

FIWE

1.0

Generated by Doxygen 1.8.18



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 Aff_Point_s Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 x	5
3.1.2.2 y	5
3.2 Montg_Curve_s Struct Reference	6
3.2.1 Detailed Description	6
3.2.2 Member Data Documentation	6
3.2.2.1 A	6
3.2.2.2 B	6
3.2.2.3 n	6
3.3 Pro_Point_s Struct Reference	7
3.3.1 Detailed Description	7
3.3.2 Member Data Documentation	7
3.3.2.1 X	7
3.3.2.2 Y	7
3.3.2.3 Z	7
<b>4 File Documentation</b>	<b>9</b>
4.1 ecm.c File Reference	9
4.1.1 Detailed Description	9
4.1.2 Function Documentation	9
4.1.2.1 ecm()	9
4.2 ecm.h File Reference	10
4.2.1 Detailed Description	10
4.2.2 Macro Definition Documentation	10
4.2.2.1 B_THRESHOLD	10
4.2.2.2 CRV_THRESHOLD	11
4.2.3 Function Documentation	11
4.2.3.1 ecm()	11
4.3 main.c File Reference	11
4.3.1 Function Documentation	12
4.3.1.1 aff_curve_point_test()	12
4.3.1.2 ecm_test()	12
4.3.1.3 main()	12
4.3.1.4 pro_add_gmp_test()	12

4.3.1.5 pro_add_magma_test()	12
4.3.1.6 pro_curve_point_test()	12
4.3.1.7 pro_dbl_magma_test()	13
4.3.1.8 pro_ladder_gmp_test()	13
4.3.1.9 pro_ladder_magma_test()	13
4.4 montgomery.c File Reference	13
4.4.1 Detailed Description	14
4.4.2 Function Documentation	14
4.4.2.1 aff_add()	14
4.4.2.2 aff_curve_point()	14
4.4.2.3 aff_dbl()	15
4.4.2.4 aff_is_on_curve()	15
4.4.2.5 aff_ladder()	15
4.4.2.6 pro_add()	16
4.4.2.7 pro_curve_point()	16
4.4.2.8 pro_dbl()	17
4.4.2.9 pro_is_on_curve()	17
4.4.2.10 pro_ladder()	17
4.5 montgomery.h File Reference	18
4.5.1 Detailed Description	19
4.5.2 Typedef Documentation	19
4.5.2.1 AFF_POINT	19
4.5.2.2 AFF_POINT_t	19
4.5.2.3 MONTG_CURVE	19
4.5.2.4 MONTG_CURVE_t	20
4.5.2.5 PRO_POINT	20
4.5.2.6 PRO_POINT_t	20
4.5.3 Function Documentation	20
4.5.3.1 aff_add()	20
4.5.3.2 aff_curve_point()	21
4.5.3.3 aff_dbl()	21
4.5.3.4 aff_is_on_curve()	21
4.5.3.5 aff_ladder()	22
4.5.3.6 pro_add()	22
4.5.3.7 pro_curve_point()	22
4.5.3.8 pro_dbl()	23
4.5.3.9 pro_is_on_curve()	23
4.5.3.10 pro_ladder()	24
4.6 mplib.c File Reference	24
4.6.1 Detailed Description	25
4.6.2 Function Documentation	25
4.6.2.1 barret_reduction()	25

4.6.2.2 barret_reduction_UL()	26
4.6.2.3 big_add()	26
4.6.2.4 big_gcd()	26
4.6.2.5 big_get_A24()	27
4.6.2.6 big_get_mu()	27
4.6.2.7 big_invert()	27
4.6.2.8 big_is_equal()	28
4.6.2.9 big_is_equal_ui()	28
4.6.2.10 big_mod_add()	28
4.6.2.11 big_mod_mul()	29
4.6.2.12 big_mod_rand()	30
4.6.2.13 big_mod_sub()	30
4.6.2.14 big_mul()	31
4.6.2.15 big_print()	31
4.6.2.16 big_rand()	31
4.6.2.17 big_sub()	32
4.7 mplib.h File Reference	32
4.7.1 Detailed Description	33
4.7.2 Macro Definition Documentation	34
4.7.2.1 big_cpy	34
4.7.2.2 W	34
4.7.3 Typedef Documentation	34
4.7.3.1 ui	34
4.7.3.2 ui_t	35
4.7.3.3 uni	35
4.7.3.4 uni_t	35
4.7.4 Function Documentation	35
4.7.4.1 barret_reduction()	35
4.7.4.2 barret_reduction_UL()	36
4.7.4.3 big_add()	36
4.7.4.4 big_gcd()	36
4.7.4.5 big_get_A24()	37
4.7.4.6 big_get_mu()	37
4.7.4.7 big_invert()	37
4.7.4.8 big_is_equal()	38
4.7.4.9 big_is_equal_ui()	38
4.7.4.10 big_mod_add()	38
4.7.4.11 big_mod_mul()	39
4.7.4.12 big_mod_rand()	40
4.7.4.13 big_mod_sub()	40
4.7.4.14 big_mul()	41
4.7.4.15 big_print()	41

4.7.4.16 <a href="#">big_rand()</a> . . . . .	41
4.7.4.17 <a href="#">big_sub()</a> . . . . .	42

<b><a href="#">Index</a></b>	<b>43</b>
------------------------------	-----------

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">Aff_Point_s</a>	Structure to represent an affine point . . . . .	5
<a href="#">Montg_Curve_s</a>	Structure to represent a Montgomery curve . . . . .	6
<a href="#">Pro_Point_s</a>	Structure to represent a projective point . . . . .	7





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">ecm.c</a>	Implementation of <a href="#">ecm.h</a> library . . . . .	9
<a href="#">ecm.h</a>	A library to implement Elliptic Curve Method to factorize a composite number . . . . .	10
<a href="#">main.c</a>	. . . . .	11
<a href="#">montgomery.c</a>	Implementation of <a href="#">montgomery.h</a> library . . . . .	13
<a href="#">montgomery.h</a>	A library to represent Montgomery curves and points on them, and to do arithmetic with that points . . . . .	18
<a href="#">mplib.c</a>	Implementation of <a href="#">mplib.h</a> library . . . . .	24
<a href="#">mplib.h</a>	A library to represent multi-precision integers and do arithmetic on them . . . . .	32



## Chapter 3

# Class Documentation

### 3.1 Aff\_Point\_s Struct Reference

Structure to represent an affine point.

```
#include <montgomery.h>
```

#### Public Attributes

- [ui x](#)
- [ui y](#)

#### 3.1.1 Detailed Description

Structure to represent an affine point.

#### 3.1.2 Member Data Documentation

##### 3.1.2.1 x

```
ui Aff_Point_s::x
```

x coordinate of the point

##### 3.1.2.2 y

```
ui Aff_Point_s::y
```

y coordinate of the point

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

- [montgomery.h](#)

## 3.2 Montg\_Curve\_s Struct Reference

Structure to represent a Montgomery curve.

```
#include <montgomery.h>
```

### Public Attributes

- [ui A](#)
- [ui B](#)
- [ui n](#)

### 3.2.1 Detailed Description

Structure to represent a Montgomery curve.

