ELiPS for BN and BLS12 curve

v 1.0.1

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

ELiPS: Stands for Efficient Library for Pairing based Security. The main goal of this library is to give the reaserchers a tool that can be easy to install, configure and use. With a basic idea of pairing-based crypto anyone can be able to use this library for their reserach of protocols. We would like to update this library as incremental basis along side of our research activity to include the contemporary algorithmic improvements.

1.1 Licensing

ELiPS is released under an LGPL version 3.1-or-above license to encourage collaboration with other research groups and contributions from the industry.

1.2 Disclaimer

ELiPS is still under development software. Implementations may not be correct or secure and may include patented algorithms. Backward API compatibility with early versions may not necessarily be maintained. Use at your own risk.

Chapter 2

ELiPS Installation

This document describes how to make existing ELiPS library working in Linux environment. This is expected that it will work any 32-bit and 64bit Unix distribution (not tested). Autotools intallation may vary. If found any bug related to installation, please infrom in khandaker@s.okayama-u.ac.jp

1. Follow the instructions to install GMP library. Latest vesion is ok.
2. Check if autoconf is installed in your environment. autoconf --version. You migh see someting like this . If it is not installed then follow point 3. autoconf (GNU Autoconf) 2.69 Copyright (C) 2012 Free Software Foundation, Inc. License GPLv3+/Autoconf: GNU GPL version 3 or later http://gnu.org/licenses/gpl.html, http://gnu.org/licenses/exceptions.html This is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

   Written by David J. MacKenzie and Akim Demaille.
3. Install autoconf as follows sudo apt-get update sudo apt-get install autoconf
4. Install libtool as follows sudo apt-get install libtool-bin
5. git clone https://github.com/eNipu/elips_bn_bls.git
6. From terminal enter to <elips_bn_bls> directory.
7. Run the following commands autoreconf -i

The output will be almost as follows

```
libtoolize: Consider adding 'AC_CONFIG_MACRO_DIRS([m4])' to configure.ac,
libtoolize: and rerunning libtoolize and aclocal.
libtoolize: Consider adding '-I m4' to ACLOCAL_AMFLAGS in Makefile.am.
libtoolize: 'AC_PROG_RANLIB' is rendered obsolete by 'LT_INIT'
configure.ac:7: warning: AM_INIT_AUTOMAKE: two- and three-arguments forms are deprecated.  For more info, see:
configure.ac:7: http://www.gnu.org/software/automake/manual/automake.html#Modernize-AM_005fINIT_005fAUTOMAKE-i
```

1. Next run ./confgure
2. Then make
3. Finally sudo make install

1. To uninstall sudo make uninstall from the directory

If you face `cannot open shared object file: No such file or directory` while running then follow this steps:

1. Run from terminal sudo ldconfig
2. echo $LD_LIBRARY_PATH
3. If the command of point 2 gives blank result then LD_LIBRARY_PATH=/usr/local/lib
4. Check if again of `echo $LD_LIBRARY_PATH` . If path is set then run again.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

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

Chapter 6

Module Documentation

6.1 Paring over BARRETO-LYNN-SCOTT curve of embedding degree 12 related files.

Files

- file bls12_finalexp.h
- file bls12_frobenius.h
- file bls12_G3_exp.h
- file bls12_generate_points.h
- file bls12_inits.h
- file bls12_line_ate.h
- file bls12_line_tate.h
- file bls12_miller_ate.h
- file bls12_miller_optate.h
- file bls12_miller_tate.h
- file bls12_p8sparse.h
- file bls12_pairings.h
- file bls12_scm.h
- file bls12_skew_frobenius.h
- file bls12_test_pairings.h
- file bls12_timeprint.h
- file bls12_twist.h
- file bn_bls12_precoms.h

6.1.1 Detailed Description

6.2 Finite field construction

Files

- file bn_fp.h
- file field_dtype.h

6.2.1 Detailed Description

6.3 Paring over BARRETO-NAEHRIG (BN) Curve related files

Files

- file bn_efp.h
- file bn_efp12.h
- file bn_efp2.h
- file bn_efp6.h
- file bn_final_exp.h
- file bn_inits.h

6.3.1 Detailed Description

6.4 Curve parameter generation and settings

Files

- file curve_dtypes.h
- file curve_settings.h

6.4.1 Detailed Description

Chapter 7

Data Structure Documentation

7.1 curve_params Struct Reference

Curve Parameters.

`#include <curve_settings.h>`

Data Fields

- mpz_t prime
- mpz_t X
- mpz_t trace_t
- mpz_t order
- mpz_t EFp_total
- mpz_t EFp2_total
- mpz_t EFp6_total
- mpz_t EFp12_total
- mpz_t EFpd_total
- mpz_t curve_a
- mpz_t curve_b

7.1.1 Detailed Description

Curve Parameters.

This curve_params structure contains member variable related to pairing friendly curves.

7.1.2 Field Documentation

7.1.2.1 curve_a

```
mpz_t curve_a
```

Curves coeffient y^2=x^3+ax+b curve_params::a

Referenced by clear_parameters(), Fp_mul_basis_KSS16(), print_curve_parameters(), and weil().

7.1.2.2 curve_b

```
mpz_t curve_b
```

Curves coeffient y^2=x^3+ax+b curve_params::b

Referenced by bls12_set_curve_parameter(), clear_parameters(), EFp12_rational_point_bls12(), EFp12_rational_point_bn(), EFp2_rational_point(), EFp6_rational_point(), EFp_rational_point_bls12(), EFp_rational_point_bn(), init_bls12_parameters(), print_curve_parameters(), set_bn_curve_parameter(), and weil().

7.1.2.3 EFp12_total

```
mpz_t EFp12_total
```

Number of rational points in curve defined in Fp12 curve_params::EFp12_total.

Referenced by init_bls12_parameters().

7.1.2.4 EFp2_total

```
mpz_t EFp2_total
```

Number of rational points in curve defined in Fp2 curve_params::EFp2_total.

Referenced by init_bls12_parameters().

7.1.2.5 EFp6_total

```
mpz_t EFp6_total
```

Number of rational points in curve defined in Fp6 curve_params::EFp6_total.

Referenced by init_bls12_parameters().

7.1.2.6 EFp_total

```
mpz_t EFp_total
```

Number of rational points in curve defined in Fp curve_params::EFp_total.

Referenced by bls12_generate_G1_point(), bls12_weil(), clear_parameters(), init_bls12_parameters(), and weil().

7.1.2.7 EFpd_total

```
mpz_t EFpd_total
```

Number of rational points in twisted curve defined in Fp^k/d curve_params::EFp_total.

Referenced by bls12_generate_G2_point(), bls12_weil(), and weil().

7.1.2.8 order

```
mpz_t order
```

Curves order r given by polynomial formula of X curve_params::order.

Referenced by bls12_finalexp_plain(), bls12_generate_G1_point(), bls12_generate_G2_point(), bls12_Miller_algo_for_tate(), bls12_test_G1_scm(), bls12_test_G2_scm(), bls12_test_G3_exp(), bls12_test_opt_ate_pairing(), bls12_test_plain_ate_pairing(), bls12_test_tate_pairing(), bn_final_exp_plain(), clear_parameters(), generate_bn_order(), init_bls12_parameters(), and print_curve_parameters().

7.1.2.9 prime

```
mpz_t prime
```

Prime number genereted using polynomial formula of X curve_params::prime.

Referenced by bls12_finalexp_plain(), bls12_weil(), bn_final_exp_plain(), clear_parameters(), Fp_add(), Fp_add_mpz(), Fp_add_ui(), Fp_inv(), Fp_isCNR(), Fp_legendre(), Fp_mul(), Fp_mul_basis(), Fp_mul_mpz(), Fp_mul_ui(), Fp_neg(), Fp_set_random(), Fp_sqrt(), Fp_sub(), Fp_sub_mpz(), Fp_sub_ui(), generate_bn_prime(), init_bls12_parameters(), init_precoms(), print_curve_parameters(), and weil().

7.1.2.10 trace_t

```
mpz_t trace_t
```

Curves Frobenius trace curve_params::trace_t.

Referenced by bls12_generate_trace(), bls12_Miller_algo_for_plain_ate(), bls12_weil(), clear_parameters(), generate_bn_trace(), init_bls12_parameters(), print_curve_parameters(), and weil().

7.1.2.11 X

```
mpz_t X
```

Mother parameter curve_params::X.

Referenced by clear_parameters(), generate_bn_mother_parameter(), generate_bn_order(), generate_bn_prime(), generate_bn_trace(), init_bls12_parameters(), and print_curve_parameters().

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

- include/ELiPS_bn_bls/curve_settings.h

7.2 EFp Struct Reference

Data Fields

- Fp y
- int infinity

7.2.1 Field Documentation

7.2.1.1 infinity

```
int infinity
```

Flag to identify rational point as Point at infinity. 1 is TRUE, default vale 0. curve_params::infinity

Referenced by bls12_2split_G1_scm(), bls12_EFp12_to_EFp(), bls12_EFp_to_EFp12(), bls12_f_ltp_vtp_for_tate(), bls12_ff_ltt_vtt_for_tate(), bls12_generate_G1_point(), EFp_ECA(), EFp_ECD(), EFp_init(), EFp_printf(), EFp_S↩CM(), EFp_set(), EFp_set_mpz(), EFp_set_neg(), and EFp_set_ui().

7.2.1.2 y

```
Fp y
```

Coordinate of curve curve_params::x curve_params::y.

Referenced by bls12_EFp12_to_EFp(), bls12_EFp_skew_frobenius_map_p2(), bls12_EFp_to_EFp12(), bls12_f_↩ ltp_vtp_for_tate(), bls12_ff_ltt_vtt_for_tate(), bls12_Pseudo_8_sparse_mapping(), EFp_clear(), EFp_ECA(), EFp↩ _ECD(), EFp_init(), EFp_printf(), EFp_rational_point_bls12(), EFp_rational_point_bn(), EFp_set(), EFp_set_mpz(), EFp_set_neg(), and EFp_set_ui().

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

- include/ELiPS_bn_bls/curve_dtypes.h

7.3 EFp12 Struct Reference

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

- include/ELiPS_bn_bls/curve_dtypes.h

7.4 EFp16 Struct Reference

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

- include/ELiPS_bn_bls/curve_dtypes.h

7.5 EFp2 Struct Reference

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

- include/ELiPS_bn_bls/curve_dtypes.h

7.6 EFp4 Struct Reference

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

- include/ELiPS_bn_bls/curve_dtypes.h

7.7 EFp6 Struct Reference

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

- include/ELiPS_bn_bls/curve_dtypes.h

7.8 EFp8 Struct Reference

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

- include/ELiPS_bn_bls/curve_dtypes.h

7.9 Fp Struct Reference

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

- include/ELiPS_bn_bls/field_dtype.h

7.10 Fp12 Struct Reference

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

- include/ELiPS_bn_bls/field_dtype.h

7.11 Fp16 Struct Reference

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

- include/ELiPS_bn_bls/field_dtype.h

7.12 Fp2 Struct Reference

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

- include/ELiPS_bn_bls/field_dtype.h

7.13 Fp4 Struct Reference

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

- include/ELiPS_bn_bls/field_dtype.h

7.14 Fp6 Struct Reference

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

- include/ELiPS_bn_bls/field_dtype.h

7.15 Fp8 Struct Reference

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

- include/ELiPS_bn_bls/field_dtype.h

Chapter 8

File Documentation

8.1 include/ELiPS_bn_bls/bls12_finalexp.h File Reference

```
#include <ELiPS_bn_bls/bls12_frobenius.h>
```
Include dependency graph for bls12_finalexp.h:

8.2 include/ELiPS_bn_bls/bls12_frobenius.h File Reference

```
#include <ELiPS_bn_bls/bn_efp12.h>
#include <ELiPS_bn_bls/bn_bls12_precoms.h>
```
Include dependency graph for bls12_frobenius.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_Fp12_frobenius_map_p1 (Fp12 *ANS, Fp12 *A)
- void bls12_Fp12_frobenius_map_p2 (Fp12 *ANS, Fp12 *A)
- void bls12_Fp12_frobenius_map_p3 (Fp12 *ANS, Fp12 *A)
- void bls12_Fp12_frobenius_map_p4 (Fp12 *ANS, Fp12 *A)
- void bls12_Fp12_frobenius_map_p6 (Fp12 *ANS, Fp12 *A)
- void bls12_Fp12_frobenius_map_p8 (Fp12 *ANS, Fp12 *A)
- void bls12_Fp12_frobenius_map_p10 (Fp12 *ANS, Fp12 *A)

8.2.1 Detailed Description

Header of the Frobenius mapping implementation for BLS12 over Fp12 extension field.

8.2.2 Function Documentation

8.2.2.1 bls12_Fp12_frobenius_map_p1()

```
void bls12_Fp12_frobenius_map_p1 (
            Fp12 * ANS,
            Fp12 * A )
```
Calculate Frobenius map of A in Fp12 extension field as A^p where p is the BLS12 prime.

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.

References d12_frobenius_constant, Fp_init(), Fp_set(), and Fp_set_neg().

Referenced by bls12_4split_G3_exp(), bls12_finalexp_optimal(), and bls12_generate_G2_point().

```
31                                                  {
32     Fp tmp;
33     Fp_init(&tmp);
34
35     //x0
36     Fp_set(&ANS->x0.x0.x0,&A->x0.x0.x0);
37     Fp_set_neg(&ANS->x0.x0.x1,&A->x0.x0.x1);
38     Fp_set(&tmp,&A->x0.x1.x0);
39     Fp_set(&ANS->x0.x1.x0,&A->x0.x1.x1);
40     Fp_set(&ANS->x0.x1.x1,&tmp);
41     Fp2_mul_mpz(&ANS->x0.x1,&ANS->x0.x1,d12_frobenius_constant[f_p1][1].x1.x0);
42     Fp_set(&ANS->x0.x2.x0,&A->x0.x2.x0);
43     Fp_set_neg(&ANS->x0.x2.x1,&A->x0.x2.x1);
44     Fp2_mul_mpz(&ANS->x0.x2,&ANS->x0.x2,d12_frobenius_constant[f_p1][2].x0.x0);
45     //x1
46     Fp_set(&ANS->x1.x0.x0,&A->x1.x0.x0);
47     Fp_set_neg(&ANS->x1.x0.x1,&A->x1.x0.x1);
48     Fp2_mul(&ANS->x1.x0,&ANS->x1.x0,&d12_frobenius_constant[f_p1][3]);
49     Fp_set(&ANS->x1.x1.x0,&A->x1.x1.x0);
50     Fp_set_neg(&ANS->x1.x1.x1,&A->x1.x1.x1);
51     Fp2_mul(&ANS->x1.x1,&ANS->x1.x1,&d12_frobenius_constant[f_p1][4]);
52     Fp_set(&ANS->x1.x2.x0,&A->x1.x2.x0);
53     Fp_set_neg(&ANS->x1.x2.x1,&A->x1.x2.x1);
54     Fp2_mul(&ANS->x1.x2,&ANS->x1.x2,&d12_frobenius_constant[f_p1][5]);
55
56     Fp_clear(&tmp);
57 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.2.2.2 bls12_Fp12_frobenius_map_p10()

```
void bls12_Fp12_frobenius_map_p10 (
             Fp12 * ANS,
             Fp12 * A )
```

Calculate Frobenius map of A in Fp12 extension field as A^p^10 where p is the BLS12 prime.

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.

References d12_frobenius_constant.

```
125                                                   {
126     //x0
127     Fp2_set(&ANS->x0.x0,&A->x0.x0);
128     Fp2_mul_mpz(&ANS->x0.x1,&A->x0.x1,d12_frobenius_constant[f_p10][1].x0.x0);
129     Fp2_mul_mpz(&ANS->x0.x2,&A->x0.x2,d12_frobenius_constant[f_p10][2].x0.x0);
130     //x1
131     Fp2_mul_mpz(&ANS->x1.x0,&A->x1.x0,d12_frobenius_constant[f_p10][3].x0.x0);
132     Fp2_set_neg(&ANS->x1.x1,&A->x1.x1);
133     Fp2_mul_mpz(&ANS->x1.x2,&A->x1.x2,d12_frobenius_constant[f_p10][5].x0.x0);
134 }
```

8.2.2.3 bls12_Fp12_frobenius_map_p2()

```
void bls12_Fp12_frobenius_map_p2 (
            Fp12 * ANS,
            Fp12 * A )
```

Calculate Frobenius map of A in Fp12 extension field as A^p^2 where p is the BLS12 prime.

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.

References d12_frobenius_constant.

Referenced by bls12_2split_G3_exp(), bls12_4split_G3_exp(), bls12_finalexp_optimal(), and bls12_finalexp_↩plain().

```
59                                                  {
60     //x0
61     Fp2_set(&ANS->x0.x0,&A->x0.x0);
62     Fp2_mul_mpz(&ANS->x0.x1,&A->x0.x1,d12_frobenius_constant[f_p2][1].x0.x0);
63     Fp2_mul_mpz(&ANS->x0.x2,&A->x0.x2,d12_frobenius_constant[f_p2][2].x0.x0);
64     //x1
65     Fp2_mul_mpz(&ANS->x1.x0,&A->x1.x0,d12_frobenius_constant[f_p2][3].x0.x0);
66     Fp2_set_neg(&ANS->x1.x1,&A->x1.x1);
67     Fp2_mul_mpz(&ANS->x1.x2,&A->x1.x2,d12_frobenius_constant[f_p2][5].x0.x0);
68 }
```

Here is the caller graph for this function:

8.2.2.4 bls12_Fp12_frobenius_map_p3()

```
void bls12_Fp12_frobenius_map_p3 (
            Fp12 * ANS,
            Fp12 * A )
```

Calculate Frobenius map of A in Fp12 extension field as A^p^3 where p is the BLS12 prime.

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.

References d12_frobenius_constant, Fp_init(), Fp_set(), and Fp_set_neg().

Referenced by bls12_4split_G3_exp(), and bls12_finalexp_optimal().

```
70                                                  {
71     Fp tmp;
72     Fp_init(&tmp);
```

```
73
74     //x0
75     Fp_set(&ANS->x0.x0.x0,&A->x0.x0.x0);
76     Fp_set_neg(&ANS->x0.x0.x1,&A->x0.x0.x1);
77     Fp_set(&tmp,&A->x0.x1.x0);
78     Fp_set(&ANS->x0.x1.x0,&A->x0.x1.x1);
79     Fp_set(&ANS->x0.x1.x1,&tmp);
80     Fp_set_neg(&ANS->x0.x2.x0,&A->x0.x2.x0);
81     Fp_set(&ANS->x0.x2.x1,&A->x0.x2.x1);
82     //x1
83     Fp_set(&ANS->x1.x0.x0,&A->x1.x0.x0);
84     Fp_set_neg(&ANS->x1.x0.x1,&A->x1.x0.x1);
85     Fp2_mul(&ANS->x1.x0,&ANS->x1.x0,&d12_frobenius_constant[f_p3][3]);
86     Fp_set(&ANS->x1.x1.x0,&A->x1.x1.x0);
87     Fp_set_neg(&ANS->x1.x1.x1,&A->x1.x1.x1);
88     Fp2_mul(&ANS->x1.x1,&ANS->x1.x1,&d12_frobenius_constant[f_p3][4]);
89     Fp_set(&ANS->x1.x2.x0,&A->x1.x2.x0);
90     Fp_set_neg(&ANS->x1.x2.x1,&A->x1.x2.x1);
91     Fp2_mul(&ANS->x1.x2,&ANS->x1.x2,&d12_frobenius_constant[f_p3][5]);
92
93     Fp_clear(&tmp);
94 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.2.2.5 bls12_Fp12_frobenius_map_p4()

```
void bls12_Fp12_frobenius_map_p4 (
              Fp12 * ANS,
              Fp12 * A )
```

Calculate Frobenius map of A in Fp12 extension field as A^p^4 where p is the BLS12 prime.

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.

References d12_frobenius_constant.

```
96                                                  {
97     //x0
98     Fp2_set(&ANS->x0.x0,&A->x0.x0);
99     Fp2_mul_mpz(&ANS->x0.x1,&A->x0.x1,d12_frobenius_constant[f_p4][1].x0.x0);
100     Fp2_mul_mpz(&ANS->x0.x2,&A->x0.x2,d12_frobenius_constant[f_p4][2].x0.x0);
101     //x1
102     Fp2_mul_mpz(&ANS->x1.x0,&A->x1.x0,d12_frobenius_constant[f_p4][3].x0.x0);
103     Fp2_set(&ANS->x1.x1,&A->x1.x1);
104     Fp2_mul_mpz(&ANS->x1.x2,&A->x1.x2,d12_frobenius_constant[f_p4][5].x0.x0);
105 }
```

8.2.2.6 bls12_Fp12_frobenius_map_p6()

```
void bls12_Fp12_frobenius_map_p6 (
              Fp12 * ANS,
              Fp12 * A )
```

Calculate Frobenius map of A in Fp12 extension field as A^p^6 where p is the BLS12 prime.

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.

Referenced by bls12_finalexp_optimal(), and bls12_finalexp_plain().

```
107                                                 {
108     //x0
109     Fp6_set(&ANS->x0,&A->x0);
110     //x1
111     Fp6_set_neg(&ANS->x1,&A->x1);
112 }
```

Here is the caller graph for this function:

8.2.2.7 bls12_Fp12_frobenius_map_p8()

```
void bls12_Fp12_frobenius_map_p8 (
            Fp12 * ANS,
            Fp12 * A )
```

Calculate Frobenius map of A in Fp12 extension field as A^p^8 where p is the BLS12 prime.

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.

References d12_frobenius_constant.

```
114                                                 {
115     //x0
116     Fp2_set(&ANS->x0.x0,&A->x0.x0);
117     Fp2_mul_mpz(&ANS->x0.x1,&A->x0.x1,d12_frobenius_constant[f_p8][1].x0.x0);
118     Fp2_mul_mpz(&ANS->x0.x2,&A->x0.x2,d12_frobenius_constant[f_p8][2].x0.x0);
119     //x1
120     Fp2_mul_mpz(&ANS->x1.x0,&A->x1.x0,d12_frobenius_constant[f_p8][3].x0.x0);
121     Fp2_set(&ANS->x1.x1,&A->x1.x1);
122     Fp2_mul_mpz(&ANS->x1.x2,&A->x1.x2,d12_frobenius_constant[f_p8][5].x0.x0);
123 }
```

8.3 include/ELiPS_bn_bls/bls12_G3_exp.h File Reference

```
#include <ELiPS_bn_bls/bn_utils.h>
#include <ELiPS_bn_bls/bls12_timeprint.h>
#include <ELiPS_bn_bls/bls12_frobenius.h>
```

Include dependency graph for bls12_G3_exp.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_plain_G3_exp (Fp12 *ANS, Fp12 *A, mpz_t scalar)
- void bls12_2split_G3_exp (Fp12 *ANS, Fp12 *A, mpz_t scalar)
- void bls12_4split_G3_exp (Fp12 *ANS, Fp12 *A, mpz_t scalar)

8.3.1 Detailed Description

Header of the Group G3 element exponentiation for BLS12 over Fp12 extension field.

8.3.2 Function Documentation

8.3.2.1 bls12_2split_G3_exp()

```
void bls12_2split_G3_exp (
            Fp12 * ANS,
            Fp12 * A,
            mpz_t scalar )
```

Calculate exponentiation of element in G3 group in Fp12 extension field. where G1∗G2->G3 is the bilinear pairing. This method is divides the scallar into 2 parts and and calculate the exponentiation in parallel

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.
in	*scalar*	- the input exponent by splitting into 2 parts.

References bls12_Fp12_frobenius_map_p2(), and Fp_set_ui().

Referenced by bls12_test_G3_exp().

```
40                                                          {
41     gettimeofday(&t0,NULL);
42
43     int i,length_s[2],loop_length;
44     Fp12 Buf;
45     Fp12_init(&Buf);
46     Fp12 next_f,f,frobenius_f_2x;
47     Fp12_init(&next_f);
48     Fp12_init(&f);
49     Fp12_init(&frobenius_f_2x);
50     mpz_t s[2],buf;
51     mpz_init(buf);
52     for(i=0; i<2; i++){
53         mpz_init(s[i]);
54     }
55     //table
56     Fp12 table[4];
57     for(i=0; i<4; i++){
58         Fp12_init(&table[i]);
59     }
60
61     //set
62     Fp12_set(&f, A);
63     bls12_Fp12_frobenius_map_p2(&frobenius_f_2x, A);
64
65     //set table
66     Fp_set_ui(&table[0].x0.x0.x0,1);    //00
67     Fp12_set(&table[1],&f);                //01
68     Fp12_set(&table[2],&frobenius_f_2x);    //10
69     Fp12_mul(&table[3],&table[1],&table[2]);    //11
70
71     //s0,s1
72     mpz_neg(buf,bls12_X);
73     mpz_pow_ui(buf,buf,2);
74     mpz_tdiv_qr(s[1],s[0],scalar,buf);
```

```
75      //binary
76      loop_length=0;
77      for(i=0; i<2; i++){
78          length_s[i]=(int)mpz_sizeinbase(s[i],2);
79          if(loop_length<length_s[i]){
80              loop_length=length_s[i];
81          }
82      }
83      //set binary
84      char binary_s[2][loop_length+1];
85      char str[5],*e;
86      int binary[loop_length+1];
87      for(i=0; i<2; i++){
88          if(length_s[i]==loop_length){
89              mpz_get_str(binary_s[i],2,s[i]);
90          }else{
91              char binary_buf[loop_length+1];
92              mpz_get_str(binary_buf,2,s[i]);
93              memset(binary_s[i],'0',sizeof(binary_s[i]));
94              memmove(binary_s[i]+loop_length-length_s[i],binary_buf,sizeof(binary_buf));
95          }
96      }
97      for(i=0; i<loop_length; i++){
98          sprintf(str,"%c%c",binary_s[1][i],binary_s[0][i]);
99          binary[i]=(int)strtol(str,&e,2);
100     }
101     Fp12_set(&next_f,&table[binary[0]]);
102
103     //SCM
104     for(i=1; i<loop_length; i++){
105         Fp12_mul(&next_f,&next_f,&next_f);
106         Fp12_mul(&next_f,&next_f,&table[binary[i]]);
107     }
108
109     Fp12_set(ANS,&next_f);
110
111     mpz_clear(buf);
112     Fp12_clear(&next_f);
113     Fp12_clear(&f);
114     Fp12_clear(&frobenius_f_2x);
115     Fp12_clear(&Buf);
116     for(i=0; i<2; i++){
117         mpz_clear(s[i]);
118     }
119     for(i=0; i<4; i++){
120         Fp12_clear(&table[i]);
121     }
122
123     gettimeofday(&t1,NULL);
124     bls12_G3EXP_2SPLIT=timedifference_msec(t0,t1);
125 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.3.2.2 bls12_4split_G3_exp()

```
void bls12_4split_G3_exp (
            Fp12 * ANS,
            Fp12 * A,
            mpz_t scalar )
```

Calculate exponentiation of element in G3 group in Fp12 extension field. where G1∗G2->G3 is the bilinear pairing. This method is divides the scallar into 4 parts and and calculate the exponentiation in parallel

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.
in	*scalar*	- the input exponent by splitting into 4 parts.

References bls12_Fp12_frobenius_map_p1(), bls12_Fp12_frobenius_map_p2(), bls12_Fp12_frobenius_map_↩

p3(), and Fp_set_ui().

Referenced by bls12_test_G3_exp().

```
127                                                     {
128     gettimeofday(&t0,NULL);
129 
130     int i,length_s[4],loop_length;
131     Fp12 Buf;
132     Fp12_init(&Buf);
133     Fp12 next_f,f,frobenius_f_x,frobenius_f_2x,frobenius_f_3x;
134     Fp12_init(&next_f);
135     Fp12_init(&f);
136     Fp12_init(&frobenius_f_x);
137     Fp12_init(&frobenius_f_2x);
138     Fp12_init(&frobenius_f_3x);
139     mpz_t C,D,s[4],x_2,x_1;
140     mpz_init(C);
141     mpz_init(D);
142     for(i=0; i<4; i++){
143         mpz_init(s[i]);
144     }
145     mpz_init(x_1);
146     mpz_init(x_2);
147     //table
148     Fp12 table[16];
149     for(i=0; i<16; i++){
150         Fp12_init(&table[i]);
151     }
152 
153     //set
154     Fp12_set(&f, A);
155     bls12_Fp12_frobenius_map_p1(&frobenius_f_x, A);
156     Fp12_inv(&frobenius_f_x,&frobenius_f_x);
157     bls12_Fp12_frobenius_map_p2(&frobenius_f_2x, A);
158     bls12_Fp12_frobenius_map_p3(&frobenius_f_3x, A);
159     Fp12_inv(&frobenius_f_3x,&frobenius_f_3x);
160     //set table
161     Fp_set_ui(&table[0].x0.x0.x0,1);            //0000
162     Fp12_set(&table[1],&f);                     //0001
163     Fp12_set(&table[2],&frobenius_f_x);         //0010
164     Fp12_mul(&table[3],&table[1],&table[2]);    //0011
165     Fp12_set(&table[4],&frobenius_f_2x);        //0100
166     Fp12_mul(&table[5],&table[4],&table[1]);    //0101
167     Fp12_mul(&table[6],&table[2],&table[4]);    //0110
168     Fp12_mul(&table[7],&table[6],&table[1]);    //0111
169     Fp12_set(&table[8],&frobenius_f_3x);        //1000
170     Fp12_mul(&table[9],&table[8],&table[1]);    //1001
171     Fp12_mul(&table[10],&table[8],&table[2]);    //1010
172     Fp12_mul(&table[11],&table[10],&table[1]);    //1011
173     Fp12_mul(&table[12],&table[8],&table[4]);    //1100
174     Fp12_mul(&table[13],&table[12],&table[1]);    //1101
175     Fp12_mul(&table[14],&table[12],&table[2]);    //1110
176     Fp12_mul(&table[15],&table[14],&table[1]);    //1111
177     //set
178     //s0,s1,s2,s3
179     mpz_neg(x_1,bls12_X);
180     mpz_mul(x_2,x_1,x_1);
181     mpz_tdiv_qr(D,C,scalar,x_2);
182     mpz_tdiv_qr(s[1],s[0],C,x_1);
183     mpz_tdiv_qr(s[3],s[2],D,x_1);
184 
185     //binary
186     loop_length=0;
187     for(i=0; i<4; i++){
188         length_s[i]=(int)mpz_sizeinbase(s[i],2);
189         if(loop_length<length_s[i]){
190             loop_length=length_s[i];
191         }
192     }
193     //set binary
194     char binary_s[4][loop_length+1];
195     char str[5],*e;
196     int binary[loop_length+1];
197     for(i=0; i<4; i++){
198         if(length_s[i]==loop_length){
199             mpz_get_str(binary_s[i],2,s[i]);
200         }else{
201             char binary_buf[loop_length+1];
202             mpz_get_str(binary_buf,2,s[i]);
203             memset(binary_s[i],'0',sizeof(binary_s[i]));
204             memmove(binary_s[i]+loop_length-length_s[i],binary_buf,sizeof(binary_buf));
205         }
206     }
207     for(i=0; i<loop_length; i++){
```

```
208         sprintf(str,"%c%c%c%c",binary_s[3][i],binary_s[2][i],binary_s[1][i],binary_s[0][i]);
209         binary[i]=(int)strtol(str,&e,2);
210     }
211 
212     Fp12_set(&next_f,&table[binary[0]]);
213 
214     //SCM
215     for(i=1; i<loop_length; i++){
216         Fp12_mul(&next_f,&next_f,&next_f);
217         Fp12_mul(&next_f,&next_f,&table[binary[i]]);
218     }
219 
220     Fp12_set(ANS,&next_f);
221 
222     Fp12_clear(&Buf);
223     Fp12_clear(&next_f);
224     Fp12_clear(&f);
225     Fp12_clear(&frobenius_f_x);
226     Fp12_clear(&frobenius_f_2x);
227     Fp12_clear(&frobenius_f_3x);
228     mpz_clear(x_1);
229     mpz_clear(x_2);
230     for(i=0; i<4; i++){
231         mpz_clear(s[i]);
232     }
233     for(i=0; i<16; i++){
234         Fp12_clear(&table[i]);
235     }
236 
237     gettimeofday(&t1,NULL);
238     bls12_G3EXP_4SPLIT=timedifference_msec(t0,t1);
239 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.3.2.3 bls12_plain_G3_exp()

```
void bls12_plain_G3_exp (
             Fp12 * ANS,
             Fp12 * A,
             mpz_t scalar )
```

Calculate exponentiation of element in G3 group in Fp12 extension field. where G1∗G2->G3 is the bilinear pairing. This method is less efficient than 2 and 4 split

Parameters

out	*ANS*	- the result.
in	*A*	- the input vecotr.
in	*scalar*	- the input exponent without splitting.

