ECDSA署名実装の解説 with TEPLA

ECDSAの実装

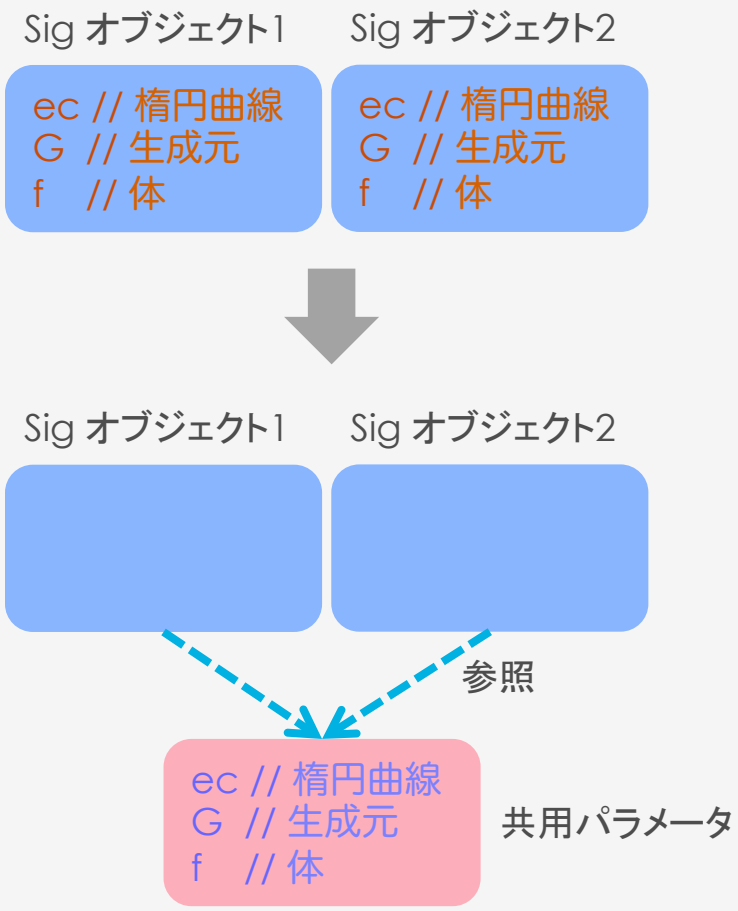
- ライブラリ
  - TEPLA 1.0
    - 曲線: ec_bn254_fp
  - GMP
  - open_ssl
    - SHA256
- 言語: C++
- ECDSAアルゴリズム
  - http://www.cryptrec.go.jp/estimation/rep_ID0003.pdf 参照

Class構造

```
1  class Sig
2  {
3    public:
4      Sig();
5      ~Sig();
6      mpz_t* get_sig(char*);  // char c = "r" or "s"
7      void key_gen();
8      void sign( const string );
9      bool vrfy( const string );
10     static void init() { /* 省略 */ }
11     static void fin() { /* 省略 */ }
12
13   private:
14     static EC_GROUP ec; // elliptic curve
15     static EC_POINT G;  // generator
16     static Field f ;
17     static mpz_t tmp;
18     static mpz_t n;      // order of curve
19     static gmp_randstate_t r_state;  // radomn state
20     EC_POINT Q;          // public key
21     mpz_t d;             // secret key
22     mpz_t r,s;           // Signature: S = (r,s)
23     static void gen_G();
24     void set_mpz_from_element( mpz_t , Element );
25 };
```

- クラス変数
  - 公開鍵，秘密鍵，署名など
  - static がついている変数は共用パラメータ
- メソッド
  - key_gen() : 鍵生成
  - sign() : 署名の生成
  - vrfy: 署名の検証
  - init(), fin() :
    - 共用パラメータの初期化及び後処理
  - gen_G() : 生成元の生成 (TEPLA 1.1では必要なくなる)
  - set_mpz_from_lement() :
    - Element型からmpz_t型への型変換