Curve Equations:

- Affine Coordinates:  $B*y^2 = x^3 + A*x^2 + x$
- Projective Coordinates:  $B*y^2*Z = x^3 + A*x^2*Z + x*Z^2$

### 3.2.2 Member Data Documentation

#### 3.2.2.1 A

```
ui Montg_Curve_s::A
```

Coefficient A in the equation

#### 3.2.2.2 B

```
ui Montg_Curve_s::B
```

Coefficient B in the equation

#### 3.2.2.3 n

```
ui Montg_Curve_s::n
```

Modular base of the curve

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

- [montgomery.h](#)

## 3.3 Pro\_Point\_s Struct Reference

Structure to represent a projective point.

```
#include <montgomery.h>
```

### Public Attributes

- [ui X](#)
- [ui Y](#)
- [ui Z](#)

### 3.3.1 Detailed Description

Structure to represent a projective point.

### 3.3.2 Member Data Documentation

#### 3.3.2.1 X

```
ui Pro_Point_s::X
```

X coordinate of the point

#### 3.3.2.2 Y

```
ui Pro_Point_s::Y
```

Y coordinate of the point

#### 3.3.2.3 Z

```
ui Pro_Point_s::Z
```

Z coordinate of the point

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

- [montgomery.h](#)



## Chapter 4

# File Documentation

### 4.1 ecm.c File Reference

Implementation of [ecm.h](#) library.

```
#include <gmp.h>
#include <stdlib.h>
#include <time.h>
#include "montgomery.h"
#include "mplib.h"
#include "ecm.h"
```

#### Functions

- int [ecm](#) (ui d, ui n, ui\_t nl)  
*Factorizes the given composite using Elliptic Curve Method.*

#### 4.1.1 Detailed Description

Implementation of [ecm.h](#) library.

#### 4.1.2 Function Documentation

##### 4.1.2.1 ecm()

```
int ecm (
    ui d,
    ui n,
    ui_t nl )
```

Factorizes the given composite using Elliptic Curve Method.

**Parameters**

out	$d$	factor of $n$
in	$n$	number to be factorized
in	$nl$	number of digits of $n$ in base $2^W$

**Returns**

an integer, positive when ECM succeeds, 0 otherwise

## 4.2 ecm.h File Reference

A library to implement Elliptic Curve Method to factorize a composite number.

```
#include "mplib.h"
```

**Macros**

- `#define B_THRESHOLD 1000`  
*Number of different B values to try during the factorization.*
- `#define CRV_THRESHOLD 20`  
*Number of different curves to try during the factorization.*

**Functions**

- `int ecm(ui d, ui n, ui_t nl)`  
*Factorizes the given composite using Elliptic Curve Method.*

### 4.2.1 Detailed Description

A library to implement Elliptic Curve Method to factorize a composite number.

### 4.2.2 Macro Definition Documentation

#### 4.2.2.1 B\_THRESHOLD

```
#define B_THRESHOLD 1000
```

Number of different B values to try during the factorization.

If a B value does not produce a factor, another one is tried until the number of trials exceed B\_THRESHOLD.



### 4.2.2.2 CRV\_THRESHOLD

```
#define CRV_THRESHOLD 20
```

Number of different curves to try during the factorization.

If a curve does not produce a factor, another one is tried until the number of trials exceed CRV\_THRESHOLD.

## 4.2.3 Function Documentation

### 4.2.3.1 ecm()

```
int ecm (
    ui d,
    ui n,
    ui_t nl )
```

Factorizes the given composite using Elliptic Curve Method.

#### Parameters

out	<i>d</i>	factor of n
in	<i>n</i>	number to be factorized
in	<i>nl</i>	number of digits of n in base $2^W$

#### Returns

an integer, positive when ECM succeeds, 0 otw

## 4.3 main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>
#include "mplib.h"
#include "montgomery.h"
#include "ecm.h"
```

### Functions

- void [pro\\_curve\\_point\\_test](#) ()
- void [aff\\_curve\\_point\\_test](#) ()
- void [pro\\_add\\_gmp\\_test](#) ()
- void [pro\\_add\\_magma\\_test](#) ()

- void [pro\\_dbl\\_magma\\_test](#) ()
- void [pro\\_ladder\\_gmp\\_test](#) ()
- void [pro\\_ladder\\_magma\\_test](#) ()
- void [ecm\\_test](#) ()
- int [main](#) ()

## 4.3.1 Function Documentation

### 4.3.1.1 [aff\\_curve\\_point\\_test\(\)](#)

```
void aff_curve_point_test ( )
```

### 4.3.1.2 [ecm\\_test\(\)](#)

```
void ecm_test ( )
```

### 4.3.1.3 [main\(\)](#)

```
int main ( )
```

### 4.3.1.4 [pro\\_add\\_gmp\\_test\(\)](#)

```
void pro_add_gmp_test ( )
```

### 4.3.1.5 [pro\\_add\\_magma\\_test\(\)](#)

```
void pro_add_magma_test ( )
```

### 4.3.1.6 [pro\\_curve\\_point\\_test\(\)](#)

```
void pro_curve_point_test ( )
```

**4.3.1.7 pro\_dbl\_magma\_test()**

```
void pro_dbl_magma_test ( )
```

**4.3.1.8 pro\_ladder\_gmp\_test()**

```
void pro_ladder_gmp_test ( )
```

**4.3.1.9 pro\_ladder\_magma\_test()**

```
void pro_ladder_magma_test ( )
```

**4.4 montgomery.c File Reference**

Implementation of [montgomery.h](#) library.

```
#include <gmp.h>
#include <stdlib.h>
#include <time.h>
#include "montgomery.h"
#include "mplib.h"
```

**Functions**

- void [pro\\_curve\\_point](#) (ui d, [MONTG\\_CURVE](#) c, [PRO\\_POINT](#) p, ui n, [ui\\_t](#) nl, ui mu, [ui\\_t](#) mul, int \*flag)  
*Initializes a randomly generated Montgomery curve and a projective point on the curve.*
- void [aff\\_curve\\_point](#) (ui d, [MONTG\\_CURVE](#) c, [AFF\\_POINT](#) p, ui n, [ui\\_t](#) nl, ui mu, [ui\\_t](#) mul, int \*flag)  
*Initializes a randomly generated Montgomery curve and a projective point on the curve.*
- void [pro\\_add](#) ([PRO\\_POINT](#) p, [PRO\\_POINT](#) p1, [PRO\\_POINT](#) p2, [PRO\\_POINT](#) pd, ui n, [ui\\_t](#) nl, ui mu, [ui\\_t](#) mul)  
*Adds two projective points on a curve.*
- void [aff\\_add](#) ([AFF\\_POINT](#) p, [AFF\\_POINT](#) p1, [AFF\\_POINT](#) p2, ui A, ui B, ui n, [ui\\_t](#) nl, ui mu, [ui\\_t](#) mul)  
*Adds two affine points on a curve.*
- void [pro\\_dbl](#) ([PRO\\_POINT](#) p, [PRO\\_POINT](#) p1, ui A24, ui n, [ui\\_t](#) nl, ui mu, [ui\\_t](#) mul)  
*Doubles a projective point on a curve.*
- void [aff\\_dbl](#) (ui x, ui z, ui x1, ui y1, ui A, ui B, ui n)
- void [pro\\_ladder](#) ([PRO\\_POINT](#) p, [PRO\\_POINT](#) p1, ui A24, ui k, [ui\\_t](#) kl, ui n, [ui\\_t](#) nl, ui mu, [ui\\_t](#) mul)  
*Multiplies a projective point with a constant.*
- void [aff\\_ladder](#) (ui x, ui y, ui x1, ui y1, ui k, ui n)
- int [pro\\_is\\_on\\_curve](#) (ui A, ui B, ui X, ui Y, ui Z, ui n)
- int [aff\\_is\\_on\\_curve](#) (ui A, ui B, ui x, ui y, ui n)

### 4.4.1 Detailed Description

Implementation of [montgomery.h](#) library.

### 4.4.2 Function Documentation

#### 4.4.2.1 `aff_add()`