Referenced by bls12_test_G3_exp().

```
31                                                  {
32     gettimeofday(&t0,NULL);
33 
34     Fp12_pow(ANS,A,scalar);
35 
36     gettimeofday(&t1,NULL);
37     bls12_G3EXP_PLAIN=timedifference_msec(t0,t1);
38 }
```

Here is the caller graph for this function:

8.4 include/ELiPS_bn_bls/bls12_generate_points.h File Reference

```
#include <ELiPS_bn_bls/curve_settings.h>
#include <ELiPS_bn_bls/bls12_twist.h>
#include <ELiPS_bn_bls/bls12_frobenius.h>
```

Include dependency graph for bls12_generate_points.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_generate_G1_point (EFp12 *P)
- void bls12_generate_G2_point (EFp12 *Q)
- void bls12_generate_random_point (EFp12 *R)

8.4.1 Detailed Description

Header of generating rational point for BLS12 over Fp12 extension field.

8.4.2 Function Documentation

8.4.2.1 bls12_generate_G1_point()

```
void bls12_generate_G1_point (
            EFp12 * P )
```

Generates rational point P in G1 group. Usally generated in prime field but maps it to Fp12 for using in pairing. where G1*G2->G3 is the bilinear pairing.

Parameters

out	*P*	- the generated point.
in	*P*	- the input vecotr initialized as 'EFp12 P' point by 'EFp12_init' function.

References bls12_EFp_to_EFp12(), curve_parameters, EFp12_SCM(), EFp_clear(), EFp_init(), EFp_rational_point_bls12(), curve_params::EFp_total, EFp::infinity, and curve_params::order.

Referenced by bls12_test_G1_scm(), bls12_test_G3_exp(), bls12_test_opt_ate_pairing(), bls12_test_plain_ate_pairing(), and bls12_test_tate_pairing().

```
33                                      {
34     EFp Tmp_P;
35     EFp_init(&Tmp_P);
36     mpz_t scalar;
37     mpz_init(scalar);
38
39     EFp_rational_point_bls12(&Tmp_P);
40     bls12_EFp_to_EFp12(P,&Tmp_P);
41     mpz_tdiv_q(scalar,curve_parameters.EFp_total,
```

```
      curve_parameters.order);
42     EFp12_SCM(P,P,scalar);
43     P->infinity=Tmp_P.infinity;
44
45     EFp_clear(&Tmp_P);
46     mpz_clear(scalar);
47 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.4.2.2 bls12_generate_G2_point()

```
void bls12_generate_G2_point (
             EFp12 * Q )
```

Generates special rational point Q in G2 group. where G1∗G2->G3 is the bilinear pairing.

Parameters

out	*Q*	- the generated point.
in	*Q*	- the input vecotr initialized as 'EFp12 Q' point by 'EFp12_init' function.

References bls12_Fp12_frobenius_map_p1(), curve_parameters, EFp12_clear(), EFp12_ECA(), EFp12_init(), E↩ Fp12_rational_point_bls12(), EFp12_SCM(), EFp12_set_neg(), curve_params::EFpd_total, and curve_params↩ ::order.

Referenced by bls12_test_G2_scm(), bls12_test_G3_exp(), bls12_test_opt_ate_pairing(), and bls12_test_plain_↩ ate_pairing().

```
49                                      {
50     EFp12 random_P,P,frobenius_P;
51     EFp12_init(&random_P);
52     EFp12_init(&P);
53     EFp12_init(&frobenius_P);
54     mpz_t exp;
55     mpz_init(exp);
56
57     EFp12_rational_point_bls12(&random_P);
58     mpz_pow_ui(exp,curve_parameters.order,2);
59     mpz_tdiv_q(exp,curve_parameters.EFpd_total,exp);
60     EFp12_SCM(&P,&random_P,exp);
61     bls12_Fp12_frobenius_map_p1(&frobenius_P.x,&P.x);
62     bls12_Fp12_frobenius_map_p1(&frobenius_P.y,&P.y);
63     frobenius_P.infinity=P.infinity;
64     EFp12_set_neg(&P,&P);
65     EFp12_ECA(Q,&P,&frobenius_P);
66
67     mpz_clear(exp);
68     EFp12_clear(&random_P);
69     EFp12_clear(&P);
70     EFp12_clear(&frobenius_P);
71 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.4.2.3 bls12_generate_random_point()

```
void bls12_generate_random_point (
             EFp12 * R )
```

Generates random rational point R in the BLS12 curve over Fp12.

Parameters

out	*R*	- the generated point.
in	*R*	- the input vecotr initialized as 'EFp12 P' point by 'EFp12_init'.

References EFp12_init(), and EFp12_rational_point_bls12().

```
73                                          {
74     EFp12_init(R);
75     EFp12_rational_point_bls12(R);
76 }
```

Here is the call graph for this function:

8.5 include/ELiPS_bn_bls/bls12_inits.h File Reference

`#include <ELiPS_bn_bls/bn_bls12_precoms.h>`
Include dependency graph for bls12_inits.h:

Functions

- void bls12_inits (void)

8.5.1 Detailed Description

Interaface for pairing using BLS12 curve. It needs to included for initializing BLS12 curve in the applications.

8.5.2 Function Documentation

8.5.2.1 bls12_inits()

```
void bls12_inits (
             void  )
```

This methods needs to be called before using the BLS12 curve for pairing. It initialized the curve as y^2=x^3+4 with parameters suggested in https://eprint.iacr.org/2017/334.

References init_bls12_settings(), and init_precoms().

```
31                  {
32     init_bls12_settings();
33     init_precoms(2);
34 }
```

Here is the call graph for this function:

8.6 include/ELiPS_bn_bls/bls12_line_ate.h File Reference

```
#include <ELiPS_bn_bls/bls12_p8sparse.h>
```

Include dependency graph for bls12_line_ate.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_ff_ltt (Fp12 *f, EFp2 *T, EFp *P, Fp *L)
- void bls12_f_ltq (Fp12 *f, EFp2 *T, EFp2 *Q, EFp *P, Fp *L)

8.6.1 Detailed Description

Interaface for BLS12 line evaluation for ate pairing.

8.6.2 Function Documentation

8.6.2.1 bls12_f_ltq()

```
void bls12_f_ltq (
            Fp12 * f,
            EFp2 * T,
            EFp2 * Q,
            EFp * P,
            Fp * L )
```

Calculate line equation ltq that goes through T,Q and P in Miller's algo.

Parameters

out	*f*	- output in Fp12
in	*T*	- Input rational point T in Fp2.
in	*Q*	- Input rational point Q mapped in Fp2.
in	*P*	- Input rational point P mapped in Fp.
in	*L*	- Input element L in mapped Fp after precomputation in Miller's algo.

References bls12_Pseudo_8_sparse_mul(), EFp2_clear(), EFp2_init(), EFp2_set(), and Fp_set_ui().

Referenced by bls12_Miller_algo_for_opt_ate(), and bls12_Miller_algo_for_plain_ate().

```
80                                                  {
81     EFp2 Tmp_T;
82     EFp2_init(&Tmp_T);
83     Fp12 ltq;
84     Fp12_init(&ltq);
85     Fp2 A,B,C,D,E;
86     Fp2_init(&A);
87     Fp2_init(&B);
```

```
88      Fp2_init(&C);
89      Fp2_init(&D);
90      Fp2_init(&E);
91      EFp2_set(&Tmp_T,T);
92
93      //ltq
94      Fp2_sub(&A,&Q->x,&Tmp_T.x);         //A=(Q->x-T.x)^-1
95      Fp2_inv(&A,&A);
96      Fp2_sub(&B,&Q->y,&Tmp_T.y);         //B=(Q->y-T.y)
97      Fp2_mul(&C,&A,&B);              //C=A*B
98      Fp2_add(&D,&Tmp_T.x,&Q->x);         //D=Q->x+T.x
99      Fp2_sqr(&T->x,&C);              //next_T.x=C^2-D
100      Fp2_sub(&T->x,&T->x,&D);
101      Fp2_mul(&E,&C,&Tmp_T.x);          //E=C*T.x-T.y
102      Fp2_sub(&E,&E,&Tmp_T.y);
103      Fp2_mul(&T->y,&C,&T->x);          //next_T.y=E-C*next_T.x
104      Fp2_sub(&T->y,&E,&T->y);
105
106      //set ltq
107      Fp_set_ui(&ltq.x0.x0.x0,1);
108      Fp2_set_neg(&ltq.x1.x2,&C);
109      Fp2_inv_basis(&ltq.x1.x2,&ltq.x1.x2);
110      Fp2_mul_mpz(&ltq.x1.x1,&E,L->x0);
111      Fp2_inv_basis(&ltq.x1.x1,&ltq.x1.x1);
112
113      bls12_Pseudo_8_sparse_mul(f,f,&ltq);
114
115      EFp2_clear(&Tmp_T);
116      Fp12_clear(&ltq);
117      Fp2_clear(&A);
118      Fp2_clear(&B);
119      Fp2_clear(&C);
120      Fp2_clear(&D);
121  }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.6.2.2 bls12_ff_ltt()

```
void bls12_ff_ltt (
              Fp12 * f,
              EFp2 * T,
              EFp * P,
              Fp * L )
```

Calculate line equation ltt that goes through T and P in Miller's algo.

Parameters

out	*f*	- output in Fp12
in	*T*	- Input rational point T in Fp2.
in	*P*	- Input rational point P in mapped Fp.
in	*L*	- Input element L in mapped Fp after precomputation in Miller's algo.

References bls12_Pseudo_8_sparse_mul(), EFp2_clear(), EFp2_init(), EFp2_set(), and Fp_set_ui().

Referenced by bls12_Miller_algo_for_opt_ate(), and bls12_Miller_algo_for_plain_ate().

```
31                                                  {
32      EFp2 Tmp_T;
33      EFp2_init(&Tmp_T);
34      Fp12 ff,ltt;
35      Fp12_init(&ff);
36      Fp12_init(&ltt);
37      Fp2 A,B,C,D,E;
38      Fp2_init(&A);
39      Fp2_init(&B);
```

```
40      Fp2_init(&C);
41      Fp2_init(&D);
42      Fp2_init(&E);
43      EFp2_set(&Tmp_T,T);
44
45      Fp12_sqr(&ff,f);
46
47      //ltt
48      Fp2_add(&A,&Tmp_T.y,&Tmp_T.y);          //A=1/(2*T.y)
49      Fp2_inv(&A,&A);
50      Fp2_sqr(&B,&Tmp_T.x);              //B=3(T.x)^2
51      Fp2_mul_ui(&B,&B,3);
52      Fp2_mul(&C,&A,&B);                  //C=A*B
53      Fp2_add(&D,&Tmp_T.x,&Tmp_T.x);          //D=2T.x
54      Fp2_sqr(&T->x,&C);                  //next_T.x=C^2-D
55      Fp2_sub(&T->x,&T->x,&D);
56      Fp2_mul(&E,&C,&Tmp_T.x);              //E=C*T.x-T.y
57      Fp2_sub(&E,&E,&Tmp_T.y);
58      Fp2_mul(&T->y,&C,&T->x);              //next_T.y=E-C*next_T.x
59      Fp2_sub(&T->y,&E,&T->y);
60
61      //set ltt
62      Fp_set_ui(&ltt.x0.x0.x0,1);
63      Fp2_set_neg(&ltt.x1.x2,&C);
64      Fp2_inv_basis(&ltt.x1.x2,&ltt.x1.x2);
65      Fp2_mul_mpz(&ltt.x1.x1,&E,L->x0);
66      Fp2_inv_basis(&ltt.x1.x1,&ltt.x1.x1);
67
68      bls12_Pseudo_8_sparse_mul(f,&ff,&ltt);
69
70      EFp2_clear(&Tmp_T);
71      Fp2_clear(&A);
72      Fp2_clear(&B);
73      Fp2_clear(&C);
74      Fp2_clear(&D);
75      Fp2_clear(&E);
76      Fp12_clear(&ff);
77      Fp12_clear(&ltt);
78  }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.7 include/ELiPS_bn_bls/bls12_line_tate.h File Reference

`#include <ELiPS_bn_bls/bn_efp12.h>`
Include dependency graph for bls12_line_tate.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_ff_ltt_vtt_for_tate (Fp12 *f, EFp *T, EFp12 *Q)
- void bls12_f_ltp_vtp_for_tate (Fp12 *f, EFp *T, EFp *P, EFp12 *Q)

8.7.1 Detailed Description

Interaface for BLS12 line evaluation for tate pairing.

8.7.2 Function Documentation

8.7.2.1 bls12_f_ltp_vtp_for_tate()

```
void bls12_f_ltp_vtp_for_tate (
             Fp12 * f,
             EFp * T,
             EFp * P,
             EFp12 * Q )
```

Calculate line equation for tate pairing in BLS12 curve.

Parameters

out	*f*	- output in Fp12
in	*T*	- Input rational point T in Fp.
in	*P*	- Input rational point P in Fp.
in	*Q*	- Input rational point Q mapped in Fp22.

References EFp_clear(), EFp_init(), EFp_set(), Fp_clear(), Fp_init(), Fp_sub(), EFp::infinity, and EFp::y.

Referenced by bls12_Miller_algo_for_tate().

```
151                                                                 {
152     EFp Next_T;
153     EFp_init(&Next_T);
154     Fp12 tmp1,tmp2,tmp3;
155     Fp12_init(&tmp1);
156     Fp12_init(&tmp2);
157     Fp12_init(&tmp3);
158     Fp lambda;
159     Fp_init(&lambda);
160 
161     bls12_EFp_ECA_return_lambda(&Next_T,&lambda,T,P);
162 
163     switch(Next_T.infinity){
164         case 0:
165             Fp12_set(&tmp3,&Q->x);
166             Fp_sub(&tmp3.x0.x0.x0,&Q->x.x0.x0.x0,&P->x);
167             Fp12_mul_mpz(&tmp3,&tmp3,lambda.x0);
168             Fp12_set(&tmp1,&Q->y);
169             Fp_sub(&tmp1.x0.x0.x0,&Q->y.x0.x0.x0,&P->y);
170             Fp12_sub(&tmp1,&tmp1,&tmp3);
171 
172             Fp12_set(&tmp2,&Q->x);
173             Fp_sub(&tmp2.x0.x0.x0,&Q->x.x0.x0.x0,&Next_T.x);
174 
175             Fp12_inv(&tmp3,&tmp2);
176             Fp12_mul(&tmp2,f,&tmp1);
177             Fp12_mul(f,&tmp2,&tmp3);
178             break;
179         case 1:
180             Fp12_set(&tmp1,&Q->x);
181             Fp_sub(&tmp1.x0.x0.x0,&Q->x.x0.x0.x0,&T->x);
182             Fp12_mul(f,f,&tmp1);
183             break;
184         default:
185             break;
186     }
187     EFp_set(T,&Next_T);
188 
189     EFp_clear(&Next_T);
190     Fp12_clear(&tmp1);
191     Fp12_clear(&tmp2);
192     Fp12_clear(&tmp3);
193     Fp_clear(&lambda);
194 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.7.2.2 bls12_ff_ltt_vtt_for_tate()

```
void bls12_ff_ltt_vtt_for_tate (
            Fp12 * f,
            EFp * T,
            EFp12 * Q )
```

Calculate line equation for tate pairing in BLS12 curve.

Parameters

out	*f*	- output in Fp12
in	*T*	- Input rational point T in Fp.
in	*Q*	- Input rational point Q mapped in Fp22.

References EFp_clear(), EFp_init(), EFp_set(), Fp_clear(), Fp_init(), Fp_sub(), EFp::infinity, and EFp::y.

Referenced by bls12_Miller_algo_for_tate().

```
106                                                     {
107     EFp Next_T;
108     EFp_init(&Next_T);
109     Fp12 tmp1,tmp2,tmp3;
110     Fp12_init(&tmp1);
111     Fp12_init(&tmp2);
112     Fp12_init(&tmp3);
113     Fp lambda;
114     Fp_init(&lambda);
115 
116     bls12_EFp_ECD_return_lambda(&Next_T,&lambda,T);
117     switch(Next_T.infinity){
118         case 0:
119             Fp12_sqr(&tmp1,f);
120             Fp12_set(&tmp3,&Q->x);
121             Fp_sub(&tmp3.x0.x0.x0,&Q->x.x0.x0.x0,&T->x);
122             Fp12_mul_mpz(&tmp3,&tmp3,lambda.x0);
123             Fp12_set(&tmp2,&Q->y);
124             Fp_sub(&tmp2.x0.x0.x0,&Q->y.x0.x0.x0,&T->y);
125             Fp12_sub(&tmp2,&tmp2,&tmp3);
126 
127             Fp12_set(&tmp3,&Q->x);
128             Fp_sub(&tmp3.x0.x0.x0,&Q->x.x0.x0.x0,&Next_T.x);
129             Fp12_inv(&tmp3,&tmp3);
130             Fp12_mul(&tmp1,&tmp1,&tmp2);
131             Fp12_mul(f,&tmp1,&tmp3);
132             break;
133         case 1:
134             Fp12_sqr(&tmp1,f);
135             Fp12_set(&tmp2,&Q->x);
136             Fp_sub(&tmp2.x0.x0.x0,&Q->x.x0.x0.x0,&T->x);
137             Fp12_mul(f,&tmp1,&tmp2);
138             break;
139         default:
140             break;
141     }
142     EFp_set(T,&Next_T);
143 
144     EFp_clear(&Next_T);
145     Fp12_clear(&tmp1);
146     Fp12_clear(&tmp2);
147     Fp12_clear(&tmp3);
148     Fp_clear(&lambda);
149 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.8 include/ELiPS_bn_bls/bls12_miller_ate.h File Reference

```
#include <ELiPS_bn_bls/bls12_twist.h>
#include <ELiPS_bn_bls/bls12_p8sparse.h>
#include <ELiPS_bn_bls/bls12_line_ate.h>
```

Include dependency graph for bls12_miller_ate.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_Miller_algo_for_plain_ate (Fp12 *ANS, EFp12 *Q, EFp12 *P)

8.8.1 Detailed Description

Interaface for BLS12 Miller's algo for unoptimized version of ate pairing.

8.8.2 Function Documentation

8.8.2.1 bls12_Miller_algo_for_plain_ate()

```
void bls12_Miller_algo_for_plain_ate (
            Fp12 * ANS,
            EFp12 * Q,
            EFp12 * P )
```

Calculate Miller's loop for ate pairing in BLS12 curve.

Parameters

out	*ANS*	- output in Fp12
in	*Q*	- Input rational point Q in G2.
in	*P*	- Input rational point P in G1.

References bls12_EFp12_to_EFp(), bls12_EFp12_to_EFp2(), bls12_f_ltq(), bls12_ff_ltt(), bls12_Pseudo_8_sparse_mapping(), curve_parameters, EFp12_clear(), EFp12_init(), EFp2_clear(), EFp2_init(), EFp2_set(), EFp2_set_neg(), EFp_clear(), EFp_init(), Fp_clear(), Fp_init(), Fp_set_ui(), and curve_params::trace_t.

Referenced by bls12_plain_ate().

```
31                                                                  {
32     EFp12 Buf;
33     EFp12_init(&Buf);
34     EFp2 T;
35     EFp2_init(&T);
36     EFp2 mapped_Q,mapped_Q_neg,mapped_Q1,mapped_Q2_neg;
37     EFp2_init(&mapped_Q);
38     EFp2_init(&mapped_Q_neg);
39     EFp2_init(&mapped_Q1);
40     EFp2_init(&mapped_Q2_neg);
```

```
41      EFp mapped_P;
42      EFp_init(&mapped_P);
43      Fp12 f;
44      Fp12_init(&f);
45      Fp L;
46      Fp_init(&L);
47      int i,length;
48      mpz_t loop;
49      mpz_init(loop);
50
51      //set
52      bls12_EFp12_to_EFp(&mapped_P,P);  //set P
53      bls12_EFp12_to_EFp2(&mapped_Q,Q);//set mapped_Q
54      bls12_Pseudo_8_sparse_mapping(&mapped_P,&mapped_Q,&L);
55      EFp2_set_neg(&mapped_Q_neg,&mapped_Q);//set mapped_Q_neg
56
57      EFp2_set(&T,&mapped_Q_neg);    //set T
58      Fp12_set_ui(&f,0);            //set f
59      Fp_set_ui(&f.x0.x0.x0,1);
60
61      mpz_sub_ui(loop,curve_parameters.trace_t,1);
62      mpz_neg(loop,loop);
63
64      length=(int)mpz_sizeinbase(loop,2);
65      char binary[length];
66      mpz_get_str(binary,2,loop);
67
68      //miller
69      for(i=1; i<length; i++){
70          bls12_ff_ltt(&f,&T,&mapped_P,&L);
71          if(binary[i]=='1'){
72              bls12_f_ltq(&f,&T,&mapped_Q_neg,&mapped_P,&L);
73          }
74      }
75
76      Fp12_set(ANS,&f);
77
78      mpz_clear(loop);
79      EFp12_clear(&Buf);
80      Fp12_clear(&f);
81      EFp2_clear(&T);
82      EFp2_clear(&mapped_Q);
83      EFp2_clear(&mapped_Q_neg);
84      EFp2_clear(&mapped_Q1);
85      EFp2_clear(&mapped_Q2_neg);
86      EFp_clear(&mapped_P);
87      Fp_clear(&L);
88  }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.9 include/ELiPS_bn_bls/bls12_miller_optate.h File Reference

```
#include <ELiPS_bn_bls/bls12_twist.h>
#include <ELiPS_bn_bls/bls12_p8sparse.h>
#include <ELiPS_bn_bls/bls12_line_ate.h>
```

Include dependency graph for bls12_miller_optate.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_Miller_algo_for_opt_ate (Fp12 *ANS, EFp12 *Q, EFp12 *P)

8.9.1 Detailed Description

Interaface for BLS12 Miller's algo of optimal-ate pairing.

8.9.2 Function Documentation

8.9.2.1 bls12_Miller_algo_for_opt_ate()

```
void bls12_Miller_algo_for_opt_ate (
            Fp12 * ANS,
            EFp12 * Q,
            EFp12 * P )
```

Calculate Miller's loop for optimal ate pairing in BLS12 curve.

Parameters

out	*ANS*	- output in Fp12
in	*Q*	- Input rational point Q in G2.
in	*P*	- Input rational point P in G1.

References bls12_EFp12_to_EFp(), bls12_EFp12_to_EFp2(), bls12_f_ltq(), bls12_ff_ltt(), bls12_Pseudo_8_sparse_mapping(), bls12_X_binary, bls12_X_length, EFp2_clear(), EFp2_init(), EFp2_set(), EFp2_set_neg(), EFp_clear(), EFp_init(), Fp_clear(), Fp_init(), and Fp_set_ui().

Referenced by bls12_opt_ate().

```
31                                                                  {
32     EFp2 T;
33     EFp2_init(&T);
34     EFp2 mapped_Q,mapped_Q_neg,mapped_Q1,mapped_Q2_neg;
35     EFp2_init(&mapped_Q);
36     EFp2_init(&mapped_Q_neg);
37     EFp2_init(&mapped_Q1);
38     EFp2_init(&mapped_Q2_neg);
39     EFp mapped_P;
40     EFp_init(&mapped_P);
41     Fp12 f;
42     Fp12_init(&f);
43     Fp L;
44     Fp_init(&L);
45     int i;
46
47     //set
48     bls12_EFp12_to_EFp(&mapped_P,P);  //set P
49     bls12_EFp12_to_EFp2(&mapped_Q,Q);//set mapped_Q
50     bls12_Pseudo_8_sparse_mapping(&mapped_P,&mapped_Q,&L);
51
52     EFp2_set_neg(&mapped_Q_neg,&mapped_Q);//set mapped_Q_neg
53
54     EFp2_set(&T,&mapped_Q_neg);        //set T
55     Fp12_set_ui(&f,0);             //set f
56     Fp_set_ui(&f.x0.x0.x0,1);
57     //miller
58     for(i=bls12_X_length-1; i>=0; i--){
59         switch(bls12_X_binary[i]){
60             case 0:
61                 bls12_ff_ltt(&f,&T,&mapped_P,&L);
62                 break;
63             case 1:
64                 bls12_ff_ltt(&f,&T,&mapped_P,&L);
65                 bls12_f_ltq(&f,&T,&mapped_Q,&mapped_P,&L);
66                 break;
67             case -1:
68                 bls12_ff_ltt(&f,&T,&mapped_P,&L);
69                 bls12_f_ltq(&f,&T,&mapped_Q_neg,&mapped_P,&L);
70                 break;
71             default:
72                 break;
```

```
73         }
74
75     }
76
77     Fp12_set(ANS,&f);
78
79     Fp12_clear(&f);
80     EFp2_clear(&T);
81     EFp2_clear(&mapped_Q);
82     EFp2_clear(&mapped_Q_neg);
83     EFp2_clear(&mapped_Q1);
84     EFp2_clear(&mapped_Q2_neg);
85     EFp_clear(&mapped_P);
86     Fp_clear(&L);
87 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.10 include/ELiPS_bn_bls/bls12_miller_tate.h File Reference

```
#include <ELiPS_bn_bls/bls12_line_tate.h>
#include <ELiPS_bn_bls/bls12_twist.h>
```

Include dependency graph for bls12_miller_tate.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_Miller_algo_for_tate (Fp12 *ANS, EFp12 *Q, EFp12 *P)

8.10.1 Detailed Description

Interaface for BLS12 Miller's algo of tate pairing.

8.10.2 Function Documentation

8.10.2.1 bls12_Miller_algo_for_tate()

```
void bls12_Miller_algo_for_tate (
            Fp12 * ANS,
            EFp12 * Q,
            EFp12 * P )
```

Calculate Miller's loop for tate pairing in BLS12 curve.

Parameters

out	*ANS*	- output in Fp12
in	*Q*	- Input rational point Q in G2.
in	*P*	- Input rational point P in G1.

References bls12_EFp12_to_EFp(), bls12_f_ltp_vtp_for_tate(), bls12_ff_ltt_vtt_for_tate(), curve_parameters, E↩ Fp_clear(), EFp_init(), EFp_set(), Fp_set_ui(), and curve_params::order.

Referenced by bls12_tate().

```
32                                                          {
33     EFp Tmp_P,T;
34     EFp_init(&Tmp_P);
35     EFp_init(&T);
36     Fp12 f;
37     Fp12_init(&f);
38     int i,length;
39     length=(int)mpz_sizeinbase(curve_parameters.order,2);
40     char binary[length];
41     mpz_get_str(binary,2,curve_parameters.order);
42 
43     //set
44     bls12_EFp12_to_EFp(&Tmp_P,P);
45     EFp_set(&T,&Tmp_P);
46     Fp12_set_ui(&f,0);
47     Fp_set_ui(&f.x0.x0.x0,1);
48 
49     //miller
50     for(i=1; i<length; i++){
51         bls12_ff_ltt_vtt_for_tate(&f,&T,Q);
52         if(binary[i]=='1'){
53             bls12_f_ltp_vtp_for_tate(&f,&T,&Tmp_P,Q);
54         }
55     }
56     Fp12_set(ANS,&f);
57 
58     Fp12_clear(&f);
59     EFp_clear(&T);
60     EFp_clear(&Tmp_P);
61 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.11 include/ELiPS_bn_bls/bls12_p8sparse.h File Reference

`#include <ELiPS_bn_bls/bn_efp2.h>`
Include dependency graph for bls12_p8sparse.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_Pseudo_8_sparse_mapping (EFp ∗P, EFp2 ∗Q, Fp ∗L)
- void bls12_Pseudo_8_sparse_mul (Fp12 ∗ANS, Fp12 ∗A, Fp12 ∗B)

8.11.1 Detailed Description

Interaface for BLS12 Pseudo 8-sparse multplication in Miller's algo of tate pairing https://eprint.iacr.↩ org/2017/1174.pdf

8.11.2 Function Documentation

8.11.2.1 bls12_Pseudo_8_sparse_mapping()

```
void bls12_Pseudo_8_sparse_mapping (
            EFp * P,
            EFp2 * Q,
            Fp * L )
```

Calculate Precomputations for Pseudo 8-sparse multplication and Millers algo.

Parameters

in	*P*	- P in G1
in	*Q*	- Q in G2.
in	*L*	- L in Fp from Miller's algo.

References EFp2_clear(), EFp2_init(), EFp2_set(), EFp_clear(), EFp_init(), EFp_set(), Fp_clear(), Fp_init(), Fp_↩ inv(), Fp_mul(), Fp_set(), and EFp::y.

Referenced by bls12_Miller_algo_for_opt_ate(), and bls12_Miller_algo_for_plain_ate().

```
31                                                          {
32     EFp2 Tmp_Q;
33     EFp2_init(&Tmp_Q);
34     EFp Tmp_P;
35     EFp_init(&Tmp_P);
36     Fp A,B,C,D,c;
37     Fp_init(&A);
38     Fp_init(&B);
39     Fp_init(&C);
40     Fp_init(&D);
41     Fp_init(&c);
42
43     EFp_set(&Tmp_P,P);
44     EFp2_set(&Tmp_Q,Q);
45
46     Fp_mul(&A,&Tmp_P.x,&Tmp_P.y);
47     Fp_inv(&A,&A);
48     Fp_mul(&B,&Tmp_P.x,&Tmp_P.x);
49     Fp_mul(&B,&B,&A);
50     Fp_mul(&C,&Tmp_P.y,&A);
51     Fp_mul(&D,&B,&B);
52
53     Fp2_mul_mpz(&Q->x,&Tmp_Q.x,D.x0);
54     Fp_mul(&c,&B,&D);
55     Fp2_mul_mpz(&Q->y,&Tmp_Q.y,c.x0);
56
57     Fp_mul(&P->x,&D,&Tmp_P.x);
58     Fp_set(&P->y,&P->x);
59
60     Fp_mul(L,&C,&Tmp_P.y);
61     Fp_mul(L,L,L);
62     Fp_mul(L,L,&C);
63
64
65     EFp2_clear(&Tmp_Q);
66     EFp_clear(&Tmp_P);
67     Fp_clear(&A);
68     Fp_clear(&B);
69     Fp_clear(&C);
70     Fp_clear(&D);
71     Fp_clear(&c);
72 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.11.2.2 bls12_Pseudo_8_sparse_mul()

```
void bls12_Pseudo_8_sparse_mul (
             Fp12 * ANS,
             Fp12 * A,
             Fp12 * B )
```

Calculate Precomputations for Pseudo 8-sparse multplication and Millers algo.

Parameters

out	*ANS*	- output of Pseudo 8-sparse multplication
in	*A*	- Fp12 vector.
in	*B*	- Fp12 vector.

