Documentation

5.21.3.1 Fra_YYY

```
const BIG_XXX Fra_YYY [extern]
```

real part of BN curve Frobenius Constant

5.21.3.2 Frb_YYY

```
const BIG_XXX Frb_YYY [extern]
```

imaginary part of BN curve Frobenius Constant

5.22 fp4.h File Reference

FP4 Header File.

```
#include "fp2_YYY.h"
#include "config_curve_ZZZ.h"
```

Classes

- struct FP4_YYY

  FP4 Structure - towered over two FP2.

Functions

- int FP4_YYY_iszilch (FP4_YYY *x)

  Tests for FP4 equal to zero.
- int FP4_YYY_islarger (FP4_YYY *x)

  Tests for lexically larger.
- void FP4_YYY_toBytes (char *b, FP4_YYY *x)

  Serialize out FP4
- void FP4_YYY_fromBytes (FP4_YYY *x, char *b)

  Serialize in FP4
- int FP4_YYY_isunity (FP4_YYY *x)

  Tests for FP4 equal to unity.
- int FP4_YYY_equals (FP4_YYY *x, FP4_YYY *y)

  Tests for equality of two FP4s.
- int FP4_YYY_isreal (FP4_YYY *x)

  Tests for FP4 having only a real part and no imaginary part.
- void FP4_YYY_from_FP2s (FP4_YYY *x, FP2_YYY *a, FP2_YYY *b)

  Initialise FP4 from two FP2s.
- void FP4_YYY_from_FP2 (FP4_YYY *x, FP2_YYY *a)

  Initialise FP4 from single FP2.
- void FP4_YYY_from_FP2H (FP4_YYY *x, FP2_YYY *a)

  Initialise FP4 from single FP2.
- void FP4_YYY_from_FP (FP4_YYY *x, FP_YYY *a)

  Initialise FP4 from single FP.
- void FP4_YYY_copy (FP4_YYY *x, FP4_YYY *y)

  Copy FP4 to another FP4.
- void FP4_YYY_zero (FP4_YYY *x)

  Set FP4 to zero.
- void FP4_YYY_one (FP4_YYY *x)

  Set FP4 to unity.
- int FP4_YYY_sign (FP4_YYY *x)

  Sign of FP4.
- void FP4_YYY_neg (FP4_YYY *x, FP4_YYY *y)

  Negation of FP4.
- void FP4_YYY_conj (FP4_YYY *x, FP4_YYY *y)

  Conjugation of FP4.
- void FP4_YYY_nconj (FP4_YYY *x, FP4_YYY *y)

  Negative conjugation of FP4.
- void FP4_YYY_add (FP4_YYY *x, FP4_YYY *y, FP4_YYY *z)

  addition of two FP4s
- void FP4_YYY_sub (FP4_YYY *x, FP4_YYY *y, FP4_YYY *z)

  subtraction of two FP4s
- void FP4_YYY_pmul (FP4_YYY *x, FP4_YYY *y, FP2_YYY *a)

  Multiplication of an FP4 by an FP2.
- void FP4_YYY_qmul (FP4_YYY *x, FP4_YYY *y, FP_YYY *a)

Multiplication of an FP4 by an FP.

- void FP4_YYY_imul (FP4_YYY ∗x, FP4_YYY ∗y, int i)

    Multiplication of an FP4 by a small integer.
- void FP4_YYY_sqr (FP4_YYY ∗x, FP4_YYY ∗y)

    Squaring an FP4.
- void FP4_YYY_mul (FP4_YYY ∗x, FP4_YYY ∗y, FP4_YYY ∗z)

    Multiplication of two FP4s.
- void FP4_YYY_inv (FP4_YYY ∗x, FP4_YYY ∗y, FP_YYY ∗h)

    Inverting an FP4.
- void FP4_YYY_output (FP4_YYY ∗x)

    Formats and outputs an FP4 to the console.
- void FP4_YYY_rawoutput (FP4_YYY ∗x)

    Formats and outputs an FP4 to the console in raw form (for debugging)
- void FP4_YYY_times_i (FP4_YYY ∗x)

    multiplies an FP4 instance by irreducible polynomial sqrt(1+sqrt(-1))
- void FP4_YYY_norm (FP4_YYY ∗x)

    Normalises the components of an FP4.
- void FP4_YYY_reduce (FP4_YYY ∗x)

    Reduces all components of possibly unreduced FP4 mod Modulus.
- void FP4_YYY_pow (FP4_YYY ∗x, FP4_YYY ∗y, BIG_XXX b)

    Raises an FP4 to the power of a BIG.
- void FP4_YYY_frob (FP4_YYY ∗x, FP2_YYY ∗f)

    Raises an FP4 to the power of the internal modulus p, using the Frobenius.
- void FP4_YYY_xtr_A (FP4_YYY ∗r, FP4_YYY ∗w, FP4_YYY ∗x, FP4_YYY ∗y, FP4_YYY ∗z)

    Calculates the XTR addition function r=w∗x-conj(x)∗y+z.
- void FP4_YYY_xtr_D (FP4_YYY ∗r, FP4_YYY ∗x)

    Calculates the XTR doubling function r=x^2-2∗conj(x)
- void FP4_YYY_xtr_pow (FP4_YYY ∗r, FP4_YYY ∗x, BIG_XXX b)

    Calculates FP4 trace of an FP12 raised to the power of a BIG number.
- void FP4_YYY_xtr_pow2 (FP4_YYY ∗r, FP4_YYY ∗c, FP4_YYY ∗d, FP4_YYY ∗e, FP4_YYY ∗f, BIG_XXX a, BIG_XXX b)

    Calculates FP4 trace of c^a.d^b, where c and d are derived from FP4 traces of FP12s.
- void FP4_YYY_cmove (FP4_YYY ∗x, FP4_YYY ∗y, int s)

    Conditional copy of FP4 number.
- int FP4_YYY_qr (FP4_YYY ∗r, FP_YYY ∗h)

    Test FP4 for QR.
- void FP4_YYY_sqrt (FP4_YYY ∗r, FP4_YYY ∗x, FP_YYY ∗h)

    Calculate square root of an FP4.
- void FP4_YYY_div_i (FP4_YYY ∗x)

    Divide FP4 number by QNR.
- void FP4_YYY_div_2i (FP4_YYY ∗x)

    Divide an FP4 by QNR/2.
- void FP4_YYY_div2 (FP4_YYY ∗x, FP4_YYY ∗y)

    Divide an FP4 by 2.
- void FP4_YYY_rand (FP4_YYY ∗x, csprng ∗rng)

    Generate random FP4.

5.22.1 Detailed Description

FP4 Header File.

Author

Mike Scott

5.22.2 Function Documentation

5.22.2.1 FP4_YYY_iszilch()

```
int FP4_YYY_iszilch (
            FP4_YYY * x )
```

Tests for FP4 equal to zero.

Parameters

x	FP4 number to be tested

Returns

1 if zero, else returns 0

5.22.2.2 FP4_YYY_islarger()

```
int FP4_YYY_islarger (
            FP4_YYY * x )
```

Tests for lexically larger.

Parameters

x	FP4 number to be tested if larger than -x

Returns

1 if larger, else returns 0

5.22.2.3 FP4_YYY_toBytes()

```
void FP4_YYY_toBytes (
            char * b,
            FP4_YYY * x )
```

Serialize out FP4

Parameters

b	buffer for output
x	FP4 number to be serialized

5.22.2.4 FP4_YYY_fromBytes()

```
void FP4_YYY_fromBytes (
            FP4_YYY * x,
            char * b )
```

Serialize in FP4

Parameters

x	FP4 number to be serialized
b	buffer for input

5.22.2.5 FP4_YYY_isunity()

```
int FP4_YYY_isunity (
            FP4_YYY * x )
```

Tests for FP4 equal to unity.

Parameters

x	FP4 number to be tested

Returns

1 if unity, else returns 0

5.22.2.6 FP4_YYY_equals()

```
int FP4_YYY_equals (
            FP4_YYY * x,
            FP4_YYY * y )
```

Tests for equality of two FP4s.

Parameters

x	FP4 instance to be compared
y	FP4 instance to be compared

Returns

1 if x=y, else returns 0

5.22.2.7 FP4_YYY_isreal()

```
int FP4_YYY_isreal (
            FP4_YYY * x )
```

Tests for FP4 having only a real part and no imaginary part.

Parameters

x	FP4 number to be tested

Returns

1 if real, else returns 0

5.22.2.8 FP4_YYY_from_FP2s()

```
void FP4_YYY_from_FP2s (
            FP4_YYY * x,
            FP2_YYY * a,
            FP2_YYY * b )
```

Initialise FP4 from two FP2s.

Parameters

x	FP4 instance to be initialised
a	FP2 to form real part of FP4
b	FP2 to form imaginary part of FP4

5.22.2.9 FP4_YYY_from_FP2()

```
void FP4_YYY_from_FP2 (
            FP4_YYY * x,
            FP2_YYY * a )
```

Initialise FP4 from single FP2.
Imaginary part is set to zero

Parameters

x	FP4 instance to be initialised
a	FP2 to form real part of FP4

5.22.2.10 FP4_YYY_from_FP2H()

```
void FP4_YYY_from_FP2H (
            FP4_YYY * x,
            FP2_YYY * a )
```

Initialise FP4 from single FP2.
real part is set to zero

Parameters

x	FP4 instance to be initialised
a	FP2 to form imaginary part of FP4

5.22.2.11 FP4_YYY_from_FP()

```
void FP4_YYY_from_FP (
            FP4_YYY * x,
            FP_YYY * a )
```

Initialise FP4 from single FP.

Parameters

x	FP4 instance to be initialised
a	FP to form real part of FP4

5.22.2.12 FP4_YYY_copy()

```
void FP4_YYY_copy (
            FP4_YYY * x,
            FP4_YYY * y )
```

Copy FP4 to another FP4.

Parameters

x	FP4 instance, on exit = y
y	FP4 instance to be copied

5.22.2.13 FP4_YYY_zero()

```
void FP4_YYY_zero (
            FP4_YYY * x )
```

Set FP4 to zero.

Parameters

x	FP4 instance to be set to zero

5.22.2.14 FP4_YYY_one()

```
void FP4_YYY_one (
            FP4_YYY * x )
```

Set FP4 to unity.

Parameters

x	FP4 instance to be set to one

5.22.2.15 FP4_YYY_sign()

```
int FP4_YYY_sign (
            FP4_YYY * x )
```

Sign of FP4.

Parameters

x	FP4 instance

Returns

"sign" of FP4

5.22.2.16 FP4_YYY_neg()

```
void FP4_YYY_neg (
            FP4_YYY * x,
            FP4_YYY * y )
```

Negation of FP4.

Parameters

x	FP4 instance, on exit = -y
y	FP4 instance

5.22.2.17 FP4_YYY_conj()

```
void FP4_YYY_conj (
            FP4_YYY * x,
            FP4_YYY * y )
```

Conjugation of FP4.
If y=(a,b) on exit x=(a,-b)

Parameters

x	FP4 instance, on exit = conj(y)
y	FP4 instance

5.22.2.18 FP4_YYY_nconj()

```
void FP4_YYY_nconj (
            FP4_YYY * x,
            FP4_YYY * y )
```

Negative conjugation of FP4.
If y=(a,b) on exit x=(-a,b)

Parameters

x	FP4 instance, on exit = -conj(y)
y	FP4 instance

5.22.2.19 FP4_YYY_add()

```
void FP4_YYY_add (
            FP4_YYY * x,
            FP4_YYY * y,
            FP4_YYY * z )
```

addition of two FP4s

Parameters

x	FP4 instance, on exit = y+z
y	FP4 instance
z	FP4 instance

5.22.2.20 FP4_YYY_sub()

```
void FP4_YYY_sub (
            FP4_YYY * x,
            FP4_YYY * y,
            FP4_YYY * z )
```

subtraction of two FP4s

Parameters

x	FP4 instance, on exit = y-z
y	FP4 instance
z	FP4 instance

5.22.2.21 FP4_YYY_pmul()

```
void FP4_YYY_pmul (
            FP4_YYY * x,
            FP4_YYY * y,
            FP2_YYY * a )
```

Multiplication of an FP4 by an FP2.

Parameters

x	FP4 instance, on exit = y∗a
y	FP4 instance
a	FP2 multiplier

5.22.2.22 FP4_YYY_qmul()

```
void FP4_YYY_qmul (
            FP4_YYY * x,
            FP4_YYY * y,
            FP_YYY * a )
```

Multiplication of an FP4 by an FP.

Parameters

x	FP4 instance, on exit = y∗a
y	FP4 instance
a	FP multiplier

5.22.2.23 FP4_YYY_imul()

```
void FP4_YYY_imul (
            FP4_YYY * x,
            FP4_YYY * y,
            int i )
```

Multiplication of an FP4 by a small integer.

Parameters

x	FP4 instance, on exit = y∗i
y	FP4 instance
i	an integer

5.22.2.24 FP4_YYY_sqr()

```
void FP4_YYY_sqr (
            FP4_YYY * x,
            FP4_YYY * y )
```

Squaring an FP4.

Parameters

x	FP4 instance, on exit = y^2
y	FP4 instance

5.22.2.25 FP4_YYY_mul()

```
void FP4_YYY_mul (
            FP4_YYY * x,
            FP4_YYY * y,
            FP4_YYY * z )
```

Multiplication of two FP4s.

Parameters

x	FP4 instance, on exit = y∗z
y	FP4 instance
z	FP4 instance

5.22.2.26 FP4_YYY_inv()

```
void FP4_YYY_inv (
            FP4_YYY * x,
            FP4_YYY * y,
            FP_YYY * h )
```

Inverting an FP4.

Parameters

x	FP4 instance, on exit = 1/y

Parameters

y	FP4 instance
h	optional input hint

5.22.2.27 FP4_YYY_output()

```
void FP4_YYY_output (
        FP4_YYY * x )
```

Formats and outputs an FP4 to the console.