```
void aff_add (
    AFF_POINT p,
    AFF_POINT p1,
    AFF_POINT p2,
    ui A,
    ui B,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Adds two affine points on a curve.

##### Parameters

out	<i>p</i>	resulted projective point
in	<i>p1</i>	first operand of the addition
in	<i>p2</i>	second operand of the addition
in	<i>A</i>	coefficient A of the curve
in	<i>B</i>	coefficient B of the curve
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$

#### 4.4.2.2 `aff_curve_point()`

```
void aff_curve_point (
    ui d,
    MONTG_CURVE c,
    AFF_POINT p,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul,
    int * flag )
```

Initializes a randomly generated Montgomery curve and a projective point on the curve.

## Parameters

out	<i>d</i>	factor of n, that may be found while generating the curve
out	<i>c</i>	curve to be initialized
out	<i>p</i>	affine point to be initialized
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$
in	<i>flag</i>	0 when factor found, 1 when curve and point generated, -1 when function failed due to singular curve generation

## 4.4.2.3 aff\_dbl()

```
void aff_dbl (
    ui x,
    ui z,
    ui x1,
    ui y1,
    ui A,
    ui B,
    ui n )
```

## 4.4.2.4 aff\_is\_on\_curve()

```
int aff_is_on_curve (
    ui A,
    ui B,
    ui x,
    ui y,
    ui n )
```

## 4.4.2.5 aff\_ladder()

```
void aff_ladder (
    ui x,
    ui y,
    ui x1,
    ui y1,
    ui k,
    ui n )
```

#### 4.4.2.6 pro\_add()

```
void pro_add (
    PRO_POINT p,
    PRO_POINT p1,
    PRO_POINT p2,
    PRO_POINT pd,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Adds two projective points on a curve.

##### Parameters

out	<i>p</i>	resulted projective point
in	<i>p1</i>	first operand of the addition
in	<i>p2</i>	second operand of the addition
in	<i>pd</i>	differences of the first and second operands
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$

#### 4.4.2.7 pro\_curve\_point()

```
void pro_curve_point (
    ui d,
    MONTG_CURVE c,
    PRO_POINT p,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul,
    int * flag )
```

Initializes a randomly generated Montgomery curve and a projective point on the curve.

##### Parameters

out	<i>d</i>	factor of n, that may be found while generating the curve
out	<i>c</i>	curve to be initialized
out	<i>p</i>	projective point to be initialized
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$
in	<i>flag</i>	0 when factor found, 1 when curve and point generated, -1 when function failed due to singular curve generation

#### 4.4.2.8 pro\_dbl()

```
void pro_dbl (
    PRO_POINT p,
    PRO_POINT p1,
    ui A24,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Doubles a projective point on a curve.

##### Parameters

out	<i>p</i>	resulted projective point
in	<i>p1</i>	point to be doubled
in	<i>A24</i>	$(A + 2)/4$ where A is the coefficient of the curve
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$

#### 4.4.2.9 pro\_is\_on\_curve()

```
int pro_is_on_curve (
    ui A,
    ui B,
    ui X,
    ui Y,
    ui Z,
    ui n )
```

#### 4.4.2.10 pro\_ladder()

```
void pro_ladder (
    PRO_POINT p,
    PRO_POINT p1,
    ui A24,
    ui k,
    ui_t k1,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Multiplies a projective point with a constant.

**Parameters**

out	$p$	resulted projective point
in	$p1$	point to be multiplied
in	$A24$	$(A + 2)/4$ where A is the coefficient of the curve
in	$k$	constant to multiply with p1
in	$kl$	number of digits of k in base $2^W$
in	$n$	modular base of the curve that is going to be generated
in	$nl$	number of digits of n in base $2^W$
in	$mu$	precalculated value of $(2^W)^{2*nl}/n$
in	$mul$	number of digits of mu in base $2^W$

## 4.5 montgomery.h File Reference

A library to represent Montgomery curves and points on them, and to do arithmetic with that points.

```
#include "mplib.h"
```

**Classes**

- struct [Montg\\_Curve\\_s](#)  
*Structure to represent a Montgomery curve.*
- struct [Pro\\_Point\\_s](#)  
*Structure to represent a projective point.*
- struct [Aff\\_Point\\_s](#)  
*Structure to represent an affine point.*

**Typedefs**

- typedef struct [Montg\\_Curve\\_s](#) [MONTG\\_CURVE\\_t](#)[1]  
*Structure to represent a Montgomery curve.*
- typedef struct [Montg\\_Curve\\_s](#) \* [MONTG\\_CURVE](#)[1]
- typedef struct [Pro\\_Point\\_s](#) [PRO\\_POINT\\_t](#)[1]  
*Structure to represent a projective point.*
- typedef struct [Pro\\_Point\\_s](#) \* [PRO\\_POINT](#)[1]
- typedef struct [Aff\\_Point\\_s](#) [AFF\\_POINT\\_t](#)[1]  
*Structure to represent an affine point.*
- typedef struct [Aff\\_Point\\_s](#) \* [AFF\\_POINT](#)[1]



## Functions

- void `pro_curve_point` (ui d, MONTG\_CURVE c, PRO\_POINT p, ui n, ui\_t nl, ui mu, ui\_t mul, int \*flag)  
*Initializes a randomly generated Montgomery curve and a projective point on the curve.*
- void `aff_curve_point` (ui d, MONTG\_CURVE c, AFF\_POINT p, ui n, ui\_t nl, ui mu, ui\_t mul, int \*flag)  
*Initializes a randomly generated Montgomery curve and a projective point on the curve.*
- void `pro_add` (PRO\_POINT p, PRO\_POINT p1, PRO\_POINT p2, PRO\_POINT pd, ui n, ui\_t nl, ui mu, ui\_t mul)  
*Adds two projective points on a curve.*
- void `aff_add` (AFF\_POINT p, AFF\_POINT p1, AFF\_POINT p2, ui A, ui B, ui n, ui\_t nl, ui mu, ui\_t mul)  
*Adds two affine points on a curve.*
- void `pro_dbl` (PRO\_POINT p, PRO\_POINT p1, ui A24, ui n, ui\_t nl, ui mu, ui\_t mul)  
*Doubles a projective point on a curve.*
- void `aff_dbl` (ui x, ui z, ui x1, ui y1, ui A, ui B, ui n)
- void `pro_ladder` (PRO\_POINT p, PRO\_POINT p1, ui A24, ui k, ui\_t kl, ui n, ui\_t nl, ui mu, ui\_t mul)  
*Multiplies a projective point with a constant.*
- void `aff_ladder` (ui x, ui y, ui x1, ui y1, ui k, ui n)
- int `pro_is_on_curve` (ui A, ui B, ui X, ui Y, ui Z, ui n)
- int `aff_is_on_curve` (ui A, ui B, ui x, ui y, ui n)

### 4.5.1 Detailed Description

A library to represent Montgomery curves and points on them, and to do arithmetic with that points.

### 4.5.2 Typedef Documentation

#### 4.5.2.1 AFF\_POINT