Referenced by bls12_f_ltq(), and bls12_ff_ltt().

```
74                                                          {
75     //A= f0 + f1^2 + f2^4 + f3+ f4^3 + f5^5
76     //B= 1  +                        a^3 +  b^5
77     // x0.x0  x0.x1   x0.x2  x1.x0  x1.x1   x1.x2
78     Fp12 ans;
79     Fp12_init(&ans);
80     Fp2 tmp0,tmp1,tmp2,tmp3;
81     Fp2_init(&tmp0);
82     Fp2_init(&tmp1);
83     Fp2_init(&tmp2);
84     Fp2_init(&tmp3);
85
86     Fp2_mul(&tmp0,&A->x0.x0,&B->x1.x1);        //tmp0←a*f0
87     Fp2_mul(&tmp1,&A->x0.x1,&B->x1.x2);        //tmp1←b*f1
88     Fp2_add(&tmp2,&A->x0.x0,&A->x0.x1);        //tmp2←f0+f1
89     Fp2_add(&tmp3,&B->x1.x1,&B->x1.x2);        //tmp3←a+b
90     Fp2_mul(&tmp2,&tmp2,&tmp3);             //tmp2←tmp2*tmp3
91     Fp2_sub(&tmp2,&tmp2,&tmp0);             //tmp2←tmp2-tmp0
92     Fp2_sub(&tmp2,&tmp2,&tmp1);             //tmp2←tmp2-tmp1
93
94     Fp2_add(&ans.x1.x2,&tmp2,&A->x1.x2);    //ans[^5]←tmp2+f5
95     Fp2_add(&ans.x1.x1,&tmp0,&A->x1.x1);    //ans[^3]←tmp0+f4
96     Fp2_mul(&tmp2,&A->x0.x2,&B->x1.x2);        //tmp2←b*f2
97     Fp2_mul_basis(&tmp2,&tmp2);             //tmp2←tmp2*
98     Fp2_add(&ans.x1.x1,&ans.x1.x1,&tmp2);    //ans[^3]←ans[^3]+tmp2
99     Fp2_mul(&tmp0,&A->x0.x2,&B->x1.x1);        //tmp0←a*f2
100     Fp2_add(&tmp0,&tmp0,&tmp1);             //tmp0←tmp0+tmp1
101     Fp2_mul_basis(&tmp0,&tmp0);             //tmp0←tmp0*
102     Fp2_add(&ans.x1.x0,&tmp0,&A->x1.x0);    //ans[]←tmp0+f3
103
104     Fp2_mul(&tmp0,&A->x1.x0,&B->x1.x1);        //tmp0←a*f3
105     Fp2_mul(&tmp1,&A->x1.x1,&B->x1.x2);        //tmp1←b*f4
106     Fp2_add(&tmp2,&A->x1.x0,&A->x1.x1);        //tmp2←f3+f4
107     Fp2_mul(&tmp2,&tmp2,&tmp3);             //tmp2←tmp2+tmp3
108     Fp2_sub(&tmp2,&tmp2,&tmp0);             //tmp2←tmp2-tmp0
109     Fp2_sub(&tmp2,&tmp2,&tmp1);             //tmp2←tmp2-tmp1
110
111     Fp2_mul_basis(&tmp2,&tmp2);             //tmp2←tmp2*
112     Fp2_add(&ans.x0.x0,&tmp2,&A->x0.x0);    //ans[1]←tmp2+f0
113
114     Fp2_mul(&tmp2,&A->x1.x2,&B->x1.x1);        //tmp2←a*f5
115     Fp2_add(&tmp2,&tmp1,&tmp2);             //tmp2←tmp1+tmp2
116     Fp2_mul_basis(&tmp2,&tmp2);             //tmp2←tmp2*
117     Fp2_add(&ans.x0.x1,&tmp2,&A->x0.x1);    //ans[^2]←tmp2+f1
118     Fp2_mul(&tmp3,&A->x1.x2,&B->x1.x2);        //tmp3←b*f5
119     Fp2_mul_basis(&tmp3,&tmp3);             //tmp3←tmp3*
120
121     Fp2_add(&tmp0,&tmp0,&tmp3);             //tmp0←tmp0+tmp3
122     Fp2_add(&ans.x0.x2,&tmp0,&A->x0.x2);    //ans[^2]←tmp0+f2
123
124     Fp12_set(ANS,&ans);
125
126     Fp12_clear(&ans);
127     Fp2_clear(&tmp0);
128     Fp2_clear(&tmp1);
129     Fp2_clear(&tmp2);
130     Fp2_clear(&tmp3);
131 }
```

Here is the caller graph for this function:

8.12 include/ELiPS_bn_bls/bls12_pairings.h File Reference

```
#include <ELiPS_bn_bls/bls12_miller_ate.h>
#include <ELiPS_bn_bls/bls12_miller_tate.h>
#include <ELiPS_bn_bls/bls12_miller_optate.h>
#include <ELiPS_bn_bls/bls12_finalexp.h>
#include <ELiPS_bn_bls/bls12_timeprint.h>
#include <ELiPS_bn_bls/bn_utils.h>
```

Include dependency graph for bls12_pairings.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_tate (Fp12 *ANS, EFp12 *P, EFp12 *Q)
- void bls12_plain_ate (Fp12 *ANS, EFp12 *P, EFp12 *Q)
- void bls12_opt_ate (Fp12 *ANS, EFp12 *P, EFp12 *Q)

8.12.1 Detailed Description

Interaface for BLS12 Pairing implementations

8.12.2 Function Documentation

8.12.2.1 bls12_opt_ate()

```
void bls12_opt_ate (
            Fp12 * ANS,
            EFp12 * P,
            EFp12 * Q )
```

Calculate optimal ate pairing in BLS12 curve.

Parameters

out	*ANS*	- output
in	*P*	- P in G1 mapped to Fp12.
in	*Q*	- Q in G2.

References bls12_Miller_algo_for_opt_ate().

Referenced by bls12_test_G3_exp(), and bls12_test_opt_ate_pairing().

```
59                                                  {
60     //miller
61     gettimeofday(&t0,NULL);
62     bls12_Miller_algo_for_opt_ate(ANS,P,Q);
63     gettimeofday(&t1,NULL);
64     bls12_MILLER_OPTATE=timedifference_msec(t0,t1);
65
66     //final exp
67     gettimeofday(&t0,NULL);
68     bls12_finalexp_optimal(ANS,ANS);
69     gettimeofday(&t1,NULL);
70     bls12_FINALEXP_OPT=timedifference_msec(t0,t1);
71 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.12.2.2 bls12_plain_ate()

```
void bls12_plain_ate (
            Fp12 * ANS,
            EFp12 * P,
            EFp12 * Q )
```

Calculate ate pairing in BLS12 curve.

Parameters

out	*ANS*	- output
in	*P*	- P in G1 mapped to Fp12.
in	*Q*	- Q in G2.

References bls12_Miller_algo_for_plain_ate().

Referenced by bls12_test_plain_ate_pairing().

```
45                                                  {
46     //miller
47     gettimeofday(&t0,NULL);
48     bls12_Miller_algo_for_plain_ate(ANS,P,Q);
49     gettimeofday(&t1,NULL);
50     bls12_MILLER_PLAINATE=timedifference_msec(t0,t1);
51
52     //final exp
53     gettimeofday(&t0,NULL);
54     bls12_finalexp_optimal(ANS,ANS);
55     gettimeofday(&t1,NULL);
56     bls12_FINALEXP_OPT=timedifference_msec(t0,t1);
57 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.12.2.3 bls12_tate()

```
void bls12_tate (
            Fp12 * ANS,
            EFp12 * P,
            EFp12 * Q )
```

Calculate tate pairing in BLS12 curve.

Parameters

out	*ANS*	- output
in	*P*	- P in G1 mapped to Fp12.
in	*Q*	- Q in G2.

References bls12_finalexp_optimal(), bls12_Miller_algo_for_tate(), and bls12_MILLER_TATE.

Referenced by bls12_test_tate_pairing().

```
31                                          {
```

```
32     //miller
33     gettimeofday(&t0,NULL);
34     bls12_Miller_algo_for_tate(ANS,P,Q);
35     gettimeofday(&t1,NULL);
36     bls12_MILLER_TATE=timedifference_msec(t0,t1);
37
38     //final exp
39     gettimeofday(&t0,NULL);
40     bls12_finalexp_optimal(ANS,ANS);
41     gettimeofday(&t1,NULL);
42     bls12_FINALEXP_OPT=timedifference_msec(t0,t1);
43 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.13 include/ELiPS_bn_bls/bls12_scm.h File Reference

```
#include <ELiPS_bn_bls/bn_efp12.h>
#include <ELiPS_bn_bls/bn_utils.h>
#include <ELiPS_bn_bls/bls12_timeprint.h>
#include <ELiPS_bn_bls/bls12_twist.h>
#include <ELiPS_bn_bls/bls12_skew_frobenius.h>
```

Include dependency graph for bls12_scm.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_plain_G1_scm (EFp12 *ANS, EFp12 *P, mpz_t scalar)
- void bls12_2split_G1_scm (EFp12 *ANS, EFp12 *P, mpz_t scalar)
- void bls12_plain_G2_scm (EFp12 *ANS, EFp12 *Q, mpz_t scalar)
- void bls12_2split_G2_scm (EFp12 *ANS, EFp12 *Q, mpz_t scalar)
- void bls12_4split_G2_scm (EFp12 *ANS, EFp12 *Q, mpz_t scalar)

8.13.1 Detailed Description

Interaface for BLS12 scalar multiplication implementations in G1 and G2

8.13.2 Function Documentation

8.13.2.1 bls12_2split_G1_scm()

```
void bls12_2split_G1_scm (
            EFp12 * ANS,
            EFp12 * P,
            mpz_t scalar )
```

Calculate scalar multiplication G1 in BLS12 curve splitting of scallar into two parts.

Parameters

out	*ANS*	- output
in	*P*	- P in G1 mapped to Fp12.
in	*s*	- gmp mpz_t type big integer.

References bls12_EFp12_to_EFp(), bls12_EFp_skew_frobenius_map_p2(), EFp12_init(), EFp_ECA(), EFp_init(), EFp_set(), and EFp::infinity.

Referenced by bls12_test_G1_scm(), and bls12_test_G3_exp().

```
47                                                          {
48      gettimeofday(&t0,NULL);
49
50      int i,length_s[2],loop_length;
51      EFp12 Buf;
52      EFp12_init(&Buf);
53      EFp next_P,tmp_P,skew_P_2;
54      EFp_init(&next_P);
55      EFp_init(&tmp_P);
56      EFp_init(&skew_P_2);
57      mpz_t s[2],buf;
58      mpz_init(buf);
59      for(i=0; i<2; i++){
60          mpz_init(s[i]);
61      }
62      //table
63      EFp table[4];
64      for(i=0; i<4; i++){
65          EFp_init(&table[i]);
66      }
67
68      //set
69      bls12_EFp12_to_EFp(&tmp_P,P);
70      bls12_EFp_skew_frobenius_map_p2(&skew_P_2,&tmp_P);
71      //set table
72      table[0].infinity=1;     //00
73      EFp_set(&table[1],&tmp_P);     //01
74      EFp_set(&table[2],&skew_P_2);     //10
75      EFp_ECA(&table[3],&tmp_P,&skew_P_2);     //11
76
77      //s0,s1
78      mpz_neg(buf,bls12_X);
79      mpz_pow_ui(buf,buf,2);
80      mpz_tdiv_qr(s[1],s[0],scalar,buf);
81      //binary
82      loop_length=0;
83      for(i=0; i<2; i++){
84          length_s[i]=(int)mpz_sizeinbase(s[i],2);
85          if(loop_length<length_s[i]){
86              loop_length=length_s[i];
87          }
88      }
89      //set binary
90      char binary_s[2][loop_length+1];
91      char str[5],*e;
92      int binary[loop_length+1];
93      for(i=0; i<2; i++){
94          if(length_s[i]==loop_length){
95              mpz_get_str(binary_s[i],2,s[i]);
96          }else{
97              char binary_buf[loop_length+1];
98              mpz_get_str(binary_buf,2,s[i]);
99              memset(binary_s[i],'0',sizeof(binary_s[i]));
100              memmove(binary_s[i]+loop_length-length_s[i],binary_buf,sizeof(binary_buf));
101          }
102      }
103      for(i=0; i<loop_length; i++){
104          sprintf(str,"%c%c",binary_s[1][i],binary_s[0][i]);
105          binary[i]=(int)strtol(str,&e,2);
106      }
107      EFp_set(&next_P,&table[binary[0]]);
108
109      //SCM
110      for(i=1; i<loop_length; i++){
111          EFp_ECD(&next_P,&next_P);
112          EFp_ECA(&next_P,&next_P,&table[binary[i]]);
113      }
114
```

```
115     bls12_EFp_to_EFp12(ANS,&next_P);
116
117
118     EFp12_clear(&Buf);
119     EFp_clear(&next_P);
120     EFp_clear(&tmp_P);
121     EFp_clear(&skew_P_2);
122     mpz_clear(buf);
123     for(i=0; i<2; i++){
124         mpz_clear(s[i]);
125     }
126     //table
127     for(i=0; i<4; i++){
128         EFp_clear(&table[i]);
129     }
130
131     gettimeofday(&t1,NULL);
132     bls12_G1SCM_2SPLIT=timedifference_msec(t0,t1);
133 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.13.2.2 bls12_2split_G2_scm()

```
void bls12_2split_G2_scm (
              EFp12 * ANS,
              EFp12 * Q,
              mpz_t scalar )
```

Calculate scalar multiplication G1 in BLS12 curve splitting of scallar into 2 parts.

Parameters

out	*ANS*	- output
in	*P*	- P in G1 mapped to Fp12.
in	*s*	- gmp mpz_t type big integer.

References bls12_EFp12_to_EFp2(), bls12_EFp2_skew_frobenius_map_p2(), EFp2_ECA(), EFp2_init(), and E↩ Fp2_set().

Referenced by bls12_test_G2_scm().

```
151                                                     {
152     gettimeofday(&t0,NULL);
153
154     int i,length_s[2],loop_length;
155     EFp2 next_twisted_Q,twisted_Q,twisted_Q_2x;
156     EFp2_init(&next_twisted_Q);
157     EFp2_init(&twisted_Q);
158     EFp2_init(&twisted_Q_2x);
159     mpz_t s[2],buf;
160     mpz_init(buf);
161     for(i=0; i<2; i++){
162         mpz_init(s[i]);
163     }
164     //table
165     EFp2 table[4];
166     for(i=0; i<4; i++){
167         EFp2_init(&table[i]);
168     }
169
170     //set
171     bls12_EFp12_to_EFp2(&twisted_Q,Q);                //twisted_Q
172     bls12_EFp2_skew_frobenius_map_p2(&twisted_Q_2x,&twisted_Q);//
    twisted_Q_2x
173     //set table
174     table[0].infinity=1;                          //00
175     EFp2_set(&table[1],&twisted_Q);              //01
```

```
176     EFp2_set(&table[2],&twisted_Q_2x);            //10
177     EFp2_ECA(&table[3],&twisted_Q,&twisted_Q_2x);    //11
178
179     //s0,s1
180     mpz_neg(buf,bls12_X);
181     mpz_pow_ui(buf,buf,2);
182     mpz_tdiv_qr(s[1],s[0],scalar,buf);
183     //binary
184     loop_length=0;
185     for(i=0; i<2; i++){
186         length_s[i]=(int)mpz_sizeinbase(s[i],2);
187         if(loop_length<length_s[i]){
188             loop_length=length_s[i];
189         }
190     }
191     //set binary
192     char binary_s[2][loop_length+1];
193     char str[5],*e;
194     int binary[loop_length+1];
195     for(i=0; i<2; i++){
196         if(length_s[i]==loop_length){
197             mpz_get_str(binary_s[i],2,s[i]);
198         }else{
199             char binary_buf[loop_length+1];
200             mpz_get_str(binary_buf,2,s[i]);
201             memset(binary_s[i],'0',sizeof(binary_s[i]));
202             memmove(binary_s[i]+loop_length-length_s[i],binary_buf,sizeof(binary_buf));
203         }
204     }
205     for(i=0; i<loop_length; i++){
206         sprintf(str,"%c%c",binary_s[1][i],binary_s[0][i]);
207         binary[i]=(int)strtol(str,&e,2);
208     }
209     EFp2_set(&next_twisted_Q,&table[binary[0]]);
210
211     //SCM
212     for(i=1; i<loop_length; i++){
213         EFp2_ECD(&next_twisted_Q,&next_twisted_Q);
214         EFp2_ECA(&next_twisted_Q,&next_twisted_Q,&table[binary[i]]);
215     }
216
217     bls12_EFp2_to_EFp12(ANS,&next_twisted_Q);
218
219     mpz_clear(buf);
220     EFp2_clear(&next_twisted_Q);
221     EFp2_clear(&twisted_Q);
222     EFp2_clear(&twisted_Q_2x);
223     for(i=0; i<2; i++){
224         mpz_clear(s[i]);
225     }
226     for(i=0; i<4; i++){
227         EFp2_clear(&table[i]);
228     }
229
230     gettimeofday(&t1,NULL);
231     bls12_G2SCM_2SPLIT=timedifference_msec(t0,t1);
232 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.13.2.3 bls12_4split_G2_scm()

```
void bls12_4split_G2_scm (
            EFp12 * ANS,
            EFp12 * Q,
            mpz_t scalar )
```

Calculate scalar multiplication G1 in BLS12 curve splitting of scallar into 4 parts.

Parameters

out	*ANS*	- output
in	*P*	- P in G1 mapped to Fp12.
in	*s*	- gmp mpz_t type big integer.

References bls12_EFp12_to_EFp2(), bls12_EFp2_skew_frobenius_map_p1(), bls12_EFp2_skew_frobenius_map_p2(), bls12_EFp2_skew_frobenius_map_p3(), EFp2_ECA(), EFp2_init(), EFp2_set(), and EFp2_set_neg().

Referenced by bls12_test_G2_scm(), and bls12_test_G3_exp().

```
234                                                     {
235     gettimeofday(&t0,NULL);
236 
237     int i,length_s[4],loop_length;
238     EFp2 next_twisted_Q,twisted_Q,twisted_Q_x,twisted_Q_2x,twisted_Q_3x;
239     EFp2_init(&next_twisted_Q);
240     EFp2_init(&twisted_Q);
241     EFp2_init(&twisted_Q_x);
242     EFp2_init(&twisted_Q_2x);
243     EFp2_init(&twisted_Q_3x);
244     mpz_t A,B,s[4],x_2,x_1;
245     mpz_init(A);
246     mpz_init(B);
247     mpz_init(x_1);
248     mpz_init(x_2);
249     for(i=0; i<4; i++){
250         mpz_init(s[i]);
251     }
252     //table
253     EFp2 table[16];
254     for(i=0; i<16; i++){
255         EFp2_init(&table[i]);
256     }
257 
258     //set twisted_Q
259     bls12_EFp12_to_EFp2(&twisted_Q,Q);                    //twisted_Q
260     bls12_EFp2_skew_frobenius_map_p1(&twisted_Q_x,&twisted_Q);        //
    twisted_Q_x
261     EFp2_set_neg(&twisted_Q_x,&twisted_Q_x);
262     bls12_EFp2_skew_frobenius_map_p2(&twisted_Q_2x,&twisted_Q);    //
    twisted_Q_2x
263     bls12_EFp2_skew_frobenius_map_p3(&twisted_Q_3x,&twisted_Q);    //
    twisted_Q_3x
264     EFp2_set_neg(&twisted_Q_3x,&twisted_Q_3x);
265 
266     //set table
267     table[0].infinity=1;                            //0000
268     EFp2_set(&table[1],&twisted_Q);                //0001
269     EFp2_set(&table[2],&twisted_Q_x);                //0010
270     EFp2_ECA(&table[3],&twisted_Q_x,&twisted_Q);        //0011
271     EFp2_set(&table[4],&twisted_Q_2x);                //0100
272     EFp2_ECA(&table[5],&twisted_Q_2x,&twisted_Q);        //0101
273     EFp2_ECA(&table[6],&twisted_Q_2x,&twisted_Q_x);        //0110
274     EFp2_ECA(&table[7],&table[6],&twisted_Q);            //0111
275     EFp2_set(&table[8],&twisted_Q_3x);                //1000
276     EFp2_ECA(&table[9],&twisted_Q_3x,&twisted_Q);        //1001
277     EFp2_ECA(&table[10],&twisted_Q_3x,&twisted_Q_x);    //1010
278     EFp2_ECA(&table[11],&twisted_Q_3x,&table[3]);        //1011
279     EFp2_ECA(&table[12],&twisted_Q_3x,&twisted_Q_2x);    //1100
280     EFp2_ECA(&table[13],&table[12],&twisted_Q);        //1101
281     EFp2_ECA(&table[14],&table[12],&twisted_Q_x);        //1110
282     EFp2_ECA(&table[15],&table[14],&twisted_Q);        //1111
283 
284     //set
285     //s0,s1,s2,s3
286     mpz_neg(x_1,bls12_X);
287     mpz_mul(x_2,x_1,x_1);
288     mpz_tdiv_qr(B,A,scalar,x_2);
289     mpz_tdiv_qr(s[1],s[0],A,x_1);
290     mpz_tdiv_qr(s[3],s[2],B,x_1);
291 
292     //binary
293     loop_length=0;
294     for(i=0; i<4; i++){
295         length_s[i]=(int)mpz_sizeinbase(s[i],2);
296         if(loop_length<length_s[i]){
297             loop_length=length_s[i];
298         }
299     }
300     //set binary
301     char binary_s[4][loop_length+1];
302     char str[5],*e;
303     int binary[loop_length+1];
304     for(i=0; i<4; i++){
305         if(length_s[i]==loop_length){
306             mpz_get_str(binary_s[i],2,s[i]);
307         }else{
308             char binary_buf[loop_length+1];
309             mpz_get_str(binary_buf,2,s[i]);
```

```
310             memset(binary_s[i],'0',sizeof(binary_s[i]));
311             memmove(binary_s[i]+loop_length-length_s[i],binary_buf,sizeof(binary_buf));
312         }
313     }
314     for(i=0; i<loop_length; i++){
315         sprintf(str,"%c%c%c%c",binary_s[3][i],binary_s[2][i],binary_s[1][i],binary_s[0][i]);
316         binary[i]=(int)strtol(str,&e,2);
317     }
318
319     EFp2_set(&next_twisted_Q,&table[binary[0]]);
320
321     //SCM
322     for(i=1; i<loop_length; i++){
323         EFp2_ECD(&next_twisted_Q,&next_twisted_Q);
324         EFp2_ECA(&next_twisted_Q,&next_twisted_Q,&table[binary[i]]);
325     }
326
327     bls12_EFp2_to_EFp12(ANS,&next_twisted_Q);
328
329     EFp2_clear(&next_twisted_Q);
330     EFp2_clear(&twisted_Q);
331     EFp2_clear(&twisted_Q_x);
332     EFp2_clear(&twisted_Q_2x);
333     EFp2_init(&twisted_Q_3x);
334     mpz_clear(x_1);
335     mpz_clear(x_2);
336     for(i=0; i<4; i++){
337         mpz_clear(s[i]);
338     }
339     for(i=0; i<16; i++){
340         EFp2_clear(&table[i]);
341     }
342
343     gettimeofday(&t1,NULL);
344     bls12_G2SCM_4SPLIT=timedifference_msec(t0,t1);
345 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.13.2.4 bls12_plain_G1_scm()

```
void bls12_plain_G1_scm (
              EFp12 * ANS,
              EFp12 * P,
              mpz_t scalar )
```

Calculate scalar multiplication G1 in BLS12 curve without any optimization/splitting of scallar.

Parameters

out	*ANS*	- output
in	*P*	- P in G1 mapped to Fp12.
in	*s*	- gmp mpz_t type big integer.

References bls12_EFp12_to_EFp(), bls12_EFp_to_EFp12(), EFp_clear(), EFp_init(), and EFp_SCM().

Referenced by bls12_test_G1_scm().

```
31                                                  {
32     gettimeofday(&t0,NULL);
33
34     EFp Tmp_P;
35     EFp_init(&Tmp_P);
36
37     bls12_EFp12_to_EFp(&Tmp_P,P);
38     EFp_SCM(&Tmp_P,&Tmp_P,scalar);
39     bls12_EFp_to_EFp12(ANS,&Tmp_P);
40
```

```
41     EFp_clear(&Tmp_P);
42
43     gettimeofday(&t1,NULL);
44     bls12_G1SCM_PLAIN=timedifference_msec(t0,t1);
45 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.13.2.5 bls12_plain_G2_scm()

```
void bls12_plain_G2_scm (
             EFp12 * ANS,
             EFp12 * Q,
             mpz_t scalar )
```

Calculate scalar multiplication G2 in BLS12 curve without any optimization/splitting of scallar.

Parameters

out	*ANS*	- output
in	*Q*	- Q in G2 mapped to Fp12.
in	*s*	- gmp mpz_t type big integer.

References bls12_EFp12_to_EFp2(), bls12_EFp2_to_EFp12(), EFp2_clear(), EFp2_init(), and EFp2_SCM().

Referenced by bls12_test_G2_scm().

```
135                                                    {
136     gettimeofday(&t0,NULL);
137
138     EFp2 twisted_Q;
139     EFp2_init(&twisted_Q);
140
141     bls12_EFp12_to_EFp2(&twisted_Q,Q);
142     EFp2_SCM(&twisted_Q,&twisted_Q,scalar);
143     bls12_EFp2_to_EFp12(ANS,&twisted_Q);
144
145     EFp2_clear(&twisted_Q);
146
147     gettimeofday(&t1,NULL);
148     bls12_G2SCM_PLAIN=timedifference_msec(t0,t1);
149 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.14 include/ELiPS_bn_bls/bls12_skew_frobenius.h File Reference

```
#include <ELiPS_bn_bls/bn_bls12_precoms.h>
#include <ELiPS_bn_bls/bn_efp12.h>
```

Include dependency graph for bls12_skew_frobenius.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_EFp_skew_frobenius_map_p2 (EFp ∗ANS, EFp ∗A)
- void bls12_EFp2_skew_frobenius_map_p1 (EFp2 ∗ANS, EFp2 ∗A)
- void bls12_EFp2_skew_frobenius_map_p2 (EFp2 ∗ANS, EFp2 ∗A)
- void bls12_EFp2_skew_frobenius_map_p3 (EFp2 ∗ANS, EFp2 ∗A)
- void bls12_EFp2_skew_frobenius_map_p10 (EFp2 ∗ANS, EFp2 ∗A)

8.14.1 Detailed Description

Interaface for BLS12's Skew Frobenius map implementations

8.14.2 Function Documentation

8.14.2.1 bls12_EFp2_skew_frobenius_map_p1()

```
void bls12_EFp2_skew_frobenius_map_p1 (
            EFp2 * ANS,
            EFp2 * A )
```

Calculate Skew Frobenius map for pwer of p in G2 of BLS12 curve

Parameters

out	*ANS*	- output in G2'
in	*A*	- A in EFp2.

References d12_skew_frobenius_constant, Fp_set(), and Fp_set_neg().

Referenced by bls12_4split_G2_scm().

```
36                                                      {
37     //x
38     Fp_set(&ANS->x.x0,&A->x.x0);
39     Fp_set_neg(&ANS->x.x1,&A->x.x1);
40     Fp2_mul(&ANS->x,&ANS->x,&d12_skew_frobenius_constant[f_p1][0]);
41     //y
42     Fp_set(&ANS->y.x0,&A->y.x0);
43     Fp_set_neg(&ANS->y.x1,&A->y.x1);
44     Fp2_mul(&ANS->y,&ANS->y,&d12_skew_frobenius_constant[f_p1][1]);
45 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.14.2.2 bls12_EFp2_skew_frobenius_map_p10()

```
void bls12_EFp2_skew_frobenius_map_p10 (
            EFp2 * ANS,
            EFp2 * A )
```

Calculate Skew Frobenius map for pwer of p^10 in G2 of BLS12 curve

Parameters

out	*ANS*	- output in G2'
in	*A*	- A in EFp2.

References d12_skew_frobenius_constant.

```
65                                                          {
66     //x
67     Fp2_mul(&ANS->x,&A->x,&d12_skew_frobenius_constant[f_p10][0]);
68     //y
69     Fp2_mul(&ANS->y,&A->y,&d12_skew_frobenius_constant[f_p10][1]);
70 }
```

8.14.2.3 bls12_EFp2_skew_frobenius_map_p2()

```
void bls12_EFp2_skew_frobenius_map_p2 (
             EFp2 * ANS,
             EFp2 * A )
```

Calculate Skew Frobenius map for pwer of p^2 in G2 of BLS12 curve

Parameters

out	*ANS*	- output in G2'
in	*A*	- A in EFp2.

References d12_skew_frobenius_constant.

Referenced by bls12_2split_G2_scm(), and bls12_4split_G2_scm().

```
47                                                          {
48     //x
49     Fp2_mul(&ANS->x,&A->x,&d12_skew_frobenius_constant[f_p2][0]);
50     //y
51     Fp2_mul(&ANS->y,&A->y,&d12_skew_frobenius_constant[f_p2][1]);
52 }
```

Here is the caller graph for this function:

8.14.2.4 bls12_EFp2_skew_frobenius_map_p3()

```
void bls12_EFp2_skew_frobenius_map_p3 (
             EFp2 * ANS,
             EFp2 * A )
```

Calculate Skew Frobenius map for pwer of p^3 in G2 of BLS12 curve

Parameters

out	*ANS*	- output in G2'
in	*A*	- A in EFp2.

References d12_skew_frobenius_constant, Fp_set(), and Fp_set_neg().

Referenced by bls12_4split_G2_scm().

```
54                                                      {
55     //x
56     Fp_set(&ANS->x.x0,&A->x.x0);
57     Fp_set_neg(&ANS->x.x1,&A->x.x1);
58     Fp2_mul(&ANS->x,&ANS->x,&d12_skew_frobenius_constant[f_p3][0]);
59     //y
60     Fp_set(&ANS->y.x0,&A->y.x0);
61     Fp_set_neg(&ANS->y.x1,&A->y.x1);
62     Fp2_mul(&ANS->y,&ANS->y,&d12_skew_frobenius_constant[f_p3][1]);
63 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.14.2.5 bls12_EFp_skew_frobenius_map_p2()

```
void bls12_EFp_skew_frobenius_map_p2 (
             EFp * ANS,
             EFp * A )
```

Calculate Skew Frobenius map for p^2 in G1 of BLS12 curve

Parameters

out	*ANS*	- output in G1
in	*A*	- A in EFp.

References epsilon1, Fp_mul_mpz(), Fp_set_neg(), and EFp::y.

Referenced by bls12_2split_G1_scm().

```
31                                                      {
32     Fp_mul_mpz(&ANS->x,&A->x,epsilon1);
33     Fp_set_neg(&ANS->y,&A->y);
34 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.15 include/ELiPS_bn_bls/bls12_test_pairings.h File Reference

```
#include <ELiPS_bn_bls/bls12_pairings.h>
#include <ELiPS_bn_bls/bls12_generate_points.h>
#include <ELiPS_bn_bls/bls12_timeprint.h>
#include <ELiPS_bn_bls/bls12_scm.h>
#include <ELiPS_bn_bls/bls12_G3_exp.h>
```

Include dependency graph for bls12_test_pairings.h:

Functions

- void bls12_test_tate_pairing (void)
- void bls12_test_plain_ate_pairing (void)
- void bls12_test_opt_ate_pairing (void)
- void bls12_test_G1_scm (void)
- void bls12_test_G2_scm (void)
- void bls12_test_G3_exp (void)

8.15.1 Detailed Description

Interaface for BLS12's Usefull pairing and scm functionality

8.15.2 Function Documentation

8.15.2.1 bls12_test_G1_scm()

```
void bls12_test_G1_scm (
            void  )
```

Test G1 scm in BLS12 curve

References bls12_2split_G1_scm(), bls12_generate_G1_point(), bls12_plain_G1_scm(), bls12_print_2split_G1↩_scm_time(), bls12_print_plain_G1_scm_time(), curve_parameters, EFp12_clear(), EFp12_init(), EFp12_printf(), and curve_params::order.

