TEPLA
(University of Tsukuba Elliptic Curve and Pairing Library) マニュアル

筑波大学 暗号・情報セキュリティ研究室

平成 25 年 1 月 22 日
ver. 1.0.0

1　TEPLA の概要

TEPLA (University of Tsukuba Elliptic Curve and Pairing Library) は C 言語で利用するソフトウェアライブラリです。2 入力 1 出力で、双線形性などの特徴を持つペアリングと呼ばれる関数があります。ペアリングを使った暗号プロトコルを実装するときには、ペアリング自体の演算だけでなく、楕円曲線上の点の演算や、有限体の元の演算など複数の演算が必要とされます。TEPLA はそういったペアリングを使った暗号プロトコルの実装に必要となる様々な機能を備えています。

様々なプラットフォームでペアリングを使った暗号システムを構築可能にするために、TEPLA は汎用的に利用可能とすることを主な目的としています。さまざまなプラットフォームにおいて、プラットフォームの差を意識することなく使えることにすることで、ペアリングを使った暗号システムの利用促進を図ります。

提供される演算の種類は以下の通りです。

- 有限体上の元の演算（254 ビットの素体と 2 次・12 次拡大体）
- 楕円曲線上の点の演算（Barreto-Naehrig (BN) 曲線）
- ペアリング演算（BN 曲線上の Opimal Ate ペアリング）

各演算について、さまざまな機能を提供します。提供する機能は以下の通りです。

- 有限体上の元の演算
  - 加算、減算、乗算（積）、逆元、2 乗根、べき乗、ランダムな元の生成
- 楕円曲線上の点の演算
  - 加算、スカラ倍算、ランダムな点の生成、任意データの曲線上の点へのマッピング
- ペアリング演算
  - （ペアリング演算は、ペアリングの演算機能のみであり、その他の機能は提供していません）

各機能に対応する関数に関する説明は、4、5、6 章で行います。

TEPLA は、いくつかのプラットフォームで利用可能です。現在、Microsoft Windows と Apple Mac OS X、Linux での動作を確認済みです。動作確認をした環境の詳細な情報は 2 章で確認いただけます。

TEPLA はオープンソースで提供されます。筑波大学 暗号・情報セキュリティ研究室の Web からダウンロード可能です[1] 。

TEPLA は、修正 BSD ライセンス（三条項 BSD ライセンス）を基に特許についての事項を加えたものをライセンスとして、その下で提供されます。

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

OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

TEPLA で現状利用可能な有限体や楕円曲線、ペアリング、またそれぞれの演算で利用されているライブラリやアルゴリズムは以下の通りです。

- 有限体：254 ビットの素体、2 次・6 次・12 次拡大体
  - 素体上の元の演算： GMP ライブラリ
  - 拡大体上の元の演算：Beuchat らの手法[2]
- 楕円曲線： Barreto-Naehrig 曲線
  - G_1 上のスカラ倍算：Hankerson らの書籍に掲載されている手法[3]
  - G_2 上のスカラ倍算：Nogami らの手法[4]
  - Map-To-Point: IEEE P1363.3 草案に記載されている手法[5]
  - Map-To-Point でのハッシュ関数利用：OpenSSL ライブラリ
- Optimal Ate ペアリング
  - ペアリングの演算：Beuchat らの手法[6]

Barreto-Naehrig 曲線は $y^2 = x^3 + b, b \in F_p$ であらわされ、標数 p と位数 r はそれぞれ、パラメータ z を基に $p = 36z^4 + 36z^3 + 24z^2 + 6z + 1$ と $r = 36z^4+36z^3+18z^2+6z+1$ で表されます。TEPLA では $z = 2^{62}-2^{54}+2^{44}$ を利用しています。

2 TEPLA のインストール

TEPLA のインストール方法は、インストールマニュアルをご参照ください。TEPLA の利用には、GMP と OpenSSL がインストールされていることが必要になります。

TEPLA は以下のプラットフォームで動作確認がされております。

[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]上記 2 と同様

- **Windows**

  OS Windows 7 Professional SP1 (64bit)