Parameters

x	FP4 instance to be printed

5.22.2.28 FP4_YYY_rawoutput()

```
void FP4_YYY_rawoutput (
        FP4_YYY * x )
```

Formats and outputs an FP4 to the console in raw form (for debugging)

Parameters

x	FP4 instance to be printed

5.22.2.29 FP4_YYY_times_i()

```
void FP4_YYY_times_i (
        FP4_YYY * x )
```

multiplies an FP4 instance by irreducible polynomial sqrt(1+sqrt(-1))

Parameters

x	FP4 instance, on exit = sqrt(1+sqrt(-1)∗x

5.22.2.30 FP4_YYY_norm()

```
void FP4_YYY_norm (
        FP4_YYY * x )
```

Normalises the components of an FP4.

Parameters

x	FP4 instance to be normalised

5.22.2.31 FP4_YYY_reduce()

```
void FP4_YYY_reduce (
            FP4_YYY * x )
```

Reduces all components of possibly unreduced FP4 mod Modulus.

Parameters

x	FP4 instance, on exit reduced mod Modulus

5.22.2.32 FP4_YYY_pow()

```
void FP4_YYY_pow (
            FP4_YYY * x,
            FP4_YYY * y,
            BIG_XXX b )
```

Raises an FP4 to the power of a BIG.

Parameters

x	FP4 instance, on exit = y^b
y	FP4 instance
b	BIG number

5.22.2.33 FP4_YYY_frob()

```
void FP4_YYY_frob (
            FP4_YYY * x,
            FP2_YYY * f )
```

Raises an FP4 to the power of the internal modulus p, using the Frobenius.

Parameters

x	FP4 instance, on exit = x^p
f	FP2 precalculated Frobenius constant

5.22.2.34 FP4_YYY_xtr_A()

```
void FP4_YYY_xtr_A (
            FP4_YYY * r,
            FP4_YYY * w,
            FP4_YYY * x,
            FP4_YYY * y,
            FP4_YYY * z )
```

Calculates the XTR addition function r=w∗x-conj(x)∗y+z.

Parameters

r	FP4 instance, on exit = w∗x-conj(x)∗y+z
w	FP4 instance
x	FP4 instance
y	FP4 instance

Parameters

z	FP4 instance

5.22.2.35 FP4_YYY_xtr_D()

```
void FP4_YYY_xtr_D (
            FP4_YYY * r,
            FP4_YYY * x )
```

Calculates the XTR doubling function r=x^2-2*conj(x)

Parameters

r	FP4 instance, on exit = x^2-2*conj(x)
x	FP4 instance

5.22.2.36 FP4_YYY_xtr_pow()

```
void FP4_YYY_xtr_pow (
            FP4_YYY * r,
            FP4_YYY * x,
            BIG_XXX b )
```

Calculates FP4 trace of an FP12 raised to the power of a BIG number.
XTR single exponentiation

Parameters

r	FP4 instance, on exit = trace(w^b)
x	FP4 instance, trace of an FP12 w
b	BIG number

5.22.2.37 FP4_YYY_xtr_pow2()

```
void FP4_YYY_xtr_pow2 (
            FP4_YYY * r,
            FP4_YYY * c,
            FP4_YYY * d,
            FP4_YYY * e,
            FP4_YYY * f,
            BIG_XXX a,
            BIG_XXX b )
```

Calculates FP4 trace of c^a.d^b, where c and d are derived from FP4 traces of FP12s.
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

Parameters

a	BIG number
b	BIG number

5.22.2.38 FP4_YYY_cmove()

```
void FP4_YYY_cmove (
            FP4_YYY * x,
            FP4_YYY * y,
            int s )
```

Conditional copy of FP4 number.

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

5.22.2.39 FP4_YYY_qr()

```
int FP4_YYY_qr (
            FP4_YYY * r,
            FP_YYY * h )
```

Test FP4 for QR.

Parameters

r	FP4 instance
h	optional generated hint

Returns

1 x is a QR, otherwise 0

5.22.2.40 FP4_YYY_sqrt()

```
void FP4_YYY_sqrt (
            FP4_YYY * r,
            FP4_YYY * x,
            FP_YYY * h )
```

Calculate square root of an FP4.

Square root

Parameters

r	FP4 instance, on exit = sqrt(x)
x	FP4 instance
h	optional input hint

5.22.2.41 FP4_YYY_div_i()

```
void FP4_YYY_div_i (
        FP4_YYY * x )
```

Divide FP4 number by QNR.

Divide FP4 by the QNR

Parameters

x	FP4 instance

5.22.2.42 FP4_YYY_div_2i()

```
void FP4_YYY_div_2i (
        FP4_YYY * x )
```

Divide an FP4 by QNR/2.

Divide FP4 by the QNR/2

Parameters

x	FP4 instance

5.22.2.43 FP4_YYY_div2()

```
void FP4_YYY_div2 (
        FP4_YYY * x,
        FP4_YYY * y )
```

Divide an FP4 by 2.

Parameters

x	FP4 instance, on exit = y/2
y	FP4 instance

5.22.2.44 FP4_YYY_rand()

```
void FP4_YYY_rand (
        FP4_YYY * x,
        csprng * rng )
```

Generate random FP4.

Parameters

x	random FP4 number
rng	random number generator

5.23 fp48.h File Reference

FP48 Header File.

```
#include "fp16_YYY.h"
```

Classes

- struct FP48_YYY

  FP12 Structure - towered over three FP16.

Functions

- int FP48_YYY_iszilch (FP48_YYY ∗x)

  Tests for FP48 equal to zero.
- int FP48_YYY_isunity (FP48_YYY ∗x)

  Tests for FP48 equal to unity.
- void FP48_YYY_copy (FP48_YYY ∗x, FP48_YYY ∗y)

  Copy FP48 to another FP48.
- void FP48_YYY_one (FP48_YYY ∗x)

  Set FP48 to unity.
- void FP48_YYY_zero (FP48_YYY ∗x)

  Set FP48 to zero.
- int FP48_YYY_equals (FP48_YYY ∗x, FP48_YYY ∗y)

  Tests for equality of two FP48s.
- void FP48_YYY_conj (FP48_YYY ∗x, FP48_YYY ∗y)

  Conjugation of FP48.
- void FP48_YYY_from_FP16 (FP48_YYY ∗x, FP16_YYY ∗a)

  Initialise FP48 from single FP16.
- void FP48_YYY_from_FP16s (FP48_YYY ∗x, FP16_YYY ∗a, FP16_YYY ∗b, FP16_YYY ∗c)

  Initialise FP48 from three FP16s.
- void FP48_YYY_usqr (FP48_YYY ∗x, FP48_YYY ∗y)

  Fast Squaring of an FP48 in "unitary" form.
- void FP48_YYY_sqr (FP48_YYY ∗x, FP48_YYY ∗y)

  Squaring an FP48.
- void FP48_YYY_smul (FP48_YYY ∗x, FP48_YYY ∗y)

  Fast multiplication of two sparse FP24s that arises from ATE pairing line functions.
- void FP48_YYY_ssmul (FP48_YYY ∗x, FP48_YYY ∗y)

  Fast multiplication of what may be sparse multiplicands.
- void FP48_YYY_mul (FP48_YYY ∗x, FP48_YYY ∗y)

  Full unconditional Multiplication of two FP24s.
- void FP48_YYY_inv (FP48_YYY ∗x, FP48_YYY ∗y)

  Inverting an FP48.
- void FP48_YYY_pow (FP48_YYY ∗r, FP48_YYY ∗x, BIG_XXX b)

  Raises an FP48 to the power of a BIG.
- void FP48_YYY_pinpow (FP48_YYY ∗x, int i, int b)

  Raises an FP48 instance x to a small integer power, side-channel resistant.
- void FP48_YYY_compow (FP16_YYY ∗c, FP48_YYY ∗x, BIG_XXX e, BIG_XXX r)

  Raises an FP48 instance x to a BIG_XXX power, compressed to FP16.
- void FP48_YYY_pow16 (FP48_YYY ∗r, FP48_YYY ∗x, BIG_XXX ∗b)

  Calculate Pi x[i]^b[i] for i=0 to 15, side-channel resistant.
- void FP48_YYY_frob (FP48_YYY ∗x, FP2_YYY ∗f, int n)

Raises an FP48 to the power of the internal modulus p, using the Frobenius.
- void FP48_YYY_reduce (FP48_YYY ∗x)

  Reduces all components of possibly unreduced FP48 mod Modulus.
- void FP48_YYY_norm (FP48_YYY ∗x)

  Normalises the components of an FP48.
- void FP48_YYY_output (FP48_YYY ∗x)

  Formats and outputs an FP48 to the console.
- void FP48_YYY_toOctet (octet ∗S, FP48_YYY ∗x)

  Formats and outputs an FP48 instance to an octet string.
- void FP48_YYY_fromOctet (FP48_YYY ∗x, octet ∗S)

  Creates an FP48 instance from an octet string.
- void FP48_YYY_trace (FP16_YYY ∗t, FP48_YYY ∗x)

  Calculate the trace of an FP48.
- void FP48_YYY_cmove (FP48_YYY ∗x, FP48_YYY ∗y, int s)

  Conditional copy of FP48 number.

Variables

- const BIG_XXX Fra_YYY
- const BIG_XXX Frb_YYY

5.23.1 Detailed Description

FP48 Header File.

Author

Mike Scott

5.23.2 Function Documentation

5.23.2.1 FP48_YYY_iszilch()

```
int FP48_YYY_iszilch (
            FP48_YYY * x )
```

Tests for FP48 equal to zero.

Parameters

x	FP48 number to be tested

Returns

1 if zero, else returns 0

5.23.2.2 FP48_YYY_isunity()

```
int FP48_YYY_isunity (
            FP48_YYY * x )
```

Tests for FP48 equal to unity.

Parameters

x	FP48 number to be tested

Returns

1 if unity, else returns 0

5.23.2.3 FP48_YYY_copy()

```
void FP48_YYY_copy (
        FP48_YYY * x,
        FP48_YYY * y )
```

Copy FP48 to another FP48.

Parameters

x	FP48 instance, on exit = y
y	FP48 instance to be copied

5.23.2.4 FP48_YYY_one()

```
void FP48_YYY_one (
        FP48_YYY * x )
```

Set FP48 to unity.

Parameters

x	FP48 instance to be set to one

5.23.2.5 FP48_YYY_zero()

```
void FP48_YYY_zero (
        FP48_YYY * x )
```

Set FP48 to zero.

Parameters

x	FP48 instance to be set to zero

5.23.2.6 FP48_YYY_equals()

```
int FP48_YYY_equals (
        FP48_YYY * x,
        FP48_YYY * y )
```

Tests for equality of two FP48s.

Parameters

x	FP48 instance to be compared
y	FP48 instance to be compared

Returns

1 if x=y, else returns 0

5.23.2.7 FP48_YYY_conj()

```
void FP48_YYY_conj (
            FP48_YYY * x,
            FP48_YYY * y )
```

Conjugation of FP48.

If y=(a,b,c) (where a,b,c are its three FP16 components) on exit x=(conj(a),-conj(b),conj(c))

Parameters

x	FP48 instance, on exit = conj(y)
y	FP48 instance

5.23.2.8 FP48_YYY_from_FP16()

```
void FP48_YYY_from_FP16 (
            FP48_YYY * x,
            FP16_YYY * a )
```

Initialise FP48 from single FP16.

Sets first FP16 component of an FP48, other components set to zero

Parameters

x	FP48 instance to be initialised
a	FP16 to form first part of FP48

5.23.2.9 FP48_YYY_from_FP16s()

```
void FP48_YYY_from_FP16s (
            FP48_YYY * x,
            FP16_YYY * a,
            FP16_YYY * b,
            FP16_YYY * c )
```

Initialise FP48 from three FP16s.

Parameters

x	FP48 instance to be initialised
a	FP16 to form first part of FP48
b	FP16 to form second part of FP48
c	FP16 to form third part of FP48

5.23.2.10 FP48_YYY_usqr()

```
void FP48_YYY_usqr (
            FP48_YYY * x,
```

```
              FP48_YYY * y )
```
Fast Squaring of an FP48 in "unitary" form.

Parameters

x	FP48 instance, on exit = y^2
y	FP16 instance, must be unitary

5.23.2.11 FP48_YYY_sqr()

```
void FP48_YYY_sqr (
              FP48_YYY * x,
              FP48_YYY * y )
```
Squaring an FP48.

Parameters

x	FP48 instance, on exit = y^2
y	FP48 instance

5.23.2.12 FP48_YYY_smul()

```
void FP48_YYY_smul (
              FP48_YYY * x,
              FP48_YYY * y )
```
Fast multiplication of two sparse FP24s that arises from ATE pairing line functions.

Parameters

x	FP48 instance, on exit = x*y
y	FP48 instance, of special form

5.23.2.13 FP48_YYY_ssmul()

```
void FP48_YYY_ssmul (
              FP48_YYY * x,
              FP48_YYY * y )
```
Fast multiplication of what may be sparse multiplicands.

Parameters

x	FP48 instance, on exit = x*y
y	FP48 instance, of special form

5.23.2.14 FP48_YYY_mul()

```
void FP48_YYY_mul (
              FP48_YYY * x,
              FP48_YYY * y )
```

Full unconditional Multiplication of two FP24s.

Parameters

x	FP48 instance, on exit = x∗y
y	FP48 instance, the multiplier

5.23.2.15 FP48_YYY_inv()

```
void FP48_YYY_inv (
            FP48_YYY * x,
            FP48_YYY * y )
```

Inverting an FP48.

Parameters

x	FP48 instance, on exit = 1/y
y	FP48 instance

5.23.2.16 FP48_YYY_pow()

```
void FP48_YYY_pow (
            FP48_YYY * r,
            FP48_YYY * x,
            BIG_XXX b )
```

Raises an FP48 to the power of a BIG.