```
293                     {
294     printf("================================================================================\n");
295     printf("G1 SCM\n\n");
296     EFp12 P,Test1,Test2;
297     EFp12_init(&P);
298     EFp12_init(&Test1);
299     EFp12_init(&Test2);
300     mpz_t scalar;
301     mpz_init(scalar);
302 
303     //scalar
304     gmp_randstate_t state;
305     gmp_randinit_default (state);
306     gmp_randseed_ui(state,(unsigned long)time(NULL));
307     mpz_urandomm(scalar,state,curve_parameters.order);
308     //printf("scalar:");
309     //gmp_printf("%Zd",scalar);
310     //printf("\n\n");
311     //G1
312     bls12_generate_G1_point(&P);
313 
314     printf("plain G1 scm\n");
315     bls12_plain_G1_scm(&Test1,&P,scalar);
316     bls12_print_plain_G1_scm_time();
317     EFp12_printf(&Test1,"");
318     printf("\n\n");
319 
320     //num=0;
321     printf("2split G1 scm\n");
322     bls12_2split_G1_scm(&Test2,&P,scalar);
323     bls12_print_2split_G1_scm_time();
324     EFp12_printf(&Test2,"");
325     printf("\n");
326 
327     if(Fp12_cmp(&Test1.x,&Test2.x)==0 && Fp12_cmp(&Test1.y,&Test2.y)==0){
328         printf("success\n\n");
329     }else{
330         printf("failed\n\n");
331     }
332 
333     mpz_clear(scalar);
334     EFp12_clear(&P);
335     EFp12_clear(&Test1);
336     EFp12_clear(&Test2);
337 }
```

Here is the call graph for this function:

8.15.2.2 bls12_test_G2_scm()

```
void bls12_test_G2_scm (
            void  )
```

Test G2 scm in BLS12 curve

References bls12_2split_G2_scm(), bls12_4split_G2_scm(), bls12_generate_G2_point(), bls12_plain_G2_scm(), bls12_print_2split_G2_scm_time(), bls12_print_4split_G2_scm_time(), bls12_print_plain_G2_scm_time(), curve_parameters, EFp12_clear(), EFp12_init(), EFp12_printf(), and curve_params::order.

```
339                         {
340     printf("================================================================================\n");
341     printf("G2 SCM\n\n");
342     EFp12 Q,Test1,Test2,Test3;
343     EFp12_init(&Q);
344     EFp12_init(&Test1);
345     EFp12_init(&Test2);
346     EFp12_init(&Test3);
347     mpz_t scalar;
348     mpz_init(scalar);
349 
350     //scalar
351     gmp_randstate_t state;
352     gmp_randinit_default (state);
353     gmp_randseed_ui(state,(unsigned long)time(NULL));
354     mpz_urandomm(scalar,state,curve_parameters.order);
355     //printf("scalar:");
356     //gmp_printf("%Zd",scalar);
357     //printf("\n\n");
358     //G2
359     bls12_generate_G2_point(&Q);
360 
361     printf("plain G2 scm\n");
362     bls12_plain_G2_scm(&Test1,&Q,scalar);
363     bls12_print_plain_G2_scm_time();
364     EFp12_printf(&Test1,"");
365     printf("\n\n");
366 
367     printf("2split G2 scm\n");
368     bls12_2split_G2_scm(&Test2,&Q,scalar);
369     bls12_print_2split_G2_scm_time();
370     EFp12_printf(&Test2,"");
371     printf("\n\n");
372 
373     printf("4split G2 scm\n");
374     bls12_4split_G2_scm(&Test3,&Q,scalar);
375     bls12_print_4split_G2_scm_time();
376     EFp12_printf(&Test3,"");
377     printf("\n\n");
378 
379     if(Fp12_cmp(&Test1.x,&Test2.x)==0 && Fp12_cmp(&Test1.y,&Test2.y)==0
380        && Fp12_cmp(&Test1.x,&Test3.x)==0 && Fp12_cmp(&Test1.y,&Test3.y)==0){
381         printf("success\n\n");
382     }else{
383         printf("failed\n\n");
384     }
385 
386     mpz_clear(scalar);
387     EFp12_clear(&Q);
388     EFp12_clear(&Test1);
389     EFp12_clear(&Test2);
390     EFp12_clear(&Test3);
391 }
```

Here is the call graph for this function:

8.15.2.3 bls12_test_G3_exp()

```
void bls12_test_G3_exp (
            void  )
```

Test G3 exponentiation in BLS12 curve

References bls12_2split_G1_scm(), bls12_2split_G3_exp(), bls12_4split_G2_scm(), bls12_4split_G3_exp(), bls12_generate_G1_point(), bls12_generate_G2_point(), bls12_opt_ate(), bls12_plain_G3_exp(), bls12_print↩_2split_G3_exp_time(), bls12_print_4split_G3_exp_time(), bls12_print_plain_G3_exp_time(), curve_parameters, EFp12_clear(), EFp12_init(), and curve_params::order.

```
393                         {
394     printf("=============================================================================================\n");
395     printf("G3 EXP\n\n");
396     EFp12 P,Q,s1_P,s2_Q;
397     EFp12_init(&P);
398     EFp12_init(&Q);
399     EFp12_init(&s1_P);
400     EFp12_init(&s2_Q);
401     Fp12 Z,Test0,Test1,Test2,Test3;
402     Fp12_init(&Z);
403     Fp12_init(&Test0);
404     Fp12_init(&Test1);
405     Fp12_init(&Test2);
406     Fp12_init(&Test3);
407     mpz_t s1,s2,s12;
408     mpz_init(s1);
409     mpz_init(s2);
410     mpz_init(s12);
411 
412     //S
413     gmp_randstate_t state;
414     gmp_randinit_default (state);
415     gmp_randseed_ui(state,(unsigned long)time(NULL));
416     mpz_urandomm(s1,state,curve_parameters.order);    //s1
417     mpz_urandomm(s2,state,curve_parameters.order);    //s2
418     mpz_mul(s12,s1,s2);             //s12
419     mpz_mod(s12,s12,curve_parameters.order);
420 
421     bls12_generate_G1_point(&P);             //P
422     bls12_generate_G2_point(&Q);             //Q
423 
424     bls12_2split_G1_scm(&s1_P,&P,s1);         //s1_P
425     bls12_4split_G2_scm(&s2_Q,&Q,s2);    //s2_Q
426 
427     printf("x-ate([s2]Q,[s1]P)\n");
428     bls12_opt_ate(&Test0,&s1_P,&s2_Q);
429 
430     printf("x-ate(Q,P)^s12\n");
431     bls12_opt_ate(&Z,&P,&Q);
432 
433     printf("plain G3 exp\n");
434     bls12_plain_G3_exp(&Test1,&Z,s12);
435     bls12_print_plain_G3_exp_time();
436     Fp12_printf(&Test1,"");
437     printf("\n\n");
438 
439     printf("2split G3 exp\n");
440     bls12_2split_G3_exp(&Test2,&Z,s12);
441     bls12_print_2split_G3_exp_time();
442     Fp12_printf(&Test2,"");
443     printf("\n\n");
444 
445     printf("4split G3 exp\n");
446     bls12_4split_G3_exp(&Test3,&Z,s12);
447     bls12_print_4split_G3_exp_time();
448     Fp12_printf(&Test3,"");
449     printf("\n\n");
450 
451     if(Fp12_cmp(&Test0,&Test1)==0 && Fp12_cmp(&Test0,&Test2)==0 && Fp12_cmp(&Test0,&Test3)==0){
452         printf("success\n\n");
453     }else{
454         printf("failed\n\n");
455     }
456 
457     mpz_clear(s1);
458     mpz_clear(s2);
459     mpz_clear(s12);
460     EFp12_clear(&P);
461     EFp12_clear(&Q);
462     EFp12_clear(&s1_P);
463     EFp12_clear(&s2_Q);
464     Fp12_clear(&Z);
465     Fp12_clear(&Test0);
466     Fp12_clear(&Test1);
467     Fp12_clear(&Test2);
468     Fp12_clear(&Test3);
469 }
```

Here is the call graph for this function:

8.15.2.4 bls12_test_opt_ate_pairing()

```
void bls12_test_opt_ate_pairing (
            void  )
```

Test opt-ate pairing in BLS12 curve

References bls12_generate_G1_point(), bls12_generate_G2_point(), bls12_opt_ate(), bls12_print_final_exp_optimal_time(), bls12_print_opt_ate_time(), curve_parameters, EFp12_clear(), EFp12_init(), EFp12_printf(), EFp12_SCM(), and curve_params::order.

```
205                                {
206     printf("================================================================================\n");
207     printf("opt-ate pairing\n\n");
208     EFp12 P,Q,s1_P,s2_P,s1_Q,s2_Q;
209     EFp12_init(&P);
210     EFp12_init(&Q);
211     EFp12_init(&s1_P);
212     EFp12_init(&s2_P);
213     EFp12_init(&s1_Q);
214     EFp12_init(&s2_Q);
215     Fp12 Z,Test1,Test2,Test3;
216     Fp12_init(&Z);
217     Fp12_init(&Test1);
218     Fp12_init(&Test2);
219     Fp12_init(&Test3);
220     mpz_t s1,s2,s12;
221     mpz_init(s1);
222     mpz_init(s2);
223     mpz_init(s12);
224
225     gmp_randstate_t state;
226     gmp_randinit_default (state);
227     gmp_randseed_ui(state,(unsigned long)time(NULL));
228     mpz_urandomm(s1,state,curve_parameters.order);
229     mpz_urandomm(s2,state,curve_parameters.order);
230     mpz_mul(s12,s1,s2);
231     mpz_mod(s12,s12,curve_parameters.order);
232
233     printf("input\n");
234     bls12_generate_G1_point(&P);
235     EFp12_printf(&P,"P : G1 rational point\n");
236     printf("\n");
237     bls12_generate_G2_point(&Q);
238     EFp12_printf(&Q,"Q : G2 rational point\n");
239     printf("\n\n");
240
241     EFp12_SCM(&s1_P,&P,s1);
242     EFp12_SCM(&s2_P,&P,s2);
243     EFp12_SCM(&s1_Q,&Q,s1);
244     EFp12_SCM(&s2_Q,&Q,s2);
245
246     printf("bilinearity test\n");
247     //test1
248     printf("opt-ate(Q,P)^s12\n");
249     bls12_opt_ate(&Z,&P,&Q);
250     Fp12_pow(&Test1,&Z,s12);
251     bls12_print_opt_ate_time();
252     bls12_print_final_exp_optimal_time();
253     Fp12_printf(&Test1,"");
254     printf("\n\n");
255     //test2
256     printf("opt-ate([s2]Q,[s1]P)\n");
257     bls12_opt_ate(&Test2,&s2_P,&s1_Q);
258     bls12_print_opt_ate_time();
259     bls12_print_final_exp_optimal_time();
260     Fp12_printf(&Test2,"");
261     printf("\n\n");
262     //test3
263     printf("opt-ate([s1]Q,[s2]P)\n");
264     bls12_opt_ate(&Test3,&s1_P,&s2_Q);
265     bls12_print_opt_ate_time();
266     bls12_print_final_exp_optimal_time();
267     Fp12_printf(&Test3,"");
268     printf("\n\n");
269
```

```
270     if(Fp12_cmp_zero(&Test1)!=0 && Fp12_cmp_one(&Test1)!=0 && Fp12_cmp(&Test1,&Test2)==0 && Fp12_cmp(&Test2
      ,&Test3)==0 && Fp12_cmp(&Test3,&Test1)==0){
271         printf("success\n\n");
272     }else{
273         printf("failed\n\n");
274     }
275
276
277     mpz_clear(s1);
278     mpz_clear(s2);
279     mpz_clear(s12);
280     EFp12_clear(&P);
281     EFp12_clear(&Q);
282     EFp12_clear(&s1_P);
283     EFp12_clear(&s2_P);
284     EFp12_clear(&s1_Q);
285     EFp12_clear(&s2_Q);
286     Fp12_clear(&Z);
287     Fp12_clear(&Test1);
288     Fp12_clear(&Test2);
289     Fp12_clear(&Test3);
290 }
```

Here is the call graph for this function:

8.15.2.5 bls12_test_plain_ate_pairing()

```
void bls12_test_plain_ate_pairing (
            void  )
```

Test ate pairing in BLS12 curve

References bls12_generate_G1_point(), bls12_generate_G2_point(), bls12_plain_ate(), bls12_print_final_exp_↩optimal_time(), bls12_print_plain_ate_time(), curve_parameters, EFp12_clear(), EFp12_init(), EFp12_printf(), E↩Fp12_SCM(), and curve_params::order.

```
118                                 {
119     printf("=================================================================================================\n");
120     printf("plain-ate pairing\n\n");
121     EFp12 P,Q,s1_P,s2_P,s1_Q,s2_Q;
122     EFp12_init(&P);
123     EFp12_init(&Q);
124     EFp12_init(&s1_P);
125     EFp12_init(&s2_P);
126     EFp12_init(&s1_Q);
127     EFp12_init(&s2_Q);
128     Fp12 Z,Test1,Test2,Test3;
129     Fp12_init(&Z);
130     Fp12_init(&Test1);
131     Fp12_init(&Test2);
132     Fp12_init(&Test3);
133     mpz_t s1,s2,s12;
134     mpz_init(s1);
135     mpz_init(s2);
136     mpz_init(s12);
137
138     gmp_randstate_t state;
139     gmp_randinit_default (state);
140     gmp_randseed_ui(state,(unsigned long)time(NULL));
141     mpz_urandomm(s1,state,curve_parameters.order);
142     mpz_urandomm(s2,state,curve_parameters.order);
143     mpz_mul(s12,s1,s2);
144     mpz_mod(s12,s12,curve_parameters.order);
145
146     printf("input\n");
147     bls12_generate_G1_point(&P);
148     EFp12_printf(&P,"P : G1 rational point\n");
149     printf("\n");
150     bls12_generate_G2_point(&Q);
151     EFp12_printf(&Q,"Q : G2 rational point\n");
152     printf("\n\n");
153
154     EFp12_SCM(&s1_P,&P,s1);
155     EFp12_SCM(&s2_P,&P,s2);
156     EFp12_SCM(&s1_Q,&Q,s1);
```

```
157     EFp12_SCM(&s2_Q,&Q,s2);
158
159     printf("bilinearity test\n");
160     //test1
161     printf("plain-ate(Q,P)^s12\n");
162     bls12_plain_ate(&Z,&P,&Q);
163     Fp12_pow(&Test1,&Z,s12);
164     bls12_print_plain_ate_time();
165     bls12_print_final_exp_optimal_time();
166     Fp12_printf(&Test1,"");
167     printf("\n\n");
168     //test2
169     printf("plain-ate([s2]Q,[s1]P)\n");
170     bls12_plain_ate(&Test2,&s2_P,&s1_Q);
171     bls12_print_plain_ate_time();
172     bls12_print_final_exp_optimal_time();
173     Fp12_printf(&Test2,"");
174     printf("\n\n");
175     //test3
176     printf("plain-ate([s1]Q,[s2]P)\n");
177     bls12_plain_ate(&Test3,&s1_P,&s2_Q);
178     bls12_print_plain_ate_time();
179     bls12_print_final_exp_optimal_time();
180     Fp12_printf(&Test3,"");
181     printf("\n\n");
182
183     if(Fp12_cmp_zero(&Test1)!=0 && Fp12_cmp_one(&Test1)!=0 && Fp12_cmp(&Test1,&Test2)==0 && Fp12_cmp(&Test2
    ,&Test3)==0 && Fp12_cmp(&Test3,&Test1)==0){
184         printf("success\n\n");
185     }else{
186         printf("failed\n\n");
187     }
188
189
190     mpz_clear(s1);
191     mpz_clear(s2);
192     mpz_clear(s12);
193     EFp12_clear(&P);
194     EFp12_clear(&Q);
195     EFp12_clear(&s1_P);
196     EFp12_clear(&s2_P);
197     EFp12_clear(&s1_Q);
198     EFp12_clear(&s2_Q);
199     Fp12_clear(&Z);
200     Fp12_clear(&Test1);
201     Fp12_clear(&Test2);
202     Fp12_clear(&Test3);
203 }
```

Here is the call graph for this function:

8.15.2.6 bls12_test_tate_pairing()

```
void bls12_test_tate_pairing (
             void  )
```

Test tate pairing in BLS12 curve

References bls12_generate_G1_point(), bls12_print_final_exp_optimal_time(), bls12_print_tate_time(), bls12_↩ tate(), curve_parameters, EFp12_clear(), EFp12_init(), EFp12_printf(), EFp12_rational_point_bls12(), EFp12_↩ SCM(), and curve_params::order.

```
32                              {
33     printf("================================================================================================\n");
34     printf("tate pairing\n\n");
35     EFp12 P,Q,s1_P,s2_P,s1_Q,s2_Q;
36     EFp12_init(&P);
37     EFp12_init(&Q);
38     EFp12_init(&s1_P);
39     EFp12_init(&s2_P);
40     EFp12_init(&s1_Q);
41     EFp12_init(&s2_Q);
42     Fp12 Z,Test1,Test2,Test3;
43     Fp12_init(&Z);
44     Fp12_init(&Test1);
```

```
45      Fp12_init(&Test2);
46      Fp12_init(&Test3);
47      mpz_t s1,s2,s12;
48      mpz_init(s1);
49      mpz_init(s2);
50      mpz_init(s12);
51
52      gmp_randstate_t state;
53      gmp_randinit_default (state);
54      gmp_randseed_ui(state,(unsigned long)time(NULL));
55      mpz_urandomm(s1,state,curve_parameters.order);
56      mpz_urandomm(s2,state,curve_parameters.order);
57      mpz_mul(s12,s1,s2);
58      mpz_mod(s12,s12,curve_parameters.order);
59
60      printf("input\n");
61      bls12_generate_G1_point(&P);
62      EFp12_printf(&P,"P : G1 rational point\n");
63      printf("\n\n");
64      EFp12_rational_point_bls12(&Q);
65      EFp12_printf(&Q,"Q : random rational point\n");
66      printf("\n\n");
67
68      EFp12_SCM(&s1_P,&P,s1);
69      EFp12_SCM(&s2_P,&P,s2);
70      EFp12_SCM(&s1_Q,&Q,s1);
71      EFp12_SCM(&s2_Q,&Q,s2);
72
73      printf("bilinearity test\n");
74      //test1
75      printf("tate(P,Q)^s12\n");
76      bls12_tate(&Z,&P,&Q);
77      Fp12_pow(&Test1,&Z,s12);
78      bls12_print_tate_time();
79      bls12_print_final_exp_optimal_time();
80      Fp12_printf(&Test1,"");
81      printf("\n\n");
82      //test2
83      printf("tate([s1]P,[s2]Q)\n");
84      bls12_tate(&Test2,&s1_P,&s2_Q);
85      bls12_print_tate_time();
86      bls12_print_final_exp_optimal_time();
87      Fp12_printf(&Test2,"");
88      printf("\n\n");
89      //test3
90      printf("tate([s2]P,[s1]Q)\n");
91      bls12_tate(&Test3,&s2_P,&s1_Q);
92      bls12_print_tate_time();
93      bls12_print_final_exp_optimal_time();
94      Fp12_printf(&Test3,"");
95      printf("\n\n");
96
97      if(Fp12_cmp_zero(&Test1)!=0 && Fp12_cmp_one(&Test1)!=0 && Fp12_cmp(&Test1,&Test2)==0 && Fp12_cmp(&Test2
    ,&Test3)==0 && Fp12_cmp(&Test3,&Test1)==0){
98          printf("success\n\n");
99      }else{
100          printf("failed\n\n");
101      }
102
103      mpz_clear(s1);
104      mpz_clear(s2);
105      mpz_clear(s12);
106      EFp12_clear(&P);
107      EFp12_clear(&Q);
108      EFp12_clear(&s1_P);
109      EFp12_clear(&s2_P);
110      EFp12_clear(&s1_Q);
111      EFp12_clear(&s2_Q);
112      Fp12_clear(&Z);
113      Fp12_clear(&Test1);
114      Fp12_clear(&Test2);
115      Fp12_clear(&Test3);
116  }
```

Here is the call graph for this function:

8.16 include/ELiPS_bn_bls/bls12_timeprint.h File Reference

`#include <stdio.h>`

Include dependency graph for bls12_timeprint.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_print_parameters (void)
- void bls12_print_G1_point (void)
- void bls12_print_G2_point (void)
- void bls12_print_tate_time (void)
- void bls12_print_plain_ate_time (void)
- void bls12_print_opt_ate_time (void)
- void bls12_print_final_exp_plain_time (void)
- void bls12_print_final_exp_optimal_time (void)
- void bls12_print_plain_G1_scm_time (void)
- void bls12_print_2split_G1_scm_time (void)
- void bls12_print_plain_G2_scm_time (void)
- void bls12_print_2split_G2_scm_time (void)
- void bls12_print_4split_G2_scm_time (void)
- void bls12_print_plain_G3_exp_time (void)
- void bls12_print_2split_G3_exp_time (void)
- void bls12_print_4split_G3_exp_time (void)

Variables

- double bls12_MILLER_TATE

8.16.1 Detailed Description

Interaface for BLS12's time profiling

8.16.2 Function Documentation

8.16.2.1 bls12_print_2split_G1_scm_time()

```
void bls12_print_2split_G1_scm_time (
            void  )
```

Prints 2 split of scalr G1 scm time: BLS12 curve

Referenced by bls12_test_G1_scm().

```
61                                      {
62     printf("2split-G1-SCM time:%.2f[ms]\n",bls12_G1SCM_2SPLIT);
63 }
```

Here is the caller graph for this function:

8.16.2.2 bls12_print_2split_G2_scm_time()

```
void bls12_print_2split_G2_scm_time (
            void  )
```

Prints 2 split of scalr G2 scm time: BLS12 curve

Referenced by bls12_test_G2_scm().

```
69                                  {
70     printf("2split-G2-SCM time:%.2f[ms]\n",bls12_G2SCM_2SPLIT);
71 }
```

Here is the caller graph for this function:

8.16.2.3 bls12_print_2split_G3_exp_time()

```
void bls12_print_2split_G3_exp_time (
            void  )
```

Prints 2 split of scalr G3 exponentiation time: BLS12 curve

Referenced by bls12_test_G3_exp().

```
81                                  {
82     printf("2split-G3-SCM time:%.2f[ms]\n",bls12_G3EXP_2SPLIT);
83 }
```

Here is the caller graph for this function:

8.16.2.4 bls12_print_4split_G2_scm_time()

```
void bls12_print_4split_G2_scm_time (
            void  )
```

Prints 4 split of scalr G2 scm time: BLS12 curve

Referenced by bls12_test_G2_scm().

```
73                                  {
74     printf("4split-G2-SCM time:%.2f[ms]\n",bls12_G2SCM_4SPLIT);
75 }
```

Here is the caller graph for this function:

8.16.2.5 bls12_print_4split_G3_exp_time()

```
void bls12_print_4split_G3_exp_time (
             void  )
```

Prints 2 split of scalr G3 exponentiation time: BLS12 curve

Referenced by bls12_test_G3_exp().

```
85                                      {
86     printf("4split-G3-SCM time:%.2f[ms]\n",bls12_G3EXP_4SPLIT);
87 }
```

Here is the caller graph for this function:

8.16.2.6 bls12_print_final_exp_optimal_time()

```
void bls12_print_final_exp_optimal_time (
             void  )
```

Prints efficient final exp time: BLS12 curve

Referenced by bls12_test_opt_ate_pairing(), bls12_test_plain_ate_pairing(), and bls12_test_tate_pairing().

```
53                                        {
54     printf("optimal-final exp time:%.2f[ms]\n",bls12_FINALEXP_OPT);
55 }
```

Here is the caller graph for this function:

8.16.2.7 bls12_print_final_exp_plain_time()

```
void bls12_print_final_exp_plain_time (
             void  )
```

Prints plain final exp time: BLS12 curve

```
49                                       {
50     printf("plain-final exp time:%.2f[ms]\n",bls12_FINALEXP_PLAIN);
51 }
```

8.16.2.8 bls12_print_G1_point()

```
void bls12_print_G1_point (
             void  )
```

Prints G1 rational point: BLS12 curve

8.16.2.9 bls12_print_G2_point()

```
void bls12_print_G2_point (
            void  )
```

Prints G2 rational point: BLS12 curve

8.16.2.10 bls12_print_opt_ate_time()

```
void bls12_print_opt_ate_time (
            void  )
```

Prints opt-ate paring time: BLS12 curve

Referenced by bls12_test_opt_ate_pairing().

```
45                              {
46     printf("miller opt-ate time:%.2f[ms]\n",bls12_MILLER_OPTATE);
47 }
```

Here is the caller graph for this function:

8.16.2.11 bls12_print_parameters()

```
void bls12_print_parameters (
            void  )
```

Prints curve parameter: BLS12 curve

```
478                             {
479     printf("================================================================================\n");
480     printf("bls12\n\n");
481     gmp_printf("parameters\n");
482     gmp_printf("X     (%dbit length) : %Zd \n",(int)mpz_sizeinbase(bls12_X,2),bls12_X);
483     gmp_printf("prime (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.prime,2),curve_parameters.
      prime);
484     gmp_printf("order (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.order,2),curve_parameters.
      order);
485     gmp_printf("trace (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.trace_t,2),curve_parameters.
      trace_t);
486
487     gmp_printf("\nelliptic curve\n");
488     gmp_printf("E:y^2=x^3+4\n",curve_parameters.curve_b);
489
490     gmp_printf("\nmodulo polynomial\n");
491     gmp_printf("Fp2  : f(x) = x^2+1\n");
492     gmp_printf("Fp6  : f(x) = x^3-(alpha+1)\n");
493     gmp_printf("Fp12 : f(x) = x^2-beta\n");
494
495 }
```

8.16.2.12 bls12_print_plain_ate_time()

```
void bls12_print_plain_ate_time (
             void  )
```

Prints ate paring time: BLS12 curve

Referenced by bls12_test_plain_ate_pairing().

```
41                                {
42     printf("miller plain-ate time:%.2f[ms]\n",bls12_MILLER_PLAINATE);
43 }
```

Here is the caller graph for this function:

8.16.2.13 bls12_print_plain_G1_scm_time()

```
void bls12_print_plain_G1_scm_time (
             void  )
```

Prints plain G1 scm time: BLS12 curve

Referenced by bls12_test_G1_scm().

```
57                                  {
58     printf("plain-G1-SCM time:%.2f[ms]\n",bls12_G1SCM_PLAIN);
59 }
```

Here is the caller graph for this function:

8.16.2.14 bls12_print_plain_G2_scm_time()

```
void bls12_print_plain_G2_scm_time (
             void  )
```

Prints plain G2 scm time: BLS12 curve

Referenced by bls12_test_G2_scm().

```
65                                  {
66     printf("plain-G2-SCM time:%.2f[ms]\n",bls12_G2SCM_PLAIN);
67 }
```

Here is the caller graph for this function:

8.16.2.15 bls12_print_plain_G3_exp_time()

```
void bls12_print_plain_G3_exp_time (
            void  )
```

Prints no split of scalr G3 exponentiation time: BLS12 curve

Referenced by bls12_test_G3_exp().

```
77                                  {
78     printf("plain-G3-SCM time:%.2f[ms]\n",bls12_G3EXP_PLAIN);
79 }
```

Here is the caller graph for this function:

8.16.2.16 bls12_print_tate_time()

```
void bls12_print_tate_time (
            void  )
```

Prints Tate paring time: BLS12 curve

References bls12_MILLER_TATE.

Referenced by bls12_test_tate_pairing().

```
37                              {
38     printf("miller tate time:%.2f[ms]\n",bls12_MILLER_TATE);
39 }
```

Here is the caller graph for this function:

8.16.3 Variable Documentation

8.16.3.1 bls12_MILLER_TATE

```
double bls12_MILLER_TATE
```

Time types of different operations.

Referenced by bls12_print_tate_time(), and bls12_tate().

8.17 include/ELiPS_bn_bls/bls12_twist.h File Reference

```
#include <ELiPS_bn_bls/bn_efp12.h>
```

Include dependency graph for bls12_twist.h: This graph shows which files directly or indirectly include this file:

Functions

- void bls12_EFp12_to_EFp2 (EFp2 *ANS, EFp12 *P)
- void bls12_EFp2_to_EFp12 (EFp12 *ANS, EFp2 *P)
- void bls12_EFp12_to_EFp (EFp *ANS, EFp12 *P)
- void bls12_EFp_to_EFp12 (EFp12 *ANS, EFp *P)

8.17.1 Detailed Description

Interaface for BLS12's rational point mapping functionalities

8.17.2 Function Documentation

8.17.2.1 bls12_EFp12_to_EFp()

```
void bls12_EFp12_to_EFp (
            EFp * ANS,
            EFp12 * P )
```

Calculate primary twist of G1 rational point

Parameters

out	*ANS*	- output in EFp
in	*P*	- input P in EFp12.

References Fp_set(), EFp::infinity, and EFp::y.

Referenced by bls12_2split_G1_scm(), bls12_Miller_algo_for_opt_ate(), bls12_Miller_algo_for_plain_ate(), bls12_Miller_algo_for_tate(), and bls12_plain_G1_scm().

```
32                                          {
33     Fp_set(&ANS->x,&P->x.x0.x0.x0);
34     Fp_set(&ANS->y,&P->y.x0.x0.x0);
35     ANS->infinity=P->infinity;
36 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.17.2.2 bls12_EFp12_to_EFp2()

```
void bls12_EFp12_to_EFp2 (
            EFp2 * ANS,
            EFp12 * P )
```

Calculate sextic twisted map G2 rational point

Parameters

out	*ANS*	- output in EFp2
in	*P*	- input P in EFp12.

Referenced by bls12_2split_G2_scm(), bls12_4split_G2_scm(), bls12_Miller_algo_for_opt_ate(), bls12_Miller_↩ algo_for_plain_ate(), and bls12_plain_G2_scm().

```
45                                          {
46     Fp2_set_ui(&ANS->x,0);
47     Fp2_set(&ANS->x,&P->x.x0.x2);
48     Fp2_mul_basis(&ANS->x,&ANS->x);
49     Fp2_set_ui(&ANS->y,0);
50     Fp2_set(&ANS->y,&P->y.x1.x1);
51     Fp2_mul_basis(&ANS->y,&ANS->y);
52     ANS->infinity=P->infinity;
53 }
```

Here is the caller graph for this function:

8.17.2.3 bls12_EFp2_to_EFp12()

```
void bls12_EFp2_to_EFp12 (
             EFp12 * ANS,
             EFp2 * P )
```

Calculate sextic twisted map G2 rational point

Parameters

out	*ANS*	- output in EFp12
in	*P*	- input P in EFp2.

Referenced by bls12_plain_G2_scm().

```
55                                          {
56     Fp12_set_ui(&ANS->x,0);
57     Fp2_set(&ANS->x.x0.x2,&P->x);
58     Fp2_inv_basis(&ANS->x.x0.x2,&ANS->x.x0.x2);
59     Fp12_set_ui(&ANS->y,0);
60     Fp2_set(&ANS->y.x1.x1,&P->y);
61     Fp2_inv_basis(&ANS->y.x1.x1,&ANS->y.x1.x1);
62     ANS->infinity=P->infinity;
63 }
```

Here is the caller graph for this function:

8.17.2.4 bls12_EFp_to_EFp12()

```
void bls12_EFp_to_EFp12 (
             EFp12 * ANS,
             EFp * P )
```

Calculate primary twist of G1 rational point

Parameters

out	*ANS*	- output in EFp12
in	*P*	- input P in EFp.

References Fp_set(), EFp::infinity, and EFp::y.