  C コンパイラ Visual C++ 2010

  GMP (MPIR 2.6.0)

  OpenSSL 1.0.1c

- **Linux**

  Kernel 2.6.18-308.8.1.el5 (Cent OS)

  C コンパイラ gcc 4.1.2

  GMP 5.0.5

  OpenSSL 0.9.8e-fips-rhel5

- **Mac OS X**

  OS 10.6.8 (Snow Leopard)

  C コンパイラ gcc 4.2.1

  GMP 5.0.4

  OpenSSL 0.9.8r

3 ヘッダ (tepla/ec.h) の宣言

TEPLA を利用するために、ソースコードにヘッダを宣言する必要があります。

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

4 有限体の設定など

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

利用する有限体を指定します。戻り値はありません。

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

指定可能な有限体は以下の 4 つです。

- bn254_fp
- bn254_fp2

- bn254_fp6
- bn254_fp12

それぞれ素体、2 次拡大体、6 次拡大体、12 次拡大体を示します。

4.2 field_clear(Field f);

利用していた有限体を解放します。戻り値はありません。

4.3 field_get_name(const Field f);

利用している有限体の名称を取得します。名称が char 型の配列のポインタで返されます。

4.4 field_get_char(const Field f);

利用している有限体の標数を取得します。名称が mpz_t 型の配列のポインタで返されます。

4.5 field_get_degree(const Field f);

利用している有限体の次数を取得します。次数が int で返されます。

5 有限体上の元の演算

内部では 2 次拡大体、6 次拡大体、12 次拡大体の演算を分けて定義・実装してありますが、利用者はそれらを意識することなく下記関数を利用することが可能です。

5.1 element_init(Element x, const Field f)

体の元を初期化します。入力された Element 型変数 x が入力された有限体 f の元として初期化されます。

5.2 element_clear(Element x)

体の元を解放します。入力された Element 型の変数 x を解放します。

5.3 element_set(Element x, const Element y)

元の値を設定します。入力された Element 型変数 y の内容を、入力された Element 変数 x の内容に設定します。

5.4 element_set_str(Element x, const char *str)

元の値を設定します。入力された文字列 str の内容を、入力された Element 型変数 x の内容に設定します。

5.5 element_get_str(char *str, const Element x)

元の値を取得します。入力された Element 型変数 x の内容を文字列 str に設定します。

元は 16 進数表現で文字列に格納されます。拡大体の場合、各元は半角スペースを空けて 16 進数表現で文字列に格納されます。

5.6 element_set_zero(Element x)

入力された Element 型変数 x に 0 を設定します。

5.7 element_set_one(Element x)

入力された Element 型変数 x に 1 を設定します。

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

2 つの元の和を計算します。入力された Element 型変数 x と y の和を Element 型変数 z に設定します。

5.9 element_neg(Element z, const Element x)

加法に関する逆元を計算します。入力された Element 型変数 x の加法に関する逆元を Element 型変数 z に設定します。

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

2 つの元の差を計算します。入力された Element 型変数 x と y の差を Element 型変数 z に設定します。

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

2 つの元の積を計算します。入力された Element 型変数 x と y の積を Element 型変数 z に設定します。

5.12 element_sqr(Element z, const Element x)

元の 2 乗を計算します。入力された Element 型変数 x の 2 乗を Element 型変数 z に設定します。

5.13 element_inv(Element z, const Element x)

乗法に関する逆元を計算します。入力された Element 型変数 x の乗法に関する逆元を Element 型変数 z に設定します。

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

べき乗を計算します。入力された Element 型変数 x の mpz_t 型 exp 乗を Element 型変数 z に設定します。

5.15 element_sqrt(Element z, const Element x)

元の平方根を計算します。入力された Element 型変数 x の平方根を Element 型変数 z に設定します。x が平方根を持つ場合は 1、持たない場合は 0 が int で返されます。

5.16 element_is_zero(const Element x)

入力された Element 型変数 x が 0 かどうか判定します。入力された Element 型変数 x が 0 の場合 1、異なる場合は 0 が int で返されます。