Parameters

r	FP48 instance, on exit = y^b
x	FP48 instance
b	BIG number

5.23.2.17 FP48_YYY_pinpow()

```
void FP48_YYY_pinpow (
            FP48_YYY * x,
            int i,
            int b )
```

Raises an FP48 instance x to a small integer power, side-channel resistant.

Parameters

x	FP48 instance, on exit = x^i
i	small integer exponent
b	maximum number of bits in exponent

5.23.2.18 FP48_YYY_compow()

```
void FP48_YYY_compow (
            FP16_YYY * c,
            FP48_YYY * x,
            BIG_XXX e,
            BIG_XXX r )
```

Raises an FP48 instance x to a BIG_XXX power, compressed to FP16.

Parameters

c	FP16 instance, on exit = x^(e mod r) as FP16
x	FP48 input
e	BIG exponent
r	BIG group order

5.23.2.19 FP48_YYY_pow16()

```
void FP48_YYY_pow16 (
            FP48_YYY * r,
            FP48_YYY * x,
            BIG_XXX * b )
```

Calculate Pi x[i]^b[i] for i=0 to 15, side-channel resistant.

Parameters

r	FP48 instance, on exit = Pi x[i]^b[i] for i=0 to 15
x	FP48 array with 16 FP48s
b	BIG array of 16 exponents

5.23.2.20 FP48_YYY_frob()

```
void FP48_YYY_frob (
            FP48_YYY * x,
            FP2_YYY * f,
            int n )
```

Raises an FP48 to the power of the internal modulus p, using the Frobenius.

Parameters

x	FP48 instance, on exit = x^p^n
f	FP2 precalculated Frobenius constant
n	power of p

5.23.2.21 FP48_YYY_reduce()

```
void FP48_YYY_reduce (
            FP48_YYY * x )
```

Reduces all components of possibly unreduced FP48 mod Modulus.

Parameters

x	FP48 instance, on exit reduced mod Modulus

5.23.2.22 FP48_YYY_norm()

```
void FP48_YYY_norm (
            FP48_YYY * x )
```

Normalises the components of an FP48.

Parameters

x	FP48 instance to be normalised

5.23.2.23 FP48_YYY_output()

```
void FP48_YYY_output (
            FP48_YYY * x )
```

Formats and outputs an FP48 to the console.

Parameters

x	FP48 instance to be printed

5.23.2.24 FP48_YYY_toOctet()

```
void FP48_YYY_toOctet (
            octet * S,
            FP48_YYY * x )
```

Formats and outputs an FP48 instance to an octet string.
Serializes the components of an FP48 to big-endian base 256 form.

Parameters

S	output octet string
x	FP48 instance to be converted to an octet string

5.23.2.25 FP48_YYY_fromOctet()

```
void FP48_YYY_fromOctet (
            FP48_YYY * x,
            octet * S )
```

Creates an FP48 instance from an octet string.
De-serializes the components of an FP48 to create an FP48 from big-endian base 256 components.

Parameters

x	FP48 instance to be created from an octet string
S	input octet string

5.23.2.26 FP48_YYY_trace()

```
void FP48_YYY_trace (
            FP16_YYY * t,
            FP48_YYY * x )
```

Calculate the trace of an FP48.

Parameters

t	FP16 trace of x, on exit = tr(x)
x	FP48 instance

5.23.2.27 FP48_YYY_cmove()

```
void FP48_YYY_cmove (
            FP48_YYY * x,
            FP48_YYY * y,
            int s )
```

Conditional copy of FP48 number.

Conditionally copies second parameter to the first (without branching)

Parameters

x	FP48 instance, set to y if s!=0
y	another FP48 instance
s	copy only takes place if not equal to 0

5.23.3 Variable Documentation

5.23.3.1 Fra_YYY

```
const BIG_XXX Fra_YYY [extern]
```

real part of BN curve Frobenius Constant

5.23.3.2 Frb_YYY

```
const BIG_XXX Frb_YYY [extern]
```

imaginary part of BN curve Frobenius Constant

5.24 fp8.h File Reference

FP8 Header File.

```
#include "fp4_YYY.h"
#include "config_curve_ZZZ.h"
```

Classes

- struct FP8_YYY

    FP8 Structure - towered over two FP4.

Functions

- int FP8_YYY_iszilch (FP8_YYY *x)

  Tests for FP8 equal to zero.
- int FP8_YYY_islarger (FP8_YYY *x)

  Tests for lexically larger.
- void FP8_YYY_toBytes (char *b, FP8_YYY *x)

  Serialize in FP8
- void FP8_YYY_fromBytes (FP8_YYY *x, char *b)

  Serialize out FP8
- int FP8_YYY_isunity (FP8_YYY *x)

  Tests for FP8 equal to unity.
- int FP8_YYY_equals (FP8_YYY *x, FP8_YYY *y)

  Tests for equality of two FP8s.
- int FP8_YYY_isreal (FP8_YYY *x)

  Tests for FP8 having only a real part and no imaginary part.
- void FP8_YYY_from_FP4s (FP8_YYY *x, FP4_YYY *a, FP4_YYY *b)

  Initialise FP8 from two FP4s.
- void FP8_YYY_from_FP4 (FP8_YYY *x, FP4_YYY *a)

  Initialise FP8 from single FP4.
- void FP8_YYY_from_FP4H (FP8_YYY *x, FP4_YYY *a)

  Initialise FP8 from single FP4.
- void FP8_YYY_from_FP (FP8_YYY *x, FP_YYY *a)

  Initialise FP8 from single FP.
- void FP8_YYY_copy (FP8_YYY *x, FP8_YYY *y)

  Copy FP8 to another FP8.
- void FP8_YYY_zero (FP8_YYY *x)

  Set FP8 to zero.
- void FP8_YYY_one (FP8_YYY *x)

  Set FP8 to unity.
- int FP8_YYY_sign (FP8_YYY *x)

  Sign of FP8.
- void FP8_YYY_neg (FP8_YYY *x, FP8_YYY *y)

  Negation of FP8.
- void FP8_YYY_conj (FP8_YYY *x, FP8_YYY *y)

  Conjugation of FP8.
- void FP8_YYY_nconj (FP8_YYY *x, FP8_YYY *y)

  Negative conjugation of FP8.
- void FP8_YYY_add (FP8_YYY *x, FP8_YYY *y, FP8_YYY *z)

  addition of two FP8s
- void FP8_YYY_sub (FP8_YYY *x, FP8_YYY *y, FP8_YYY *z)

  subtraction of two FP8s
- void FP8_YYY_pmul (FP8_YYY *x, FP8_YYY *y, FP4_YYY *a)

  Multiplication of an FP8 by an FP4.
- void FP8_YYY_qmul (FP8_YYY *x, FP8_YYY *y, FP2_YYY *a)

  Multiplication of an FP8 by an FP2.
- void FP8_YYY_tmul (FP8_YYY *x, FP8_YYY *y, FP_YYY *a)

  Multiplication of an FP8 by an FP.
- void FP8_YYY_imul (FP8_YYY *x, FP8_YYY *y, int i)

  Multiplication of an FP8 by a small integer.

- void FP8_YYY_sqr (FP8_YYY *x, FP8_YYY *y)

  Squaring an FP8.
- void FP8_YYY_mul (FP8_YYY *x, FP8_YYY *y, FP8_YYY *z)

  Multiplication of two FP8s.
- void FP8_YYY_inv (FP8_YYY *x, FP8_YYY *y, FP_YYY *h)

  Inverting an FP8.
- void FP8_YYY_output (FP8_YYY *x)

  Formats and outputs an FP8 to the console.
- void FP8_YYY_div2 (FP8_YYY *x, FP8_YYY *y)

  Divide an FP8 by 2.
- void FP8_YYY_rawoutput (FP8_YYY *x)

  Formats and outputs an FP8 to the console in raw form (for debugging)
- void FP8_YYY_times_i (FP8_YYY *x)

  multiplies an FP8 instance by irreducible polynomial sqrt(1+sqrt(-1))
- void FP8_YYY_times_i2 (FP8_YYY *x)

  multiplies an FP8 instance by irreducible polynomial (1+sqrt(-1))
- void FP8_YYY_norm (FP8_YYY *x)

  Normalises the components of an FP8.
- void FP8_YYY_reduce (FP8_YYY *x)

  Reduces all components of possibly unreduced FP8 mod Modulus.
- void FP8_YYY_pow (FP8_YYY *x, FP8_YYY *y, BIG_XXX b)

  Raises an FP8 to the power of a BIG.
- void FP8_YYY_frob (FP8_YYY *x, FP2_YYY *f)

  Raises an FP8 to the power of the internal modulus p, using the Frobenius.
- void FP8_YYY_xtr_A (FP8_YYY *r, FP8_YYY *w, FP8_YYY *x, FP8_YYY *y, FP8_YYY *z)

  *Calculates the XTR addition function r=w*x-conj(x)*y+z.*
- void FP8_YYY_xtr_D (FP8_YYY *r, FP8_YYY *x)

  *Calculates the XTR doubling function r=x^2-2*conj(x)*
- void FP8_YYY_xtr_pow (FP8_YYY *r, FP8_YYY *x, BIG_XXX b)

  Calculates FP8 trace of an FP12 raised to the power of a BIG number.
- void FP8_YYY_xtr_pow2 (FP8_YYY *r, FP8_YYY *c, FP8_YYY *d, FP8_YYY *e, FP8_YYY *f, BIG_XXX a, BIG_XXX b)

  Calculates FP8 trace of c^a.d^b, where c and d are derived from FP8 traces of FP12s.
- int FP8_YYY_qr (FP8_YYY *r, FP_YYY *h)

  Test FP8 for QR.
- void FP8_YYY_sqrt (FP8_YYY *r, FP8_YYY *x, FP_YYY *h)

  Calculate square root of an FP8.
- void FP8_YYY_cmove (FP8_YYY *x, FP8_YYY *y, int s)

  Conditional copy of FP8 number.
- void FP8_YYY_div_i (FP8_YYY *x)

  Divide FP8 number by QNR.
- void FP8_YYY_div_i2 (FP8_YYY *x)

  Divide FP8 number by QNR twice.
- void FP8_YYY_div_2i (FP8_YYY *x)

  Divide FP8 number by QNR/2.
- void FP8_YYY_rand (FP8_YYY *x, csprng *rng)

  Generate random FP8.

5.24.1 Detailed Description

FP8 Header File.

Author

Mike Scott

5.24.2 Function Documentation

5.24.2.1 FP8_YYY_iszilch()

```
int FP8_YYY_iszilch (
        FP8_YYY * x )
```

Tests for FP8 equal to zero.

Parameters

x	FP8 number to be tested

Returns

1 if zero, else returns 0

5.24.2.2 FP8_YYY_islarger()

```
int FP8_YYY_islarger (
        FP8_YYY * x )
```

Tests for lexically larger.

Parameters

x	FP8 number to be tested if larger than -x

Returns

1 if larger, else returns 0

5.24.2.3 FP8_YYY_toBytes()

```
void FP8_YYY_toBytes (
        char * b,
        FP8_YYY * x )
```

Serialize in FP8

Parameters

b	buffer for output
x	FP8 number to be serialized

5.24.2.4 FP8_YYY_fromBytes()

```
void FP8_YYY_fromBytes (
            FP8_YYY * x,
            char * b )
```

Serialize out FP8

Parameters

x	FP8 number to be serialized
b	buffer for input

5.24.2.5 FP8_YYY_isunity()

```
int FP8_YYY_isunity (
            FP8_YYY * x )
```

Tests for FP8 equal to unity.

Parameters

x	FP8 number to be tested

Returns

1 if unity, else returns 0

5.24.2.6 FP8_YYY_equals()

```
int FP8_YYY_equals (
            FP8_YYY * x,
            FP8_YYY * y )
```

Tests for equality of two FP8s.

Parameters

x	FP8 instance to be compared
y	FP8 instance to be compared

Returns

1 if x=y, else returns 0

5.24.2.7 FP8_YYY_isreal()

```
int FP8_YYY_isreal (
            FP8_YYY * x )
```

Tests for FP8 having only a real part and no imaginary part.

Parameters

x	FP8 number to be tested

Returns

1 if real, else returns 0

5.24.2.8 FP8_YYY_from_FP4s()

```
void FP8_YYY_from_FP4s (
            FP8_YYY * x,
            FP4_YYY * a,
            FP4_YYY * b )
```

Initialise FP8 from two FP4s.

Parameters

x	FP8 instance to be initialised
a	FP4 to form real part of FP8
b	FP4 to form imaginary part of FP8

5.24.2.9 FP8_YYY_from_FP4()

```
void FP8_YYY_from_FP4 (
            FP8_YYY * x,
            FP4_YYY * a )
```

Initialise FP8 from single FP4.
Imaginary part is set to zero

Parameters

x	FP8 instance to be initialised
a	FP4 to form real part of FP8

5.24.2.10 FP8_YYY_from_FP4H()

```
void FP8_YYY_from_FP4H (
            FP8_YYY * x,
            FP4_YYY * a )
```

Initialise FP8 from single FP4.
real part is set to zero

Parameters

x	FP8 instance to be initialised
a	FP4 to form imaginary part of FP8

5.24.2.11 FP8_YYY_from_FP()

```
void FP8_YYY_from_FP (
            FP8_YYY * x,
            FP_YYY * a )
```

Initialise FP8 from single FP.

Parameters

x	FP8 instance to be initialised
a	FP to form real part of FP8

5.24.2.12 FP8_YYY_copy()

```
void FP8_YYY_copy (
            FP8_YYY * x,
            FP8_YYY * y )
```

Copy FP8 to another FP8.

Parameters

x	FP8 instance, on exit = y
y	FP8 instance to be copied

5.24.2.13 FP8_YYY_zero()

```
void FP8_YYY_zero (
            FP8_YYY * x )
```

Set FP8 to zero.