Referenced by bls12_generate_G1_point(), and bls12_plain_G1_scm().

```
38                                          {
39     Fp_set(&ANS->x.x0.x0.x0,&P->x);
40     Fp_set(&ANS->y.x0.x0.x0,&P->y);
41     ANS->infinity=P->infinity;
42 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.18 include/ELiPS_bn_bls/bn_bls12_precoms.h File Reference

```
#include <ELiPS_bn_bls/bn_fp6.h>
#include <ELiPS_bn_bls/curve_settings.h>
```

Include dependency graph for bn_bls12_precoms.h: This graph shows which files directly or indirectly include this file:

Enumerations

- enum state

Functions

- void init_precoms (int curvetype)
- void get_epsilon (void)
- void set_basis (void)
- void set_frobenius_constant (void)

Variables

- struct Fp Fp_basis
- struct Fp2 Fp2_basis
- struct Fp2 Fp2_basis_inv
- struct Fp6 Fp6_basis
- mpz_t epsilon1
- Fp2 d12_frobenius_constant [d12][6]
- Fp2 d12_skew_frobenius_constant [d12][2]

8.18.1 Detailed Description

Interaface for BLS12's pre-computationa and constants

8.18.2 Enumeration Type Documentation

8.18.2.1 state

```
enum state
```

Enumate the p-th power

```
57           {
58     f_p1,f_p2,f_p3,f_p4,f_p5,f_p6,f_p7,f_p8,f_p9,f_p10,f_p11,f_p12
59 };
```

8.18.3 Function Documentation

8.18.3.1 get_epsilon()

```
void get_epsilon (
              void  )
```

Calculate primitive cubic root of 1 in Fp.

References epsilon1, Fp_init(), Fp_inv(), Fp_mul(), Fp_set_ui(), Fp_sqrt(), and Fp_sub_ui().

```
78                  {
79     Fp inv,buf,result1,result2;
80     Fp_init(&inv);
81     Fp_init(&buf);
82     Fp_init(&result1);
83     Fp_init(&result2);
84
85     Fp_set_ui(&buf,2);
86     Fp_inv(&inv,&buf);
87     mpz_sub_ui(buf.x0,prime_p,3);
88
89     Fp_sqrt(&buf,&buf);
90     Fp_sub_ui(&buf,&buf,1);
91     Fp_mul(&result1,&buf,&inv);
92     Fp_mul(&result2,&result1,&result1);
93
94     mpz_set(epsilon1,result1.x0);
95     mpz_set(epsilon2,result2.x0);
96
97     Fp_clear(&inv);
98     Fp_clear(&buf);
99     Fp_clear(&result1);
100     Fp_clear(&result2);
101 }
```

Here is the call graph for this function:

8.18.3.2 init_precoms()

```
void init_precoms (
              int curvetype )
```

Initialize BLS12 curve pre-computations

Parameters

in	*curvetype*	- send 2 for BLS12 and 1 for BN

References curve_parameters, epsilon1, Fp2_basis, Fp2_basis_inv, Fp6_basis, Fp_basis, Fp_init(), and curve_↩ params::prime.

Referenced by bls12_inits(), and init_bn().

```
45                                  {
46
47      mpz_init(prime_p);
48      if (curvetype == 1) {
49          mpz_set(prime_p,curve_parameters.prime);
50      }
51      else if (curvetype == 2){
52          mpz_set(prime_p,curve_parameters.prime);
53      }
54
55      Fp_init(&Fp_basis);
56      Fp2_init(&Fp2_basis);
57      Fp2_init(&Fp2_basis_inv);
58      Fp6_init(&Fp6_basis);
59      mpz_init(epsilon1);
60      mpz_init(epsilon2);
61
62      int i,j;
63      for(i=0; i<d12; i++){
64          for(j=0; j<6; j++){
65              Fp2_init(&d12_frobenius_constant[i][j]);
66          }
67          for(j=0; j<2; j++){
68              Fp2_init(&d12_skew_frobenius_constant[i][j]);
69          }
70      }
71
72
73      get_epsilon();
74      set_basis();
75      set_frobenius_constant();
76  }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.18.3.3 set_basis()

```
void set_basis (
              void  )
```

Set basis elements

References Fp2_basis, Fp2_basis_inv, Fp6_basis, Fp_basis, and Fp_set_ui().

```
103                 {
104      Fp_set_ui(&Fp_basis,1);
105      Fp2_set_ui(&Fp2_basis,1);
106      Fp6_set_ui(&Fp6_basis,0);
107      Fp_set_ui(&Fp6_basis.x1.x0,1);
108      Fp2_inv(&Fp2_basis_inv,&Fp2_basis);
109  }
```

Here is the call graph for this function:

8.18.3.4 set_frobenius_constant()

```
void set_frobenius_constant (
            void  )
```

Set Frobenius map constants

References Fp2_basis, and Fp_set_ui().

```
111                                {
112     Fp2 tmp1,tmp2,tmp3;
113     Fp2_init(&tmp1);
114     Fp2_init(&tmp2);
115     Fp2_init(&tmp3);
116 
117     mpz_t exp,buf,p2,p3,p4,p6,p8,p10;
118     mpz_init(exp);
119     mpz_init(buf);
120     mpz_init(p2);
121     mpz_init(p3);
122     mpz_init(p4);
123     mpz_init(p6);
124     mpz_init(p8);
125     mpz_init(p10);
126 
127     mpz_mul(p2,prime_p,prime_p);
128     mpz_mul(p3,p2,prime_p);
129     mpz_mul(p4,p3,prime_p);
130     mpz_mul(p6,p4,p2);
131     mpz_mul(p8,p6,p2);
132     mpz_mul(p10,p8,p2);
133 
134     //frobenius_1
135     mpz_sub_ui(exp,prime_p,1);
136     mpz_tdiv_q_ui(exp,exp,3);
137     Fp2_pow(&tmp1,&Fp2_basis,exp);
138     Fp2_mul(&tmp2,&tmp1,&tmp1);
139     mpz_tdiv_q_ui(exp,exp,2);
140     Fp2_pow(&tmp3,&Fp2_basis,exp);
141     //set f_p1
142     Fp_set_ui(&d12_frobenius_constant[f_p1][0].x0,1);
143     Fp2_set(&d12_frobenius_constant[f_p1][1],&tmp1);
144     Fp2_set(&d12_frobenius_constant[f_p1][2],&tmp2);
145     Fp2_set(&d12_frobenius_constant[f_p1][3],&tmp3);
146     Fp2_mul(&d12_frobenius_constant[f_p1][4],&tmp1,&tmp3);
147     Fp2_mul(&d12_frobenius_constant[f_p1][5],&tmp2,&tmp3);
148     //set skew_f_p1
149     Fp2_inv(&tmp1,&tmp1);
150     mpz_sub_ui(exp,prime_p,1);
151     mpz_tdiv_q_ui(exp,exp,2);
152     Fp2_pow(&tmp2,&Fp2_basis,exp);
153     Fp2_inv(&tmp2,&tmp2);
154     Fp2_set(&d12_skew_frobenius_constant[f_p1][0],&tmp1);
155     Fp2_set(&d12_skew_frobenius_constant[f_p1][1],&tmp2);
156 
157     //frobenius_2
158     mpz_sub_ui(exp,p2,1);
159     mpz_tdiv_q_ui(exp,exp,3);
160     Fp2_pow(&tmp1,&Fp2_basis,exp);
161     Fp2_mul(&tmp2,&tmp1,&tmp1);
162     mpz_tdiv_q_ui(exp,exp,2);
163     Fp2_pow(&tmp3,&Fp2_basis,exp);
164     //set f_p2
165     Fp_set_ui(&d12_frobenius_constant[f_p2][0].x0,1);
166     Fp2_set(&d12_frobenius_constant[f_p2][1],&tmp1);
167     Fp2_set(&d12_frobenius_constant[f_p2][2],&tmp2);
168     Fp2_set(&d12_frobenius_constant[f_p2][3],&tmp3);
169     Fp2_mul(&d12_frobenius_constant[f_p2][4],&tmp1,&tmp3);
170     Fp2_mul(&d12_frobenius_constant[f_p2][5],&tmp2,&tmp3);
171     //set skew_f_p2
172     Fp2_inv(&tmp1,&tmp1);
173     mpz_sub_ui(exp,p2,1);
174     mpz_tdiv_q_ui(exp,exp,2);
175     Fp2_pow(&tmp2,&Fp2_basis,exp);
176     Fp2_inv(&tmp2,&tmp2);
177     Fp2_set(&d12_skew_frobenius_constant[f_p2][0],&tmp1);
178     Fp2_set(&d12_skew_frobenius_constant[f_p2][1],&tmp2);
179 
180     //frobenius_3
181     mpz_sub_ui(exp,p3,1);
182     mpz_tdiv_q_ui(exp,exp,3);
```

```
183     Fp2_pow(&tmp1,&Fp2_basis,exp);
184     Fp2_mul(&tmp2,&tmp1,&tmp1);
185     mpz_tdiv_q_ui(exp,exp,2);
186     Fp2_pow(&tmp3,&Fp2_basis,exp);
187     //set f_p3
188     Fp_set_ui(&d12_frobenius_constant[f_p3][0].x0,1);
189     Fp2_set(&d12_frobenius_constant[f_p3][1],&tmp1);
190     Fp2_set(&d12_frobenius_constant[f_p3][2],&tmp2);
191     Fp2_set(&d12_frobenius_constant[f_p3][3],&tmp3);
192     Fp2_mul(&d12_frobenius_constant[f_p3][4],&tmp1,&tmp3);
193     Fp2_mul(&d12_frobenius_constant[f_p3][5],&tmp2,&tmp3);
194     //set skew_f_p3
195     Fp2_inv(&tmp1,&tmp1);
196     mpz_sub_ui(exp,p3,1);
197     mpz_tdiv_q_ui(exp,exp,2);
198     Fp2_pow(&tmp2,&Fp2_basis,exp);
199     Fp2_inv(&tmp2,&tmp2);
200     Fp2_set(&d12_skew_frobenius_constant[f_p3][0],&tmp1);
201     Fp2_set(&d12_skew_frobenius_constant[f_p3][1],&tmp2);
202
203     //d12_frobenius_constant[f_p4]
204     mpz_sub_ui(exp,p4,1);
205     mpz_tdiv_q_ui(exp,exp,3);
206     Fp2_pow(&tmp1,&Fp2_basis,exp);
207     Fp2_mul(&tmp2,&tmp1,&tmp1);
208     mpz_tdiv_q_ui(exp,exp,2);
209     Fp2_pow(&tmp3,&Fp2_basis,exp);
210     //set d12_frobenius_constant[f_p4]
211     Fp_set_ui(&d12_frobenius_constant[f_p4][0].x0,1);
212     Fp2_set(&d12_frobenius_constant[f_p4][1],&tmp1);
213     Fp2_set(&d12_frobenius_constant[f_p4][2],&tmp2);
214     Fp2_set(&d12_frobenius_constant[f_p4][3],&tmp3);
215     Fp2_mul(&d12_frobenius_constant[f_p4][4],&tmp1,&tmp3);
216     Fp2_mul(&d12_frobenius_constant[f_p4][5],&tmp2,&tmp3);
217
218     //d12_frobenius_constant[f_p8]
219     mpz_sub_ui(exp,p8,1);
220     mpz_tdiv_q_ui(exp,exp,3);
221     Fp2_pow(&tmp1,&Fp2_basis,exp);
222     Fp2_mul(&tmp2,&tmp1,&tmp1);
223     mpz_tdiv_q_ui(exp,exp,2);
224     Fp2_pow(&tmp3,&Fp2_basis,exp);
225     //set d12_frobenius_constant[f_p8]
226     Fp_set_ui(&d12_frobenius_constant[f_p8][0].x0,1);
227     Fp2_set(&d12_frobenius_constant[f_p8][1],&tmp1);
228     Fp2_set(&d12_frobenius_constant[f_p8][2],&tmp2);
229     Fp2_set(&d12_frobenius_constant[f_p8][3],&tmp3);
230     Fp2_mul(&d12_frobenius_constant[f_p8][4],&tmp1,&tmp3);
231     Fp2_mul(&d12_frobenius_constant[f_p8][5],&tmp2,&tmp3);
232
233     //frobenius_10
234     mpz_sub_ui(exp,p10,1);
235     mpz_tdiv_q_ui(exp,exp,3);
236     Fp2_pow(&tmp1,&Fp2_basis,exp);
237     Fp2_mul(&tmp2,&tmp1,&tmp1);
238     mpz_tdiv_q_ui(exp,exp,2);
239     Fp2_pow(&tmp3,&Fp2_basis,exp);
240     //set frobenius_10
241     Fp_set_ui(&d12_frobenius_constant[f_p10][0].x0,1);
242     Fp2_set(&d12_frobenius_constant[f_p10][1],&tmp1);
243     Fp2_set(&d12_frobenius_constant[f_p10][2],&tmp2);
244     Fp2_set(&d12_frobenius_constant[f_p10][3],&tmp3);
245     Fp2_mul(&d12_frobenius_constant[f_p10][4],&tmp1,&tmp3);
246     Fp2_mul(&d12_frobenius_constant[f_p10][5],&tmp2,&tmp3);
247     //set skew_f_10
248     Fp2_inv(&tmp1,&tmp1);
249     mpz_sub_ui(exp,p10,1);
250     mpz_tdiv_q_ui(exp,exp,2);
251     Fp2_pow(&tmp2,&Fp2_basis,exp);
252     Fp2_inv(&tmp2,&tmp2);
253     Fp2_set(&d12_skew_frobenius_constant[f_p10][0],&tmp1);
254     Fp2_set(&d12_skew_frobenius_constant[f_p10][1],&tmp2);
255
256     Fp2_clear(&tmp1);
257     Fp2_clear(&tmp2);
258     Fp2_clear(&tmp3);
259     mpz_clear(exp);
260     mpz_clear(buf);
261     mpz_clear(p2);
262     mpz_clear(p3);
263     mpz_clear(p4);
264     mpz_clear(p6);
265     mpz_clear(p8);
266     mpz_clear(p10);
267 }
```

Here is the call graph for this function:

8.18.4 Variable Documentation

8.18.4.1 d12_frobenius_constant

```
Fp2 d12_frobenius_constant[d12][6]
```

Constant values for Frobenius map (FM) to multiplied during FM calculations.

Referenced by bls12_Fp12_frobenius_map_p1(), bls12_Fp12_frobenius_map_p10(), bls12_Fp12_frobenius_map_p2(), bls12_Fp12_frobenius_map_p3(), bls12_Fp12_frobenius_map_p4(), and bls12_Fp12_frobenius_map_p8().

8.18.4.2 d12_skew_frobenius_constant

```
Fp2 d12_skew_frobenius_constant[d12][2]
```

Constant values for skew Frobenius map (FM) to multiplied during skew FM calculations.

Referenced by bls12_EFp2_skew_frobenius_map_p1(), bls12_EFp2_skew_frobenius_map_p10(), bls12_EFp2_skew_frobenius_map_p2(), and bls12_EFp2_skew_frobenius_map_p3().

8.18.4.3 epsilon1

```
mpz_t epsilon1
```

Primitive Cube root of 1.

Referenced by bls12_EFp_skew_frobenius_map_p2(), clear_parameters(), get_epsilon(), and init_precoms().

8.18.4.4 Fp2_basis

```
struct Fp2 Fp2_basis
```

Basis element in Fp2 extension field.

Referenced by clear_parameters(), init_precoms(), set_basis(), and set_frobenius_constant().

8.18.4.5 Fp2_basis_inv

```
struct Fp2 Fp2_basis_inv
```

Inverted Fp2 basis element.

Referenced by init_precoms(), and set_basis().

8.18.4.6 Fp6_basis

```
struct Fp6 Fp6_basis
```

Basis element in Fp6 extension field.

Referenced by clear_parameters(), init_precoms(), and set_basis().

8.18.4.7 Fp_basis

```
struct Fp Fp_basis
```

Basis element in prime field.

Referenced by clear_parameters(), init_precoms(), and set_basis().

8.19 include/ELiPS_bn_bls/bn_clears.h File Reference

```
#include <ELiPS_bn_bls/bn_inits.h>
```
Include dependency graph for bn_clears.h:

Functions

- void clear_parameters (void)

8.19.1 Detailed Description

Interaface to clear curve params and constants after using curves for pairing.

8.19.2 Function Documentation

8.19.2.1 clear_parameters()

```
void clear_parameters (
            void  )
```

Clear curve parameter and constants

References curve_params::curve_a, curve_params::curve_b, curve_parameters, curve_params::EFp_total, epsilon1, Fp2_basis, Fp6_basis, Fp_basis, Fp_clear(), curve_params::order, curve_params::prime, curve_params::trace_t, and curve_params::X.

```
33                      {
34
35     //    if (isCleared != 1){
36     int i,j;
37 //    unsigned int base = 10;
38
39     mpz_clear(curve_parameters.X);
40     mpz_clear(curve_parameters.prime);
41     mpz_clear(curve_parameters.order);
42     mpz_clear(curve_parameters.trace_t);
43
44     mpz_clear(curve_parameters.EFp_total);
45
46 //    printf(" has exact length %zu in base %d\n", strlen(mpz_get_str(NULL, base,
       curve_parameters.EFpd_total)), base);
47 //    printf("Size in base %d\n ",(int)mpz_sizeinbase(curve_parameters.EFpd_total,10));
48 //    i = (int)strlen(mpz_get_str(NULL, base, curve_parameters.EFpd_total));
49 //    if (i > 10) {
50 //    mpz_clear(curve_parameters.EFpd_total);
51 //    }
52 //    printf("Size in base after %d\n ",(int)mpz_sizeinbase(curve_parameters.EFpd_total,2));
53     mpz_clear(curve_parameters.curve_b);
54     mpz_clear(curve_parameters.curve_a);
55
56     Fp_clear(&Fp_basis);
57     Fp2_clear(&Fp2_basis);
58     Fp6_clear(&Fp6_basis);
59     mpz_clear(epsilon1);
60     mpz_clear(epsilon2);
61
62     for(i=0; i<d12; i++){
63         for(j=0; j<6; j++){
64             Fp2_clear(&d12_frobenius_constant[i][j]);
65         }
66         for(j=0; j<2; j++){
67             Fp2_clear(&d12_skew_frobenius_constant[i][j]);
68         }
69     }
70     //    }
71     //    isCleared = 1;
72 }
```

Here is the call graph for this function:

8.20 include/ELiPS_bn_bls/bn_efp.h File Reference

```
#include <ELiPS_bn_bls/curve_dtypes.h>
```

Include dependency graph for bn_efp.h: This graph shows which files directly or indirectly include this file:

Functions

- void EFp_init (EFp *P)
- void EFp_clear (EFp *P)
- void EFp_printf (EFp *P, char *str)
- void EFp_set (EFp *ANS, EFp *P)
- void EFp_set_ui (EFp *ANS, unsigned long int UI)
- void EFp_set_mpz (EFp *ANS, mpz_t A)
- void EFp_set_neg (EFp *ANS, EFp *P)
- void EFp_rational_point_bn (EFp *P)
- void EFp_rational_point_bls12 (EFp *P)
- void EFp_ECD (EFp *ANS, EFp *P)
- void EFp_ECA (EFp *ANS, EFp *P1, EFp *P2)
- void EFp_SCM (EFp *ANS, EFp *P, mpz_t scalar)

8.20.1 Detailed Description

Interaface for ellipti cuve operation in BN and BLS12 curve

8.20.2 Function Documentation

8.20.2.1 EFp_clear()

```
void EFp_clear (
            EFp * P )
```

clear of memory an EFp type structure

Parameters

in	*P*	- input P in EFp.

References Fp_clear(), and EFp::y.

Referenced by bls12_f_ltp_vtp_for_tate(), bls12_ff_ltt_vtt_for_tate(), bls12_generate_G1_point(), bls12_Miller_algo_for_opt_ate(), bls12_Miller_algo_for_plain_ate(), bls12_Miller_algo_for_tate(), bls12_plain_G1_scm(), bls12_Pseudo_8_sparse_mapping(), EFp_ECA(), EFp_ECD(), and EFp_SCM().

```
37                      {
38     Fp_clear(&P->x);
39     Fp_clear(&P->y);
40 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.20.2.2 EFp_ECA()

```
void EFp_ECA (
              EFp * ANS,
              EFp * P1,
              EFp * P2 )
```

Elliptiptic curve addition in EFp ANS = P1+P2 for BN and BLS curve

Parameters

out	*ANS*	- output
in	*P1*	- passed input pointer in EFp
in	*P1*	- passed input pointer in EFp

References EFp_clear(), EFp_ECD(), EFp_init(), EFp_set(), Fp_clear(), Fp_cmp(), Fp_init(), Fp_inv(), Fp_mul(), Fp_sub(), EFp::infinity, and EFp::y.

Referenced by bls12_2split_G1_scm(), and EFp_SCM().

```
162                                     {
163     if(P1->infinity==1){
164         EFp_set(ANS,P2);
165         return;
166     }else if(P2->infinity==1){
167         EFp_set(ANS,P1);
168         return;
169     }else if(Fp_cmp(&P1->x,&P2->x)==0){
170         if(Fp_cmp(&P1->y,&P2->y)!=0){
171             ANS->infinity=1;
172             return;
173         }else{
174             EFp_ECD(ANS,P1);
175             return;
176         }
177     }
178
179     EFp Tmp_P1,Tmp_P2;
180     EFp_init(&Tmp_P1);
181     EFp_set(&Tmp_P1,P1);
182     EFp_init(&Tmp_P2);
183     EFp_set(&Tmp_P2,P2);
184     Fp tmp1,tmp2,lambda;
185     Fp_init(&tmp1);
186     Fp_init(&tmp2);
187     Fp_init(&lambda);
188
189     Fp_sub(&tmp1,&Tmp_P2.x,&Tmp_P1.x);
190     Fp_inv(&tmp1,&tmp1);
191     Fp_sub(&tmp2,&Tmp_P2.y,&Tmp_P1.y);
192     Fp_mul(&lambda,&tmp1,&tmp2);
193     Fp_mul(&tmp1,&lambda,&lambda);
194     Fp_sub(&tmp2,&tmp1,&Tmp_P1.x);
195     Fp_sub(&ANS->x,&tmp2,&Tmp_P2.x);
196     Fp_sub(&tmp1,&Tmp_P1.x,&ANS->x);
197     Fp_mul(&tmp2,&lambda,&tmp1);
198     Fp_sub(&ANS->y,&tmp2,&Tmp_P1.y);
199
200     //clear
201     Fp_clear(&tmp1);
202     Fp_clear(&tmp2);
203     Fp_clear(&lambda);
204     EFp_clear(&Tmp_P1);
205     EFp_clear(&Tmp_P2);
206 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.20.2.3 EFp_ECD()

```
void EFp_ECD (
              EFp * ANS,
              EFp * P )
```

Elliptiptic curve doubling ANS = 2[P] in EFp for BN and BLS curve

Parameters

out	*ANS*	- output
in	*P*	- passed input pointer in EFp

References EFp_clear(), EFp_init(), EFp_set(), Fp_clear(), Fp_cmp_zero(), Fp_init(), Fp_inv(), Fp_mul(), Fp_mul_ui(), Fp_sub(), EFp::infinity, and EFp::y.

Referenced by EFp_ECA(), and EFp_SCM().

```
129                                 {
130     if(Fp_cmp_zero(&P->y)==0){
131         ANS->infinity=1;
132         return;
133     }
134 
135     EFp Tmp_P;
136     EFp_init(&Tmp_P);
137     EFp_set(&Tmp_P,P);
138     Fp tmp1,tmp2,lambda;
139     Fp_init(&tmp1);
140     Fp_init(&tmp2);
141     Fp_init(&lambda);
142 
143     Fp_mul_ui(&tmp1,&Tmp_P.y,2);
144     Fp_inv(&tmp1,&tmp1);
145     Fp_mul(&tmp2,&Tmp_P.x,&Tmp_P.x);
146     Fp_mul_ui(&tmp2,&tmp2,3);
147     Fp_mul(&lambda,&tmp1,&tmp2);
148     Fp_mul(&tmp1,&lambda,&lambda);
149     Fp_mul_ui(&tmp2,&Tmp_P.x,2);
150     Fp_sub(&ANS->x,&tmp1,&tmp2);
151     Fp_sub(&tmp1,&Tmp_P.x,&ANS->x);
152     Fp_mul(&tmp2,&lambda,&tmp1);
153     Fp_sub(&ANS->y,&tmp2,&Tmp_P.y);
154 
155     //clear
156     Fp_clear(&tmp1);
157     Fp_clear(&tmp2);
158     Fp_clear(&lambda);
159     EFp_clear(&Tmp_P);
160 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.20.2.4 EFp_init()

```
void EFp_init (
              EFp * P )
```

Initialize an EFp type structure

Parameters

in	*P*	- input P in EFp.

References Fp_init(), EFp::infinity, and EFp::y.

Referenced by bls12_2split_G1_scm(), bls12_f_ltp_vtp_for_tate(), bls12_ff_ltt_vtt_for_tate(), bls12_generate_G1_point(), bls12_Miller_algo_for_opt_ate(), bls12_Miller_algo_for_plain_ate(), bls12_Miller_algo_for_tate(), bls12_plain_G1_scm(), bls12_Pseudo_8_sparse_mapping(), EFp_ECA(), EFp_ECD(), and EFp_SCM().

```
31                      {
32     Fp_init(&P->x);
33     Fp_init(&P->y);
34     P->infinity=0;
35 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.20.2.5 EFp_printf()

```
void EFp_printf (
             EFp * P,
             char * str )
```

Prints an EFp type structure

Parameters

in	*P*	- input P in EFp.
in	*str*	- any string to print.

References Fp_printf(), EFp::infinity, and EFp::y.

```
42                                  {
43     gmp_printf("%s",str);
44     if(P->infinity==0){
45         gmp_printf("(");
46         Fp_printf(&P->x,"");
47         gmp_printf(",");
48         Fp_printf(&P->y,"");
49         gmp_printf(")");
50     }else{
51         gmp_printf("infinity");
52     }
53 }
```

Here is the call graph for this function:

8.20.2.6 EFp_rational_point_bls12()

```
void EFp_rational_point_bls12 (
             EFp * P )
```

Generate rational point in EFp for BLS12 curve

Parameters

in	*P*	- passed input pointer in EFp and obtain genterated point

References curve_params::curve_b, curve_parameters, Fp_add_mpz(), Fp_clear(), Fp_init(), Fp_legendre(), Fp_mul(), Fp_set_random(), Fp_sqrt(), and EFp::y.

Referenced by bls12_generate_G1_point().

```
104                                     {
105     Fp tmp1,tmp2,tmp_x;
106     Fp_init(&tmp1);
107     Fp_init(&tmp2);
108     Fp_init(&tmp_x);
109     gmp_randstate_t state;
110     gmp_randinit_default (state);
111     gmp_randseed_ui(state,(unsigned long)time(NULL));
112 
113     while(1){
114         Fp_set_random(&P->x,state);
115         Fp_mul(&tmp1,&P->x,&P->x);
116         Fp_mul(&tmp2,&tmp1,&P->x);
117         Fp_add_mpz(&tmp_x,&tmp2,curve_parameters.
    curve_b);
118         if(Fp_legendre(&tmp_x)==1){
119             Fp_sqrt(&P->y,&tmp_x);
120             break;
121         }
122     }
123 
124     Fp_clear(&tmp1);
125     Fp_clear(&tmp2);
126     Fp_clear(&tmp_x);
127 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.20.2.7 EFp_rational_point_bn()

```
void EFp_rational_point_bn (
             EFp * P )
```

Generate rational point in EFp for BN curve

Parameters

in	*P*	- passed input pointer in EFp and obtain genterated point

References curve_params::curve_b, curve_parameters, Fp_clear(), Fp_init(), Fp_legendre(), Fp_mul(), Fp_set_random(), Fp_sqrt(), Fp_sub_mpz(), and EFp::y.

```
79                                     {
80     Fp tmp1,tmp2,tmp_x;
81     Fp_init(&tmp1);
82     Fp_init(&tmp2);
83     Fp_init(&tmp_x);
84     gmp_randstate_t state;
85     gmp_randinit_default (state);
86     gmp_randseed_ui(state,(unsigned long)time(NULL));
87 
88     while(1){
89         Fp_set_random(&P->x,state);
90         Fp_mul(&tmp1,&P->x,&P->x);
91         Fp_mul(&tmp2,&tmp1,&P->x);
92         Fp_sub_mpz(&tmp_x,&tmp2,curve_parameters.
    curve_b);
93         if(Fp_legendre(&tmp_x)==1){
94             Fp_sqrt(&P->y,&tmp_x);
95             break;
96         }
97     }
98 
```

```
99     Fp_clear(&tmp1);
100     Fp_clear(&tmp2);
101     Fp_clear(&tmp_x);
102 }
```

Here is the call graph for this function:

8.20.2.8 EFp_SCM()

```
void EFp_SCM (
              EFp * ANS,
              EFp * P,
              mpz_t scalar )
```

Elliptiptic Scalar Multiplication (SCM) in EFp ANS = [scalar]P for BN and BLS curve

Parameters

out	*ANS*	- output
in	*P*	- passed input pointer in EFp
in	*P1*	- input integer

References EFp_clear(), EFp_ECA(), EFp_ECD(), EFp_init(), EFp_set(), and EFp::infinity.

Referenced by bls12_plain_G1_scm().

```
208                                         {
209     if(mpz_cmp_ui(scalar,0)==0){
210         ANS->infinity=1;
211         return;
212     }else if(mpz_cmp_ui(scalar,1)==0){
213         EFp_set(ANS,P);
214         return;
215     }
216
217     EFp Tmp_P,Next_P;
218     EFp_init(&Tmp_P);
219     EFp_set(&Tmp_P,P);
220     EFp_init(&Next_P);
221     int i,length;
222     length=(int)mpz_sizeinbase(scalar,2);
223     char binary[length];
224     mpz_get_str(binary,2,scalar);
225
226     EFp_set(&Next_P,&Tmp_P);
227     for(i=1; i<length; i++){
228         EFp_ECD(&Next_P,&Next_P);
229         if(binary[i]=='1'){
230             EFp_ECA(&Next_P,&Next_P,&Tmp_P);
231         }
232     }
233
234     EFp_set(ANS,&Next_P);
235
236     EFp_clear(&Next_P);
237     EFp_clear(&Tmp_P);
238 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.20.2.9 EFp_set()

```
void EFp_set (
              EFp * ANS,
              EFp * P )
```

Sets an EFp type structure

Parameters

out	*ANS*	- output ANS is set as P in EFp.
in	*P*	- input P in EFp.

References Fp_set(), EFp::infinity, and EFp::y.

Referenced by bls12_2split_G1_scm(), bls12_f_ltp_vtp_for_tate(), bls12_ff_ltt_vtt_for_tate(), bls12_Miller_algo_↩ for_tate(), bls12_Pseudo_8_sparse_mapping(), EFp_ECA(), EFp_ECD(), and EFp_SCM().

```
55                                {
56     Fp_set(&ANS->x,&P->x);
57     Fp_set(&ANS->y,&P->y);
58     ANS->infinity=P->infinity;
59 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.20.2.10 EFp_set_mpz()

```
void EFp_set_mpz (
             EFp * ANS,
             mpz_t A )
```

Sets an EFp type structure

Parameters

out	*ANS*	- output ANS is set as EFp.
in	*UI*	- input in GMP mpz_t

References Fp_set_mpz(), EFp::infinity, and EFp::y.

```
67                                    {
68     Fp_set_mpz(&ANS->x,A);
69     Fp_set_mpz(&ANS->y,A);
70     ANS->infinity=0;
71 }
```

Here is the call graph for this function:

8.20.2.11 EFp_set_neg()

```
void EFp_set_neg (
             EFp * ANS,
             EFp * P )
```

Negate EFp

Parameters

out	*ANS*	- output pointer.
in	*UI*	- input pointer in EFp

References Fp_set(), Fp_set_neg(), EFp::infinity, and EFp::y.

```
73                                  {
74     Fp_set(&ANS->x,&P->x);
75     Fp_set_neg(&ANS->y,&P->y);
76     ANS->infinity=P->infinity;
77 }
```

Here is the call graph for this function:

8.20.2.12 EFp_set_ui()

```
void EFp_set_ui (
             EFp * ANS,
             unsigned long int UI )
```

Sets an EFp type structure

Parameters

out	*ANS*	- output ANS is set an unsigned long int.
in	*UI*	- input in unsigned long int

References Fp_set_ui(), EFp::infinity, and EFp::y.