5.17 element_is_one(const Element x)

入力された Element 型変数 x が 1 かどうか判定します。入力された Element 型変数 x が 1 の場合 1、異なる場合は 0 が int で返されます。

5.18 element_is_sqr(const Element x)

元が平方根を持つか計算します。入力された Element 型変数 x が平方根を持つ場合は 1、持たない場合は 0 が int で返されます。

5.19 element_cmp(const Element x, const Element y)

入力された 2 つの元を比較します。入力された Element 型変数 x と y を比較し、同じだったら 0 を、異なる場合は 1 を int で返します。

5.20 element_random(Element x)

ランダムな元を選択し、Element 型変数 x に設定します。
乱数の生成には GMP の乱数生成機能を利用しています。

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

元をバイト列に変換します。入力された Element 型変数 x をバイト列変換したものを unsigned char 型の配列のポインタ os に格納し、配列長が size_t 型変数のポインタ size に格納されます。

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

バイト列を元に変換します。入力された unsigned char 型配列のポインタ os を、入力された size_t 型の変数 size を長さとして、Element 型変数 z に格納します。

5.23 element_get_str_length(const Element x);

元を 16 進数表現された文字列に変換し、その文字列の長さを求めます。入力された Element 型変数 x の 16 進数表現した文字列の長さを int で返します。

5.24 element_get_oct_length(const Element x);

元をバイト列に変換し、そのバイト数を求めます。入力された Element 型変数 x のバイト列変換後のバイト長を int で返します。

6 楕円曲線の設定など

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

利用する楕円曲線を指定します。戻り値はありません。

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

指定可能な楕円曲線は以下の 1 つです。

- ec_bn254_fp
- ec_bn254_tw

それぞれ素体上の BN 曲線、ec_bn254_fp の 6 次 Twist 楕円曲線を示します。

6.2 curve_clear(EC_GROUP ec);

利用していた楕円曲線を解放します。戻り値はありません。

6.3 curve_get_name(const EC_GROUP ec);

入力された EC_GROUP 型変数 ec に設定されている楕円曲線の名称を取得します。名称が char 型の配列のポインタで返されます。

6.4 curve_get_order(const EC_GROUP ec);

入力された EC_GROUP 型変数 ec に設定されている楕円曲線上の部分群の位数を取得します。位数が mpz_t 型の配列のポインタで返されます。

7 楕円曲線上の点の演算

7.1 point_init(EC_POINT p, const EC_GROUP ec)

楕円曲線上の点を初期化します。入力された EC_PONIT 型変数 p が入力された楕円曲線 ec 上の点として初期化されます。

7.2 point_clear(EC_POINT p)

楕円曲線上の点を解放します。入力された EC_PONIT 型変数 p を解放します。

7.3 point_set(EC_POINT P, const EC_POINT Q)

楕円曲線上の点の値を設定します。入力された EC_PONIT 型変数 Q の内容を、入力された EC_PONIT 型変数 P の内容に設定します。

7.4 point_set_str(EC_POINT P, const char *s)

楕円曲線上の点の値を設定します。入力された文字列 s の内容を、入力された EC_PONIT 型変数 P の内容に設定します。

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

楕円曲線上の点を設定します。入力された Element 型変数 x、y を、点 (x,y) として入力された EC_PONIT 型変数 P の内容に設定します。

7.6 point_set_infinity(EC_POINT P)

入力された EC_PONIT 型変数 P を無限遠点に設定します。

7.7 point_get_str(char *s, const EC_POINT P)

楕円曲線上の点を取得します。入力された EC_PONIT 型変数 P の内容を文字列 str に設定します。

点 P の x 座標、y 座標がそれぞれ 16 進数表現され、カンマで区切られ [] で挟まれたものが文字列に格納されます。

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

2 つの点の和を計算します。入力された EC_PONIT 型変数 P と Q の和を入力された EC_PONIT 型変数 R に設定します。

なお、P=Q の場合は point_dob の結果が R に設定され、P=-Q の場合は無限遠点が R に設定されます。

7.9 point_dob(EC_POINT Q, const EC_POINT P)