Parameters

x	FP8 instance to be set to zero

5.24.2.14 FP8_YYY_one()

```
void FP8_YYY_one (
            FP8_YYY * x )
```

Set FP8 to unity.

Parameters

x	FP8 instance to be set to one

5.24.2.15 FP8_YYY_sign()

```
int FP8_YYY_sign (
            FP8_YYY * x )
```

Sign of FP8.

Parameters

x	FP8 instance

Returns

"sign" of FP8

5.24.2.16 FP8_YYY_neg()

```
void FP8_YYY_neg (
            FP8_YYY * x,
            FP8_YYY * y )
```

Negation of FP8.

Parameters

x	FP8 instance, on exit = -y
y	FP8 instance

5.24.2.17 FP8_YYY_conj()

```
void FP8_YYY_conj (
            FP8_YYY * x,
            FP8_YYY * y )
```

Conjugation of FP8.
If y=(a,b) on exit x=(a,-b)

Parameters

x	FP8 instance, on exit = conj(y)
y	FP8 instance

5.24.2.18 FP8_YYY_nconj()

```
void FP8_YYY_nconj (
            FP8_YYY * x,
            FP8_YYY * y )
```

Negative conjugation of FP8.
If y=(a,b) on exit x=(-a,b)

Parameters

x	FP8 instance, on exit = -conj(y)
y	FP8 instance

5.24.2.19 FP8_YYY_add()

```
void FP8_YYY_add (
            FP8_YYY * x,
            FP8_YYY * y,
            FP8_YYY * z )
```

addition of two FP8s

Parameters

x	FP8 instance, on exit = y+z
y	FP8 instance
z	FP8 instance

5.24.2.20 FP8_YYY_sub()

```
void FP8_YYY_sub (
            FP8_YYY * x,
            FP8_YYY * y,
            FP8_YYY * z )
```

subtraction of two FP8s

Parameters

x	FP8 instance, on exit = y-z
y	FP8 instance
z	FP8 instance

5.24.2.21 FP8_YYY_pmul()

```
void FP8_YYY_pmul (
            FP8_YYY * x,
            FP8_YYY * y,
            FP4_YYY * a )
```

Multiplication of an FP8 by an FP4.

Parameters

x	FP8 instance, on exit = y∗a
y	FP8 instance
a	FP4 multiplier

5.24.2.22 FP8_YYY_qmul()

```
void FP8_YYY_qmul (
            FP8_YYY * x,
            FP8_YYY * y,
            FP2_YYY * a )
```

Multiplication of an FP8 by an FP2.

Parameters

x	FP8 instance, on exit = y∗a
y	FP8 instance
a	FP2 multiplier

5.24.2.23 FP8_YYY_tmul()

```
void FP8_YYY_tmul (
            FP8_YYY * x,
            FP8_YYY * y,
            FP_YYY * a )
```

Multiplication of an FP8 by an FP.

Parameters

x	FP8 instance, on exit = y*a
y	FP8 instance
a	FP multiplier

5.24.2.24 FP8_YYY_imul()

```
void FP8_YYY_imul (
            FP8_YYY * x,
            FP8_YYY * y,
            int i )
```

Multiplication of an FP8 by a small integer.

Parameters

x	FP8 instance, on exit = y*i
y	FP8 instance
i	an integer

5.24.2.25 FP8_YYY_sqr()

```
void FP8_YYY_sqr (
            FP8_YYY * x,
            FP8_YYY * y )
```

Squaring an FP8.

Parameters

x	FP8 instance, on exit = y^2
y	FP8 instance

5.24.2.26 FP8_YYY_mul()

```
void FP8_YYY_mul (
            FP8_YYY * x,
            FP8_YYY * y,
            FP8_YYY * z )
```

Multiplication of two FP8s.

Parameters

x	FP8 instance, on exit = y*z

Parameters

y	FP8 instance
z	FP8 instance

5.24.2.27 FP8_YYY_inv()

```
void FP8_YYY_inv (
            FP8_YYY * x,
            FP8_YYY * y,
            FP_YYY * h )
```

Inverting an FP8.

Parameters

x	FP8 instance, on exit = 1/y
y	FP8 instance
h	optional input hint

5.24.2.28 FP8_YYY_output()

```
void FP8_YYY_output (
            FP8_YYY * x )
```

Formats and outputs an FP8 to the console.

Parameters

x	FP8 instance to be printed

5.24.2.29 FP8_YYY_div2()

```
void FP8_YYY_div2 (
            FP8_YYY * x,
            FP8_YYY * y )
```

Divide an FP8 by 2.

Parameters

x	FP8 instance, on exit = y/2
y	FP8 instance

5.24.2.30 FP8_YYY_rawoutput()

```
void FP8_YYY_rawoutput (
            FP8_YYY * x )
```

Formats and outputs an FP8 to the console in raw form (for debugging)

Parameters

x	FP8 instance to be printed

5.24.2.31 FP8_YYY_times_i()

```
void FP8_YYY_times_i (
            FP8_YYY * x )
```

multiplies an FP8 instance by irreducible polynomial sqrt(1+sqrt(-1))

Parameters

x	FP8 instance, on exit = sqrt(1+sqrt(-1)∗x

5.24.2.32 FP8_YYY_times_i2()

```
void FP8_YYY_times_i2 (
            FP8_YYY * x )
```

multiplies an FP8 instance by irreducible polynomial (1+sqrt(-1))

Parameters

x	FP8 instance, on exit = (1+sqrt(-1)∗x

5.24.2.33 FP8_YYY_norm()

```
void FP8_YYY_norm (
            FP8_YYY * x )
```

Normalises the components of an FP8.

Parameters

x	FP8 instance to be normalised

5.24.2.34 FP8_YYY_reduce()

```
void FP8_YYY_reduce (
            FP8_YYY * x )
```

Reduces all components of possibly unreduced FP8 mod Modulus.

Parameters

x	FP8 instance, on exit reduced mod Modulus

5.24.2.35 FP8_YYY_pow()

```
void FP8_YYY_pow (
            FP8_YYY * x,
            FP8_YYY * y,
            BIG_XXX b )
```

Raises an FP8 to the power of a BIG.

Parameters

x	FP8 instance, on exit = y^b
y	FP8 instance
b	BIG number

5.24.2.36 FP8_YYY_frob()

```
void FP8_YYY_frob (
            FP8_YYY * x,
            FP2_YYY * f )
```

Raises an FP8 to the power of the internal modulus p, using the Frobenius.

Parameters

x	FP8 instance, on exit = x^p
f	FP2 precalculated Frobenius constant

5.24.2.37 FP8_YYY_xtr_A()

```
void FP8_YYY_xtr_A (
            FP8_YYY * r,
            FP8_YYY * w,
            FP8_YYY * x,
            FP8_YYY * y,
            FP8_YYY * z )
```

Calculates the XTR addition function r=w∗x-conj(x)∗y+z.

Parameters

r	FP8 instance, on exit = w∗x-conj(x)∗y+z
w	FP8 instance
x	FP8 instance
y	FP8 instance
z	FP8 instance

5.24.2.38 FP8_YYY_xtr_D()

```
void FP8_YYY_xtr_D (
            FP8_YYY * r,
            FP8_YYY * x )
```

Calculates the XTR doubling function r=x^2-2∗conj(x)

Parameters

r	FP8 instance, on exit = x^2-2∗conj(x)
x	FP8 instance

5.24.2.39 FP8_YYY_xtr_pow()

```
void FP8_YYY_xtr_pow (
            FP8_YYY * r,
            FP8_YYY * x,
            BIG_XXX b )
```

Calculates FP8 trace of an FP12 raised to the power of a BIG number.
XTR single exponentiation

Parameters

r	FP8 instance, on exit = trace(w^b)
x	FP8 instance, trace of an FP12 w
b	BIG number

5.24.2.40 FP8_YYY_xtr_pow2()

```
void FP8_YYY_xtr_pow2 (
            FP8_YYY * r,
            FP8_YYY * c,
            FP8_YYY * d,
            FP8_YYY * e,
            FP8_YYY * f,
            BIG_XXX a,
            BIG_XXX b )
```

Calculates FP8 trace of c^a.d^b, where c and d are derived from FP8 traces of FP12s.
XTR double exponentiation Assumes c=tr(x^m), d=tr(x^n), e=tr(x^(m-n)), f=tr(x^(m-2n))

Parameters

r	FP8 instance, on exit = trace(c^a.d^b)
c	FP8 instance, trace of an FP12
d	FP8 instance, trace of an FP12
e	FP8 instance, trace of an FP12
f	FP8 instance, trace of an FP12
a	BIG number
b	BIG number

5.24.2.41 FP8_YYY_qr()

```
int FP8_YYY_qr (
            FP8_YYY * r,
            FP_YYY * h )
```

Test FP8 for QR.

Parameters

r	FP8 instance
h	optional generated hint

Returns

1 r is a QR, otherwise 0

5.24.2.42 FP8_YYY_sqrt()

```
void FP8_YYY_sqrt (
            FP8_YYY * r,
            FP8_YYY * x,
            FP_YYY * h )
```

Calculate square root of an FP8.
Square root

Parameters

r	FP8 instance, on exit = sqrt(x)
x	FP8 instance
h	optional input hint

5.24.2.43 FP8_YYY_cmove()

```
void FP8_YYY_cmove (
            FP8_YYY * x,
            FP8_YYY * y,
            int s )
```

Conditional copy of FP8 number.
Conditionally copies second parameter to the first (without branching)

Parameters

x	FP8 instance, set to y if s!=0
y	another FP8 instance
s	copy only takes place if not equal to 0

5.24.2.44 FP8_YYY_div_i()

```
void FP8_YYY_div_i (
            FP8_YYY * x )
```

Divide FP8 number by QNR.
Divide FP8 by the QNR

Parameters

x	FP8 instance

5.24.2.45 FP8_YYY_div_i2()

```
void FP8_YYY_div_i2 (
            FP8_YYY * x )
```

Divide FP8 number by QNR twice.

Divide FP8 by the QNR twice

Parameters

x	FP8 instance

5.24.2.46 FP8_YYY_div_2i()

```
void FP8_YYY_div_2i (
            FP8_YYY * x )
```

Divide FP8 number by QNR/2.
Divide FP8 by the QNR/2

Parameters

x	FP8 instance

5.24.2.47 FP8_YYY_rand()

```
void FP8_YYY_rand (
            FP8_YYY * x,
            csprng * rng )
```

Generate random FP8.

Parameters

x	random FP8 number
rng	random number generator

5.25 hpke.h File Reference

HPKE Header file.
`#include "ecdh_ZZZ.h"`

Macros

- #define HPKE_OK 0
- #define HPKE_INVALID_PUBLIC_KEY -2
- #define HPKE_ERROR -3

Functions

- int DeriveKeyPair_ZZZ (int config_id, octet ∗SK, octet ∗PK, octet ∗SEED)

    Derive a Key Pair from a seed.
- void HPKE_ZZZ_Encap (int config_id, octet ∗SK, octet ∗Z, octet ∗pkE, octet ∗pkR)

    Encapsulate function.
- void HPKE_ZZZ_Decap (int config_id, octet ∗skR, octet ∗Z, octet ∗pkE, octet ∗pkR)

    Decapsulate function.
- void HPKE_ZZZ_AuthEncap (int config_id, octet ∗skE, octet ∗skS, octet ∗Z, octet ∗pkE, octet ∗pkR, octet ∗pkS)

Encapsulate/Authenticate function.

- void HPKE_ZZZ_AuthDecap (int config_id, octet ∗skR, octet ∗Z, octet ∗pkE, octet ∗pkR, octet ∗pkS)

    Decapsulate function.

- void HPKE_ZZZ_KeySchedule (int config_id, octet ∗key, octet ∗nonce, octet ∗exp_secret, int mode, octet ∗Z, octet ∗info, octet ∗psk, octet ∗pskID)

    KeyScheduler function.

5.25.1 Detailed Description

HPKE Header file.

Author

Mike Scott

Date

2nd December 2019

declares functions

5.25.2 Macro Definition Documentation

5.25.2.1 HPKE_OK

```
#define HPKE_OK 0
```

Function completed without error

5.25.2.2 HPKE_INVALID_PUBLIC_KEY

```
#define HPKE_INVALID_PUBLIC_KEY -2
```

Public Key is Invalid

5.25.2.3 HPKE_ERROR

```
#define HPKE_ERROR -3
```

HPKE Internal Error

5.25.3 Function Documentation

5.25.3.1 DeriveKeyPair_ZZZ()

```
int DeriveKeyPair_ZZZ (
            int config_id,
            octet * SK,
            octet * PK,
            octet * SEED )
```

Derive a Key Pair from a seed.

Parameters

config←_id	is the configuration KEM/KDF/AEAD
SK	is the output secret key
PK	is the output public key
SEED	is the input random seed

Returns

1 if OK, 0 if failed

5.25.3.2 HPKE_ZZZ_Encap()

```
void HPKE_ZZZ_Encap (
            int config_id,
            octet * SK,
            octet * Z,
            octet * pkE,
            octet * pkR )
```

Encapsulate function.

Parameters

config_id	is the configuration KEM/KDF/AEAD
SK	is the input ephemeral secret
Z	is a pointer to a shared secret DH(skE,pkR)
pkE	the ephemeral public key, which is skE.G, where G is a fixed generator
pkR	the respondents public key

5.25.3.3 HPKE_ZZZ_Decap()

```
void HPKE_ZZZ_Decap (
            int config_id,
            octet * skR,
            octet * Z,
            octet * pkE,
            octet * pkR )
```

Decapsulate function.

Parameters

config_id	is the configuration KEM/KDF/AEAD
skR	the respondents private key
Z	is a pointer to a shared secret DH(skR,pkE)
pkE	the ephemeral public key
pkR	the respondents private key

5.25.3.4 HPKE_ZZZ_AuthEncap()

```
void HPKE_ZZZ_AuthEncap (
            int config_id,
            octet * skE,
            octet * skS,
            octet * Z,
            octet * pkE,
            octet * pkR,
            octet * pkS )
```

Encapsulate/Authenticate function.