```
61                                          {
62     Fp_set_ui(&ANS->x,UI);
63     Fp_set_ui(&ANS->y,UI);
64     ANS->infinity=0;
65 }
```

Here is the call graph for this function:

8.21 include/ELiPS_bn_bls/bn_efp12.h File Reference

```
#include <ELiPS_bn_bls/bn_efp6.h>
```

Include dependency graph for bn_efp12.h: This graph shows which files directly or indirectly include this file:

Functions

- void EFp12_init (EFp12 *P)
- void EFp12_clear (EFp12 *P)
- void EFp12_printf (EFp12 *P, char *str)
- void EFp12_set (EFp12 *ANS, EFp12 *P)

- void EFp12_set_ui (EFp12 *ANS, unsigned long int UI)
- void EFp12_set_mpz (EFp12 *ANS, mpz_t A)
- void EFp12_set_neg (EFp12 *ANS, EFp12 *P)
- void EFp12_rational_point_bn (EFp12 *P)
- void EFp12_rational_point_bls12 (EFp12 *P)
- void EFp12_ECD (EFp12 *ANS, EFp12 *P)
- void EFp12_ECA (EFp12 *ANS, EFp12 *P1, EFp12 *P2)
- void EFp12_SCM (EFp12 *ANS, EFp12 *P, mpz_t scalar)

8.21.1 Detailed Description

Interaface for elliptic cuve operation in BN and BLS12 curve in Fp2 extension field

8.21.2 Function Documentation

8.21.2.1 EFp12_clear()

```
void EFp12_clear (
            EFp12 * P )
```

Clear memory of an EFp12 type structure

Parameters

in	*P*	- input P in EFp12.

Referenced by bls12_generate_G2_point(), bls12_Miller_algo_for_plain_ate(), bls12_test_G1_scm(), bls12_test_G2_scm(), bls12_test_G3_exp(), bls12_test_opt_ate_pairing(), bls12_test_plain_ate_pairing(), bls12_test_tate_pairing(), EFp12_ECA(), EFp12_ECD(), and EFp12_SCM().

```
36                          {
37     Fp12_clear(&P->x);
38     Fp12_clear(&P->y);
39 }
```

Here is the caller graph for this function:

8.21.2.2 EFp12_ECA()

```
void EFp12_ECA (
            EFp12 * ANS,
            EFp12 * P1,
            EFp12 * P2 )
```

ECA in EFp12 rational point

Parameters

in	*ANS*	- output.
in	*P1*	- input P1 in EFp12.
in	*P2*	- input P2 in EFp12.

References EFp12_clear(), EFp12_ECD(), EFp12_init(), and EFp12_set().

Referenced by bls12_generate_G2_point(), and EFp12_SCM().

```
159                                                 {
160     if(P1->infinity==1){
161         EFp12_set(ANS,P2);
162         return;
163     }else if(P2->infinity==1){
164         EFp12_set(ANS,P1);
165         return;
166     }else if(Fp12_cmp(&P1->x,&P2->x)==0){
167         if(Fp12_cmp(&P1->y,&P2->y)!=0){
168             ANS->infinity=1;
169             return;
170         }else{
171             EFp12_ECD(ANS,P1);
172             return;
173         }
174     }
175
176     EFp12 Tmp_P1,Tmp_P2;
177     EFp12_init(&Tmp_P1);
178     EFp12_set(&Tmp_P1,P1);
179     EFp12_init(&Tmp_P2);
180     EFp12_set(&Tmp_P2,P2);
181     Fp12 tmp1,tmp2,lambda;
182     Fp12_init(&tmp1);
183     Fp12_init(&tmp2);
184     Fp12_init(&lambda);
185
186     Fp12_sub(&tmp1,&Tmp_P2.x,&Tmp_P1.x);
187     Fp12_inv(&tmp1,&tmp1);
188     Fp12_sub(&tmp2,&Tmp_P2.y,&Tmp_P1.y);
189     Fp12_mul(&lambda,&tmp1,&tmp2);
190     Fp12_sqr(&tmp1,&lambda);
191     Fp12_sub(&tmp2,&tmp1,&Tmp_P1.x);
192     Fp12_sub(&ANS->x,&tmp2,&Tmp_P2.x);
193     Fp12_sub(&tmp1,&Tmp_P1.x,&ANS->x);
194     Fp12_mul(&tmp2,&lambda,&tmp1);
195     Fp12_sub(&ANS->y,&tmp2,&Tmp_P1.y);
196
197     //clear
198     Fp12_clear(&tmp1);
199     Fp12_clear(&tmp2);
200     Fp12_clear(&lambda);
201     EFp12_clear(&Tmp_P1);
202     EFp12_clear(&Tmp_P2);
203 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.21.2.3 EFp12_ECD()

```
void EFp12_ECD (
              EFp12 * ANS,
              EFp12 * P )
```

ECD in EFp12 rational point

Parameters

in	*ANS*	- output.
in	*P*	- input P in EFp12.

References EFp12_clear(), EFp12_init(), and EFp12_set().

Referenced by EFp12_ECA(), and EFp12_SCM().

```
126                                     {
127     if(Fp12_cmp_zero(&P->y)==0){
128         ANS->infinity=1;
129         return;
130 
131     }
132 
133     EFp12 Tmp_P;
134     EFp12_init(&Tmp_P);
135     EFp12_set(&Tmp_P,P);
136     Fp12 tmp1,tmp2,lambda;
137     Fp12_init(&tmp1);
138     Fp12_init(&tmp2);
139     Fp12_init(&lambda);
140 
141     Fp12_mul_ui(&tmp1,&Tmp_P.y,2);
142     Fp12_inv(&tmp1,&tmp1);
143     Fp12_sqr(&tmp2,&Tmp_P.x);
144     Fp12_mul_ui(&tmp2,&tmp2,3);
145     Fp12_mul(&lambda,&tmp1,&tmp2);
146     Fp12_sqr(&tmp1,&lambda);
147     Fp12_mul_ui(&tmp2,&Tmp_P.x,2);
148     Fp12_sub(&ANS->x,&tmp1,&tmp2);
149     Fp12_sub(&tmp1,&Tmp_P.x,&ANS->x);
150     Fp12_mul(&tmp2,&lambda,&tmp1);
151     Fp12_sub(&ANS->y,&tmp2,&Tmp_P.y);
152 
153     Fp12_clear(&tmp1);
154     Fp12_clear(&tmp2);
155     Fp12_clear(&lambda);
156     EFp12_clear(&Tmp_P);
157 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.21.2.4 EFp12_init()

```
void EFp12_init (
              EFp12 * P )
```

Initialize an EFp12 type structure

Parameters

in	*P*	- input P in EFp12.

Referenced by bls12_2split_G1_scm(), bls12_generate_G2_point(), bls12_generate_random_point(), bls12_↩ Miller_algo_for_plain_ate(), bls12_test_G1_scm(), bls12_test_G2_scm(), bls12_test_G3_exp(), bls12_test_opt↩ _ate_pairing(), bls12_test_plain_ate_pairing(), bls12_test_tate_pairing(), EFp12_ECA(), EFp12_ECD(), and E↩ Fp12_SCM().

```
30                          {
31     Fp12_init(&P->x);
```

```
32     Fp12_init(&P->y);
33     P->infinity=0;
34 }
```

Here is the caller graph for this function:

8.21.2.5 EFp12_printf()

```
void EFp12_printf (
             EFp12 * P,
             char * str )
```

Prints an EFp12 rational point

Parameters

in	*P*	- input P as pointer in EFp12.
in	*str*	- input str as string.

Referenced by bls12_test_G1_scm(), bls12_test_G2_scm(), bls12_test_opt_ate_pairing(), bls12_test_plain_ate↩_pairing(), and bls12_test_tate_pairing().

```
41                                      {
42     gmp_printf("%s",str);
43     if(P->infinity==0){
44         gmp_printf("(");
45         Fp12_printf(&P->x,"");
46         gmp_printf(",");
47         Fp12_printf(&P->y,"");
48         gmp_printf(")");
49     }else{
50         gmp_printf("infinity");
51     }
52 }
```

Here is the caller graph for this function:

8.21.2.6 EFp12_rational_point_bls12()

```
void EFp12_rational_point_bls12 (
             EFp12 * P )
```

Generate an EFp12 rational point BLS12

Parameters

in	*P*	- input rational point P and.

References curve_params::curve_b, and curve_parameters.

Referenced by bls12_generate_G2_point(), bls12_generate_random_point(), and bls12_test_tate_pairing().

```
101                                     {
```

```
102     Fp12 tmp1,tmp2;
103     Fp12_init(&tmp1);
104     Fp12_init(&tmp2);
105     gmp_randstate_t state;
106     gmp_randinit_default (state);
107     gmp_randseed_ui(state,(unsigned long)time(NULL));
108 
109     while(1){
110         Fp12_set_random(&P->x,state);
111         Fp12_sqr(&tmp1,&P->x);
112         Fp12_mul(&tmp2,&tmp1,&P->x);
113         mpz_add(tmp2.x0.x0.x0.x0,tmp2.x0.x0.x0.x0, curve_parameters.
    curve_b);
114         if(Fp12_legendre(&tmp2)==1){
115             Fp12_sqrt(&P->y,&tmp2);
116             break;
117         }
118     }
119 
120     Fp12_clear(&tmp1);
121     Fp12_clear(&tmp2);
122 }
```

Here is the caller graph for this function:

8.21.2.7 EFp12_rational_point_bn()

```
void EFp12_rational_point_bn (
             EFp12 * P )
```

Generate an EFp12 rational point for BN curve

Parameters

in	*P*	- input rational point P and.

References curve_params::curve_b, and curve_parameters.

```
78                                      {
79     Fp12 tmp1,tmp2;
80     Fp12_init(&tmp1);
81     Fp12_init(&tmp2);
82     gmp_randstate_t state;
83     gmp_randinit_default (state);
84     gmp_randseed_ui(state,(unsigned long)time(NULL));
85 
86     while(1){
87         Fp12_set_random(&P->x,state);
88         Fp12_sqr(&tmp1,&P->x);
89         Fp12_mul(&tmp2,&tmp1,&P->x);
90         mpz_sub(tmp2.x0.x0.x0.x0,tmp2.x0.x0.x0.x0,curve_parameters.
    curve_b);
91         if(Fp12_legendre(&tmp2)==1){
92             Fp12_sqrt(&P->y,&tmp2);
93             break;
94         }
95     }
96 
97     Fp12_clear(&tmp1);
98     Fp12_clear(&tmp2);
99 }
```

8.21.2.8 EFp12_SCM()

```
void EFp12_SCM (
            EFp12 * ANS,
            EFp12 * P,
            mpz_t scalar )
```

SCM in EFp12 rational point

Parameters

in	*ANS*	- output.
in	*P1*	- input P1 in EFp12.
in	*scalar*	- input scalar in mpz_t integer.

References EFp12_clear(), EFp12_ECA(), EFp12_ECD(), EFp12_init(), and EFp12_set().

Referenced by bls12_generate_G1_point(), bls12_generate_G2_point(), bls12_test_opt_ate_pairing(), bls12_↩ test_plain_ate_pairing(), and bls12_test_tate_pairing().

```
205                                                 {
206     if(mpz_cmp_ui(scalar,0)==0){
207         ANS->infinity=1;
208         return;
209     }else if(mpz_cmp_ui(scalar,1)==0){
210         EFp12_set(ANS,P);
211         return;
212     }
213
214     EFp12 Tmp_P,Next_P;
215     EFp12_init(&Tmp_P);
216     EFp12_set(&Tmp_P,P);
217     EFp12_init(&Next_P);
218     int i,length;
219     length=(int)mpz_sizeinbase(scalar,2);
220     char binary[length];
221     mpz_get_str(binary,2,scalar);
222
223     EFp12_set(&Next_P,&Tmp_P);
224     for(i=1; i<length; i++){
225         EFp12_ECD(&Next_P,&Next_P);
226         if(binary[i]=='1'){
227             EFp12_ECA(&Next_P,&Next_P,&Tmp_P);
228         }
229     }
230     EFp12_set(ANS,&Next_P);
231
232     EFp12_clear(&Next_P);
233     EFp12_clear(&Tmp_P);
234 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.21.2.9 EFp12_set()

```
void EFp12_set (
            EFp12 * ANS,
            EFp12 * P )
```

Sets EFp12 point an EFp12 type structure

Parameters

in	*ANS*	- output.
in	*P*	- input P in EFp12.

Referenced by EFp12_ECA(), EFp12_ECD(), and EFp12_SCM().

```
54                                      {
55     Fp12_set(&ANS->x,&P->x);
56     Fp12_set(&ANS->y,&P->y);
57     ANS->infinity=P->infinity;
58 }
```

Here is the caller graph for this function:

8.21.2.10 EFp12_set_mpz()

```
void EFp12_set_mpz (
            EFp12 * ANS,
            mpz_t A )
```

Sets mpz_t in an EFp12 type structure

Parameters

in	*ANS*	- output.
in	*str*	- input A in mpz_t int.

```
66                                          {
67     Fp12_set_mpz(&ANS->x,A);
68     Fp12_set_mpz(&ANS->y,A);
69     ANS->infinity=0;
70 }
```

8.21.2.11 EFp12_set_neg()

```
void EFp12_set_neg (
            EFp12 * ANS,
            EFp12 * P )
```

Negate an EFp12 type structure

Parameters

in	*ANS*	- output.
in	*P*	- input P in EFp12.

Referenced by bls12_generate_G2_point().

```
72                                          {
73     Fp12_set(&ANS->x,&P->x);
74     Fp12_set_neg(&ANS->y,&P->y);
75     ANS->infinity=P->infinity;
76 }
```

Here is the caller graph for this function:

8.21.2.12 EFp12_set_ui()

```
void EFp12_set_ui (
              EFp12 * ANS,
              unsigned long int UI )
```

Sets unsigned int an EFp12 type structure

Parameters

in	*ANS*	- output.
in	*u*	- input u in us integer.

```
60                                              {
61     Fp12_set_ui(&ANS->x,UI);
62     Fp12_set_ui(&ANS->y,UI);
63     ANS->infinity=0;
64 }
```

8.22 include/ELiPS_bn_bls/bn_efp2.h File Reference

`#include <ELiPS_bn_bls/bn_efp.h>`
Include dependency graph for bn_efp2.h: This graph shows which files directly or indirectly include this file:

Functions

- void EFp2_init (EFp2 *P)
- void EFp2_clear (EFp2 *P)
- void EFp2_printf (EFp2 *P, char *str)
- void EFp2_set (EFp2 *ANS, EFp2 *P)
- void EFp2_set_ui (EFp2 *ANS, unsigned long int u)
- void EFp2_set_mpz (EFp2 *ANS, mpz_t A)
- void EFp2_set_neg (EFp2 *ANS, EFp2 *P)
- void EFp2_rational_point (EFp2 *P)
- void EFp2_ECD (EFp2 *ANS, EFp2 *P)
- void EFp2_ECA (EFp2 *ANS, EFp2 *P1, EFp2 *P2)
- void EFp2_SCM (EFp2 *ANS, EFp2 *P, mpz_t scalar)

8.22.1 Detailed Description

Interaface for elliptic cuve operation in BN and BLS12 curve in Fp2 extension field

8.22.2 Function Documentation

8.22.2.1 EFp2_clear()

```
void EFp2_clear (
            EFp2 * P )
```

Clears an EFp2 type structure

Parameters

in	*P*	- input P in EFp2.

Referenced by bls12_f_ltq(), bls12_ff_ltt(), bls12_Miller_algo_for_opt_ate(), bls12_Miller_algo_for_plain_ate(), bls12_plain_G2_scm(), bls12_Pseudo_8_sparse_mapping(), EFp2_ECA(), EFp2_ECD(), and EFp2_SCM().

```
38                          {
39     Fp2_clear(&P->x);
40     Fp2_clear(&P->y);
41 }
```

Here is the caller graph for this function:

8.22.2.2 EFp2_ECA()

```
void EFp2_ECA (
            EFp2 * ANS,
            EFp2 * P1,
            EFp2 * P2 )
```

ECA in EFp2 rational point

Parameters

in	*ANS*	- output.
in	*P1*	- input P1 in EFp2.
in	*P2*	- input P2 in EFp2.

References EFp2_clear(), EFp2_ECD(), EFp2_init(), and EFp2_set().

Referenced by bls12_2split_G2_scm(), bls12_4split_G2_scm(), and EFp2_SCM().

```
138                                         {
139     if(P1->infinity==1){
140         EFp2_set(ANS,P2);
141         return;
142     }else if(P2->infinity==1){
143         EFp2_set(ANS,P1);
144         return;
145     }else if(Fp2_cmp(&P1->x,&P2->x)==0){
```

```
146         if(Fp2_cmp(&P1->y,&P2->y)!=0){
147             ANS->infinity=1;
148             return;
149         }else{
150             EFp2_ECD(ANS,P1);
151             return;
152         }
153     }
154 
155     EFp2 Tmp_P1,Tmp_P2;
156     EFp2_init(&Tmp_P1);
157     EFp2_set(&Tmp_P1,P1);
158     EFp2_init(&Tmp_P2);
159     EFp2_set(&Tmp_P2,P2);
160     Fp2 tmp1,tmp2,lambda;
161     Fp2_init(&tmp1);
162     Fp2_init(&tmp2);
163     Fp2_init(&lambda);
164 
165     Fp2_sub(&tmp1,&Tmp_P2.x,&Tmp_P1.x);
166     Fp2_inv(&tmp1,&tmp1);
167     Fp2_sub(&tmp2,&Tmp_P2.y,&Tmp_P1.y);
168     Fp2_mul(&lambda,&tmp1,&tmp2);
169     Fp2_sqr(&tmp1,&lambda);
170     Fp2_sub(&tmp2,&tmp1,&Tmp_P1.x);
171     Fp2_sub(&ANS->x,&tmp2,&Tmp_P2.x);
172     Fp2_sub(&tmp1,&Tmp_P1.x,&ANS->x);
173     Fp2_mul(&tmp2,&lambda,&tmp1);
174     Fp2_sub(&ANS->y,&tmp2,&Tmp_P1.y);
175 
176     //clear
177     Fp2_clear(&tmp1);
178     Fp2_clear(&tmp2);
179     Fp2_clear(&lambda);
180     EFp2_clear(&Tmp_P1);
181     EFp2_clear(&Tmp_P2);
182 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.22.2.3 EFp2_ECD()

```
void EFp2_ECD (
              EFp2 * ANS,
              EFp2 * P )
```

ECD in EFp2 rational point

Parameters

in	*ANS*	- output.
in	*P*	- input P in EFp2.

References EFp2_clear(), EFp2_init(), and EFp2_set().

Referenced by EFp2_ECA(), and EFp2_SCM().

```
103                                 {
104     if(Fp2_cmp_zero(&P->y)==0){
105         ANS->infinity=1;
106         return;
107     }
108 
109     EFp2 Tmp_P;
110     EFp2_init(&Tmp_P);
111     EFp2_set(&Tmp_P,P);
112     Fp2 tmp1,tmp2,lambda;
113     Fp2_init(&tmp1);
114     Fp2_init(&tmp2);
```

```
115     Fp2_init(&lambda);
116
117     Fp2_mul_ui(&tmp1,&Tmp_P.y,2);
118
119     Fp2_inv(&tmp1,&tmp1);
120     Fp2_mul(&tmp2,&Tmp_P.x,&Tmp_P.x);
121     Fp2_mul_ui(&tmp2,&tmp2,3);
122     Fp2_mul(&lambda,&tmp1,&tmp2);
123
124     Fp2_sqr(&tmp1,&lambda);
125     Fp2_mul_ui(&tmp2,&Tmp_P.x,2);
126     Fp2_sub(&ANS->x,&tmp1,&tmp2);
127
128     Fp2_sub(&tmp1,&Tmp_P.x,&ANS->x);
129     Fp2_mul(&tmp2,&lambda,&tmp1);
130     Fp2_sub(&ANS->y,&tmp2,&Tmp_P.y);
131
132     Fp2_clear(&tmp1);
133     Fp2_clear(&tmp2);
134     Fp2_clear(&lambda);
135     EFp2_clear(&Tmp_P);
136 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.22.2.4 EFp2_init()

```
void EFp2_init (
            EFp2 * P )
```

Initialize an EFp type structure

Parameters

in	*P*	- input P in EFp2.

Referenced by bls12_2split_G2_scm(), bls12_4split_G2_scm(), bls12_f_ltq(), bls12_ff_ltt(), bls12_Miller_algo_↩ for_opt_ate(), bls12_Miller_algo_for_plain_ate(), bls12_plain_G2_scm(), bls12_Pseudo_8_sparse_mapping(), E↩ Fp2_ECA(), EFp2_ECD(), and EFp2_SCM().

```
32                      {
33     Fp2_init(&P->x);
34     Fp2_init(&P->y);
35     P->infinity=0;
36 }
```

Here is the caller graph for this function:

8.22.2.5 EFp2_printf()

```
void EFp2_printf (
            EFp2 * P,
            char * str )
```

Prints an EFp2 type structure

Parameters

in	*P*	- input P in EFp2.
in	*str*	- any string to print.

```
43                                  {
44     gmp_printf("%s",str);
45     if(P->infinity==0){
46         gmp_printf("(");
47         Fp2_printf(&P->x,"");
48         gmp_printf(",");
49         Fp2_printf(&P->y,"");
50         gmp_printf(")");
51     }else{
52         gmp_printf("infinity");
53     }
54 }
```

8.22.2.6 EFp2_rational_point()

```
void EFp2_rational_point (
             EFp2 * P )
```

Generate an EFp2 rational point

Parameters

in	*P*	- input rational point P and.

References curve_params::curve_b, and curve_parameters.

```
80                                  {
81     Fp2 tmp1,tmp2;
82     Fp2_init(&tmp1);
83     Fp2_init(&tmp2);
84     gmp_randstate_t state;
85     gmp_randinit_default (state);
86     gmp_randseed_ui(state,(unsigned long)time(NULL));
87
88     while(1){
89         Fp2_set_random(&P->x,state);
90         Fp2_sqr(&tmp1,&P->x);
91         Fp2_mul(&tmp2,&tmp1,&P->x);
92         mpz_sub(tmp2.x0.x0,tmp2.x0.x0,curve_parameters.curve_b);
93         if(Fp2_legendre(&tmp2)==1){
94             Fp2_sqrt(&P->y,&tmp2);
95             break;
96         }
97     }
98
99     Fp2_clear(&tmp1);
100     Fp2_clear(&tmp2);
101 }
```

8.22.2.7 EFp2_SCM()

```
void EFp2_SCM (
             EFp2 * ANS,
             EFp2 * P,
             mpz_t scalar )
```

ECA in EFp2 rational point

Parameters

in	*ANS*	- output.
in	*P1*	- input P1 in EFp2.
in	*scalar*	- input scalar in mpz_t integer.

References EFp2_clear(), EFp2_ECA(), EFp2_ECD(), EFp2_init(), and EFp2_set().

Referenced by bls12_plain_G2_scm().

```
184                                                 {
185     if(mpz_cmp_ui(scalar,0)==0){
186         ANS->infinity=1;
187         return;
188     }else if(mpz_cmp_ui(scalar,1)==0){
189         EFp2_set(ANS,P);
190         return;
191     }
192 
193     EFp2 Tmp_P,Next_P;
194     EFp2_init(&Tmp_P);
195     EFp2_set(&Tmp_P,P);
196     EFp2_init(&Next_P);
197     int i,length;
198     length=(int)mpz_sizeinbase(scalar,2);
199     char binary[length];
200     mpz_get_str(binary,2,scalar);
201 
202     EFp2_set(&Next_P,&Tmp_P);
203     for(i=1; i<length; i++){
204         EFp2_ECD(&Next_P,&Next_P);
205         if(binary[i]=='1'){
206             EFp2_ECA(&Next_P,&Next_P,&Tmp_P);
207         }
208     }
209     EFp2_set(ANS,&Next_P);
210 
211     EFp2_clear(&Next_P);
212     EFp2_clear(&Tmp_P);
213 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.22.2.8 EFp2_set()

```
void EFp2_set (
              EFp2 * ANS,
              EFp2 * P )
```

Sets EFp2 point an EFp2 type structure

Parameters

in	*ANS*	- output.
in	*P*	- input P in EFp2.

Referenced by bls12_2split_G2_scm(), bls12_4split_G2_scm(), bls12_f_ltq(), bls12_ff_ltt(), bls12_Miller_algo_↩ for_opt_ate(), bls12_Miller_algo_for_plain_ate(), bls12_Pseudo_8_sparse_mapping(), EFp2_ECA(), EFp2_ECD(), and EFp2_SCM().

```
56                                  {
```

```
57     Fp2_set(&ANS->x,&P->x);
58     Fp2_set(&ANS->y,&P->y);
59     ANS->infinity=P->infinity;
60 }
```

Here is the caller graph for this function:

8.22.2.9 EFp2_set_mpz()

```
void EFp2_set_mpz (
            EFp2 * ANS,
            mpz_t A )
```

Sets mpz_t in an EFp2 type structure

Parameters

in	*ANS*	- output.
in	*str*	- input A in mpz_t int.

```
68                                  {
69     Fp2_set_mpz(&ANS->x,A);
70     Fp2_set_mpz(&ANS->y,A);
71     ANS->infinity=0;
72 }
```

8.22.2.10 EFp2_set_neg()

```
void EFp2_set_neg (
            EFp2 * ANS,
            EFp2 * P )
```

Negate an EFp2 type structure

Parameters

in	*ANS*	- output.
in	*P*	- input P in EFp2.

Referenced by bls12_4split_G2_scm(), bls12_Miller_algo_for_opt_ate(), and bls12_Miller_algo_for_plain_ate().

```
74                                  {
75     Fp2_set(&ANS->x,&P->x);
76     Fp2_set_neg(&ANS->y,&P->y);
77     ANS->infinity=P->infinity;
78 }
```

Here is the caller graph for this function:

8.22.2.11 EFp2_set_ui()

```
void EFp2_set_ui (
            EFp2 * ANS,
            unsigned long int u )
```

Sets unsigned int an EFp2 type structure

Parameters

in	*ANS*	- output.
in	*u*	- input u in us integer.

```
62                                                  {
63     Fp2_set_ui(&ANS->x,UI);
64     Fp2_set_ui(&ANS->y,UI);
65     ANS->infinity=0;
66 }
```

8.23 include/ELiPS_bn_bls/bn_efp6.h File Reference

`#include <ELiPS_bn_bls/bn_efp2.h>`
Include dependency graph for bn_efp6.h: This graph shows which files directly or indirectly include this file:

Functions

- void EFp6_init (EFp6 ∗P)
- void EFp6_clear (EFp6 ∗P)
- void EFp6_printf (EFp6 ∗P, char ∗str)
- void EFp6_set (EFp6 ∗ANS, EFp6 ∗P)
- void EFp6_set_ui (EFp6 ∗ANS, unsigned long int UI)
- void EFp6_set_mpz (EFp6 ∗ANS, mpz_t A)
- void EFp6_set_neg (EFp6 ∗ANS, EFp6 ∗P)
- void EFp6_rational_point (EFp6 ∗P)
- void EFp6_ECD (EFp6 ∗ANS, EFp6 ∗P)
- void EFp6_ECA (EFp6 ∗ANS, EFp6 ∗P1, EFp6 ∗P2)
- void EFp6_SCM (EFp6 ∗ANS, EFp6 ∗P, mpz_t scalar)

8.23.1 Detailed Description

Interaface for elliptic cuve operation in BN and BLS12 curve in Fp2 extension field

8.23.2 Function Documentation

8.23.2.1 EFp6_clear()

```
void EFp6_clear (
            EFp6 * P )
```

Clears an EFp6 type structure

Parameters

in	*P*	- input P in EFp6.

Referenced by EFp6_ECA(), EFp6_ECD(), and EFp6_SCM().

```
36                          {
37     Fp6_clear(&P->x);
38     Fp6_clear(&P->y);
39 }
```

Here is the caller graph for this function:

8.23.2.2 EFp6_ECA()

```
void EFp6_ECA (
             EFp6 * ANS,
             EFp6 * P1,
             EFp6 * P2 )
```

ECA in EFp6 rational point

Parameters

in	*ANS*	- output.
in	*P1*	- input P1 in EFp6.
in	*P2*	- input P2 in EFp6.

References EFp6_clear(), EFp6_ECD(), EFp6_init(), and EFp6_set().

Referenced by EFp6_SCM().

```
134                                          {
135     if(P1->infinity==1){
136         EFp6_set(ANS,P2);
137         return;
138     }else if(P2->infinity==1){
139         EFp6_set(ANS,P1);
140         return;
141     }else if(Fp6_cmp(&P1->x,&P2->x)==0){
142         if(Fp6_cmp(&P1->y,&P2->y)!=0){
143             ANS->infinity=1;
144             return;
145         }else{
146             EFp6_ECD(ANS,P1);
147             return;
148         }
149     }
150
151     EFp6 Tmp_P1,Tmp_P2;
152     EFp6_init(&Tmp_P1);
153     EFp6_set(&Tmp_P1,P1);
154     EFp6_init(&Tmp_P2);
155     EFp6_set(&Tmp_P2,P2);
156     Fp6 tmp1,tmp2,lambda;
157     Fp6_init(&tmp1);
158     Fp6_init(&tmp2);
159     Fp6_init(&lambda);
160
161     Fp6_sub(&tmp1,&Tmp_P2.x,&Tmp_P1.x);
162     Fp6_inv(&tmp1,&tmp1);
163     Fp6_sub(&tmp2,&Tmp_P2.y,&Tmp_P1.y);
```

```
164     Fp6_mul(&lambda,&tmp1,&tmp2);
165     Fp6_sqr(&tmp1,&lambda);
166     Fp6_sub(&tmp2,&tmp1,&Tmp_P1.x);
167     Fp6_sub(&ANS->x,&tmp2,&Tmp_P2.x);
168     Fp6_sub(&tmp1,&Tmp_P1.x,&ANS->x);
169     Fp6_mul(&tmp2,&lambda,&tmp1);
170     Fp6_sub(&ANS->y,&tmp2,&Tmp_P1.y);
171
172     //clear
173     Fp6_clear(&tmp1);
174     Fp6_clear(&tmp2);
175     Fp6_clear(&lambda);
176     EFp6_clear(&Tmp_P1);
177     EFp6_clear(&Tmp_P2);
178 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.23.2.3 EFp6_ECD()

```
void EFp6_ECD (
              EFp6 * ANS,
              EFp6 * P )
```

ECD in EFp6 rational point

Parameters

in	*ANS*	- output.
in	*P*	- input P in EFp6.

References EFp6_clear(), EFp6_init(), and EFp6_set().

Referenced by EFp6_ECA(), and EFp6_SCM().

```
101                                 {
102     if(Fp6_cmp_zero(&P->y)==0){
103         ANS->infinity=1;
104         return;
105     }
106
107     EFp6 Tmp_P;
108     EFp6_init(&Tmp_P);
109     EFp6_set(&Tmp_P,P);
110     Fp6 tmp1,tmp2,lambda;
111     Fp6_init(&tmp1);
112     Fp6_init(&tmp2);
113     Fp6_init(&lambda);
114
115     Fp6_mul_ui(&tmp1,&Tmp_P.y,2);
116
117     Fp6_inv(&tmp1,&tmp1);
118     Fp6_mul(&tmp2,&Tmp_P.x,&Tmp_P.x);
119     Fp6_mul_ui(&tmp2,&tmp2,3);
120     Fp6_mul(&lambda,&tmp1,&tmp2);
121     Fp6_sqr(&tmp1,&lambda);
122     Fp6_mul_ui(&tmp2,&Tmp_P.x,2);
123     Fp6_sub(&ANS->x,&tmp1,&tmp2);
124     Fp6_sub(&tmp1,&Tmp_P.x,&ANS->x);
125     Fp6_mul(&tmp2,&lambda,&tmp1);
126     Fp6_sub(&ANS->y,&tmp2,&Tmp_P.y);
127
128     Fp6_clear(&tmp1);
129     Fp6_clear(&tmp2);
130     Fp6_clear(&lambda);
131     EFp6_clear(&Tmp_P);
132 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.23.2.4 EFp6_init()

```
void EFp6_init (
            EFp6 * P )
```

Initialize an EFp6 type structure

Parameters

in	*P*	- input P in EFp6.

Referenced by EFp6_ECA(), EFp6_ECD(), and EFp6_SCM().