共用パラメータ



□ 無駄を削減するために共用パラメータを設定

- 楕円曲線や生成，体のパラメータは一度作成すれば，使いまわすことが可能
- オブジェクトごとに毎度生成するのは非常に効率が悪い

初期化と終了処理

```
 1 static void init()
 2 {
 3   mpz_init(tmp);  mpz_init(n);
 4
 5   curve_init(ec, "ec_bn254_fp");
 6   mpz_set(n, *curve_get_order(ec));
 7   point_init(G, ec);
 8   gen_G();
 9   field_init(f, "bn254_fp");
10
11   // init of random num
12   gmp_randinit_default(r_state);
13   gmp_randseed_ui(r_state, (unsigned long)time(NULL));
14 }
15
16 static void fin()
17 {
18   field_clear(f);
19   point_clear(G);
20   // curve_clear(ec); // segmentation fault 11 ??
21   mpz_clear(tmp); mpz_clear(n);
22   gmp_randclear(r_state);
23 }
```

共用パラメータの初期化と終了処理

```
 1 Sig::Sig()
 2 {
 3   point_init(Q , ec);
 4   mpz_init(r);
 5   mpz_init(s);
 6   mpz_init(d);
 7
 8   key_gen();
 9 }
10
11 Sig::~Sig()
12 {
13   point_clear(Q);
14   mpz_clear(r);
15   mpz_clear(s);
16   mpz_clear(d);
17 }
```

クラスの初期化と
終了処理

鍵生成

処理内容

鍵生成: 署名者 U および検証者 V は、以下により鍵を生成する。

1. 署名者 U は、楕円曲線ドメインパラメータ T を用いて、秘密鍵 $d \in [1, n-1]$ を (疑似) ランダムに選び、公開鍵 $Q = dG$ を生成する。
2. 検証者 V は、署名者 U が生成した公開鍵 Q を、検証可能な方法で取得する。

コード

```
1 void Sig::key_gen()
2 {
3   // *** generation of secret key 'd' (random value) ***
4   mpz_urandomm(d, this->r_state, this->n);
5   // *** generation of public key 'Q' ***
6   point_mul(Q, d, G);  // Q ←G^d
7 }
```

署名生成 1

処理内容

アルゴリズム:

1. $k \in [1, n-1]$ を (疑似) ランダムに選ぶ。
2. $kG = (x_1, y_1)$ を計算し、x_1 を整数表現 x_1' に変換する。
3. $r = x_1' \bmod n$ を計算する。$r = 0$ の場合 step 1 へ戻る。
4. $H(M)$ を計算し、出力をビット列 m に変換する。ハッシュ関数 H の出力が invalid の場合、invalid を出力して終了する。
5. $k^{-1} \bmod n$ を計算する。
6. $s = k^{-1}(m + dr) \bmod n$ を計算する。 $s = 0$ の場合 step 1 へ戻る。
7. 署名 $S = (r, s)$ を出力する。

署名生成2

コード

```
1  void Sig::sign( const string M )
2  { // *** initialization ***
3    unsigned char hash[SHA256_DIGEST_LENGTH];
4    mpz_t m, k, t;  mpz_init(m);        mpz_init(k);        mpz_init(t);
5    EC_POINT P, T;  point_init(P, ec);  point_init(T, ec);
6
7    // *** generation of signature 'S = (r,s)' ***
8    do {
9      mpz_urandomm(k, this->r_state, this->n);
10     point_mul(P, k, G);  // P ←k*G
11
12     set_mpz_from_element(this->r, T->x);
13     mpz_mod(this->r, this->r, this->n);   // r = x1 mod n
14
15     // *** ハッシュ計算 ***
16     SHA256( (unsigned char *)M.c_str(), M.length(), hash );
17     mpz_set_str(m, get_hex_string(hash, SHA256_DIGEST_LENGTH ).c_str(), 16);
18
19     // *** compute 's' ***          s = k^(-1)*(m+d*r) mod n
20     mpz_invert(k, k, this->n);      // k = k^-1
21     mpz_mul(t, d, r);               // t = d*r
22     mpz_add(t, m, t);               // t = m+t = m+d*r
23     mpz_mul(s, k, t);               // s = t = k*t = k^(-1)*(m+d*r)
24     mpz_mod(s, s, this->n);
25   } while ( mpz_sgn(r) == 0 or mpz_sgn(s) == 0 );
26   // *** finalization ***
27   mpz_clear(t);   mpz_clear(k);   mpz_clear(m);   point_clear(P); point_clear(T);
28 }
```