Parameters

config←_id	is the configuration KEM/KDF/AEAD
skE	is the input ephemeral secret
skS	is the Initiators private key
Z	is a pointer to a shared secret DH(skE,pkR)
pkE	the ephemeral public key, which is skE.G, where G is a fixed generator
pkR	the Respondents public key
pkS	the Initiators public key

5.25.3.5 HPKE_ZZZ_AuthDecap()

```
void HPKE_ZZZ_AuthDecap (
            int config_id,
            octet * skR,
            octet * Z,
            octet * pkE,
            octet * pkR,
            octet * pkS )
```

Decapsulate function.

Parameters

config←_id	is the configuration KEM/KDF/AEAD
skR	is the Respondents private key
Z	is a pointer to a shared secret DH(skR,pkE)
pkE	the ephemeral public key
pkR	the Respondents public key
pkS	the Initiators public key

5.25.3.6 HPKE_ZZZ_KeySchedule()

```
void HPKE_ZZZ_KeySchedule (
            int config_id,
            octet * key,
            octet * nonce,
            octet * exp_secret,
            int mode,
            octet * Z,
            octet * info,
            octet * psk,
            octet * pskID )
```

KeyScheduler function.

Parameters

config_id	is the configuration KEM/KDF/AEAD
key	the output key for aead encryption

Parameters

nonce	the output nonce for aead encryption
exp_secret	the exporter secret
mode	the mode of operation
Z	the shared key
info	application dependent info
psk	pre-shared key
pskID	identifier for the psk

5.26 mpin.h File Reference

M-Pin Header file.

```
#include "pair_ZZZ.h"
```

Macros

- #define PGS_ZZZ MODBYTES_XXX
- #define PFS_ZZZ MODBYTES_XXX
- #define MPIN_OK 0
- #define MPIN_INVALID_POINT -14
- #define MPIN_BAD_PIN -19
- #define MAXPIN 10000
- #define PBLEN 14

Functions

- void MPIN_ZZZ_ENCODE_TO_CURVE (octet ∗DST, octet ∗ID, octet ∗HCID)

  Encode a string to a curve point (in constant time)
- int MPIN_ZZZ_EXTRACT_PIN (octet ∗HID, int pin, octet ∗CS)

  Extract a PIN number from a client secret.
- int MPIN_ZZZ_CLIENT_1 (octet ∗HID, csprng ∗R, octet ∗x, int pin, octet ∗T, octet ∗S, octet ∗U)

  Perform first pass of the client side of the 3-pass version of the M-Pin protocol.
- int MPIN_ZZZ_RANDOM_GENERATE (csprng ∗R, octet ∗S)

  Generate a random group element.
- int MPIN_ZZZ_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_ZZZ_SERVER (octet ∗HID, octet ∗y, octet ∗SS, octet ∗U, octet ∗V)

  Perform final pass on the server side of the M-Pin protocol.
- int MPIN_ZZZ_GET_CLIENT_SECRET (octet ∗S, octet ∗HID, octet ∗CS)

  Create a client secret in G1 from a master secret and the client ID.
- int MPIN_ZZZ_GET_SERVER_SECRET (octet ∗S, octet ∗SS)

  Create a server secret in G2 from a master secret.

5.26.1 Detailed Description

M-Pin Header file.

Author

Mike Scott and Kealan McCusker

Date

2nd June 2015

Allows some user configuration defines structures declares functions

5.26.2 Macro Definition Documentation

5.26.2.1 PGS_ZZZ

```
#define PGS_ZZZ MODBYTES_XXX
```

MPIN Group Size

5.26.2.2 PFS_ZZZ

```
#define PFS_ZZZ MODBYTES_XXX
```

MPIN Field Size

5.26.2.3 MPIN_OK

```
#define MPIN_OK 0
```

Function completed without error

5.26.2.4 MPIN_INVALID_POINT

```
#define MPIN_INVALID_POINT -14
```

Point is NOT on the curve

5.26.2.5 MPIN_BAD_PIN

```
#define MPIN_BAD_PIN -19
```

Bad PIN number entered

5.26.2.6 MAXPIN

```
#define MAXPIN 10000
```

max PIN

5.26.2.7 PBLEN

```
#define PBLEN 14
```

max length of PIN in bits

5.26.3 Function Documentation

5.26.3.1 MPIN_ZZZ_ENCODE_TO_CURVE()

```
void MPIN_ZZZ_ENCODE_TO_CURVE (
            octet * DST,
            octet * ID,
            octet * HCID )
```

Encode a string to a curve point (in constant time)

Parameters

DST	is the Domain Separation Tag

Parameters

ID	is the input string
HCID	is the output point in G1

5.26.3.2 MPIN_ZZZ_EXTRACT_PIN()

```
int MPIN_ZZZ_EXTRACT_PIN (
            octet * HID,
            int pin,
            octet * CS )
```

Extract a PIN number from a client secret.

Parameters

HID	is the hashed-to-curve 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

5.26.3.3 MPIN_ZZZ_CLIENT_1()

```
int MPIN_ZZZ_CLIENT_1 (
            octet * HID,
            csprng * R,
            octet * x,
            int pin,
            octet * T,
            octet * S,
            octet * U )
```

Perform first pass of the client side of the 3-pass version of the M-Pin protocol.

Parameters

HID	is the hashed-to-curve 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)
S	is the reconstructed client secret
U	is output = x.H(ID)

Returns

0 or an error code

5.26.3.4 MPIN_ZZZ_RANDOM_GENERATE()

```
int MPIN_ZZZ_RANDOM_GENERATE (
```

```
            csprng * R,
            octet * S )
```

Generate a random group element.

Parameters

R	is a pointer to a cryptographically secure random number generator
S	is the output random octet

Returns

0 or an error code

5.26.3.5 MPIN_ZZZ_CLIENT_2()

```
int MPIN_ZZZ_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.

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

5.26.3.6 MPIN_ZZZ_SERVER()

```
int MPIN_ZZZ_SERVER (
            octet * HID,
            octet * y,
            octet * SS,
            octet * U,
            octet * V )
```

Perform final pass on the server side of the M-Pin protocol.

Parameters

HID	is input H(ID), a hash of the client 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)
V	is an input from the client

Returns

0 or an error code

5.26.3.7 MPIN_ZZZ_GET_CLIENT_SECRET()

```
int MPIN_ZZZ_GET_CLIENT_SECRET (
            octet * S,
            octet * HID,
            octet * CS )
```

Create a client secret in G1 from a master secret and the client ID.

Parameters

S	is an input master secret
HID	is the input client identity hashed to point
CS	is the full client secret = s.H(ID)

Returns

0 or an error code

5.26.3.8 MPIN_ZZZ_GET_SERVER_SECRET()

```
int MPIN_ZZZ_GET_SERVER_SECRET (
            octet * S,
            octet * SS )
```

Create a server secret in G2 from a master secret.

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

5.27 mpin192.h File Reference

M-Pin Header file.

`#include "pair4_ZZZ.h"`

Macros

- #define PGS_ZZZ MODBYTES_XXX
- #define PFS_ZZZ MODBYTES_XXX
- #define MPIN_OK 0
- #define MPIN_INVALID_POINT -14
- #define MPIN_BAD_PIN -19
- #define MAXPIN 10000
- #define PBLEN 14

Functions

- void MPIN_ZZZ_ENCODE_TO_CURVE (octet *DST, octet *ID, octet *HCID)

  Encode a string to a curve point (in constant time)
- int MPIN_ZZZ_EXTRACT_PIN (octet *HID, int pin, octet *CS)

Extract a PIN number from a client secret.
- int MPIN_ZZZ_CLIENT_1 (octet ∗HID, csprng ∗R, octet ∗x, int pin, octet ∗T, octet ∗S, octet ∗U)

  Perform first pass of the client side of the 3-pass version of the M-Pin protocol.
- int MPIN_ZZZ_RANDOM_GENERATE (csprng ∗R, octet ∗S)

  Generate a random group element.
- int MPIN_ZZZ_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_ZZZ_SERVER (octet ∗HID, octet ∗y, octet ∗SS, octet ∗U, octet ∗V)

  Perform final pass on the server side of the M-Pin protocol.
- int MPIN_ZZZ_GET_CLIENT_SECRET (octet ∗S, octet ∗HID, octet ∗CS)

  Create a client secret in G1 from a master secret and the client ID.
- int MPIN_ZZZ_GET_SERVER_SECRET (octet ∗S, octet ∗SS)

  Create a server secret in G2 from a master secret.

5.27.1 Detailed Description

M-Pin Header file.

Author

Mike Scott and Kealan McCusker

Date

2nd June 2015

Allows some user configuration defines structures declares functions

5.27.2 Macro Definition Documentation

5.27.2.1 PGS_ZZZ

```
#define PGS_ZZZ MODBYTES_XXX
```
MPIN Group Size

5.27.2.2 PFS_ZZZ

```
#define PFS_ZZZ MODBYTES_XXX
```
MPIN Field Size

5.27.2.3 MPIN_OK

```
#define MPIN_OK 0
```
Function completed without error

5.27.2.4 MPIN_INVALID_POINT

```
#define MPIN_INVALID_POINT -14
```
Point is NOT on the curve

5.27.2.5 MPIN_BAD_PIN

```
#define MPIN_BAD_PIN -19
```
Bad PIN number entered

5.27.2.6 MAXPIN

```
#define MAXPIN 10000
```

max PIN

5.27.2.7 PBLEN

```
#define PBLEN 14
```

max length of PIN in bits

5.27.3 Function Documentation

5.27.3.1 MPIN_ZZZ_ENCODE_TO_CURVE()

```
void MPIN_ZZZ_ENCODE_TO_CURVE (
            octet * DST,
            octet * ID,
            octet * HCID )
```

Encode a string to a curve point (in constant time)

Parameters

DST	is the Domain Separation Tag
ID	is the input string
HCID	is the output point in G1

5.27.3.2 MPIN_ZZZ_EXTRACT_PIN()

```
int MPIN_ZZZ_EXTRACT_PIN (
            octet * HID,
            int pin,
            octet * CS )
```

Extract a PIN number from a client secret.

Parameters

HID	is the hashed-to-curve 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

5.27.3.3 MPIN_ZZZ_CLIENT_1()

```
int MPIN_ZZZ_CLIENT_1 (
            octet * HID,
            csprng * R,
            octet * x,
            int pin,
            octet * T,
```

```
            octet * S,
            octet * U )
```

Perform first pass of the client side of the 3-pass version of the M-Pin protocol.

Parameters

HID	is the hashed-to-curve 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)
S	is the reconstructed client secret
U	is output = x.H(ID)

Returns

0 or an error code

5.27.3.4 MPIN_ZZZ_RANDOM_GENERATE()

```
int MPIN_ZZZ_RANDOM_GENERATE (
            csprng * R,
            octet * S )
```

Generate a random group element.

Parameters

R	is a pointer to a cryptographically secure random number generator
S	is the output random octet

Returns

0 or an error code

5.27.3.5 MPIN_ZZZ_CLIENT_2()

```
int MPIN_ZZZ_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.

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

5.27.3.6 MPIN_ZZZ_SERVER()

```
int MPIN_ZZZ_SERVER (
            octet * HID,
            octet * y,
            octet * SS,
            octet * U,
            octet * V )
```

Perform final pass on the server side of the M-Pin protocol.

Parameters

HID	is input H(ID), a hash of the client 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)
V	is an input from the client

Returns

0 or an error code

5.27.3.7 MPIN_ZZZ_GET_CLIENT_SECRET()

```
int MPIN_ZZZ_GET_CLIENT_SECRET (
            octet * S,
            octet * HID,
            octet * CS )
```

Create a client secret in G1 from a master secret and the client ID.

Parameters

S	is an input master secret
HID	is the input client identity hashed to point
CS	is the full client secret = s.H(ID)

Returns

0 or an error code

5.27.3.8 MPIN_ZZZ_GET_SERVER_SECRET()

```
int MPIN_ZZZ_GET_SERVER_SECRET (
            octet * S,
            octet * SS )
```

Create a server secret in G2 from a master secret.

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

5.28 mpin256.h File Reference

M-Pin Header file.
`#include "pair8_ZZZ.h"`

Macros

- #define PGS_ZZZ MODBYTES_XXX
- #define PFS_ZZZ MODBYTES_XXX
- #define MPIN_OK 0
- #define MPIN_INVALID_POINT -14
- #define MPIN_BAD_PIN -19
- #define MAXPIN 10000
- #define PBLEN 14

Functions

- void MPIN_ZZZ_ENCODE_TO_CURVE (octet ∗DST, octet ∗ID, octet ∗HCID)

  Encode a string to a curve point (in constant time)
- int MPIN_ZZZ_EXTRACT_PIN (octet ∗HID, int pin, octet ∗CS)

  Extract a PIN number from a client secret.
- int MPIN_ZZZ_CLIENT_1 (octet ∗HID, csprng ∗R, octet ∗x, int pin, octet ∗T, octet ∗S, octet ∗U)

  Perform first pass of the client side of the 3-pass version of the M-Pin protocol.
- int MPIN_ZZZ_RANDOM_GENERATE (csprng ∗R, octet ∗S)

  Generate a random group element.
- int MPIN_ZZZ_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_ZZZ_SERVER (octet ∗HID, octet ∗y, octet ∗SS, octet ∗U, octet ∗V)

  Perform final pass on the server side of the M-Pin protocol.
- int MPIN_ZZZ_GET_CLIENT_SECRET (octet ∗S, octet ∗HID, octet ∗CS)

  Create a client secret in G1 from a master secret and the client ID.
- int MPIN_ZZZ_GET_SERVER_SECRET (octet ∗S, octet ∗SS)

  Create a server secret in G2 from a master secret.