```
typedef struct Aff_Point_s * AFF_POINT[1]
```

#### 4.5.2.2 AFF\_POINT\_t

```
typedef struct Aff_Point_s AFF_POINT_t[1]
```

Structure to represent an affine point.

#### 4.5.2.3 MONTG\_CURVE

```
typedef struct Montg_Curve_s * MONTG_CURVE[1]
```

#### 4.5.2.4 MONTG\_CURVE\_t

```
typedef struct Montg_Curve_s MONTG_CURVE_t[1]
```

Structure to represent a Montgomery curve.

Curve Equations:

- Affine Coordinates:  $B*y^2 = x^3 + A*x^2 + x$
- Projective Coordinates:  $B*y^2*Z = x^3 + A*x^2*Z + x*Z^2$

#### 4.5.2.5 PRO\_POINT

```
typedef struct Pro_Point_s * PRO_POINT[1]
```

#### 4.5.2.6 PRO\_POINT\_t

```
typedef struct Pro_Point_s PRO_POINT_t[1]
```

Structure to represent a projective point.

### 4.5.3 Function Documentation

#### 4.5.3.1 aff\_add()

```
void aff_add (
    AFF_POINT p,
    AFF_POINT p1,
    AFF_POINT p2,
    ui A,
    ui B,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Adds two affine points on a curve.

##### Parameters

out	<i>p</i>	resulted projective point
in	<i>p1</i>	first operand of the addition
in	<i>p2</i>	second operand of the addition
in	<i>A</i>	coefficient A of the curve
in	<i>B</i>	coefficient B of the curve
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of bits of the modular base
in	<i>mul</i>	number of bits of the modular base

### 4.5.3.2 aff\_curve\_point()

```
void aff_curve_point (
    ui d,
    MONTG_CURVE c,
    AFF_POINT p,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul,
    int * flag )
```

Initializes a randomly generated Montgomery curve and a projective point on the curve.

#### Parameters

out	<i>d</i>	factor of n, that may be found while generating the curve
out	<i>c</i>	curve to be initialized
out	<i>p</i>	affine point to be initialized
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$
in	<i>flag</i>	0 when factor found, 1 when curve and point generated, -1 when function failed due to singular curve generation

### 4.5.3.3 aff\_dbl()

```
void aff_dbl (
    ui x,
    ui z,
    ui x1,
    ui y1,
    ui A,
    ui B,
    ui n )
```

### 4.5.3.4 aff\_is\_on\_curve()

```
int aff_is_on_curve (
    ui A,
    ui B,
    ui x,
    ui y,
    ui n )
```

#### 4.5.3.5 aff\_ladder()

```
void aff_ladder (
    ui x,
    ui y,
    ui x1,
    ui y1,
    ui k,
    ui n )
```

#### 4.5.3.6 pro\_add()

```
void pro_add (
    PRO_POINT p,
    PRO_POINT p1,
    PRO_POINT p2,
    PRO_POINT pd,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Adds two projective points on a curve.

##### Parameters

out	<i>p</i>	resulted projective point
in	<i>p1</i>	first operand of the addition
in	<i>p2</i>	second operand of the addition
in	<i>pd</i>	differences of the first and second operands
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$

#### 4.5.3.7 pro\_curve\_point()

```
void pro_curve_point (
    ui d,
    MONTG_CURVE c,
    PRO_POINT p,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul,
    int * flag )
```

Initializes a randomly generated Montgomery curve and a projective point on the curve.

## Parameters

out	<i>d</i>	factor of n, that may be found while generating the curve
out	<i>c</i>	curve to be initialized
out	<i>p</i>	projective point to be initialized
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$
in	<i>flag</i>	0 when factor found, 1 when curve and point generated, -1 when function failed due to singular curve generation

## 4.5.3.8 pro\_dbl()

```
void pro_dbl (
    PRO_POINT p,
    PRO_POINT p1,
    ui A24,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Doubles a projective point on a curve.

## Parameters

out	<i>p</i>	resulted projective point
in	<i>p1</i>	point to be doubled
in	<i>A24</i>	$(A + 2)/4$ where A is the coefficient of the curve
in	<i>n</i>	modular base of the curve that is going to be generated
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$

## 4.5.3.9 pro\_is\_on\_curve()

```
int pro_is_on_curve (
    ui A,
    ui B,
    ui X,
    ui Y,
    ui Z,
    ui n )
```

#### 4.5.3.10 pro\_ladder()

```
void pro_ladder (
    PRO_POINT p,
    PRO_POINT p1,
    ui A24,
    ui k,
    ui_t kl,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Multiplies a projective point with a constant.

##### Parameters

out	$p$	resulted projective point
in	$p1$	point to be multiplied
in	$A24$	$(A + 2)/4$ where A is the coefficient of the curve
in	$k$	constant to multiply with p1
in	$kl$	number of digits of k in base $2^W$
in	$n$	modular base of the curve that is going to be generated
in	$nl$	number of digits of n in base $2^W$
in	$mu$	precalculated value of $(2^W)^{2*nl}/n$
in	$mul$	number of digits of mu in base $2^W$

## 4.6 mplib.c File Reference

Implementation of [mplib.h](#) library.

