TEPLA
(University of Tsukuba Elliptic Curve and Pairing Library) Manual

Laboratory of Cryptography and Information Security
University of Tsukuba

January 22, 2013
ver. 1.0.0

1 Overview of TEPLA

TEPLA (University of Tsukuba Elliptic Curve and Pairing Library) is a software library for development of applications or systems of cryptographic algorithms using pairings. Pairing is a bilinear map which has 2 inputs and 1 output. Our library implements the functions necessary for pairings, such as calculation of points on elliptic curves, calculation of elements on finite fields, etc.

TEPLA is written in C language and allows the use of pairing-based cryptography in different platforms.

TEPLA supports the following calculations:

- Calculations on Finite Fields (prime field of 254 bits, quadratic, 6th and 12th extension fields)
- Calculations on Elliptic Curves (Barreto-Naehrig (BN) curves)
- Calculations of Pairings (Optimal Ate Pairing on BN curves)

TEPLA provides the following functions:

- Calculations on Finite Fields (prime field of 254 bits, quadratic, 6th and 12th extension fields)
  - Addition, Subtraction, Multiplication, Inversion, Square Root, Exponentiation, Random Elements
- Calculations on Elliptic Curves (Barreto-Naehrig (BN) curves)
  - Addition, Scalar Multiplication, Random Points, Map-To-Point
- Calculations of Pairings (Optimal Ate Pairing on BN curves)

These functions are described in sections 4, 5 and 6.

TEPLA can run on several platforms. Currently, it is confirmed to work on Microsoft Windows, Apple Mac OS X and Linux. A detailed environment for each platform is available in section 2.

TEPLA is open source and distributed under the license similar to 3-Clause BSD License. The license add some descriptions about patent rights. The library is available for download at the web of the web of Laboratory of Cryptography and Information Security, University of Tsukuba [1] .

[1] http://www.cipher.risk.tsukuba.ac.jp/tepla/

Applied libraries and algorithms for each calculation in TEPLA are the following:

- Finite Field: 254 bits prime field , quadratic, 6th and 12th extension field
  - Calculations on Prime Field: GMP Library
  - Calculations on Extension Fields: Method by Beuchat et.al.[2]
- Elliptic Curve: Barreto-Naehrig Curves
  - Scalar Multiplication on G_1: Method on the book by Hankerson et. al. [3]
  - Scalar Multiplication on G_2: Method by Nogami et. al. [4]
  - Map-To-Point: Method on IEEE P1363.3 Draft[5]
  - Hash functions used in Mat-To-Point: OpenSSL Library
- Optimal Ate Pairing
  - Calculation of Pairings: Method by Beuchat et. al. [6]

Barreto-Naehrig Curves are expressed as $y^2 = x^3 + b, b \in F_p$. Characteristic p and order r are given based on parameter z as $p = 36z^4 + 36z^3 + 24z^2 + 6z + 1$ and $r = 36z^4 + 36z^3 + 18z^2 + 6z + 1$. $z = 2^{62} - 2^{54} + 2^{44}$ is used in TEPLA.

2 Install TEPLA

Please refer to the Install Manual for installing TEPLA. GMP Library and Open SSL are required.

TEPLA is confirmed to run on the following platforms:

- **Windows**

  OS Windows 7 Professional SP1 (64bit)

  C compiler Visual C++ 2010

  GMP (MPIR 2.6.0)

  OpenSSL 1.0.1c

[2]Beuchat, J.L., Daz, J.E.G., Mitsunari, S., Okamoto, E., Rodrguez-Henrquez, F., Teruya, T. "High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves". In Proc. of Pairing 2010. LNCS, vol. 6487, pp. 21-39., 2010

[3]Hankerson, Darrel, Menezes, Alfred J., Vanstone, Scott , "Guide to Elliptic Curve Cryptography", Springer, January 2004

[4]Nogami, Yasuyuki, Sakemi, Yumi, Okimoto, Takumi, Nekado Kenta, Akane Masataka, Morikawa, Yoshitaka, "Scalar Multiplication Using Frobenius Expansion over Twisted Elliptic Curve for Ate Pairing Based Cryptography", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Volume E92.A, Issue 1, pp.182-189 (2009).

[5]IEEE P1363.3/D6, "Draft Standard for Publickey Cryptography"

[6]Same as above 2

- **Linux**

  Kernel 2.6.18-308.8.1.el5 (Cent OS)
  C compiler gcc 4.1.2
  GMP 5.0.5
  OpenSSL 0.9.8e-fips-rhel5

- **Mac OS X**

  OS 10.6.8 (Snow Leopard)
  C compiler gcc 4.2.1
  GMP 5.0.4
  OpenSSL 0.9.8r

3 Declaration of Header (tepla/ec.h)

The header must be declared in the source code to use TEPLA.

```
#include <tepla/ec.h>
```

4 Initialization of Finite Fields

4.1 field_init(Field f, const char *param);

Initialize a finite field to use. There is no return value;

```
Field f;
field_init(f, "bn254_fp");
```

The following four finite fields can be used:

- bn254_fp
- bn254_fp2
- bn254_fp6
- bn254_fp12

Each parameter is a prime field, quadratic, 6th and 12th extension field, respectively.