署名検証 1

typo !!
$u_1G + u_2Q$

処理内容

アルゴリズム:

1. r, s がともに $[1, n-1]$ の整数であることを確認する。
2. $H(M)$ を計算し、出力をビット列 m に変換する。
3. $u_1 = ms^{-1} \bmod n$, $u_2 = rs^{-1} \bmod n$ を計算する。
4. $R = (x_r, y_r) = u_1G - u_2Q$ を計算する。$R = O$ ならば invalid を出力。
5. x_r を整数表現 x'_r へ変換し $v = x'_r \bmod n$ を計算する。
6. $v = r$ ならば valid、$v \neq r$ ならば invalid を出力する。

署名検証2

コード

```
1 bool Sig::vrfy(const string M )
2 { // *** initialization ***
3   unsigned char hash[SHA256_DIGEST_LENGTH];
4   bool rslt = false;
5   mpz_t m, u1, u2, t; mpz_init(m); mpz_init(u1); mpz_init(u2); mpz_init(t);
6   EC_POINT R, T1, T2; point_init(R, ec); point_init(T1, ec); point_init(T2, ec);
7
8   // *** ハッシュ計算 ***
9   SHA256( (unsigned char *)M.c_str(), M.length(), hash );
10  mpz_set_str(m, get_hex_string(hash, SHA256_DIGEST_LENGTH ).c_str(), 16);
11
12  // *** compute u1, u2 ***   u1 = m*s^(-1) mod n,  u2 = r*s^(-1) mod n
13  mpz_invert(t, this->s, this->n); // t = s^(-1)
14  mpz_mul(u1, m, t);         // u1 = m*t = m*s^(-1)
15  mpz_mod(u1, u1, this->n); // u1 = m*t = m*s^(-1) mod n
16  mpz_mul(u2, this->r, t);  // u2 = r*t = r*s^(-1)
17  mpz_mod(u2, u2, this->n); // u2 = r*t = r*s^(-1) mod n
18
19  // *** compute R = (xr, yr) *** R = (xr, yr) = u1*G - u2*Q
20  point_mul(T1, u1, this->G);     // T1 ←u1*G
21  point_mul(T2, u2, this->Q);     // T2 ←u2*Q
22  point_add(R, T1, T2);           // R = T1 + T2 = u1*G + u2*Q
23  set_mpz_from_element(t, R->x);  mpz_mod(t, t, this->n);
24
25  if (mpz_cmp(t, this->r) == 0) { // if t eq r then valid
26    rslt =  true;
27  } else {  // if (R = O) or (t not eq r) then invalid
28    rslt = false;
29  }
30  // *** finalization ***
31  mpz_clear(m); mpz_clear(u1); mpz_clear(u2); mpz_clear(t);
32  point_clear(R); point_clear(T1); point_clear(T2);
33  return rslt;
34 }
```

バイト配列の変換

□ get_hex_string 関数

- open_sslのSHA関数で取得したハッシュ値はバイト配列に格納
- この値を文字列に変換して，GMPの INT 型(mpz_t)の変数に格納する必要がある
- これを実現するための関数

hash = SHA256 (message)　// hash ← [‘4’, ‘A’, ‘E’, ‘6’, … , ‘9’]
hoge = get_hex_string(hash) // hoge ← “4AE6 … 9”

```
1 string get_hex_string( unsigned char *data, size_t n )
2 {
3   stringstream ss;
4   for ( size_t i = 0; i < n; i++ )
5     ss << std::hex << std::setw(2) << std::setfill('0') << (int)data[i];
6   return ss.str();
7 }
```

サンプル実行コード

```
 1 #include <iostream>
 2 #include <assert.h>
 3 #include "ecdsa.h"
 4
 5 int main()
 6 {
 7   Sig::init();
 8   Sig hoge;
 9   string m = "hogehoge";
10   bool r1, r2;
11
12   hoge.sign(m);
13   r1 = hoge.vrfy(m);
14   r2 = hoge.vrfy("hogehogf");
15   assert( true  == r1 );  assert( false == r2 );
16
17   Sig::fin();
18   return 0;
19 }
```