5.28.1 Detailed Description

M-Pin Header file.

Author

Mike Scott and Kealan McCusker

Date

2nd June 2015

Allows some user configuration defines structures declares functions

5.28.2 Macro Definition Documentation

5.28.2.1 PGS_ZZZ

```
#define PGS_ZZZ MODBYTES_XXX
```
MPIN Group Size

5.28.2.2 PFS_ZZZ

```
#define PFS_ZZZ MODBYTES_XXX
```
MPIN Field Size

5.28.2.3 MPIN_OK

```
#define MPIN_OK 0
```
Function completed without error

5.28.2.4 MPIN_INVALID_POINT

```
#define MPIN_INVALID_POINT -14
```
Point is NOT on the curve

5.28.2.5 MPIN_BAD_PIN

```
#define MPIN_BAD_PIN -19
```
Bad PIN number entered

5.28.2.6 MAXPIN

```
#define MAXPIN 10000
```
max PIN

5.28.2.7 PBLEN

```
#define PBLEN 14
```
max length of PIN in bits

5.28.3 Function Documentation

5.28.3.1 MPIN_ZZZ_ENCODE_TO_CURVE()

```
void MPIN_ZZZ_ENCODE_TO_CURVE (
            octet * DST,
            octet * ID,
            octet * HCID )
```
Encode a string to a curve point (in constant time)

Parameters

DST	is the Domain Separation Tag
ID	is the input string
HCID	is the output point in G1

5.28.3.2 MPIN_ZZZ_EXTRACT_PIN()

```
int MPIN_ZZZ_EXTRACT_PIN (
            octet * HID,
```

```
            int pin,
            octet * CS )
```

Extract a PIN number from a client secret.

Parameters

HID	is the hashed-to-curve 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

5.28.3.3 MPIN_ZZZ_CLIENT_1()

```
int MPIN_ZZZ_CLIENT_1 (
            octet * HID,
            csprng * R,
            octet * x,
            int pin,
            octet * T,
            octet * S,
            octet * U )
```

Perform first pass of the client side of the 3-pass version of the M-Pin protocol.

Parameters

HID	is the hashed-to-curve 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)
S	is the reconstructed client secret
U	is output = x.H(ID)

Returns

0 or an error code

5.28.3.4 MPIN_ZZZ_RANDOM_GENERATE()

```
int MPIN_ZZZ_RANDOM_GENERATE (
            csprng * R,
            octet * S )
```

Generate a random group element.

Parameters

R	is a pointer to a cryptographically secure random number generator
S	is the output random octet

Returns

0 or an error code

5.28.3.5 MPIN_ZZZ_CLIENT_2()

```
int MPIN_ZZZ_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.

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

5.28.3.6 MPIN_ZZZ_SERVER()

```
int MPIN_ZZZ_SERVER (
            octet * HID,
            octet * y,
            octet * SS,
            octet * U,
            octet * V )
```

Perform final pass on the server side of the M-Pin protocol.

Parameters

HID	is input H(ID), a hash of the client 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)
V	is an input from the client

Returns

0 or an error code

5.28.3.7 MPIN_ZZZ_GET_CLIENT_SECRET()

```
int MPIN_ZZZ_GET_CLIENT_SECRET (
            octet * S,
            octet * HID,
            octet * CS )
```

Create a client secret in G1 from a master secret and the client ID.

Parameters

S	is an input master secret
HID	is the input client identity hashed to point
CS	is the full client secret = s.H(ID)

Returns

0 or an error code

5.28.3.8 MPIN_ZZZ_GET_SERVER_SECRET()

```
int MPIN_ZZZ_GET_SERVER_SECRET (
        octet * S,
        octet * SS )
```

Create a server secret in G2 from a master secret.

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

5.29 newhope.h File Reference

Newhope Header File.

`#include "core.h"`

Functions

- void NHS_SERVER_1 (csprng ∗RNG, octet ∗SB, octet ∗S)
  NHS server first pass.
- void NHS_CLIENT (csprng ∗RNG, octet ∗SB, octet ∗UC, octet ∗KEY)
  NHS client pass.
- void NHS_SERVER_2 (octet ∗S, octet ∗UC, octet ∗KEY)
  NHS server second pass.

5.29.1 Detailed Description

Newhope Header File.

Author

Mike Scott

5.29.2 Function Documentation

5.29.2.1 NHS_SERVER_1()

```
void NHS_SERVER_1 (
            csprng * RNG,
            octet * SB,
            octet * S )
```

NHS server first pass.

Parameters

RNG	Random Number Generator handle
SB	seed and polynomial B concatenated - output
S	server secret - output

5.29.2.2 NHS_CLIENT()

```
void NHS_CLIENT (
            csprng * RNG,
            octet * SB,
            octet * UC,
            octet * KEY )
```

NHS client pass.

Parameters

RNG	Random Number Generator handle
SB	seed and polynomial B concatenated - input
UC	polynomial U and compressed polynomial c - output
KEY	client key

5.29.2.3 NHS_SERVER_2()

```
void NHS_SERVER_2 (
            octet * S,
            octet * UC,
            octet * KEY )
```

NHS server second pass.

Parameters

S	server secret - input
UC	polynomial U and compressed polynomial c - input
KEY	server key

5.30 pair.h File Reference

PAIR Header File.

```
#include "fp12_YYY.h"
#include "ecp2_ZZZ.h"
#include "ecp_ZZZ.h"
```

Functions

- void PAIR_ZZZ_precomp (FP4_YYY T[ ], ECP2_ZZZ ∗GV)

    Precompute line functions details for fixed G2 value.
- void PAIR_ZZZ_another (FP12_YYY r[ ], ECP2_ZZZ ∗PV, ECP_ZZZ ∗QV)

    Precompute line functions for n-pairing.
- void PAIR_ZZZ_another_pc (FP12_YYY r[ ], FP4_YYY T[ ], ECP_ZZZ ∗QV)

    Compute line functions for n-pairing, assuming precomputation on G2.
- void PAIR_ZZZ_ate (FP12_YYY ∗r, ECP2_ZZZ ∗P, ECP_ZZZ ∗Q)

    Calculate Miller loop for Optimal ATE pairing e(P,Q)
- void PAIR_ZZZ_double_ate (FP12_YYY ∗r, ECP2_ZZZ ∗P, ECP_ZZZ ∗Q, ECP2_ZZZ ∗R, ECP_ZZZ ∗S)

    Calculate Miller loop for Optimal ATE double-pairing e(P,Q).e(R,S)
- void PAIR_ZZZ_fexp (FP12_YYY ∗x)

    Final exponentiation of pairing, converts output of Miller loop to element in GT.
- void PAIR_ZZZ_G1mul (ECP_ZZZ ∗Q, BIG_XXX b)

    Fast point multiplication of a member of the group G1 by a BIG number.
- void PAIR_ZZZ_G2mul (ECP2_ZZZ ∗P, BIG_XXX b)

    Fast point multiplication of a member of the group G2 by a BIG number.
- void PAIR_ZZZ_GTpow (FP12_YYY ∗x, BIG_XXX b)

    Fast raising of a member of GT to a BIG power.
- int PAIR_ZZZ_G1member (ECP_ZZZ ∗P)

    Tests ECP for membership of G1.
- int PAIR_ZZZ_G2member (ECP2_ZZZ ∗P)

    Tests ECP2 for membership of G2.
- int PAIR_ZZZ_GTmember (FP12_YYY ∗x)

    Tests FP12 for membership of GT.
- int PAIR_ZZZ_nbits (BIG_XXX n3, BIG_XXX n)

    Prepare Ate parameter.
- void PAIR_ZZZ_initmp (FP12_YYY r[ ])

    Initialise structure for multi-pairing.
- void PAIR_ZZZ_miller (FP12_YYY ∗res, FP12_YYY r[ ])

    Miller loop.

Variables

- const BIG_XXX CURVE_Bnx_ZZZ
- const BIG_XXX CURVE_Cru_ZZZ
- const BIG_XXX CURVE_W_ZZZ [2]
- const BIG_XXX CURVE_SB_ZZZ [2][2]
- const BIG_XXX CURVE_WB_ZZZ [4]
- const BIG_XXX CURVE_BB_ZZZ [4][4]

5.30.1 Detailed Description

PAIR Header File.

Author

Mike Scott

5.30.2 Function Documentation

5.30.2.1 PAIR_ZZZ_precomp()

```
void PAIR_ZZZ_precomp (
            FP4_YYY T[ ],
            ECP2_ZZZ * GV )
```

Precompute line functions details for fixed G2 value.

Parameters

T	array of precomputed FP4 partial line functions
GV	a fixed ECP2 instance

5.30.2.2 PAIR_ZZZ_another()

```
void PAIR_ZZZ_another (
            FP12_YYY r[ ],
            ECP2_ZZZ * PV,
            ECP_ZZZ * QV )
```

Precompute line functions for n-pairing.

Parameters

r	array of precomputed FP12 products of line functions
PV	ECP2 instance, an element of G2
QV	ECP instance, an element of G1

5.30.2.3 PAIR_ZZZ_another_pc()

```
void PAIR_ZZZ_another_pc (
            FP12_YYY r[ ],
            FP4_YYY T[ ],
            ECP_ZZZ * QV )
```

Compute line functions for n-pairing, assuming precomputation on G2.

Parameters

r	array of precomputed FP12 products of line functions
T	array contains precomputed partial line fucntions from G2
QV	ECP instance, an element of G1

5.30.2.4 PAIR_ZZZ_ate()

```
void PAIR_ZZZ_ate (
            FP12_YYY * r,
            ECP2_ZZZ * P,
            ECP_ZZZ * Q )
```

Calculate Miller loop for Optimal ATE pairing e(P,Q)

Parameters

r	FP12 result of the pairing calculation e(P,Q)

Parameters

P	ECP2 instance, an element of G2
Q	ECP instance, an element of G1

5.30.2.5 PAIR_ZZZ_double_ate()

```
void PAIR_ZZZ_double_ate (
        FP12_YYY * r,
        ECP2_ZZZ * P,
        ECP_ZZZ * Q,
        ECP2_ZZZ * R,
        ECP_ZZZ * S )
```

Calculate Miller loop for Optimal ATE double-pairing e(P,Q).e(R,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

5.30.2.6 PAIR_ZZZ_fexp()

```
void PAIR_ZZZ_fexp (
        FP12_YYY * x )
```

Final exponentiation of pairing, converts output of Miller loop to element in GT.
Here p is the internal modulus, and r is the group order

Parameters

x	FP12, on exit = x^((p^12-1)/r)

5.30.2.7 PAIR_ZZZ_G1mul()

```
void PAIR_ZZZ_G1mul (
        ECP_ZZZ * Q,
        BIG_XXX b )
```

Fast point multiplication of a member of the group G1 by a BIG number.
May exploit endomorphism for speed.

Parameters

Q	ECP member of G1.
b	BIG multiplier

5.30.2.8 PAIR_ZZZ_G2mul()

```
void PAIR_ZZZ_G2mul (
            ECP2_ZZZ * P,
            BIG_XXX b )
```

Fast point multiplication of a member of the group G2 by a BIG number.
May exploit endomorphism for speed.

Parameters

P	ECP2 member of G1.
b	BIG multiplier

5.30.2.9 PAIR_ZZZ_GTpow()

```
void PAIR_ZZZ_GTpow (
            FP12_YYY * x,
            BIG_XXX b )
```

Fast raising of a member of GT to a BIG power.
May exploit endomorphism for speed.

Parameters

x	FP12 member of GT.
b	BIG exponent

5.30.2.10 PAIR_ZZZ_G1member()

```
int PAIR_ZZZ_G1member (
            ECP_ZZZ * P )
```

Tests ECP for membership of G1.

Parameters

P	ECP member of G1

Returns

true or false

5.30.2.11 PAIR_ZZZ_G2member()

```
int PAIR_ZZZ_G2member (
            ECP2_ZZZ * P )
```

Tests ECP2 for membership of G2.

Parameters

P	ECP2 member of G2

Returns

true or false

5.30.2.12 PAIR_ZZZ_GTmember()

```
int PAIR_ZZZ_GTmember (
            FP12_YYY * x )
```

Tests FP12 for membership of GT.

Parameters

x	FP12 instance

Returns

1 if x is in GT, else return 0

5.30.2.13 PAIR_ZZZ_nbits()

```
int PAIR_ZZZ_nbits (
            BIG_XXX n3,
            BIG_XXX n )
```

Prepare Ate parameter.

Parameters

n	BIG parameter
n3	BIG paramter = 3*n

Returns

number of nits in n3

5.30.2.14 PAIR_ZZZ_initmp()

```
void PAIR_ZZZ_initmp (
            FP12_YYY r[ ] )
```

Initialise structure for multi-pairing.

Parameters

r	FP12 array, to be initialised to 1

5.30.2.15 PAIR_ZZZ_miller()

```
void PAIR_ZZZ_miller (
            FP12_YYY * res,
            FP12_YYY r[ ] )
```

Miller loop.

Parameters

res	FP12 result
r	FP12 precomputed array of accumulated line functions

5.30.3 Variable Documentation

5.30.3.1 CURVE_Bnx_ZZZ

```
const BIG_XXX CURVE_Bnx_ZZZ [extern]
```

BN curve x parameter

5.30.3.2 CURVE_Cru_ZZZ

```
const BIG_XXX CURVE_Cru_ZZZ [extern]
```

BN curve Cube Root of Unity

5.30.3.3 CURVE_W_ZZZ

```
const BIG_XXX CURVE_W_ZZZ[2] [extern]
```

BN curve constant for GLV decomposition

5.30.3.4 CURVE_SB_ZZZ

```
const BIG_XXX CURVE_SB_ZZZ[2][2] [extern]
```

BN curve constant for GLV decomposition

5.30.3.5 CURVE_WB_ZZZ

```
const BIG_XXX CURVE_WB_ZZZ[4] [extern]
```

BN curve constant for GS decomposition

5.30.3.6 CURVE_BB_ZZZ

```
const BIG_XXX CURVE_BB_ZZZ[4][4] [extern]
```

BN curve constant for GS decomposition

5.31 pair4.h File Reference

PAIR Header File.