```
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>
#include "mplib.h"
```

### Functions

- void [big\\_rand](#) (ui z, ui\_t l)  
*Initializes z with a random multi-precision number.*
- void [big\\_mod\\_rand](#) (ui z, ui\_t l, ui n, ui\_t nl, ui mu, ui\_t mul)  
*Initializes z with a random multi-precision number mod n.*
- void [big\\_print](#) (FILE \*fp, ui a, ui\_t al, char \*s, char \*R)  
*Prints the given multi-precision number in Magma assignment format.*
- void [big\\_is\\_equal](#) (int \*z, ui a, ui b, ui\_t l)  
*Checks if two multi-precision numbers are equal.*
- void [big\\_is\\_equal\\_ui](#) (int \*z, ui a, ui\_t al, ui\_t b)

- Checks if a multi-precision number is equal to given unsigned integer.
- void `big_add` (`ui z`, `ui a`, `ui_t al`, `ui b`, `ui_t bl`)  
Adds two multi-precision numbers.
- void `big_mod_add` (`ui z`, `ui a`, `ui_t al`, `ui b`, `ui_t bl`, `ui n`, `ui_t nl`)  
Adds two multi-precision numbers in mod  $n$ .
- void `big_sub` (`ui z`, `int *d`, `ui a`, `ui_t al`, `ui b`, `ui_t bl`)  
Subtracts two multi-precision numbers.
- void `big_mod_sub` (`ui z`, `ui a`, `ui_t al`, `ui b`, `ui_t bl`, `ui n`, `ui_t nl`)  
Subtracts two multi-precision numbers in mod  $n$ .
- void `big_mul` (`ui z`, `ui a`, `ui_t al`, `ui b`, `ui_t bl`)  
Multiplies two multi-precision numbers.
- void `big_mod_mul` (`ui z`, `ui a`, `ui_t al`, `ui b`, `ui_t bl`, `ui n`, `ui_t nl`, `ui mu`, `ui_t mul`)  
Multiplies two multi-precision numbers in mod  $n$ .
- void `big_get_mu` (`ui mu`, `ui n`, `ui_t nl`)  
Calculates  $(2^W)^{2*nl}/n$ .
- void `big_get_A24` (`ui z`, `ui A`, `ui n`, `ui_t nl`, `ui mu`, `ui_t mul`, `int *flag`)  
Calculates  $(A + 2)/4$ .
- `ui_t barret_reduction_UL` (`ui_t p`, `ui_t b`, `ui_t k`, `ui_t z`, `ui_t m`, `ui_t L`)
- void `barret_reduction` (`ui z`, `ui m`, `ui_t ml`, `ui n`, `ui_t nl`, `ui mu`, `ui_t mul`)  
Calculates  $m \bmod n$ .
- void `big_gcd` (`ui d`, `ui_t dl`, `ui a`, `ui_t al`, `ui b`, `ui_t bl`)
- int `big_invert` (`ui z`, `ui a`, `ui_t al`, `ui b`, `ui_t bl`)

### 4.6.1 Detailed Description

Implementation of `mplib.h` library.

### 4.6.2 Function Documentation

#### 4.6.2.1 `barret_reduction()`

```
void barret_reduction (
    ui z,
    ui m,
    ui_t ml,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Calculates  $m \bmod n$ .

#### Parameters

out	<code>z</code>	result of the reduction
in	<code>m</code>	number to be reduced
in	<code>ml</code>	number of digits of $m$ in base $2^W$
in	<code>n</code>	modular base for the reduction
Generated by Doxygen	<code>nl</code>	number of digits of $n$ in base $2^W$
in	<code>mu</code>	precalculated value of $(2^W)^{2*nl}/n$
in	<code>mul</code>	number of digits of $mu$ in base $2^W$

#### 4.6.2.2 barret\_reduction\_UL()

```

uni_t barret_reduction_UL (
    uni_t p,
    uni_t b,
    uni_t k,
    uni_t z,
    uni_t m,
    uni_t L )

```

#### 4.6.2.3 big\_add()

```

void big_add (
    ui z,
    ui a,
    ui_t al,
    ui b,
    ui_t bl )

```

Adds two multi-precision numbers.

##### Parameters

out	<i>z</i>	result of the addition
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$

#### 4.6.2.4 big\_gcd()

```

void big_gcd (
    ui d,
    ui_t dl,
    ui a,
    ui_t al,
    ui b,
    ui_t bl )

```



#### 4.6.2.5 big\_get\_A24()

```
void big_get_A24 (
    ui A24,
    ui A,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul,
    int * flag )
```

Calculates  $(A + 2)/4$ .

##### Parameters

out	<i>A24</i>	result of $(A + 2)/4$ or a factor of n
in	<i>A</i>	A in the equation
in	<i>n</i>	n modular base for the calculation
in	<i>number</i>	of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$
in	<i>flag</i>	1 when calculation succeeds, 0 when factor gets found

#### 4.6.2.6 big\_get\_mu()

```
void big_get_mu (
    ui z,
    ui n,
    ui_t nl )
```

Calculates  $(2^W)^{2*nl}/n$ .

##### Parameters

out	<i>z</i>	result of the calculation
in	<i>n</i>	n in the equation
in	<i>nl</i>	number of digits of n in base $2^W$

#### 4.6.2.7 big\_invert()

```
int big_invert (
    ui z,
    ui a,
    ui_t al,
```

```

    ui b,
    ui_t bl )

```

#### 4.6.2.8 big\_is\_equal()

```

void big_is_equal (
    int * z,
    ui a,
    ui b,
    ui_t l )

```

Checks if two multi-precision numbers are equal.

##### Parameters

out	<i>z</i>	1 if equal, 0 otw
in	<i>a</i>	first number
in	<i>b</i>	second number
in	<i>l</i>	number of digits of a and b in base $2^W$

#### 4.6.2.9 big\_is\_equal\_ui()

```

void big_is_equal_ui (
    int * z,
    ui a,
    ui_t al,
    ui_t b )

```

Checks if a multi-precision number is equal to given unsigned integer.

##### Parameters

out	<i>z</i>	1 if equal, 0 otw
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	unsigned int to be compared

#### 4.6.2.10 big\_mod\_add()

```

void big_mod_add (
    ui z,
    ui a,

```

```

    ui_t al,
    ui b,
    ui_t bl,
    ui n,
    ui_t nl )

```

Adds two multi-precision numbers in mod n.

#### Parameters

out	<i>z</i>	result of the addition
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$
in	<i>n</i>	modular base for the addition
in	<i>nl</i>	number of digits of n in base $2^W$

#### 4.6.2.11 big\_mod\_mul()

```

void big_mod_mul (
    ui z,
    ui a,
    ui_t al,
    ui b,
    ui_t bl,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )

```

Multiplies two multi-precision numbers in mod n.