点の 2 倍点を計算します。入力された EC_PONIT 型変数 P を 2 倍した結果を入力された EC_PONIT 型変数 Q に設定します。

7.10 point_neg(EC_POINT Q, const EC_POINT P)

入力された EC_PONIT 型変数 P に対し、-P を計算し、その結果を入力された EC_PONIT 型変数 Q に設定します。

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

2 つの点の差を計算します。入力された EC_PONIT 型変数 P と Q の差を入力された EC_PONIT 型変数 R に設定します。

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

楕円曲線上の点のスカラ倍算を計算します。入力された EC_PONIT 型変数 P と入力された mpz_t 型変数 s に対し、sP を計算した結果を入力された EC_PONIT 型変数 Q に設定します。

7.13 point_is_infinity(const EC_POINT P)

入力された EC_PONIT 型変数 P が無限遠点かどうか判定します。入力された EC_PONIT 型変数 P が無限遠点の場合 1、異なる場合は 0 が int で返されます。

7.14 point_is_on_curve(const EC_POINT P)

入力された EC_PONIT 型変数 P が楕円曲線上の点であるかどうか判定します。入力された EC_PONIT 型変数 P が楕円曲線上の点の場合 1、異なる場合は 0 が int で返されます。

7.15 point_cmp(const EC_POINT P, const EC_POINT Q)

入力された EC_PONIT 型変数 P と Q が一致するか判定します。一致する場合 0、異なる場合は 1 が int で返されます。

7.16 point_make_affine(EC_POINT Q, const EC_POINT P)

入力された EC_PONIT 型変数 P をアフィン変換した結果を入力された EC_PONIT 型変数 Q に設定します。

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

入力された char 型変数のポインタ s の文字列を、文字列の長さを入力された size_t 型変数 slen とし、その結果を入力された EC_POINT 型変数 P に設定します。その際、利用されるハッシュ関数のパラメータを int 型変数 t で設定します。パラメータは以下の 5 つから選択可能です。

- 80
- 112
- 128
- 192
- 256

パラメータが 80 の場合 SHA1、112 の場合は SHA-224、128 の場合 SHA-256、256 の場合 SHA-512 が使用されます。

7.18 point_random(EC_POINT P)

楕円曲線上の点をランダムに選び、入力された EC_PONIT 型変数 P に設定します。

この関数では乱数生成のために内部で element_random が呼ばれています。

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

楕円曲線上の点をバイト列に変換します。入力された EC_PONIT 型変数 P をバイト列変換したものを unsigned char 型の配列のポインタ os に格納し、配列長が size_t 型変数のポインタ size に格納されます。

P が無限遠点の場合、無限遠点を表す 1 オクテット (0x00) を os に格納します。P が無限遠点でない場合、1 オクテット (0x04)、x 座標、y 座標を連結したものを os に格納します。

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

バイト列を元に変換します。入力された unsigned char 型配列のポインタ os を、入力された size_t 型の変数 size を長さとして、入力された EC_PONIT 型変数 P に格納します。

8 ペアリングの計算など

8.1 pairing_init(EC_PAIRING p, char *param)

利用するペアリングを指定します。戻り値はありません。

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

指定可能な楕円曲線は以下の 1 つです。

- ECBN254

8.2 pairing_clear(EC_PAIRING p)

利用していたペアリングを解放します。戻り値はありません。

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

ペアリングを計算します。入力された EC_PONIT 型変数 P、Q のペアリングを入力された EC_PAIRING 型変数 p に基づき計算し、その結果を Element 型変数 g に格納します。

8.4 pairing_get_order(const EC_PAIRING p)

EC_PAIRING 型変数 p のペアリングで利用する巡回群の位数を mpz_t で返します。

A　協力者

TEPLA は以下の協力者により企画・開発・管理が行われてきました。協力いただいた皆様に深く感謝いたします。

- 岡本栄司
- 金岡晃
- 金山直樹
- 齋藤和孝
- 石井健太