```
#include "fp24_YYY.h"
#include "ecp4_ZZZ.h"
#include "ecp_ZZZ.h"
```

Functions

- void PAIR_ZZZ_precomp (FP8_YYY T[ ], ECP4_ZZZ ∗GV)

    Precompute line functions details for fixed G2 value.
- void PAIR_ZZZ_another (FP24_YYY r[ ], ECP4_ZZZ ∗PV, ECP_ZZZ ∗QV)

    Precompute line functions for n-pairing.
- void PAIR_ZZZ_another_pc (FP24_YYY r[ ], FP8_YYY T[ ], ECP_ZZZ ∗QV)

    Compute line functions for n-pairing, assuming precomputation on G2.
- void PAIR_ZZZ_ate (FP24_YYY ∗r, ECP4_ZZZ ∗P, ECP_ZZZ ∗Q)

Calculate Miller loop for Optimal ATE pairing e(P,Q)
- void PAIR_ZZZ_double_ate (FP24_YYY ∗r, ECP4_ZZZ ∗P, ECP_ZZZ ∗Q, ECP4_ZZZ ∗R, ECP_ZZZ ∗S)

  Calculate Miller loop for Optimal ATE double-pairing e(P,Q).e(R,S)
- void PAIR_ZZZ_fexp (FP24_YYY ∗x)

  Final exponentiation of pairing, converts output of Miller loop to element in GT.
- void PAIR_ZZZ_G1mul (ECP_ZZZ ∗Q, BIG_XXX b)

  Fast point multiplication of a member of the group G1 by a BIG number.
- void PAIR_ZZZ_G2mul (ECP4_ZZZ ∗P, BIG_XXX b)

  Fast point multiplication of a member of the group G2 by a BIG number.
- void PAIR_ZZZ_GTpow (FP24_YYY ∗x, BIG_XXX b)

  Fast raising of a member of GT to a BIG power.
- int PAIR_ZZZ_G1member (ECP_ZZZ ∗P)

  Tests ECP for membership of G1.
- int PAIR_ZZZ_G2member (ECP4_ZZZ ∗P)

  Tests ECP4 for membership of G2.
- int PAIR_ZZZ_GTmember (FP24_YYY ∗x)

  Tests FP24 for membership of GT.
- int PAIR_ZZZ_nbits (BIG_XXX n3, BIG_XXX n)

  Prepare Ate parameter.
- void PAIR_ZZZ_initmp (FP24_YYY r[ ])

  Initialise structure for multi-pairing.
- void PAIR_ZZZ_miller (FP24_YYY ∗res, FP24_YYY r[ ])

  Miller loop.

Variables

- const BIG_XXX CURVE_Bnx_ZZZ
- const BIG_XXX CURVE_Cru_ZZZ
- const BIG_XXX CURVE_W_ZZZ [2]
- const BIG_XXX CURVE_SB_ZZZ [2][2]
- const BIG_XXX CURVE_WB_ZZZ [4]
- const BIG_XXX CURVE_BB_ZZZ [4][4]

5.31.1 Detailed Description

PAIR Header File.

Author

Mike Scott

5.31.2 Function Documentation

5.31.2.1 PAIR_ZZZ_precomp()

```
void PAIR_ZZZ_precomp (
            FP8_YYY T[ ],
            ECP4_ZZZ * GV )
```

Precompute line functions details for fixed G2 value.

Parameters

T	array of precomputed FP8 partial line functions
GV	a fixed ECP4 instance

5.31.2.2 PAIR_ZZZ_another()

```
void PAIR_ZZZ_another (
            FP24_YYY r[ ],
            ECP4_ZZZ * PV,
            ECP_ZZZ * QV )
```

Precompute line functions for n-pairing.

Parameters

r	array of precomputed FP24 products of line functions
PV	ECP4 instance, an element of G2
QV	ECP instance, an element of G1

5.31.2.3 PAIR_ZZZ_another_pc()

```
void PAIR_ZZZ_another_pc (
            FP24_YYY r[ ],
            FP8_YYY T[ ],
            ECP_ZZZ * QV )
```

Compute line functions for n-pairing, assuming precomputation on G2.

Parameters

r	array of precomputed FP24 products of line functions
T	array contains precomputed partial line fucntions from G2
QV	ECP instance, an element of G1

5.31.2.4 PAIR_ZZZ_ate()

```
void PAIR_ZZZ_ate (
            FP24_YYY * r,
            ECP4_ZZZ * P,
            ECP_ZZZ * Q )
```

Calculate Miller loop for Optimal ATE pairing e(P,Q)

Parameters

r	FP24 result of the pairing calculation e(P,Q)
P	ECP4 instance, an element of G2
Q	ECP instance, an element of G1

5.31.2.5 PAIR_ZZZ_double_ate()

```
void PAIR_ZZZ_double_ate (
            FP24_YYY * r,
            ECP4_ZZZ * P,
            ECP_ZZZ * Q,
            ECP4_ZZZ * R,
```

```
            ECP_ZZZ * S )
```

Calculate Miller loop for Optimal ATE double-pairing e(P,Q).e(R,S)
Faster than calculating two separate pairings

Parameters

r	FP24 result of the pairing calculation e(P,Q).e(R,S), an element of GT
P	ECP4 instance, an element of G2
Q	ECP instance, an element of G1
R	ECP4 instance, an element of G2
S	ECP instance, an element of G1

5.31.2.6 PAIR_ZZZ_fexp()

```
void PAIR_ZZZ_fexp (
            FP24_YYY * x )
```

Final exponentiation of pairing, converts output of Miller loop to element in GT.
Here p is the internal modulus, and r is the group order

Parameters

x	FP24, on exit = x^((p^12-1)/r)

5.31.2.7 PAIR_ZZZ_G1mul()

```
void PAIR_ZZZ_G1mul (
            ECP_ZZZ * Q,
            BIG_XXX b )
```

Fast point multiplication of a member of the group G1 by a BIG number.
May exploit endomorphism for speed.

Parameters

Q	ECP member of G1.
b	BIG multiplier

5.31.2.8 PAIR_ZZZ_G2mul()

```
void PAIR_ZZZ_G2mul (
            ECP4_ZZZ * P,
            BIG_XXX b )
```

Fast point multiplication of a member of the group G2 by a BIG number.
May exploit endomorphism for speed.

Parameters

P	ECP4 member of G1.
b	BIG multiplier

5.31.2.9 PAIR_ZZZ_GTpow()

```
void PAIR_ZZZ_GTpow (
            FP24_YYY * x,
            BIG_XXX b )
```

Fast raising of a member of GT to a BIG power.
May exploit endomorphism for speed.

Parameters

x	FP24 member of GT.
b	BIG exponent

5.31.2.10 PAIR_ZZZ_G1member()

```
int PAIR_ZZZ_G1member (
            ECP_ZZZ * P )
```

Tests ECP for membership of G1.

Parameters

P	ECP member of G1

Returns

true or false

5.31.2.11 PAIR_ZZZ_G2member()

```
int PAIR_ZZZ_G2member (
            ECP4_ZZZ * P )
```

Tests ECP4 for membership of G2.

Parameters

P	ECP4 member of G2

Returns

true or false

5.31.2.12 PAIR_ZZZ_GTmember()

```
int PAIR_ZZZ_GTmember (
            FP24_YYY * x )
```

Tests FP24 for membership of GT.

Parameters

x	FP24 instance

Returns

1 if x is in GT, else return 0

5.31.2.13 PAIR_ZZZ_nbits()

```
int PAIR_ZZZ_nbits (
            BIG_XXX n3,
            BIG_XXX n )
```

Prepare Ate parameter.

Parameters

n	BIG parameter
n3	BIG paramter = 3∗n

Returns

number of nits in n3

5.31.2.14 PAIR_ZZZ_initmp()

```
void PAIR_ZZZ_initmp (
            FP24_YYY r[ ] )
```

Initialise structure for multi-pairing.

Parameters

r	FP24 array, to be initialised to 1

5.31.2.15 PAIR_ZZZ_miller()

```
void PAIR_ZZZ_miller (
            FP24_YYY * res,
            FP24_YYY r[ ] )
```

Miller loop.

Parameters

res	FP24 result
r	FP24 precomputed array of accumulated line functions

5.31.3 Variable Documentation

5.31.3.1 CURVE_Bnx_ZZZ

```
const BIG_XXX CURVE_Bnx_ZZZ [extern]
```

BN curve x parameter

5.31.3.2 CURVE_Cru_ZZZ

`const BIG_XXX CURVE_Cru_ZZZ [extern]`

BN curve Cube Root of Unity

5.31.3.3 CURVE_W_ZZZ

`const BIG_XXX CURVE_W_ZZZ[2] [extern]`

BN curve constant for GLV decomposition

5.31.3.4 CURVE_SB_ZZZ

`const BIG_XXX CURVE_SB_ZZZ[2][2] [extern]`

BN curve constant for GLV decomposition

5.31.3.5 CURVE_WB_ZZZ

`const BIG_XXX CURVE_WB_ZZZ[4] [extern]`

BN curve constant for GS decomposition

5.31.3.6 CURVE_BB_ZZZ

`const BIG_XXX CURVE_BB_ZZZ[4][4] [extern]`

BN curve constant for GS decomposition

5.32 pair8.h File Reference

PAIR Header File.

```
#include "fp48_YYY.h"
#include "ecp8_ZZZ.h"
#include "ecp_ZZZ.h"
```

Functions

- void PAIR_ZZZ_precomp (FP16_YYY T[ ], ECP8_ZZZ ∗GV)

    Precompute line functions details for fixed G2 value.
- void PAIR_ZZZ_another (FP48_YYY r[ ], ECP8_ZZZ ∗PV, ECP_ZZZ ∗QV)

    Precompute line functions for n-pairing.
- void PAIR_ZZZ_another_pc (FP48_YYY r[ ], FP16_YYY T[ ], ECP_ZZZ ∗QV)

    Compute line functions for n-pairing, assuming precomputation on G2.
- void PAIR_ZZZ_ate (FP48_YYY ∗r, ECP8_ZZZ ∗P, ECP_ZZZ ∗Q)

    Calculate Miller loop for Optimal ATE pairing e(P,Q)
- void PAIR_ZZZ_double_ate (FP48_YYY ∗r, ECP8_ZZZ ∗P, ECP_ZZZ ∗Q, ECP8_ZZZ ∗R, ECP_ZZZ ∗S)

    Calculate Miller loop for Optimal ATE double-pairing e(P,Q).e(R,S)
- void PAIR_ZZZ_fexp (FP48_YYY ∗x)

    Final exponentiation of pairing, converts output of Miller loop to element in GT.
- void PAIR_ZZZ_G1mul (ECP_ZZZ ∗Q, BIG_XXX b)

    Fast point multiplication of a member of the group G1 by a BIG number.
- void PAIR_ZZZ_G2mul (ECP8_ZZZ ∗P, BIG_XXX b)

    Fast point multiplication of a member of the group G2 by a BIG number.
- void PAIR_ZZZ_GTpow (FP48_YYY ∗x, BIG_XXX b)

    Fast raising of a member of GT to a BIG power.
- int PAIR_ZZZ_G1member (ECP_ZZZ ∗P)

    Tests ECP for membership of G1.

- int PAIR_ZZZ_G2member (ECP8_ZZZ *P)

  Tests ECP8 for membership of G2.
- int PAIR_ZZZ_GTmember (FP48_YYY *x)

  Tests FP48 for membership of GT.
- int PAIR_ZZZ_nbits (BIG_XXX n3, BIG_XXX n)

  Prepare Ate parameter.
- void PAIR_ZZZ_initmp (FP48_YYY r[ ])

  Initialise structure for multi-pairing.
- void PAIR_ZZZ_miller (FP48_YYY *res, FP48_YYY r[ ])

  Miller loop.

Variables

- const BIG_XXX CURVE_Bnx_ZZZ
- const BIG_XXX CURVE_Cru_ZZZ
- const BIG_XXX CURVE_W_ZZZ [2]
- const BIG_XXX CURVE_SB_ZZZ [2][2]
- const BIG_XXX CURVE_WB_ZZZ [4]
- const BIG_XXX CURVE_BB_ZZZ [4][4]

5.32.1 Detailed Description

PAIR Header File.

Author

Mike Scott

5.32.2 Function Documentation

5.32.2.1 PAIR_ZZZ_precomp()

```
void PAIR_ZZZ_precomp (
            FP16_YYY T[ ],
            ECP8_ZZZ * GV )
```

Precompute line functions details for fixed G2 value.

Parameters

T	array of precomputed FP16 partial line functions
GV	a fixed ECP8 instance

5.32.2.2 PAIR_ZZZ_another()

```
void PAIR_ZZZ_another (
            FP48_YYY r[ ],
            ECP8_ZZZ * PV,
            ECP_ZZZ * QV )
```

Precompute line functions for n-pairing.

Parameters

r	array of precomputed FP48 products of line functions

Parameters

PV	ECP8 instance, an element of G2
QV	ECP instance, an element of G1

5.32.2.3 PAIR_ZZZ_another_pc()

```
void PAIR_ZZZ_another_pc (
            FP48_YYY r[ ],
            FP16_YYY T[ ],
            ECP_ZZZ * QV )
```

Compute line functions for n-pairing, assuming precomputation on G2.

Parameters

r	array of precomputed FP48 products of line functions
T	array contains precomputed partial line fucntions from G2
QV	ECP instance, an element of G1

5.32.2.4 PAIR_ZZZ_ate()