#### Parameters

out	<i>z</i>	result of the multiplication
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$
in	<i>n</i>	modular base for the multiplication
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$

#### 4.6.2.12 big\_mod\_rand()

```
void big_mod_rand (
    ui z,
    ui_t l,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Initializes z with a random multi-precision number mod n.

##### Parameters

out	<i>z</i>	multi-precision number to be initialized
in	<i>l</i>	number of digits of z in base $2^W$
in	<i>n</i>	modular base for z
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$

#### 4.6.2.13 big\_mod\_sub()

```
void big_mod_sub (
    ui z,
    ui a,
    ui_t al,
    ui b,
    ui_t bl,
    ui n,
    ui_t nl )
```

Subtracts two multi-precision numbers in mod n.

##### Parameters

out	<i>z</i>	result of the subtraction
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$
in	<i>n</i>	modular base for the subtraction
in	<i>nl</i>	number of digits of n in base $2^W$

**4.6.2.14 big\_mul()**

```
void big_mul (
    ui z,
    ui a,
    ui_t al,
    ui b,
    ui_t bl )
```

Multiplies two multi-precision numbers.

**Parameters**

out	<i>z</i>	result of the multiplication
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$

**4.6.2.15 big\_print()**

```
void big_print (
    FILE * fp,
    ui a,
    ui_t al,
    char * s,
    char * R )
```

Prints the given multi-precision number in Magma assignment format.

**Parameters**

in	<i>fp</i>	pointer to the file to print
in	<i>a</i>	multi-precision number to be printed
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>s</i>	name of the variable going to be assigned to a
in	<i>R</i>	name of the ring that a is going to be defined in (optional)

**4.6.2.16 big\_rand()**

```
void big_rand (
    ui z,
    ui_t l )
```

Initializes z with a random multi-precision number.

**Parameters**

out	<i>z</i>	multi-precision number to be initialized
in	<i>l</i>	number of digits of <i>z</i> in base $2^W$

**4.6.2.17 big\_sub()**

```
void big_sub (
    ui z,
    int * d,
    ui a,
    ui_t al,
    ui b,
    ui_t bl )
```

Subtracts two multi-precision numbers.

**Parameters**

out	<i>z</i>	result of the subtraction
in	<i>a</i>	first number
in	<i>al</i>	number of digits of <i>a</i> in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of <i>b</i> in base $2^W$

**4.7 mplib.h File Reference**

A library to represent multi-precision integers and do arithmetic on them.

```
#include <stdio.h>
#include <stdlib.h>
```

**Macros**

- #define *W* 32  
*Size of a digit of a multi-precision integer.*
- #define *big\_cpy*(*z*, *a*, *start*, *end*)  
*Copies end — start elements from a to z.*

## Typedefs

- typedef unsigned long \* [uni](#)  
*Type definition for unsigned long pointer.*
- typedef unsigned long [uni\\_t](#)  
*Type definition for unsigned long.*
- typedef unsigned int \* [ui](#)  
*Type definition for unsigned integer pointer.*
- typedef unsigned int [ui\\_t](#)  
*Type definition for unsigned integer.*

## Functions

- void [big\\_rand](#) ([ui](#) z, [ui\\_t](#) l)  
*Initializes z with a random multi-precision number.*
- void [big\\_mod\\_rand](#) ([ui](#) z, [ui\\_t](#) l, [ui](#) n, [ui\\_t](#) nl, [ui](#) mu, [ui\\_t](#) mul)  
*Initializes z with a random multi-precision number mod n.*
- void [big\\_print](#) (FILE \*fp, [ui](#) a, [ui\\_t](#) al, char \*s, char \*R)  
*Prints the given multi-precision number in Magma assignment format.*
- void [big\\_is\\_equal](#) (int \*z, [ui](#) a, [ui](#) b, [ui\\_t](#) l)  
*Checks if two multi-precision numbers are equal.*
- void [big\\_is\\_equal\\_ui](#) (int \*z, [ui](#) a, [ui\\_t](#) al, [ui\\_t](#) b)  
*Checks if a multi-precision number is equal to given unsigned integer.*
- void [big\\_add](#) ([ui](#) z, [ui](#) a, [ui\\_t](#) al, [ui](#) b, [ui\\_t](#) bl)  
*Adds two multi-precision numbers.*
- void [big\\_mod\\_add](#) ([ui](#) z, [ui](#) a, [ui\\_t](#) al, [ui](#) b, [ui\\_t](#) bl, [ui](#) n, [ui\\_t](#) nl)  
*Adds two multi-precision numbers in mod n.*
- void [big\\_sub](#) ([ui](#) z, int \*d, [ui](#) a, [ui\\_t](#) al, [ui](#) b, [ui\\_t](#) bl)  
*Subtracts two multi-precision numbers.*
- void [big\\_mod\\_sub](#) ([ui](#) z, [ui](#) a, [ui\\_t](#) al, [ui](#) b, [ui\\_t](#) bl, [ui](#) n, [ui\\_t](#) nl)  
*Subtracts two multi-precision numbers in mod n.*
- void [big\\_mul](#) ([ui](#) z, [ui](#) a, [ui\\_t](#) al, [ui](#) b, [ui\\_t](#) bl)  
*Multiplies two multi-precision numbers.*
- void [big\\_mod\\_mul](#) ([ui](#) z, [ui](#) a, [ui\\_t](#) al, [ui](#) b, [ui\\_t](#) bl, [ui](#) n, [ui\\_t](#) nl, [ui](#) mu, [ui\\_t](#) mul)  
*Multiplies two multi-precision numbers in mod n.*
- void [big\\_get\\_mu](#) ([ui](#) z, [ui](#) n, [ui\\_t](#) nl)  
*Calculates  $(2^W)^{2*nl}/n$ .*
- void [big\\_get\\_A24](#) ([ui](#) A24, [ui](#) A, [ui](#) n, [ui\\_t](#) nl, [ui](#) mu, [ui\\_t](#) mul, int \*flag)  
*Calculates  $(A + 2)/4$ .*
- [ui\\_t](#) [barret\\_reduction\\_UL](#) ([ui\\_t](#) p, [ui\\_t](#) b, [ui\\_t](#) k, [ui\\_t](#) z, [ui\\_t](#) m, [ui\\_t](#) L)
- void [barret\\_reduction](#) ([ui](#) z, [ui](#) m, [ui\\_t](#) ml, [ui](#) n, [ui\\_t](#) nl, [ui](#) mu, [ui\\_t](#) mul)  
*Calculates m mod n.*
- void [big\\_gcd](#) ([ui](#) d, [ui\\_t](#) dl, [ui](#) a, [ui\\_t](#) al, [ui](#) b, [ui\\_t](#) bl)
- int [big\\_invert](#) ([ui](#) z, [ui](#) a, [ui\\_t](#) al, [ui](#) b, [ui\\_t](#) bl)

### 4.7.1 Detailed Description

A library to represent multi-precision integers and do arithmetic on them.

## 4.7.2 Macro Definition Documentation

### 4.7.2.1 big\_cpy

```
#define big_cpy(
    z,
    a,
    start,
    end )
```

**Value:**

```
if(1) { \
    int i, j; \
    for(i = 0, j = (start); i < (end); i++, j++) { \
        z[i] = a[j]; \
    } \
};
```

Copies  $end - start$  elements from a to z.

**Parameters**

out	z	destination of the copy operation
in	a	source of the copy operation
in	start	starting index for the copy operation
in	end	ending index for the copy operation

### 4.7.2.2 W

```
#define W 32
```

Size of a digit of a multi-precision integer.

The numbers are represented in base  $2^W$  such that a digit of the number cannot exceed  $2^W$ .

## 4.7.3 Typedef Documentation

### 4.7.3.1 ui

```
typedef unsigned int* ui
```

Type definition for unsigned integer pointer.



### 4.7.3.2 ui\_t

```
typedef unsigned int ui_t
```

Type definition for unsigned integer.

### 4.7.3.3 uni

```
typedef unsigned long* uni
```

Type definition for unsigned long pointer.

### 4.7.3.4 uni\_t

```
typedef unsigned long uni_t
```

Type definition for unsigned long.

## 4.7.4 Function Documentation

### 4.7.4.1 barret\_reduction()

```
void barret_reduction (
    ui z,
    ui m,
    ui_t ml,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Calculates  $m \bmod n$ .