```
30                          {
31     Fp6_init(&P->x);
32     Fp6_init(&P->y);
33     P->infinity=0;
34 }
```

Here is the caller graph for this function:

8.23.2.5 EFp6_printf()

```
void EFp6_printf (
            EFp6 * P,
            char * str )
```

Prints an EFp6 type structure

Parameters

in	*P*	- input P in EFp6.
in	*str*	- any string to print.

```
41                                  {
42     gmp_printf("%s",str);
43     if(P->infinity==0){
44         gmp_printf("(");
45         Fp6_printf(&P->x,"");
46         gmp_printf(",");
47         Fp6_printf(&P->y,"");
48         gmp_printf(")");
49     }else{
50         gmp_printf("infinity");
51     }
52 }
```

8.23.2.6 EFp6_rational_point()

```
void EFp6_rational_point (
            EFp6 * P )
```

Generate an EFp6 rational point

Parameters

in	*P*	- input rational point P and.

References curve_params::curve_b, and curve_parameters.

```
78                                  {
79     Fp6 tmp1,tmp2;
80     Fp6_init(&tmp1);
81     Fp6_init(&tmp2);
82     gmp_randstate_t state;
83     gmp_randinit_default (state);
84     gmp_randseed_ui(state,(unsigned long)time(NULL));
85 
86     while(1){
87         Fp6_set_random(&P->x,state);
88         Fp6_sqr(&tmp1,&P->x);
89         Fp6_mul(&tmp2,&tmp1,&P->x);
90         mpz_sub(tmp2.x0.x0.x0,tmp2.x0.x0.x0,curve_parameters.
     curve_b);
91         if(Fp6_legendre(&tmp2)==1){
92             Fp6_sqrt(&P->y,&tmp2);
93             break;
94         }
95     }
96 
97     Fp6_clear(&tmp1);
98     Fp6_clear(&tmp2);
99 }
```

8.23.2.7 EFp6_SCM()

```
void EFp6_SCM (
              EFp6 * ANS,
              EFp6 * P,
              mpz_t scalar )
```

SCM in EFp6 rational point

Parameters

in	*ANS*	- output.
in	*P1*	- input P1 in EFp6.
in	*scalar*	- input scalar in mpz_t integer.

References EFp6_clear(), EFp6_ECA(), EFp6_ECD(), EFp6_init(), and EFp6_set().

```
180                                             {
181     if(mpz_cmp_ui(scalar,0)==0){
182         ANS->infinity=1;
183         return;
184     }else if(mpz_cmp_ui(scalar,1)==0){
185         EFp6_set(ANS,P);
186         return;
187     }
188 
189     EFp6 Tmp_P,Next_P;
190     EFp6_init(&Tmp_P);
191     EFp6_set(&Tmp_P,P);
192     EFp6_init(&Next_P);
193     int i,length;
```

```
194     length=(int)mpz_sizeinbase(scalar,2);
195     char binary[length];
196     mpz_get_str(binary,2,scalar);
197
198     EFp6_set(&Next_P,&Tmp_P);
199     for(i=1; i<length; i++){
200         EFp6_ECD(&Next_P,&Next_P);
201         if(binary[i]=='1'){
202             EFp6_ECA(&Next_P,&Next_P,&Tmp_P);
203         }
204     }
205
206     EFp6_set(ANS,&Next_P);
207
208     EFp6_clear(&Next_P);
209     EFp6_clear(&Tmp_P);
210 }
```

Here is the call graph for this function:

8.23.2.8 EFp6_set()

```
void EFp6_set (
             EFp6 * ANS,
             EFp6 * P )
```

Sets EFp6 point an EFp6 type structure

Parameters

in	*ANS*	- output.
in	*P*	- input P in EFp6.

Referenced by EFp6_ECA(), EFp6_ECD(), and EFp6_SCM().

```
54                                  {
55     Fp6_set(&ANS->x,&P->x);
56     Fp6_set(&ANS->y,&P->y);
57     ANS->infinity=P->infinity;
58 }
```

Here is the caller graph for this function:

8.23.2.9 EFp6_set_mpz()

```
void EFp6_set_mpz (
             EFp6 * ANS,
             mpz_t A )
```

Sets mpz_t in an EFp6 type structure

Parameters

in	*ANS*	- output.
in	*str*	- input A in mpz_t int.

```
66                                      {
67     Fp6_set_mpz(&ANS->x,A);
68     Fp6_set_mpz(&ANS->y,A);
69     ANS->infinity=0;
70 }
```

8.23.2.10 EFp6_set_neg()

```
void EFp6_set_neg (
              EFp6 * ANS,
              EFp6 * P )
```

Negate an EFp6 type structure

Parameters

in	*ANS*	- output.
in	*P*	- input P in EFp6.

```
72                                      {
73     Fp6_set(&ANS->x,&P->x);
74     Fp6_set_neg(&ANS->y,&P->y);
75     ANS->infinity=P->infinity;
76 }
```

8.23.2.11 EFp6_set_ui()

```
void EFp6_set_ui (
              EFp6 * ANS,
              unsigned long int UI )
```

Sets unsigned int an EFp6 type structure

Parameters

in	*ANS*	- output.
in	*u*	- input u in us integer.

```
60                                              {
61     Fp6_set_ui(&ANS->x,UI);
62     Fp6_set_ui(&ANS->y,UI);
63     ANS->infinity=0;
64 }
```

8.24 include/ELiPS_bn_bls/bn_final_exp.h File Reference

```
#include <ELiPS_bn_bls/bn_frobenius.h>
```

Include dependency graph for bn_final_exp.h: This graph shows which files directly or indirectly include this file:

Functions

- void bn_final_exp_plain (Fp12 *ANS, Fp12 *A)
- void bn_final_exp_optimal (Fp12 *ANS, Fp12 *A)
- void bn_fp12_power_motherparam (Fp12 *ANS, Fp12 *A)

8.24.1 Detailed Description

Interaface for final exponentiation in BN curve

8.24.2 Function Documentation

8.24.2.1 bn_final_exp_optimal()

```
void bn_final_exp_optimal (
            Fp12 * ANS,
            Fp12 * A )
```

Optimized final exp for BN curve

Parameters

out	*ANS*	- output in Fp12.
in	*A*	- input P in Fp12 as obtained from Miller's algo output.

References bn_fp12_power_motherparam(), and X_binary.

```
59                                          {
60      Fp12 t0,t1,t2,t3;
61      Fp12_init(&t0);
62      Fp12_init(&t1);
63      Fp12_init(&t2);
64      Fp12_init(&t3);
65
66
67      //f←f^(p^6)*f^-1
68      Fp12_frobenius_map_p6(&t0,A);//f^(p^6)
69      Fp12_inv(&t1,A);//f^-1
70      Fp12_mul(A,&t0,&t1);//f^(p^6)*f^-1
71
72      //f←f^(p^2)*f
73      Fp12_frobenius_map_p2(&t0,A);//f^(p^2)
74      Fp12_mul(A,&t0,A);//f^(p^2)*f
75
76      //f←f^((p^4-p^2+1)/r)
77
78      bn_fp12_power_motherparam(&t0,A);         //t0←f^(u)
79      Fp12_frobenius_map_p6(&t0,&t0);             //t0←f^(-u)
80
81      Fp12_sqr(&t1,&t0);                  //t1←t0^2
82      Fp12_sqr(&t0,&t1);                  //t0←t1^2
83      Fp12_mul(&t0,&t1,&t0);                  //t0←t1*t0
84      bn_fp12_power_motherparam(&t2,&t0);     //t2←t0^(u)
85      Fp12_frobenius_map_p6(&t2,&t2);             //t2←t0^(-u)
86
87      X_binary[0]=0;
88      bn_fp12_power_motherparam(&t3,&t2);     //t3←t2^(u+1)
89      X_binary[0]=-1;
```

```
90      Fp12_sqr(&t3,&t3);                  //t3←t3^2
91      Fp12_frobenius_map_p6(&t0,&t0);           //t0←t0^(-1)
92      Fp12_mul(&t3,&t3,&t0);                 //t3←t3*t0
93      Fp12_frobenius_map_p6(&t2,&t2);           //t2←t2^(-1)
94      Fp12_mul(&t2,&t3,&t2);                 //t2←t3*t2
95      Fp12_mul(&t3,&t3,A);                 //t3←t3*f
96
97      Fp12_mul(&t1,&t1,&t2);                 //t1←t1*t2
98      Fp12_frobenius_map_p6(&t0,A);            //t0←f^(-1)
99      Fp12_mul(&t0,&t1,&t0);                 //t0←t1*t0
100
101      Fp12_frobenius_map_p3(&t0,&t0);           //t0←t0^(p^3)
102      Fp12_mul(&t0,&t0,&t3);                //t0←t0*t3
103      Fp12_frobenius_map_p1(&t1,&t1);           //t1←t1^p
104      Fp12_mul(&t0,&t0,&t1);                //t0←t0*t1
105      Fp12_frobenius_map_p2(&t2,&t2);           //t2←t2^(p^2)
106      Fp12_mul(&t0,&t0,&t2);                //t0←t0*t2
107
108      Fp12_set(ANS,&t0);
109
110      Fp12_clear(&t0);
111      Fp12_clear(&t1);
112      Fp12_clear(&t2);
113      Fp12_clear(&t3);
114  }
```

Here is the call graph for this function:

8.24.2.2 bn_final_exp_plain()

```
void bn_final_exp_plain (
              Fp12 * ANS,
              Fp12 * A )
```

Un-optimized final exp for BN curve

Parameters

in	*P*	- input P in EFp12.

References curve_parameters, curve_params::order, and curve_params::prime.

```
31                                          {
32      Fp12 t0,t1;
33      Fp12_init(&t0);
34      Fp12_init(&t1);
35      mpz_t exp,buf;
36      mpz_init(exp);
37      mpz_init(buf);
38
39      Fp12_frobenius_map_p6(&t0,A);
40      Fp12_inv(&t1,A);
41      Fp12_mul(A,&t0,&t1);
42
43      Fp12_frobenius_map_p2(&t0,A);
44      Fp12_mul(A,&t0,A);
45
46      mpz_pow_ui(exp,curve_parameters.prime,4);
47      mpz_pow_ui(buf,curve_parameters.prime,2);
48      mpz_sub(exp,exp,buf);
49      mpz_add_ui(exp,exp,1);
50      mpz_tdiv_q(exp,exp,curve_parameters.order);
51      Fp12_pow(ANS,A,exp);
52
53      mpz_clear(exp);
54      mpz_clear(buf);
55      Fp12_clear(&t0);
56      Fp12_clear(&t1);
57  }
```

8.24.2.3 bn_fp12_power_motherparam()

```
void bn_fp12_power_motherparam (
             Fp12 * ANS,
             Fp12 * A )
```

Efficient exponentiation by mother parameter

Parameters

out	*ANS*	- output in Fp12.
in	*A*	- input P in EFp12.

References bn_X_length, and X_binary.

Referenced by bn_final_exp_optimal().

```
116                                                 {
117     int i;
118     Fp12 tmp,A_inv;
119     Fp12_init(&tmp);
120     Fp12_init(&A_inv);
121     Fp12_frobenius_map_p6(&A_inv,A);
122
123     Fp12_set(&tmp,A);
124     for(i=bn_X_length-1; i>=0; i--){
125         switch(X_binary[i]){
126             case 0:
127                 Fp12_sqr(&tmp,&tmp);
128                 break;
129             case 1:
130                 Fp12_sqr(&tmp,&tmp);
131                 Fp12_mul(&tmp,&tmp,A);
132                 break;
133             case -1:
134                 Fp12_sqr(&tmp,&tmp);
135                 Fp12_mul(&tmp,&tmp,&A_inv);
136                 break;
137             default:
138                 break;
139         }
140     }
141     Fp12_set(ANS,&tmp);
142
143     Fp12_clear(&tmp);
144     Fp12_clear(&A_inv);
145 }
```

Here is the caller graph for this function:

8.25 include/ELiPS_bn_bls/bn_fp.h File Reference

```
#include <ELiPS_bn_bls/curve_settings.h>
#include <ELiPS_bn_bls/field_dtype.h>
```

Include dependency graph for bn_fp.h: This graph shows which files directly or indirectly include this file:

Functions

- void Fp_init (Fp *A)
- void Fp_clear (Fp *A)
- void Fp_printf (Fp *A, char *str)
- void Fp_set (Fp *ANS, Fp *A)
- void Fp_set_ui (Fp *ANS, unsigned long int A)
- void Fp_set_mpz (Fp *ANS, mpz_t B)
- void Fp_set_neg (Fp *ANS, Fp *A)
- void Fp_set_random (Fp *ANS, gmp_randstate_t state)
- void Fp_mul (Fp *ANS, Fp *A, Fp *B)
- void Fp_mul_ui (Fp *ANS, Fp *A, unsigned long int B)
- void Fp_mul_mpz (Fp *ANS, Fp *A, mpz_t B)
- void Fp_mul_basis (Fp *ANS, Fp *A)
- void Fp_mul_basis_KSS16 (Fp *ANS, Fp *A)
- void Fp_add (Fp *ANS, Fp *A, Fp *B)
- void Fp_add_ui (Fp *ANS, Fp *A, unsigned long int B)
- void Fp_add_mpz (Fp *ANS, Fp *A, mpz_t B)
- void Fp_sub (Fp *ANS, Fp *A, Fp *B)
- void Fp_sub_ui (Fp *ANS, Fp *A, unsigned long int B)
- void Fp_sub_mpz (Fp *ANS, Fp *A, mpz_t B)
- void Fp_inv (Fp *ANS, Fp *A)
- int Fp_legendre (Fp *A)
- int Fp_isCNR (Fp *A)
- void Fp_sqrt (Fp *ANS, Fp *A)
- void Fp_pow (Fp *ANS, Fp *A, mpz_t scalar)
- int Fp_cmp (Fp *A, Fp *B)
- int Fp_cmp_ui (Fp *A, unsigned long int UI)
- int Fp_cmp_mpz (Fp *A, mpz_t B)
- int Fp_cmp_zero (Fp *A)
- int Fp_cmp_one (Fp *A)
- void Fp_neg (struct Fp *ANS, struct Fp *A)

8.25.1 Detailed Description

Interaface of prime field operations. Primarily targeted for BN and BLS12 curve

8.25.2 Function Documentation

8.25.2.1 Fp_add()

```
void Fp_add (
            Fp * ANS,
            Fp * A,
            Fp * B )
```

Prime field addition with reduction as ANS= A+B mod prime

Parameters

out	*ANS*	- output ANS < prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send pointer B in Fp.

References curve_parameters, and curve_params::prime.

```
69                                  {
70     mpz_add(ANS->x0,A->x0,B->x0);
71     mpz_mod(ANS->x0,ANS->x0,curve_parameters.prime);
72 }
```

8.25.2.2 Fp_add_mpz()

```
void Fp_add_mpz (
            Fp * ANS,
            Fp * A,
            mpz_t B )
```

Prime field addition with reduction as ANS= A+B mod prime

Parameters

out	*ANS*	- output ANS < prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send mpz_t int B.

References curve_parameters, and curve_params::prime.

Referenced by EFp_rational_point_bls12().

```
79                                      {
80     mpz_add(ANS->x0,A->x0,B);
81     mpz_mod(ANS->x0,ANS->x0,curve_parameters.prime);
82 }
```

Here is the caller graph for this function:

8.25.2.3 Fp_add_ui()

```
void Fp_add_ui (
            Fp * ANS,
            Fp * A,
            unsigned long int B )
```

Prime field addition with reduction as ANS= A+B mod prime

Parameters

out	*ANS*	- output ANS $<$ prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send unsigned int B.

References curve_parameters, and curve_params::prime.

```
74                                                  {
75     mpz_add_ui(ANS->x0,A->x0,UI);
76     mpz_mod(ANS->x0,ANS->x0,curve_parameters.prime);
77 }
```

8.25.2.4 Fp_clear()

```
void Fp_clear (
            Fp * A )
```

Clears memory an Fp type struct

Parameters

in	*A*	- input A is a pointer of Fp type struct.

Referenced by bls12_f_ltp_vtp_for_tate(), bls12_ff_ltt_vtt_for_tate(), bls12_Miller_algo_for_opt_ate(), bls12_↩ Miller_algo_for_plain_ate(), bls12_Pseudo_8_sparse_mapping(), clear_parameters(), EFp_clear(), EFp_ECA(), EFp_ECD(), EFp_rational_point_bls12(), EFp_rational_point_bn(), Fp_isCNR(), Fp_pow(), and Fp_sqrt().

```
17                     {
18     mpz_clear(A->x0);
19 }
```

Here is the caller graph for this function:

8.25.2.5 Fp_cmp()

```
int Fp_cmp (
            Fp * A,
            Fp * B )
```

Compares A and B in Fp

Parameters

out	*int*	- return 0 if A==B and 1 otherwise.
in	*A*	- send pointer A in Fp.
in	*B*	- send pointer B in Fp.

Referenced by EFp_ECA().

```
224                         {
225     if(mpz_cmp(A->x0,B->x0)==0){
226         return 0;
227     }
228     return 1;
229 }
```

Here is the caller graph for this function:

8.25.2.6 Fp_cmp_mpz()

```
int Fp_cmp_mpz (
            Fp * A,
            mpz_t B )
```

Compares A and B in Fp

Parameters

out	*int*	- return 0 if A==B and 1 otherwise.
in	*A*	- send pointer A in Fp.
in	*B*	- send pointer B in mpz_t.

```
238                             {
239     if(mpz_cmp(A->x0,B)==0){
240         return 0;
241     }
242     return 1;
243 }
```

8.25.2.7 Fp_cmp_one()

```
int Fp_cmp_one (
            Fp * A )
```

Compares A == 1 in Fp

Parameters

out	*int*	- return 0 if A==1 and 1 otherwise.
in	*A*	- send pointer A in Fp.

Referenced by Fp_isCNR().

```
252                         {
253     if(mpz_cmp_ui(A->x0,1)==0){
254         return 0;
255     }
256     return 1;
257 }
```

Here is the caller graph for this function:

8.25.2.8 Fp_cmp_ui()

```
int Fp_cmp_ui (
            Fp * A,
            unsigned long int UI )
```

Compares A and B in Fp

Parameters

out	*int*	- return 0 if A==B and 1 otherwise.
in	*A*	- send pointer A in Fp.
in	*B*	- send pointer B in int.

```
231                                             {
232     if(mpz_cmp_ui(A->x0,UI)==0){
233         return 0;
234     }
235     return 1;
236 }
```

8.25.2.9 Fp_cmp_zero()

```
int Fp_cmp_zero (
            Fp * A )
```

Compares A == 0 in Fp

Parameters

out	*int*	- return 0 if A==0 and 1 otherwise.
in	*A*	- send pointer A in Fp.
in	*B*	- send pointer B in int.

Referenced by EFp_ECD().

```
245                         {
246     if(mpz_cmp_ui(A->x0,0)==0){
247         return 0;
248     }
249     return 1;
250 }
```

Here is the caller graph for this function:

8.25.2.10 Fp_init()

```
void Fp_init (
              Fp * A )
```

Initializes an Fp type struct

Parameters

in	*A*	- input A is a pointer of Fp type struct.

Referenced by bls12_f_ltp_vtp_for_tate(), bls12_ff_ltt_vtt_for_tate(), bls12_Fp12_frobenius_map_p1(), bls12_Fp12_frobenius_map_p3(), bls12_Miller_algo_for_opt_ate(), bls12_Miller_algo_for_plain_ate(), bls12_Pseudo_8_sparse_mapping(), EFp_ECA(), EFp_ECD(), EFp_init(), EFp_rational_point_bls12(), EFp_rational_point_bn(), Fp_isCNR(), Fp_pow(), Fp_sqrt(), get_epsilon(), and init_precoms().

```
13                  {
14     mpz_init(A->x0);
15 }
```

Here is the caller graph for this function:

8.25.2.11 Fp_inv()

```
void Fp_inv (
              Fp * ANS,
              Fp * A )
```

Prime field inversion A^-1 mod prime

Parameters

out	*ANS*	- output A^-1 mod prime in Fp.
in	*A*	- send pointer A in Fp.

References curve_parameters, and curve_params::prime.

Referenced by bls12_Pseudo_8_sparse_mapping(), EFp_ECA(), EFp_ECD(), and get_epsilon().

```
99                          {
100     mpz_invert(ANS->x0,A->x0,curve_parameters.prime);
101 }
```

Here is the caller graph for this function:

8.25.2.12 Fp_isCNR()

```
int Fp_isCNR (
              Fp * A )
```

Check if A has qubic root or not in prime field.

Parameters

out	*int*	- return 1 if A has CR and -1 CNR.
in	*A*	- send pointer A in Fp.

References curve_parameters, Fp_clear(), Fp_cmp_one(), Fp_init(), Fp_pow(), and curve_params::prime.

```
112                     {
113     Fp tmp;
114     Fp_init(&tmp);
115     mpz_t exp;
116     mpz_init(exp);
117 
118     mpz_sub_ui(exp,curve_parameters.prime,1);
119     mpz_tdiv_q_ui(exp,exp,3);
120     Fp_pow(&tmp,A,exp);
121 
122     if(Fp_cmp_one(&tmp)==0){
123         mpz_clear(exp);
124         Fp_clear(&tmp);
125         return 1;
126     }else{
127         mpz_clear(exp);
128         Fp_clear(&tmp);
129         return -1;
130     }
131 }
```

Here is the call graph for this function:

8.25.2.13 Fp_legendre()

```
int Fp_legendre (
              Fp * A )
```

Calculate legendre symbol (A/prime) to determine A has any sqare root in prime field.

Parameters

out	*int*	- return 1 if A has QR and -1 for QNR.
in	*A*	- send pointer A in Fp.

References curve_parameters, and curve_params::prime.

Referenced by EFp_rational_point_bls12(), EFp_rational_point_bn(), and Fp_sqrt().

```
108                       {
109     return mpz_legendre(A->x0,curve_parameters.prime);
110 }
```

Here is the caller graph for this function:

8.25.2.14 Fp_mul()

```
void Fp_mul (
              Fp * ANS,
              Fp * A,
              Fp * B )
```

Prime field multiplication with reduction as ANS= A∗B mod prime

Parameters

out	*ANS*	- output ANS < prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send pointer B in Fp.

References curve_parameters, and curve_params::prime.

Referenced by bls12_Pseudo_8_sparse_mapping(), EFp_ECA(), EFp_ECD(), EFp_rational_point_bls12(), EFp_↩ rational_point_bn(), Fp_pow(), Fp_sqrt(), and get_epsilon().

```
45                                  {
46      mpz_mul(ANS->x0,A->x0,B->x0);
47      mpz_mod(ANS->x0,ANS->x0,curve_parameters.prime);
48  }
```

Here is the caller graph for this function:

8.25.2.15 Fp_mul_basis()

```
void Fp_mul_basis (
              Fp * ANS,
              Fp * A )
```

Prime field multiplication with reduction as ANS= A∗B mod prime

Parameters

out	*ANS*	- output ANS < prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send basis element ^2 = some int

References curve_parameters, and curve_params::prime.

```
60                                  {
61      mpz_sub(ANS->x0,curve_parameters.prime,A->x0);
62  }
```

8.25.2.16 Fp_mul_basis_KSS16()

```
void Fp_mul_basis_KSS16 (
              Fp * ANS,
              Fp * A )
```

Prime field multiplication with reduction as ANS= A∗B mod prime

Parameters

out	*ANS*	- output ANS < prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send basis element ^2 = c is a int

References curve_params::curve_a, and curve_parameters.

```
64                                      {
65     mpz_mul_ui(ANS->x0,A->x0,c1);
66     mpz_mod(ANS->x0,ANS->x0,curve_parameters.curve_a);
67 }
```

8.25.2.17 Fp_mul_mpz()

```
void Fp_mul_mpz (
            Fp * ANS,
            Fp * A,
            mpz_t B )
```

Prime field multiplication with reduction as ANS= A∗B mod prime

Parameters

out	*ANS*	- output ANS < prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send mpz_t .

References curve_parameters, and curve_params::prime.

Referenced by bls12_EFp_skew_frobenius_map_p2().

```
55                                      {
56     mpz_mul(ANS->x0,A->x0,B);
57     mpz_mod(ANS->x0,ANS->x0,curve_parameters.prime);
58 }
```

Here is the caller graph for this function:

8.25.2.18 Fp_mul_ui()

```
void Fp_mul_ui (
            Fp * ANS,
            Fp * A,
            unsigned long int B )
```

Prime field multiplication with reduction as ANS= A∗B mod prime

Parameters

out	*ANS*	- output ANS < prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send unsigned int.

References curve_parameters, and curve_params::prime.

Referenced by EFp_ECD().

```
50                                                  {
51     mpz_mul_ui(ANS->x0,A->x0,UI);
52     mpz_mod(ANS->x0,ANS->x0,curve_parameters.prime);
53 }
```

Here is the caller graph for this function:

8.25.2.19 Fp_neg()

```
void Fp_neg (
             struct Fp * ANS,
             struct Fp * A )
```

Calculate -A in Fp

Parameters

out	*ANS*	- return -A.
in	*A*	- send pointer A in Fp.

References curve_parameters, and curve_params::prime.

```
259                                         {
260     mpz_sub(ANS->x0,curve_parameters.prime,A->x0);
261 }
```

8.25.2.20 Fp_pow()

```
void Fp_pow (
             Fp * ANS,
             Fp * A,
             mpz_t scalar )
```

Calculate A^mpz_t mod prime in prime field.

Parameters

out	*ANS*	- return 1 if A has CR and -1 CNR.
in	*A*	- send pointer A in Fp.

References Fp_clear(), Fp_init(), Fp_mul(), and Fp_set().

Referenced by Fp_isCNR(), and Fp_sqrt().

```
203                                         {
204     int i,length;
205     length=(int)mpz_sizeinbase(scalar,2);
206     char binary[length];
207     mpz_get_str(binary,2,scalar);
208     Fp tmp;
209     Fp_init(&tmp);
210 
211     Fp_set(&tmp,A);
212 
213     for(i=1; i<length; i++){
214         Fp_mul(&tmp,&tmp,&tmp);
215         if(binary[i]=='1'){
216             Fp_mul(&tmp,A,&tmp);
217         }
218     }
219     Fp_set(ANS,&tmp);
220 
221     Fp_clear(&tmp);
222 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.25.2.21 Fp_printf()

```
void Fp_printf (
            Fp * A,
            char * str )
```

Prints an Fp type struct

Parameters

out	*A*	- input A is a pointer of Fp type struct.
in	*str*	- any string pointer.

Referenced by EFp_printf().

```
21                                  {
22     gmp_printf("%s%Zd",str,A->x0);
23 }
```

Here is the caller graph for this function:

8.25.2.22 Fp_set()

```
void Fp_set (
            Fp * ANS,
            Fp * A )
```

Set an Fp type struct to ANS

Parameters

out	*ANS*	- set A in ANS.
in	*A*	- send A as pointer.

Referenced by bls12_EFp12_to_EFp(), bls12_EFp2_skew_frobenius_map_p1(), bls12_EFp2_skew_frobenius_↩ map_p3(), bls12_EFp_to_EFp12(), bls12_Fp12_frobenius_map_p1(), bls12_Fp12_frobenius_map_p3(), bls12_↩ Pseudo_8_sparse_mapping(), EFp_set(), EFp_set_neg(), Fp_pow(), and Fp_sqrt().

```
25                          {
26     mpz_set(ANS->x0,A->x0);
27 }
```

Here is the caller graph for this function:

8.25.2.23 Fp_set_mpz()

```
void Fp_set_mpz (
            Fp * ANS,
            mpz_t B )
```

Set an gmp int mpz in ANS Fp type struct.

Parameters

out	*ANS*	- set A in ANS.
in	*A*	- send A as mpz_t.

Referenced by EFp_set_mpz().

```
33                                  {
34     mpz_set(ANS->x0,A);
35 }
```

Here is the caller graph for this function:

8.25.2.24 Fp_set_neg()

```
void Fp_set_neg (
            Fp * ANS,
            Fp * A )
```

Negate A and set it in ANS

Parameters

out	*ANS*	- set prime-A in ANS.
in	*A*	- send A as Fp.

Referenced by bls12_EFp2_skew_frobenius_map_p1(), bls12_EFp2_skew_frobenius_map_p3(), bls12_EFp_↩ skew_frobenius_map_p2(), bls12_Fp12_frobenius_map_p1(), bls12_Fp12_frobenius_map_p3(), and EFp_set_↩ neg().

```
37                                  {
38     mpz_neg(ANS->x0,A->x0);
39 }
```

Here is the caller graph for this function:

8.25.2.25 Fp_set_random()

```
void Fp_set_random (
            Fp * ANS,
            gmp_randstate_t state )
```

Set random Fp of size <prime in ANS

Parameters

out	*ANS*	- output an Fp < prime.
in	*A*	- send a gmp_randstate_t.

References curve_parameters, and curve_params::prime.

Referenced by EFp_rational_point_bls12(), EFp_rational_point_bn(), and Fp_sqrt().

```
41                                                  {
42     mpz_urandomm(ANS->x0,state,curve_parameters.prime);
43 }
```

Here is the caller graph for this function:

8.25.2.26 Fp_set_ui()

```
void Fp_set_ui (
            Fp * ANS,
            unsigned long int A )
```

Set an unsigned int in ANS Fp type struct.

Parameters

out	*ANS*	- set A in ANS.
in	*A*	- send A as UI.

Referenced by bls12_2split_G3_exp(), bls12_4split_G3_exp(), bls12_f_ltq(), bls12_ff_ltt(), bls12_Miller_algo_for_↩ opt_ate(), bls12_Miller_algo_for_plain_ate(), bls12_Miller_algo_for_tate(), EFp_set_ui(), get_epsilon(), set_basis(), and set_frobenius_constant().

```
29                                          {
30     mpz_set_ui(ANS->x0,UI);
31 }
```

Here is the caller graph for this function:

8.25.2.27 Fp_sqrt()

```
void Fp_sqrt (
            Fp * ANS,
            Fp * A )
```

Calculate sqrt(A) mod prime using Toneli-Shanks alogrithm in prime field.

Parameters

out	*ANS*	- return 1 if A has CR and -1 CNR.
in	*A*	- send pointer A in Fp.

References curve_parameters, Fp_clear(), Fp_init(), Fp_legendre(), Fp_mul(), Fp_pow(), Fp_set(), Fp_set_↩ random(), and curve_params::prime.