```
void PAIR_ZZZ_ate (
            FP48_YYY * r,
            ECP8_ZZZ * P,
            ECP_ZZZ * Q )
```

Calculate Miller loop for Optimal ATE pairing e(P,Q)

Parameters

r	FP48 result of the pairing calculation e(P,Q)
P	ECP8 instance, an element of G2
Q	ECP instance, an element of G1

5.32.2.5 PAIR_ZZZ_double_ate()

```
void PAIR_ZZZ_double_ate (
            FP48_YYY * r,
            ECP8_ZZZ * P,
            ECP_ZZZ * Q,
            ECP8_ZZZ * R,
            ECP_ZZZ * S )
```

Calculate Miller loop for Optimal ATE double-pairing e(P,Q).e(R,S)
Faster than calculating two separate pairings

Parameters

r	FP48 result of the pairing calculation e(P,Q).e(R,S), an element of GT
P	ECP8 instance, an element of G2
Q	ECP instance, an element of G1
R	ECP8 instance, an element of G2
S	ECP instance, an element of G1

5.32.2.6 PAIR_ZZZ_fexp()

```
void PAIR_ZZZ_fexp (
            FP48_YYY * x )
```

Final exponentiation of pairing, converts output of Miller loop to element in GT.

Here p is the internal modulus, and r is the group order

Parameters

x	FP48, on exit = x^((p^12-1)/r)

5.32.2.7 PAIR_ZZZ_G1mul()

```
void PAIR_ZZZ_G1mul (
            ECP_ZZZ * Q,
            BIG_XXX b )
```

Fast point multiplication of a member of the group G1 by a BIG number.

May exploit endomorphism for speed.

Parameters

Q	ECP member of G1.
b	BIG multiplier

5.32.2.8 PAIR_ZZZ_G2mul()

```
void PAIR_ZZZ_G2mul (
            ECP8_ZZZ * P,
            BIG_XXX b )
```

Fast point multiplication of a member of the group G2 by a BIG number.

May exploit endomorphism for speed.

Parameters

P	ECP8 member of G1.
b	BIG multiplier

5.32.2.9 PAIR_ZZZ_GTpow()

```
void PAIR_ZZZ_GTpow (
            FP48_YYY * x,
            BIG_XXX b )
```

Fast raising of a member of GT to a BIG power.

May exploit endomorphism for speed.

Parameters

x	FP48 member of GT.
b	BIG exponent

5.32.2.10 PAIR_ZZZ_G1member()

```
int PAIR_ZZZ_G1member (
            ECP_ZZZ * P )
```

Tests ECP for membership of G1.

Parameters

P	ECP member of G1

Returns

true or false

5.32.2.11 PAIR_ZZZ_G2member()

```
int PAIR_ZZZ_G2member (
            ECP8_ZZZ * P )
```

Tests ECP8 for membership of G2.

Parameters

P	ECP8 member of G2

Returns

true or false

5.32.2.12 PAIR_ZZZ_GTmember()

```
int PAIR_ZZZ_GTmember (
            FP48_YYY * x )
```

Tests FP48 for membership of GT.

Parameters

x	FP48 instance

Returns

1 if x is in GT, else return 0

5.32.2.13 PAIR_ZZZ_nbits()

```
int PAIR_ZZZ_nbits (
            BIG_XXX n3,
            BIG_XXX n )
```

Prepare Ate parameter.

Parameters

n	BIG parameter

Parameters

n3	BIG paramter = 3*n

Returns

number of nits in n3

5.32.2.14 PAIR_ZZZ_initmp()

```
void PAIR_ZZZ_initmp (
            FP48_YYY r[] )
```

Initialise structure for multi-pairing.

Parameters

r	FP48 array, to be initialised to 1

5.32.2.15 PAIR_ZZZ_miller()

```
void PAIR_ZZZ_miller (
            FP48_YYY * res,
            FP48_YYY r[] )
```

Miller loop.

Parameters

res	FP48 result
r	FP48 precomputed array of accumulated line functions

5.32.3 Variable Documentation

5.32.3.1 CURVE_Bnx_ZZZ

```
const BIG_XXX CURVE_Bnx_ZZZ [extern]
```

BN curve x parameter

5.32.3.2 CURVE_Cru_ZZZ

```
const BIG_XXX CURVE_Cru_ZZZ [extern]
```

BN curve Cube Root of Unity

5.32.3.3 CURVE_W_ZZZ

```
const BIG_XXX CURVE_W_ZZZ[2] [extern]
```

BN curve constant for GLV decomposition

5.32.3.4 CURVE_SB_ZZZ

```
const BIG_XXX CURVE_SB_ZZZ[2][2] [extern]
```

BN curve constant for GLV decomposition

5.32.3.5 CURVE_WB_ZZZ

```
const BIG_XXX CURVE_WB_ZZZ[4] [extern]
```

BN curve constant for GS decomposition

5.32.3.6 CURVE_BB_ZZZ

```
const BIG_XXX CURVE_BB_ZZZ[4][4] [extern]
```

BN curve constant for GS decomposition

5.33 randapi.h File Reference

PRNG API File.

```
#include "core.h"
```

Functions

- void CREATE_CSPRNG (csprng *R, octet *S)

    Initialise a random number generator.
- void KILL_CSPRNG (csprng *R)

    Kill a random number generator.

5.33.1 Detailed Description

PRNG API File.

Author

Mike Scott

5.33.2 Function Documentation

5.33.2.1 CREATE_CSPRNG()

```
void CREATE_CSPRNG (
            csprng * R,
            octet * S )
```

Initialise a random number generator.

Parameters

R	is a pointer to a cryptographically secure random number generator
S	is an input truly random seed value

5.33.2.2 KILL_CSPRNG()

```
void KILL_CSPRNG (
            csprng * R )
```

Kill a random number generator.
Deletes all internal state

Parameters

R	is a pointer to a cryptographically secure random number generator

5.34 rsa.h File Reference

RSA Header file for implementation of RSA protocol.

```
#include "ff_WWW.h"
```

Classes

- struct rsa_public_key_WWW

  Integer Factorisation Public Key.
- struct rsa_private_key_WWW

  Integer Factorisation Private Key.

Macros

- #define HASH_TYPE_RSA_WWW SHA256
- #define RFS_WWW MODBYTES_XXX*FFLEN_WWW

Functions

- void RSA_WWW_KEY_PAIR (csprng *R, sign32 e, rsa_private_key_WWW *PRIV, rsa_public_key_WWW *PUB, octet *P, octet *Q)

  RSA Key Pair Generator.
- void RSA_WWW_ENCRYPT (rsa_public_key_WWW *PUB, octet *F, octet *G)

  RSA encryption of suitably padded plaintext.
- void RSA_WWW_DECRYPT (rsa_private_key_WWW *PRIV, octet *G, octet *F)

  RSA decryption of ciphertext.
- void RSA_WWW_PRIVATE_KEY_KILL (rsa_private_key_WWW *PRIV)

  Destroy an RSA private Key.
- void RSA_WWW_fromOctet (BIG_XXX *x, octet *S)

  Populates an RSA public key from an octet string.

5.34.1 Detailed Description

RSA Header file for implementation of RSA protocol.

Author

Mike Scott

declares functions

5.34.2 Macro Definition Documentation

5.34.2.1 HASH_TYPE_RSA_WWW

```
#define HASH_TYPE_RSA_WWW SHA256
```

Chosen Hash algorithm

5.34.2.2 RFS_WWW

```
#define RFS_WWW MODBYTES_XXX*FFLEN_WWW
```

RSA Public Key Size in bytes

5.34.3 Function Documentation

5.34.3.1 RSA_WWW_KEY_PAIR()

```
void RSA_WWW_KEY_PAIR (
            csprng * R,
            sign32 e,
            rsa_private_key_WWW * PRIV,
            rsa_public_key_WWW * PUB,
            octet * P,
            octet * Q )
```

RSA Key Pair Generator.

Parameters

R	is a pointer to a cryptographically secure random number generator
e	the encryption exponent
PRIV	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

5.34.3.2 RSA_WWW_ENCRYPT()

```
void RSA_WWW_ENCRYPT (
            rsa_public_key_WWW * PUB,
            octet * F,
            octet * G )
```

RSA encryption of suitably padded plaintext.

Parameters

PUB	the input RSA public key
F	is input padded message
G	is the output ciphertext

5.34.3.3 RSA_WWW_DECRYPT()

```
void RSA_WWW_DECRYPT (
            rsa_private_key_WWW * PRIV,
            octet * G,
            octet * F )
```

RSA decryption of ciphertext.

Parameters

PRIV	the input RSA private key
G	is the input ciphertext
F	is output plaintext (requires unpadding)

5.34.3.4 RSA_WWW_PRIVATE_KEY_KILL()

```
void RSA_WWW_PRIVATE_KEY_KILL (
            rsa_private_key_WWW * PRIV )
```

Destroy an RSA private Key.

Parameters

PRIV	the input RSA private key. Destroyed on output.

5.34.3.5 RSA_WWW_fromOctet()

```
void RSA_WWW_fromOctet (
            BIG_XXX * x,
            octet * S )
```

Populates an RSA public key from an octet string.
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

5.35 x509.h File Reference

X509 function Header File.

Classes

- struct pktype

    Public key type.

Macros

- #define X509_ECC 1
- #define X509_RSA 2
- #define X509_H256 2
- #define X509_H384 3
- #define X509_H512 4
- #define USE_NIST256 0
- #define USE_C25519 1
- #define USE_BRAINPOOL 2
- #define USE_ANSSI 3
- #define USE_NIST384 10
- #define USE_NIST521 12

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_self_signed (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)
- int X509_find_extensions (octet ∗c)
- int X509_find_extension (octet ∗c, octet ∗S, int s, int ∗f)
- int X509_find_alt_name (octet ∗c, int s, char ∗name)

Variables

- octet X509_CN
- octet X509_ON
- octet X509_EN
- octet X509_LN
- octet X509_UN
- octet X509_MN
- octet X509_SN
- octet X509_AN
- octet X509_KU
- octet X509_BC

5.35.1 Detailed Description

X509 function Header File.

Author

Mike Scott

5.35.2 Macro Definition Documentation

5.35.2.1 X509_ECC

```
#define X509_ECC 1
```

Uses Elliptic Curve Cryptography

5.35.2.2 X509_RSA

```
#define X509_RSA 2
```

Uses RSA Cryptography

5.35.2.3 X509_H256

```
#define X509_H256 2
```

Using SHA256 hashing

5.35.2.4 X509_H384

```
#define X509_H384 3
```

Using SHA384 hashing

5.35.2.5 X509_H512

```
#define X509_H512 4
```

Using SHA512 hashing

5.35.2.6 USE_NIST256

```
#define USE_NIST256 0
```

For the NIST 256-bit standard curve - WEIERSTRASS only

5.35.2.7 USE_C25519

```
#define USE_C25519 1
```

Bernstein's Modulus 2^255-19 - EDWARDS or MONTGOMERY only

5.35.2.8 USE_BRAINPOOL

```
#define USE_BRAINPOOL 2
```

For Brainpool 256-bit curve - WEIERSTRASS only

5.35.2.9 USE_ANSSI

```
#define USE_ANSSI 3
```

For French 256-bit standard curve - WEIERSTRASS only

5.35.2.10 USE_NIST384

```
#define USE_NIST384 10
```

For the NIST 384-bit standard curve - WEIERSTRASS only

5.35.2.11 USE_NIST521

```
#define USE_NIST521 12
```

For the NIST 521-bit standard curve - WEIERSTRASS only

5.35.3 Function Documentation

5.35.3.1 X509_extract_cert_sig()

```
pktype X509_extract_cert_sig (
            octet * c,
            octet * s )
```

Extract certificate signature.

Parameters

c	an X.509 certificate
s	the extracted signature

Returns

0 on failure, or indicator of signature type (ECC or RSA)

5.35.3.2 X509_extract_cert()

```
int X509_extract_cert (
```

```
            octet * sc,
            octet * c )
```

Parameters

sc	a signed certificate
c	the extracted certificate

Returns

0 on failure

5.35.3.3 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)

5.35.3.4 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

5.35.3.5 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

5.35.3.6 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

5.35.3.7 X509_self_signed()

```
int X509_self_signed (
            octet * c )
```

Parameters

c	an X.509 certificate

Returns

true if self-signed, else false

5.35.3.8 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

5.35.3.9 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

5.35.3.10 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

5.35.3.11 X509_find_extensions()

```
int X509_find_extensions (
            octet * c )
```

Parameters

c	an X.509 certificate

Returns

0 on failure (or no extensions), or pointer to extensions field in cert

5.35.3.12 X509_find_extension()

```
int X509_find_extension (
            octet * c,
            octet * S,
            int s,
            int * f )
```

Parameters

c	an X.509 certificate
S	is OID of particular extension we are looking for
s	is a pointer to the section of interest in the cert

Parameters

f	is pointer to the length of the extension

Returns

0 on failure, or pointer to the extension

5.35.3.13 X509_find_alt_name()

```
int X509_find_alt_name (
            octet * c,
            int s,
            char * name )
```

Parameters

c	an X.509 certificate
s	is a pointer to certificate extension SubjectAltNames
name	is a URL

Returns

0 on failure, 1 if URL is in list of alt names

5.35.4 Variable Documentation

5.35.4.1 X509_CN

`octet X509_CN [extern]`

Country Name

5.35.4.2 X509_ON

`octet X509_ON [extern]`

Organisation Name

5.35.4.3 X509_EN

`octet X509_EN [extern]`

Email

5.35.4.4 X509_LN

`octet X509_LN [extern]`

Local Name

5.35.4.5 X509_UN

`octet X509_UN [extern]`

Unit Name

5.35.4.6 X509_MN

`octet X509_MN [extern]`

My name

5.35.4.7 X509_SN

`octet X509_SN [extern]`

State Name

5.35.4.8 X509_AN

`octet X509_AN [extern]`

Alternate Name

5.35.4.9 X509_KU

`octet X509_KU [extern]`

Key Usage

5.35.4.10 X509_BC

`octet X509_BC [extern]`

Basic Constraints

Index