4.2 field_clear(Field f);

Clear the finite field. There is no return value.

4.3 field_get_name(const Field f);

Get a name of current finite field. Returned value is an array of char type.

4.4 field_get_char(const Field f);

Get a value of characteristics of the finite field. Returned value is an array of mpz_t type.

4.5 field_get_degree(const Field f);

Get a value of degree of the finite field. Returned value is int.

5 Calculations on Finite Fields

Although quadratic, 6th and 12th extension fields are defined and implemented separately, the following functions can be used regardless of differences between them.

5.1 element_init(Element x, const Field f)

Initialize an element of the finite field. Input value x of Element type is initialized as an element on input finite field f.

5.2 element_clear(Element x)

Clear the element. Input value x of Element type is cleared by this function.

5.3 element_set(Element x, const Element y)

Set a value of an element. Input value y of Element type is set to x of Element type.

5.4 element_set_str(Element x, const char *str)

Set a value of an element. Input string str is set to x of Element type.

5.5 element_get_str(char *str, const Element x)

Get the value of the element. Contents of input value x of Element type is set to string str.

An element is set as string of hexadecimal number. If the element is on extension field, each element is set using a space

5.6 element_set_zero(Element x)

Input value x of Element type is set to zero.

5.7 element_set_one(Element x)

Input value x of Element type is set to one.

5.8 element_add(Element z, const Element x, const Element y)

Add two elements. Input value x and y of Element type are added. The result value is set to z of Element type.

5.9 element_neg(Element z, const Element x)

Obtain an inverse element on addition. An inverse element of input value x of Element type is set to z.

5.10 element_sub(Element z, const Element x, const Element y)

Subtract input value y of Element type from input value x of Element type. The result value is set to z of Element type.

5.11 element_mul(Element z, const Element x, const Element y)

Multiply two elements. Input value x and y of Element type are multiplied. The result value is set to z of Element type.

5.12 element_sqr(Element z, const Element x)

Square an element. Input value x of Element type is squared. The result value is set to z of Element type.

5.13 element_inv(Element z, const Element x)

Obtain an inverse element on multiplication. An inverse element of input value x of Element type is set to z.

5.14 element_pow(Element z, const Element x, const mpz_t exp)

Exponentiate an element. Input value x of Element type is exponentiated using exp of mpz_t type. The result value is set to z of Element type.

5.15 element_sqrt(Element z, const Element x)

Calculate an square root of an element. Input value x of Element type is square rooted. The result value is set to z. If x has square root, 1 of int type is returned. If not, 0 of int type is returned.

5.16 element_is_zero(const Element x)

Check if an element is zero. If input value x of Element is zero, 1 of int type is returned. If not, 0 of int type is returned.

5.17 element_is_one(const Element x)

Check if an element is one. If input value x of Element is one, 1 of int type is returned. If not, 0 of int type is returned.

5.18 element_is_sqr(const Element x)

Check if an element has square root. If input value x of Element type has square root, 1 of int type is returned. If not, 0 of int type is returned.

5.19 element_cmp(const Element x, const Element y)

Compare two elements. If input value x and y are same, 0 of int type is returned. If not, 1 of int type is returned.

5.20 element_random(Element x)

Choose a random element, and set to x.

The function of random number generation on GMP is used for the function.

5.21 element_to_oct(unsigned char *os, size_t *size, Element x)

Translate an element to an array of byte type. Input value x of Element type is translated to byte sequence then set to an array of unsigned char type os. Length of the array is set to an array size* of size_t type.

5.22 element_from_oct(Element z, const unsigned char *os, size_t size)

Translate an array of byte type to an element . Input array of unsigned char type os is translated to element then set to x of Element type. Length of the array is set to an array size of size_t type.

5.23 element_get_str_length(const Element x);

Translate an element to string of hexadecimal number, then obtain a length of the string. The length is returned as int type.

5.24 element_get_oct_length(const Element x);

Translate an element to an array of byte type, then obtain a size of the array. The size is returned as int type.

6 Setting of Elliptic Curve

6.1 curve_init(EC_GROUP ec, const char *param);

Initialize an elliptic curve to use. There is no return value;

```
EC_GROUP ec;
curve_init(ec, "ec_bn254_fp");
```

The following elliptic curve can be used:

- ec_bn254_fp
- ec_bn254_tw

The parameter "ec_bn254_fp" means the BN curve on prime fields. The parameter "ec_bn254_tw" means the sextic twist of "ec_bn254_fp" on quadratic extension fields.

6.2 curve_clear(EC_GROUP ec);

Clear the elliptic curve. There is no return value.

6.3 curve_get_name(const EC_GROUP ec);

Get a name of current elliptic curve. Returned value is an array of char type.

6.4 curve_get_order(const EC_GROUP ec);

Get an order of the subgroup on current elliptic curve. Returned value is an array of char mpz_t type.