Referenced by EFp_rational_point_bls12(), EFp_rational_point_bn(), and get_epsilon().

```
133                             {
134     Fp x,y,t,k,n,tmp;
135     Fp_init(&x);
136     Fp_init(&y);
137     Fp_init(&t);
138     Fp_init(&k);
139     Fp_init(&n);
140     Fp_init(&tmp);
141     unsigned long int e,m;
142     mpz_t exp,q,z,result;
143     mpz_init(exp);
144     mpz_init(q);
145     mpz_init(z);
146     mpz_init(result);
147     gmp_randstate_t state;
148     gmp_randinit_default (state);
149     gmp_randseed_ui(state,(unsigned long)time(NULL));
150     Fp_set_random(&n,state);
151 
152     while(Fp_legendre(&n)!=-1){
153         Fp_set_random(&n,state);
154     }
155     mpz_sub_ui(q,curve_parameters.prime,1);
156     mpz_mod_ui(result,q,2);
157     e=0;
158     while(mpz_cmp_ui(result,0)==0){
159         mpz_tdiv_q_ui(q,q,2);
160         mpz_mod_ui(result,q,2);
161         e++;
162     }
163     Fp_pow(&y,&n,q);
164     mpz_set_ui(z,e);
165     mpz_sub_ui(exp,q,1);
166     mpz_tdiv_q_ui(exp,exp,2);
167     Fp_pow(&x,A,exp);
168     Fp_mul(&tmp,&x,&x);
169     Fp_mul(&k,&tmp,A);
170     Fp_mul(&x,&x,A);
171     while(mpz_cmp_ui(k.x0,1)!=0){
172         m=1;
173         mpz_ui_pow_ui(exp,2,m);
174         Fp_pow(&tmp,&k,exp);
175         while(mpz_cmp_ui(tmp.x0,1)!=0){
176             m++;
177             mpz_ui_pow_ui(exp,2,m);
```

```
178             Fp_pow(&tmp,&k,exp);
179         }
180         mpz_sub_ui(exp,z,m);
181         mpz_sub_ui(exp,exp,1);
182         mpz_ui_pow_ui(result,2,mpz_get_ui(exp));
183         Fp_pow(&t,&y,result);
184         Fp_mul(&y,&t,&t);
185         mpz_set_ui(z,m);
186         Fp_mul(&x,&x,&t);
187         Fp_mul(&k,&k,&y);
188     }
189     Fp_set(ANS,&x);
190 
191     mpz_clear(exp);
192     mpz_clear(q);
193     mpz_clear(z);
194     mpz_clear(result);
195     Fp_clear(&x);
196     Fp_clear(&y);
197     Fp_clear(&t);
198     Fp_clear(&k);
199     Fp_clear(&n);
200     Fp_clear(&tmp);
201 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.25.2.28 Fp_sub()

```
void Fp_sub (
              Fp * ANS,
              Fp * A,
              Fp * B )
```

Prime field subtraction with reduction as ANS= A-B mod prime

Parameters

out	*ANS*	- output ANS $<$ prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send pointer B.

References curve_parameters, and curve_params::prime.

Referenced by bls12_f_ltp_vtp_for_tate(), bls12_ff_ltt_vtt_for_tate(), EFp_ECA(), and EFp_ECD().

```
84                                  {
85     mpz_sub(ANS->x0,A->x0,B->x0);
86     mpz_mod(ANS->x0,ANS->x0,curve_parameters.prime);
87 }
```

Here is the caller graph for this function:

8.25.2.29 Fp_sub_mpz()

```
void Fp_sub_mpz (
              Fp * ANS,
              Fp * A,
              mpz_t B )
```

Prime field subtraction with reduction as ANS = A-B mod prime

Parameters

out	*ANS*	- output ANS < prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send mpz_t int B.

References curve_parameters, and curve_params::prime.

Referenced by EFp_rational_point_bn().

```
94                                          {
95     mpz_sub(ANS->x0,A->x0,B);
96     mpz_mod(ANS->x0,ANS->x0,curve_parameters.prime);
97 }
```

Here is the caller graph for this function:

8.25.2.30 Fp_sub_ui()

```
void Fp_sub_ui (
             Fp * ANS,
             Fp * A,
             unsigned long int B )
```

Prime field subtraction with reduction as ANS= A-B mod prime

Parameters

out	*ANS*	- output ANS < prime in Fp.
in	*A*	- send pointer A in Fp.
in	*B*	- send int B.

References curve_parameters, and curve_params::prime.

Referenced by get_epsilon().

```
89                                                  {
90     mpz_sub_ui(ANS->x0,A->x0,UI);
91     mpz_mod(ANS->x0,ANS->x0,curve_parameters.prime);
92 }
```

Here is the caller graph for this function:

8.26 include/ELiPS_bn_bls/bn_inits.h File Reference

```
#include <ELiPS_bn_bls/bn_bls12_precoms.h>
#include <ELiPS_bn_bls/curve_settings.h>
```

Include dependency graph for bn_inits.h: This graph shows which files directly or indirectly include this file:

Functions

- void init_bn (void)

8.26.1 Detailed Description

Interaface for pairing using BLS12 curve. It needs to included for initializing BLS12 curve in the applications.

8.26.2 Function Documentation

8.26.2.1 init_bn()

```
void init_bn (
             void  )
```

This methods needs to be called before using the BN curve for pairing. It initialized the curve as y^2=x^3-4 with parameters suggested in https://eprint.iacr.org/2017/334.

References init_bn_settings(), and init_precoms().

```
31                  {
32     init_bn_settings();
33     init_precoms(1);
34 }
```

Here is the call graph for this function:

8.27 include/ELiPS_bn_bls/Commont_headers.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <gmp.h>
#include <time.h>
#include <sys/time.h>
#include <unistd.h>
#include <string.h>
```

Include dependency graph for Commont_headers.h: This graph shows which files directly or indirectly include this file:

8.27.1 Detailed Description

Interface for Common header files.

8.28 include/ELiPS_bn_bls/curve_dtypes.h File Reference

```
#include <ELiPS_bn_bls/bn_fp12.h>
#include <ELiPS_bn_bls/field_dtype.h>
```

Include dependency graph for curve_dtypes.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct EFp
- struct EFp2
- struct EFp6
- struct EFp12
- struct EFp4
- struct EFp8
- struct EFp16

Typedefs

- typedef struct EFp EFp
- typedef struct EFp2 EFp2
- typedef struct EFp6 EFp6
- typedef struct EFp12 EFp12

8.28.1 Detailed Description

Interface for pairing friendly curve's data types.

8.28.2 Typedef Documentation

8.28.2.1 EFp

```
typedef struct EFp EFp
```

EFp is the basic type that represent rational point in prime field element. Consist of affine coordinate x, y and an extra flag to infinity to point if the rational point is additive unity in EFp.

8.28.2.2 EFp12

```
typedef struct EFp12 EFp12
```

EFp12 is the basic type that represent rational point curve EFp12. Consist of affine coordinate x, y in Fp12 . Flag to infinity to point if the rational point is additive unity in EFp12.

8.28.2.3 EFp2

```
typedef struct EFp2 EFp2
```

EFp2 is the basic type that represent rational point curve EFp2. Consist of affine coordinate x, y in Fp2 . Flag to infinity to point if the rational point is additive unity in EFp2.

8.28.2.4 EFp6

```
typedef struct EFp6 EFp6
```

EFp6 is the basic type that represent rational point curve EFp6. Consist of affine coordinate x, y in Fp6 . Flag to infinity to point if the rational point is additive unity in EFp6.

8.29 include/ELiPS_bn_bls/curve_settings.h File Reference

```
#include <ELiPS_bn_bls/Commont_headers.h>
```

Include dependency graph for curve_settings.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct curve_params

  Curve Parameters.

Macros

- #define bn_X_length 114

Functions

- void init_bn_settings (void)
- void init_bn_parameters (void)
- void generate_bn_mother_parameter (void)
- int generate_bn_prime (void)
- int generate_bn_order (void)
- void generate_bn_trace (void)
- void set_bn_curve_parameter (void)
- void weil (void)
- void print_curve_parameters (void)
- void init_bls12_settings (void)
- void init_bls12_parameters (void)
- void bls12_generate_X (void)
- int bls12_generate_prime (void)
- int bls12_generate_order (void)
- void bls12_generate_trace (void)
- void bls12_set_curve_parameter (void)
- void bls12_weil (void)
- void bls12_print_parameters (void)

Variables

- struct curve_params curve_parameters

  bn curve's systematically obtained parameters.
- gmp_randstate_t state
- char X_binary [bn_X_length+1]
- char X_binary_opt [bn_X_length+3]
- int bls12_X_length
- int bls12_X_binary [78]

8.29.1 Detailed Description

Interface for curve parameters.

8.29.2 Macro Definition Documentation

8.29.2.1 bn_X_length

```
#define bn_X_length 114
```

BN curve's mother parameter length in bit.

Referenced by bn_fp12_power_motherparam(), generate_bn_mother_parameter(), and init_bls12_parameters().

8.29.3 Function Documentation

8.29.3.1 bls12_generate_order()

```
int bls12_generate_order (
            void  )
```

Initialize BLS12 curve order.

Referenced by init_bls12_settings().

```
410                          {
411     mpz_t buf1,buf2;
412     mpz_init(buf1);
413     mpz_init(buf2);
414
415     mpz_pow_ui(buf1,bls12_X,4);
416     mpz_pow_ui(buf2,bls12_X,2);
417     mpz_sub(curve_parameters.order,buf1,buf2);
418     mpz_add_ui(curve_parameters.order,curve_parameters.
    order,1);
419
420     mpz_clear(buf1);
421     mpz_clear(buf2);
422
423     return 0;
424 }
```

Here is the caller graph for this function:

8.29.3.2 bls12_generate_prime()

```
int bls12_generate_prime (
            void  )
```

Initialize BLS12 curve prime.

Referenced by init_bls12_settings().

```
362                             {
363     mpz_t result,buf1,buf2,modtest;
364     mpz_init(result);
365     mpz_init(buf1);
366     mpz_init(buf2);
367     mpz_init(modtest);
368 
369     mpz_sub_ui(result,bls12_X,1);
370     mpz_pow_ui(result,result,2);
371 
372     mpz_pow_ui(buf1,bls12_X,4);
373     mpz_pow_ui(buf2,bls12_X,2);
374     mpz_sub(buf1,buf1,buf2);
375     mpz_add_ui(buf1,buf1,1);
376 
377     mpz_mul(result,result,buf1);
378 
379     //check div3
380     mpz_mod_ui(modtest,result,3);
381     if(mpz_cmp_ui(modtest,0)!=0){
382         mpz_init(result);
383         mpz_init(buf1);
384         mpz_init(buf2);
385         mpz_init(modtest);
386         return 0;
387     }
388 
389     mpz_tdiv_q_ui(result,result,3);
390     mpz_add(result,result,bls12_X);
391 
392     //isprime
393     if(mpz_probab_prime_p(result,25)==0){
394         mpz_init(result);
395         mpz_init(buf1);
396         mpz_init(buf2);
397         mpz_init(modtest);
398         return 0;
399     }
400 
401     mpz_set(curve_parameters.prime,result);
402 
403     mpz_init(result);
404     mpz_init(buf1);
405     mpz_init(buf2);
406     mpz_init(modtest);
407     return 1;
408 }
```

Here is the caller graph for this function:

8.29.3.3 bls12_generate_trace()

```
void bls12_generate_trace (
            void  )
```

Initialize BLS12 curve Frobenius trace.

References curve_parameters, and curve_params::trace_t.

Referenced by init_bls12_settings().

```
426                             {
427     mpz_add_ui(curve_parameters.trace_t,bls12_X,1);
428 }
```

Here is the caller graph for this function:

8.29.3.4 bls12_generate_X()

```
void bls12_generate_X (
            void  )
```

Initialize BLS12 curve mother parameter.

References bls12_X_binary.

Referenced by init_bls12_settings().

```
333                         {
334     int i;
335     mpz_t buf,set_2;
336     mpz_init(buf);
337     mpz_init(set_2);
338     mpz_set_ui(set_2,2);
339 
340     //bls12_X_binary
341     bls12_X_binary[77]=-1;
342     bls12_X_binary[50]=1;
343     bls12_X_binary[33]=1;
344 
345     //bls12_X
346     mpz_init(bls12_X);
347     mpz_set_ui(bls12_X,0);
348     for(i=bls12_X_length; i>=0; i--){
349         if(bls12_X_binary[i]==1){
350             mpz_pow_ui(buf,set_2,i);
351             mpz_add(bls12_X,bls12_X,buf);
352         }else if(bls12_X_binary[i]==-1){
353             mpz_pow_ui(buf,set_2,i);
354             mpz_sub(bls12_X,bls12_X,buf);
355         }
356     }
357 
358     mpz_clear(buf);
359     mpz_clear(set_2);
360 }
```

Here is the caller graph for this function:

8.29.3.5 bls12_print_parameters()

```
void bls12_print_parameters (
            void  )
```

Prints BLS12 curve parameters.

Prints curve parameter: BLS12 curve

```
478                             {
479     printf("================================================================================================\n");
480     printf("bls12\n\n");
481     gmp_printf("parameters\n");
482     gmp_printf("X     (%dbit length) : %Zd \n",(int)mpz_sizeinbase(bls12_X,2),bls12_X);
483     gmp_printf("prime (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.prime,2),curve_parameters.
      prime);
484     gmp_printf("order (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.order,2),curve_parameters.
      order);
485     gmp_printf("trace (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.trace_t,2),curve_parameters.
      trace_t);
486 
487     gmp_printf("\nelliptic curve\n");
488     gmp_printf("E:y^2=x^3+4\n",curve_parameters.curve_b);
489 
490     gmp_printf("\nmodulo polynomial\n");
491     gmp_printf("Fp2  : f(x) = x^2+1\n");
492     gmp_printf("Fp6  : f(x) = x^3-(alpha+1)\n");
493     gmp_printf("Fp12 : f(x) = x^2-beta\n");
494 
495 }
```

8.29.3.6 bls12_set_curve_parameter()

```
void bls12_set_curve_parameter (
            void  )
```

Sets BLS12 curve coefficients b.

References curve_params::curve_b, and curve_parameters.

Referenced by init_bls12_settings().

```
430                                 {
431     mpz_set_ui(curve_parameters.curve_b,4);
432 }
```

Here is the caller graph for this function:

8.29.3.7 bls12_weil()

```
void bls12_weil (
            void  )
```

Calculate BLS12 curve field size EFp.

References curve_parameters, curve_params::EFp_total, curve_params::EFpd_total, curve_params::prime, and curve_params::trace_t.

Referenced by init_bls12_settings().

```
434                 {
435     mpz_t t2,t6,t12,p2,p6,buf;
436     mpz_init(t2);
437     mpz_init(t6);
438     mpz_init(t12);
439     mpz_init(p2);
440     mpz_init(p6);
441     mpz_init(buf);
442
443     //EFp_total
444     mpz_add_ui(buf,curve_parameters.prime,1);
445     mpz_sub(curve_parameters.EFp_total,buf,
      curve_parameters.trace_t);
446
447     //t2←^2+^2
448     mpz_pow_ui(t2,curve_parameters.trace_t,2);
449     mpz_mul_ui(buf,curve_parameters.prime,2);
450     mpz_sub(t2,t2,buf);
451     mpz_pow_ui(p2,curve_parameters.prime,2);
452
453     //^6+^6
454     mpz_pow_ui(t6,t2,3);
455     mpz_mul(buf,t2,p2);
456     mpz_mul_ui(buf,buf,3);
457     mpz_sub(t6,t6,buf);
458     mpz_pow_ui(p6,p2,3);
459
460     //^12+^12
461     mpz_pow_ui(t12,t6,2);
462     mpz_mul_ui(buf,p6,2);
463     mpz_sub(t12,t12,buf);
464
465     //EFp12_total
466     mpz_pow_ui(buf,p6,2);
467     mpz_sub(buf,buf,t12);
468     mpz_add_ui(curve_parameters.EFpd_total,buf,1);
469
470     mpz_clear(t2);
471     mpz_clear(t6);
472     mpz_clear(t12);
473     mpz_clear(p2);
474     mpz_clear(p6);
475     mpz_clear(buf);
476 }
```

Here is the caller graph for this function:

8.29.3.8 generate_bn_mother_parameter()

```
void generate_bn_mother_parameter (
            void  )
```

Generate BN curves mother parameter X.

References bn_X_length, curve_parameters, curve_params::X, X_binary, and X_binary_opt.

Referenced by init_bn_settings().

```
133                                 {
134     int i;
135     mpz_t buf;
136     mpz_init(buf);
137 
138     //X_binary
139     X_binary[114]=1;
140     X_binary[101]=1;
141     X_binary[14]=-1;
142     X_binary[0]=-1;
143 
144     //X_binary_opt
145     X_binary_opt[116]=1;
146     X_binary_opt[115]=1;
147     X_binary_opt[103]=1;
148     X_binary_opt[102]=1;
149     X_binary_opt[16]=-1;
150     X_binary_opt[15]=-1;
151     X_binary_opt[2]=-1;
152 
153     //X
154     mpz_set_ui(curve_parameters.X,0);
155     for(i=bn_X_length; i>=0; i--){
156         if(X_binary[i]==1){
157             mpz_ui_pow_ui(buf,2,i);
158             mpz_add(curve_parameters.X,curve_parameters.
      X,buf);
159         }else if(X_binary[i]==-1){
160             mpz_ui_pow_ui(buf,2,i);
161             mpz_sub(curve_parameters.X,curve_parameters.
      X,buf);
162         }
163     }
164 
165     mpz_clear(buf);
166 }
```

Here is the caller graph for this function:

8.29.3.9 generate_bn_order()

```
int generate_bn_order (
            void  )
```

Generate BN curve order.

References curve_parameters, curve_params::order, and curve_params::X.

Referenced by init_bn_settings().

```
200                         {
201     mpz_t buf,result;
202     mpz_init(buf);
203     mpz_init(result);
204 
205     //prime
206     mpz_pow_ui(buf,curve_parameters.X,4);
207     mpz_mul_ui(buf,buf,36);
208     mpz_set(result,buf);
209     mpz_pow_ui(buf,curve_parameters.X,3);
210     mpz_mul_ui(buf,buf,36);
211     mpz_add(result,result,buf);
212     mpz_pow_ui(buf,curve_parameters.X,2);
213     mpz_mul_ui(buf,buf,18);
214     mpz_add(result,result,buf);
215     mpz_mul_ui(buf,curve_parameters.X,6);
216     mpz_add(result,result,buf);
217     mpz_add_ui(result,result,1);
218 
219     //isprime
220     if(mpz_probab_prime_p(result,25)==0){
221         mpz_clear(buf);
222         mpz_clear(result);
223         return 0;
224     }else{
225         mpz_set(curve_parameters.order,result);
226         mpz_clear(buf);
227         mpz_clear(result);
228         return 1;
229     }
230 }
```

Here is the caller graph for this function:

8.29.3.10 generate_bn_prime()

```
int generate_bn_prime (
             void  )
```

Generate BN curve prime.

References curve_parameters, curve_params::prime, and curve_params::X.

Referenced by init_bn_settings().

```
168                         {
169     mpz_t buf,result;
170     mpz_init(buf);
171     mpz_init(result);
172 
173     //prime
174     mpz_pow_ui(buf,curve_parameters.X,4);
175     mpz_mul_ui(buf,buf,36);
176     mpz_set(result,buf);
177     mpz_pow_ui(buf,curve_parameters.X,3);
178     mpz_mul_ui(buf,buf,36);
179     mpz_add(result,result,buf);
180     mpz_pow_ui(buf,curve_parameters.X,2);
181     mpz_mul_ui(buf,buf,24);
182     mpz_add(result,result,buf);
183     mpz_mul_ui(buf,curve_parameters.X,6);
184     mpz_add(result,result,buf);
185     mpz_add_ui(result,result,1);
186 
187     //isprime
188     if(mpz_probab_prime_p(result,25)==0){
189         mpz_clear(buf);
190         mpz_clear(result);
191         return 0;
192     }else{
193         mpz_set(curve_parameters.prime,result);
194         mpz_clear(buf);
195         mpz_clear(result);
196         return 1;
197     }
198 }
```

Here is the caller graph for this function:

8.29.3.11 generate_bn_trace()

```
void generate_bn_trace (
            void  )
```

Generate BN curve Frobenius trace.

References curve_parameters, curve_params::trace_t, and curve_params::X.

Referenced by init_bn_settings().

```
233                         {
234     mpz_t buf;
235     mpz_init(buf);
236 
237     mpz_pow_ui(buf,curve_parameters.X,2);
238     mpz_mul_ui(buf,buf,6);
239     mpz_add_ui(curve_parameters.trace_t,buf,1);
240 
241     mpz_clear(buf);
242 }
```

Here is the caller graph for this function:

8.29.3.12 init_bls12_parameters()

```
void init_bls12_parameters (
            void  )
```

Initialize BLS12 curve parameters.

References bls12_X_binary, bls12_X_length, bn_X_length, curve_params::curve_b, curve_parameters, curve_↩ params::EFp12_total, curve_params::EFp2_total, curve_params::EFp6_total, curve_params::EFp_total, curve_↩ params::order, curve_params::prime, curve_params::trace_t, curve_params::X, X_binary, and X_binary_opt.

Referenced by init_bls12_settings().

```
84                             {
85     //parameters
86     mpz_init(curve_parameters.prime);
87     mpz_init(curve_parameters.X);
88     mpz_init(curve_parameters.trace_t);
89     mpz_init(curve_parameters.order);
90     mpz_init(curve_parameters.EFp_total);
91     mpz_init(curve_parameters.EFp2_total);
92     mpz_init(curve_parameters.EFp6_total);
93     mpz_init(curve_parameters.EFp12_total);
94     //    mpz_init(curve_parameters.curve_a);
95     mpz_init(curve_parameters.curve_b);
96 
97     int i;
98     for(i=0; i<bn_X_length+1; i++){
99         X_binary[i]=0;
100     }
101     for(i=0; i<bn_X_length+3; i++){
102         X_binary_opt[i]=0;
103     }
104 
105     bls12_X_length=77;
106     for(i=0; i<bls12_X_length+1; i++){
107         bls12_X_binary[i]=0;
108     }
109 }
```

Here is the caller graph for this function:

8.29.3.13 init_bls12_settings()

```
void init_bls12_settings (
            void  )
```

Initialize BLS12 curve settings.

References bls12_generate_order(), bls12_generate_prime(), bls12_generate_trace(), bls12_generate_X(), bls12_set_curve_parameter(), bls12_weil(), and init_bls12_parameters().

Referenced by bls12_inits().

```
36                          {
37     init_bls12_parameters();
38
39     bls12_generate_X();
40     bls12_generate_prime();
41     bls12_generate_order();
42     bls12_generate_trace();
43
44     bls12_weil();
45     bls12_set_curve_parameter();
46 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.29.3.14 init_bn_parameters()

```
void init_bn_parameters (
            void  )
```

Initialize BN curves parameter generation.

Referenced by init_bn_settings().

```
59                          {
60     //parameters
61     mpz_init(C1_INV);
62     mpz_set_str(C1_INV,"
      3078116910153375752510331093757217190320537831741050592539701174176283547113174512665143269626197827 91",10);
63
64     mpz_init(curve_parameters.prime);
65     mpz_init(curve_parameters.X);
66     mpz_init(curve_parameters.trace_t);
67     mpz_init(curve_parameters.order);
68     mpz_init(curve_parameters.EFp_total);
69     mpz_init(curve_parameters.EFp2_total);
70     mpz_init(curve_parameters.EFp6_total);
71     mpz_init(curve_parameters.EFp12_total);
72 //     mpz_init(curve_parameters.curve_a);
73     mpz_init(curve_parameters.curve_b);
74
75     int i;
76     for(i=0; i<bn_X_length+1; i++){
77         X_binary[i]=0;
78     }
79     for(i=0; i<bn_X_length+3; i++){
80         X_binary_opt[i]=0;
81     }
82 }
```

Here is the caller graph for this function:

8.29.3.15 init_bn_settings()

```
void init_bn_settings (
             void  )
```

Initialize BN curves settings.

References generate_bn_mother_parameter(), generate_bn_order(), generate_bn_prime(), generate_bn_trace(), init_bn_parameters(), set_bn_curve_parameter(), and weil().

Referenced by init_bn().

```
24                      {
25     init_bn_parameters();
26
27     generate_bn_mother_parameter();
28     generate_bn_prime();
29     generate_bn_order();
30     generate_bn_trace();
31
32     weil();
33     set_bn_curve_parameter();
34 }
```

Here is the call graph for this function: Here is the caller graph for this function:

8.29.3.16 print_curve_parameters()

```
void print_curve_parameters (
             void  )
```

Prints curves public parameters.

References curve_params::curve_a, curve_params::curve_b, curve_parameters, curve_params::order, curve_↩ params::prime, curve_params::trace_t, and curve_params::X.

```
303                          {
304     printf("=============================================================================================\n");
305     printf("bn12\n\n");
306     gmp_printf("parameters\n");
307     gmp_printf("X     (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.X,2),curve_parameters.X);
308     gmp_printf("prime (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.prime,2),curve_parameters.
      prime);
309     gmp_printf("order (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.order,2),curve_parameters.
      order);
310     gmp_printf("trace (%dbit length) : %Zd \n",(int)mpz_sizeinbase(
      curve_parameters.trace_t,2),curve_parameters.
      trace_t);
311
312     if (mpz_cmp_ui(curve_parameters. curve_b, 0) > 0) {
313         print_bn_blscurve();
314     }
315     if (mpz_cmp_ui(curve_parameters.curve_a, 0) > 0) {
316         print_kss16curve();
317     }
318 //    gmp_printf("\nmodulo polynomial\n");
319 //    gmp_printf("Fp2  : f(x) = x^2+1\n");
320 //    gmp_printf("Fp6  : f(x) = x^3-(alpha+1)\n");
321 //    gmp_printf("Fp12 : f(x) = x^2-beta\n");
322
323 //    gmp_printf("\nnumber of the total rational points\n");
324 //    gmp_printf("EFp total   : %Zd\n",curve_parameters.EFp_total);
325 //    gmp_printf("EFp12 total : %Zd\n",curve_parameters.EFpd_total);
326
327 //    gmp_printf("\ncubic root of 1\n");
328 //    gmp_printf("epsilon1 : %Zd\n",epsilon1);
329 //    gmp_printf("epsilon2 : %Zd\n",epsilon2);
330
331 }
```

8.29.3.17 set_bn_curve_parameter()

```
void set_bn_curve_parameter (
            void  )
```

Set BN curve coefficients.

References curve_params::curve_b, and curve_parameters.

Referenced by init_bn_settings().

```
244                             {
245     mpz_set_ui(curve_parameters.curve_b,4);
246 }
```

Here is the caller graph for this function:

8.29.3.18 weil()

```
void weil (
            void  )
```

Generate BN curve's number of elements in EFp.

References curve_params::curve_a, curve_params::curve_b, curve_parameters, curve_params::EFp_total, curve_params::EFpd_total, curve_params::prime, and curve_params::trace_t.

Referenced by init_bn_settings().

```
248             {
249     mpz_t t2,t6,t12,p2,p6,buf;
250     mpz_init(t2);
251     mpz_init(t6);
252     mpz_init(t12);
253     mpz_init(p2);
254     mpz_init(p6);
255     mpz_init(buf);
256 
257     //EFp_total
258     mpz_add_ui(buf,curve_parameters.prime,1);
259     mpz_sub(curve_parameters.EFp_total,buf,
      curve_parameters.trace_t);
260 
261     //t2←^2+^2
262     mpz_pow_ui(t2,curve_parameters.trace_t,2);
263     mpz_mul_ui(buf,curve_parameters.prime,2);
264     mpz_sub(t2,t2,buf);
265     mpz_pow_ui(p2,curve_parameters.prime,2);
266 
267     //^6+^6
268     mpz_pow_ui(t6,t2,3);
269     mpz_mul(buf,t2,p2);
270     mpz_mul_ui(buf,buf,3);
271     mpz_sub(t6,t6,buf);
272     mpz_pow_ui(p6,p2,3);
273 
274     //^12+^12
275     mpz_pow_ui(t12,t6,2);
276     mpz_mul_ui(buf,p6,2);
277     mpz_sub(t12,t12,buf);
278 
279     //EFp12_total
280     mpz_pow_ui(buf,p6,2);
281     mpz_sub(buf,buf,t12);
282     mpz_add_ui(curve_parameters.EFpd_total,buf,1);
283 
284     mpz_clear(t2);
285     mpz_clear(t6);
286     mpz_clear(t12);
287     mpz_clear(p2);
288     mpz_clear(p6);
289     mpz_clear(buf);
290 }
```

Here is the caller graph for this function:

8.29.4 Variable Documentation

8.29.4.1 bls12_X_binary

```
int bls12_X_binary[78]
```

Array to hold BLS curve's mother parameter.

Referenced by bls12_finalexp_optimal(), bls12_generate_X(), bls12_Miller_algo_for_opt_ate(), and init_bls12_parameters().

8.29.4.2 bls12_X_length

```
int bls12_X_length
```

BLS12 curve's mother parameter length in bit.

Referenced by bls12_finalexp_optimal(), bls12_Miller_algo_for_opt_ate(), and init_bls12_parameters().

8.29.4.3 curve_parameters

```
struct curve_params curve_parameters
```

bn curve's systematically obtained parameters.

It's a global variable that give access to bn public parameters.

Referenced by bls12_finalexp_plain(), bls12_generate_G1_point(), bls12_generate_G2_point(), bls12_generate_trace(), bls12_Miller_algo_for_plain_ate(), bls12_Miller_algo_for_tate(), bls12_set_curve_parameter(), bls12_test_G1_scm(), bls12_test_G2_scm(), bls12_test_G3_exp(), bls12_test_opt_ate_pairing(), bls12_test_plain_ate_pairing(), bls12_test_tate_pairing(), bls12_weil(), bn_final_exp_plain(), clear_parameters(), EFp12_rational_point_bls12(), EFp12_rational_point_bn(), EFp2_rational_point(), EFp6_rational_point(), EFp_rational_point_bls12(), EFp_rational_point_bn(), Fp_add(), Fp_add_mpz(), Fp_add_ui(), Fp_inv(), Fp_isCNR(), Fp_legendre(), Fp_mul(), Fp_mul_basis(), Fp_mul_basis_KSS16(), Fp_mul_mpz(), Fp_mul_ui(), Fp_neg(), Fp_set_random(), Fp_sqrt(), Fp_sub(), Fp_sub_mpz(), Fp_sub_ui(), generate_bn_mother_parameter(), generate_bn_order(), generate_bn_prime(), generate_bn_trace(), init_bls12_parameters(), init_precoms(), print_curve_parameters(), set_bn_curve_parameter(), and weil().

8.29.4.4 state

```
gmp_randstate_t state
```

Global random state to generate random element

8.29.4.5 X_binary

```
char X_binary[bn_X_length+1]
```

Arrary to hold mother parameter of BN curve

Referenced by bn_final_exp_optimal(), bn_fp12_power_motherparam(), generate_bn_mother_parameter(), and init_bls12_parameters().

8.29.4.6 X_binary_opt

```
char X_binary_opt[bn_X_length+3]
```

Arrary to hold loop length Miller's algo for opt-ate pairing of BN curve

Referenced by generate_bn_mother_parameter(), and init_bls12_parameters().

8.30 include/ELiPS_bn_bls/field_dtype.h File Reference

```
#include <ELiPS_bn_bls/Commont_headers.h>
```

Include dependency graph for field_dtype.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct Fp
- struct Fp2
- struct Fp4
- struct Fp6
- struct Fp8
- struct Fp12
- struct Fp16

Typedefs

- typedef struct Fp Fp
- typedef struct Fp2 Fp2
- typedef struct Fp4 Fp4
- typedef struct Fp6 Fp6
- typedef struct Fp8 Fp8
- typedef struct Fp12 Fp12
- typedef struct Fp16 Fp16

8.30.1 Detailed Description

Interaface of finite field data types.

8.30.2 Typedef Documentation

8.30.2.1 Fp

```
typedef struct Fp Fp
```

Fp is the basic type that represent prime field element. Consist of one member element of type mpz_t

8.30.2.2 Fp12

```
typedef struct Fp12 Fp12
```

Fp12 is degree 2 extension over Fp6. Consist of three Fp2 element.

8.30.2.3 Fp16

```
typedef struct Fp16 Fp16
```

Fp16 is degree 2 extension over Fp8. Consist of three Fp8 element.

8.30.2.4 Fp2

```
typedef struct Fp2 Fp2
```

Fp2 is degree 2 extension over Fp. Consist of two Fp element.

8.30.2.5 Fp4

```
typedef struct Fp4 Fp4
```

Fp4 is degree 2 extension over Fp2. Consist of two Fp2 element.

8.30.2.6 Fp6

```
typedef struct Fp6 Fp6
```

Fp6 is degree 3 extension over Fp2. Consist of three Fp2 element.

8.30.2.7 Fp8

```
typedef struct Fp8 Fp8
```

Fp8 is degree 2 extension over Fp4. Consist of three Fp4 element.