#### Parameters

out	$z$	result of the reduction
in	$m$	number to be reduced
in	$ml$	number of digits of $m$ in base $2^W$
in	$n$	modular base for the reduction
in	$nl$	number of digits of $n$ in base $2^W$
in	$mu$	precalculated value of $(2^W)^{2*nl}/n$
in	$mul$	number of digits of $mu$ in base $2^W$

#### 4.7.4.2 barret\_reduction\_UL()

```
uni_t barret_reduction_UL (
    uni_t p,
    uni_t b,
    uni_t k,
    uni_t z,
    uni_t m,
    uni_t L )
```

#### 4.7.4.3 big\_add()

```
void big_add (
    ui z,
    ui a,
    ui_t al,
    ui b,
    ui_t bl )
```

Adds two multi-precision numbers.

##### Parameters

out	<i>z</i>	result of the addition
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$

#### 4.7.4.4 big\_gcd()

```
void big_gcd (
    ui d,
    ui_t dl,
    ui a,
    ui_t al,
    ui b,
    ui_t bl )
```

#### 4.7.4.5 big\_get\_A24()

```
void big_get_A24 (
    ui A24,
    ui A,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul,
    int * flag )
```

Calculates  $(A + 2)/4$ .

##### Parameters

out	<i>A24</i>	result of $(A + 2)/4$ or a factor of n
in	<i>A</i>	A in the equation
in	<i>n</i>	n modular base for the calculation
in	<i>number</i>	of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$
in	<i>flag</i>	1 when calculation succeeds, 0 when factor gets found

#### 4.7.4.6 big\_get\_mu()

```
void big_get_mu (
    ui z,
    ui n,
    ui_t nl )
```

Calculates  $(2^W)^{2*nl}/n$ .

##### Parameters

out	<i>z</i>	result of the calculation
in	<i>n</i>	n in the equation
in	<i>nl</i>	number of digits of n in base $2^W$

#### 4.7.4.7 big\_invert()

```
int big_invert (
    ui z,
    ui a,
    ui_t al,
```

```

    ui b,
    ui_t bl )

```

#### 4.7.4.8 big\_is\_equal()

```

void big_is_equal (
    int * z,
    ui a,
    ui b,
    ui_t l )

```

Checks if two multi-precision numbers are equal.

##### Parameters

out	<i>z</i>	1 if equal, 0 otw
in	<i>a</i>	first number
in	<i>b</i>	second number
in	<i>l</i>	number of digits of a and b in base $2^W$

#### 4.7.4.9 big\_is\_equal\_ui()

```

void big_is_equal_ui (
    int * z,
    ui a,
    ui_t al,
    ui_t b )

```

Checks if a multi-precision number is equal to given unsigned integer.

##### Parameters

out	<i>z</i>	1 if equal, 0 otw
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	unsigned int to be compared

#### 4.7.4.10 big\_mod\_add()

```

void big_mod_add (
    ui z,
    ui a,

```

```

    ui_t al,
    ui b,
    ui_t bl,
    ui n,
    ui_t nl )

```

Adds two multi-precision numbers in mod n.

#### Parameters

out	<i>z</i>	result of the addition
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$
in	<i>n</i>	modular base for the addition
in	<i>nl</i>	number of digits of n in base $2^W$

#### 4.7.4.11 big\_mod\_mul()

```

void big_mod_mul (
    ui z,
    ui a,
    ui_t al,
    ui b,
    ui_t bl,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )

```

Multiplies two multi-precision numbers in mod n.

#### Parameters

out	<i>z</i>	result of the multiplication
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$
in	<i>n</i>	modular base for the multiplication
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$

#### 4.7.4.12 big\_mod\_rand()

```
void big_mod_rand (
    ui z,
    ui_t l,
    ui n,
    ui_t nl,
    ui mu,
    ui_t mul )
```

Initializes z with a random multi-precision number mod n.

##### Parameters

out	<i>z</i>	multi-precision number to be initialized
in	<i>l</i>	number of digits of z in base $2^W$
in	<i>n</i>	modular base for z
in	<i>nl</i>	number of digits of n in base $2^W$
in	<i>mu</i>	precalculated value of $(2^W)^{2*nl}/n$
in	<i>mul</i>	number of digits of mu in base $2^W$

#### 4.7.4.13 big\_mod\_sub()

```
void big_mod_sub (
    ui z,
    ui a,
    ui_t al,
    ui b,
    ui_t bl,
    ui n,
    ui_t nl )
```

Subtracts two multi-precision numbers in mod n.

##### Parameters

out	<i>z</i>	result of the subtraction
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$
in	<i>n</i>	modular base for the subtraction
in	<i>nl</i>	number of digits of n in base $2^W$

**4.7.4.14 big\_mul()**

```
void big_mul (
    ui z,
    ui a,
    ui_t al,
    ui b,
    ui_t bl )
```

Multiplies two multi-precision numbers.

**Parameters**

out	<i>z</i>	result of the multiplication
in	<i>a</i>	first number
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of b in base $2^W$

**4.7.4.15 big\_print()**

```
void big_print (
    FILE * fp,
    ui a,
    ui_t al,
    char * s,
    char * R )
```

Prints the given multi-precision number in Magma assignment format.

**Parameters**

in	<i>fp</i>	pointer to the file to print
in	<i>a</i>	multi-precision number to be printed
in	<i>al</i>	number of digits of a in base $2^W$
in	<i>s</i>	name of the variable going to be assigned to a
in	<i>R</i>	name of the ring that a is going to be defined in (optional)

**4.7.4.16 big\_rand()**

```
void big_rand (
    ui z,
    ui_t l )
```

Initializes z with a random multi-precision number.

**Parameters**

out	<i>z</i>	multi-precision number to be initialized
in	<i>l</i>	number of digits of <i>z</i> in base $2^W$

**4.7.4.17 big\_sub()**

```
void big_sub (
    ui z,
    int * d,
    ui a,
    ui_t al,
    ui b,
    ui_t bl )
```

Subtracts two multi-precision numbers.