7 Calculations on Elliptic Curves

7.1 point_init(EC_POINT p, const EC_GROUP ec)

Initialize a point of the elliptic curve. Input value p of EC_PONIT p type is initialized as a point on input elliptic curve ec.

7.2 point_clear(EC_POINT p)

Clear the point. Input value p of EC_PONIT is cleared by this function.

7.3 point_set(EC_POINT P, const EC_POINT Q)

Set a value of a point. Input value Q of EC_PONIT type is set to P of EC_PONIT type.

7.4 point_set_str(EC_POINT P, const char *s)

Set a value of a point. Input string str is set to P of EC_PONIT type.

7.5 point_set_xy(EC_POINT P, const Element x, const Element y)

Set a value of a point. Input value x, y of Element type is set to P of EC_PONIT type as P=(x,y).

7.6 point_set_infinity(EC_POINT P)

Input value P of EC_PONIT type is set to the point at infinity.

7.7 point_get_str(char *s, const EC_POINT P)

Get a value of the point. Contents of input value P of EC_PONIT type is set to string s. A point is set as string of two elements with hexadecimal expression. two elements is devided by comma, and put between "[]".

7.8 point_add(EC_POINT R, const EC_POINT P, const EC_POINT Q)

Add two points. Input value P and Q of EC_PONIT type are added. The result value is set to R of EC_PONIT type.

7.9 point_dob(EC_POINT Q, const EC_POINT P)

Double a point. Input value P of EC_PONIT type is doubled. The result value is set to Q of EC_PONIT type.

7.10 point_neg(EC_POINT Q, const EC_POINT P)

Calculate -P from input value P of EC_PONIT type, then set to Q of EC_PONIT type.

7.11 point_sub(EC_POINT R, const EC_POINT P, const EC_POINT Q)

Subtract input value Q of EC_PONIT type from input value P of EC_PONIT type. The result value is set to R of EC_PONIT type.

7.12 point_mul(EC_POINT Q, const mpz_t s, const EC_POINT P)

Calculate scalar multiplication of a point. Calculate sP with input value P of EC_PONIT type and input value s of mpz_t. The result value is set to Q of EC_PONIT type.

7.13 point_is_infinity(const EC_POINT P)

Check if a point is a point at infinity. If input value P of EC_PONIT type is a point at infinity, 1 of int type is returned. If not, 0 of int type is returned.

7.14 point_is_on_curve(const EC_POINT P)

Check if a point is on the elliptic curve. If input value P of EC_PONIT type is on the elliptic curve, 1 of int type is returned. If not, 0 of int type is returned.

7.15 point_cmp(const EC_POINT P, const EC_POINT Q)

Compare two points. If input value P and Q of EC_PONIT type are some, 0 of int type is returned. If not, 1 of int type is returned.

7.16 point_make_affine(EC_POINT Q, const EC_POINT P)

Calculate affine translation of input value P of EC_PONIT type. The result value is set to Q of EC_PONIT type.

7.17 point_map_to_point(EC_POINT P, const char *s, size_t slen, int t)

Input string s of an array of char type is mapped to a point on the elliptic curve. slen is a length of the input string. The result value is set to P of EC_PONIT type. Hash function is used to calculate map_to_point with the parameter of input value t of int type. The following parameters are available for hash functions:

- 80
- 112

- 128
- 192
- 256

If the parameter is 80, SHA1 is used. SHA-224, SHA-256, SHA-384 and SHA-512 are used with 112, 128, 192 and 254, respectively.

7.18 point_random(EC_POINT P)

Choose a random point on the elliptic curve, and set to P of EC_PONIT type. element_random is called for random number generation in the function.

7.19 point_to_oct(unsigned char* os, size_t *size, EC_POINT P)

Translate a point to an array of byte type. Input value P of EC_PONIT type is translated to byte sequence then set to an array of unsigned char type os. Length of the array is set to an array size* of size_t type.

7.20 point_from_oct(EC_POINT P, const unsigned char *os, size_t size)

Translate an array of byte type to a point. Input array of unsigned char type os is translated to element then set to P of EC_PONIT type. Length of the array is set to array size of size_t type.

8 Operations on Pairings

8.1 pairing_init(EC_PAIRING p, char *param)

Initialize a pairing to use. There is no return value.

```
EC_PAIRING p;
pairing_init(p, "ECBN254");
```

The following pairing can be used:

- ECBN254

8.2 pairing_clear(EC_PAIRING p)

Clear the pairing. There is no return value.

8.3 pairing_map(Element g, const EC_POINT P, const EC_POINT Q, const EC_PAIRING p)

Calculate pairing. Input value P and Q of EC_PONIT type are used to calculate pairing based on p of EC_PAIRING type. The result value is set to g of Element type.

8.4 pairing_get_order(const EC_PAIRING p)

Get an order of the cyclic group used in the pairing p of EC_PAIRING type. Returned value is mpz_t.

A Contributors

TEPLA is designed, developed, managed and translated by many contributors. We appreciate their help.

- Eiji Okamoto
- Akira Kanaoka
- Naoki Kanayama
- Kazutaka Saito
- Alberto Moreno Tablado
- Kenta Ishii