**Parameters**

out	<i>z</i>	result of the subtraction
in	<i>a</i>	first number
in	<i>al</i>	number of digits of <i>a</i> in base $2^W$
in	<i>b</i>	second number
in	<i>bl</i>	number of digits of <i>b</i> in base $2^W$



# Index

## A

- Montg\_Curve\_s, 6
- aff\_add
  - montgomery.c, 14
  - montgomery.h, 20
- aff\_curve\_point
  - montgomery.c, 14
  - montgomery.h, 21
- aff\_curve\_point\_test
  - main.c, 12
- aff\_dbl
  - montgomery.c, 15
  - montgomery.h, 21
- aff\_is\_on\_curve
  - montgomery.c, 15
  - montgomery.h, 21
- aff\_ladder
  - montgomery.c, 15
  - montgomery.h, 21
- AFF\_POINT
  - montgomery.h, 19
- Aff\_Point\_s, 5
  - x, 5
  - y, 5
- AFF\_POINT\_t
  - montgomery.h, 19

## B

- Montg\_Curve\_s, 6
- B\_THRESHOLD
  - ecm.h, 10
- barret\_reduction
  - mplib.c, 25
  - mplib.h, 35
- barret\_reduction\_UL
  - mplib.c, 26
  - mplib.h, 36
- big\_add
  - mplib.c, 26
  - mplib.h, 36
- big\_cpy
  - mplib.h, 34
- big\_gcd
  - mplib.c, 26
  - mplib.h, 36
- big\_get\_A24
  - mplib.c, 26
  - mplib.h, 36
- big\_get\_mu
  - mplib.c, 27

- mplib.h, 37
- big\_invert
  - mplib.c, 27
  - mplib.h, 37
- big\_is\_equal
  - mplib.c, 28
  - mplib.h, 38
- big\_is\_equal\_ui
  - mplib.c, 28
  - mplib.h, 38
- big\_mod\_add
  - mplib.c, 28
  - mplib.h, 38
- big\_mod\_mul
  - mplib.c, 29
  - mplib.h, 39
- big\_mod\_rand
  - mplib.c, 29
  - mplib.h, 39
- big\_mod\_sub
  - mplib.c, 30
  - mplib.h, 40
- big\_mul
  - mplib.c, 30
  - mplib.h, 40
- big\_print
  - mplib.c, 31
  - mplib.h, 41
- big\_rand
  - mplib.c, 31
  - mplib.h, 41
- big\_sub
  - mplib.c, 32
  - mplib.h, 42

- CRV\_THRESHOLD
  - ecm.h, 10

## ecm

- ecm.c, 9
- ecm.h, 11
- ecm.c, 9
  - ecm, 9
- ecm.h, 10
  - B\_THRESHOLD, 10
  - CRV\_THRESHOLD, 10
  - ecm, 11
- ecm\_test
  - main.c, 12

- main
  - main.c, 12
- main.c, 11
  - aff\_curve\_point\_test, 12
  - ecm\_test, 12
  - main, 12
  - pro\_add\_gmp\_test, 12
  - pro\_add\_magma\_test, 12
  - pro\_curve\_point\_test, 12
  - pro\_dbl\_magma\_test, 12
  - pro\_ladder\_gmp\_test, 13
  - pro\_ladder\_magma\_test, 13
- MONTG\_CURVE
  - montgomery.h, 19
- Montg\_Curve\_s, 6
  - A, 6
  - B, 6
  - n, 6
- MONTG\_CURVE\_t
  - montgomery.h, 19
- montgomery.c, 13
  - aff\_add, 14
  - aff\_curve\_point, 14
  - aff\_dbl, 15
  - aff\_is\_on\_curve, 15
  - aff\_ladder, 15
  - pro\_add, 15
  - pro\_curve\_point, 16
  - pro\_dbl, 17
  - pro\_is\_on\_curve, 17
  - pro\_ladder, 17
- montgomery.h, 18
  - aff\_add, 20
  - aff\_curve\_point, 21
  - aff\_dbl, 21
  - aff\_is\_on\_curve, 21
  - aff\_ladder, 21
  - AFF\_POINT, 19
  - AFF\_POINT\_t, 19
  - MONTG\_CURVE, 19
  - MONTG\_CURVE\_t, 19
  - pro\_add, 22
  - pro\_curve\_point, 22
  - pro\_dbl, 23
  - pro\_is\_on\_curve, 23
  - pro\_ladder, 23
  - PRO\_POINT, 20
  - PRO\_POINT\_t, 20
- mplib.c, 24
  - barret\_reduction, 25
  - barret\_reduction\_UL, 26
  - big\_add, 26
  - big\_gcd, 26
  - big\_get\_A24, 26
  - big\_get\_mu, 27
  - big\_invert, 27
  - big\_is\_equal, 28
  - big\_is\_equal\_ui, 28
  - big\_mod\_add, 28
  - big\_mod\_mul, 29
  - big\_mod\_rand, 29
  - big\_mod\_sub, 30
  - big\_mul, 30
  - big\_print, 31
  - big\_rand, 31
  - big\_sub, 32
- mplib.h, 32
  - barret\_reduction, 35
  - barret\_reduction\_UL, 36
  - big\_add, 36
  - big\_cpy, 34
  - big\_gcd, 36
  - big\_get\_A24, 36
  - big\_get\_mu, 37
  - big\_invert, 37
  - big\_is\_equal, 38
  - big\_is\_equal\_ui, 38
  - big\_mod\_add, 38
  - big\_mod\_mul, 39
  - big\_mod\_rand, 39
  - big\_mod\_sub, 40
  - big\_mul, 40
  - big\_print, 41
  - big\_rand, 41
  - big\_sub, 42
  - ui, 34
  - ui\_t, 34
  - uni, 35
  - uni\_t, 35
  - W, 34
- n
  - Montg\_Curve\_s, 6
- pro\_add
  - montgomery.c, 15
  - montgomery.h, 22
- pro\_add\_gmp\_test
  - main.c, 12
- pro\_add\_magma\_test
  - main.c, 12
- pro\_curve\_point
  - montgomery.c, 16
  - montgomery.h, 22
- pro\_curve\_point\_test
  - main.c, 12
- pro\_dbl
  - montgomery.c, 17
  - montgomery.h, 23
- pro\_dbl\_magma\_test
  - main.c, 12
- pro\_is\_on\_curve
  - montgomery.c, 17
  - montgomery.h, 23
- pro\_ladder
  - montgomery.c, 17
  - montgomery.h, 23

- pro\_ladder\_gmp\_test
  - main.c, [13](#)
- pro\_ladder\_magma\_test
  - main.c, [13](#)
- PRO\_POINT
  - montgomery.h, [20](#)
- Pro\_Point\_s, [7](#)
  - X, [7](#)
  - Y, [7](#)
  - Z, [7](#)
- PRO\_POINT\_t
  - montgomery.h, [20](#)
- ui
  - mplib.h, [34](#)
- ui\_t
  - mplib.h, [34](#)
- uni
  - mplib.h, [35](#)
- uni\_t
  - mplib.h, [35](#)
- W
  - mplib.h, [34](#)
- X
  - Pro\_Point\_s, [7](#)
- x
  - Aff\_Point\_s, [5](#)
- Y
  - Pro\_Point\_s, [7](#)
- y
  - Aff\_Point\_s, [5](#)
- Z
  - Pro\_Point\_s, [7](#)