Boneh等のアグリゲート署名実装解説 with TEPLA

実装

- 対象アグリゲート署名：

  D. Boneh and M.K. Franklin, “Identity-based encryption from the Weil pairing,” CRYPTO, ed. by J. Kilian, vol.2139, pp.213–229, Lecture Notes in Computer Science, Springer, 2001.

- 実装言語：　C++
- 利用したライブラリ：
  - TEPLA 1.0，GMP

署名クラス

□署名クラス(Sig Class, Co-GDH scheme)

- 署名単体の鍵生成，署名，検証を行う

Key Generation. For a particular user, pick random $x \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The user's public key is $v \in G_2$. The user's secret key is $x \in \mathbb{Z}_p$.

Signing. For a particular user, given the secret key x and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in G_1$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

Verification. Given user's public key v, a message M, and a signature σ, compute $h \leftarrow H(M)$; accept if $e(\sigma, g_2) = e(h, v)$ holds.

クラス構造

```
41 class Sig {
42   public:
43     void key_gen();
44     void sign( const string );
45     bool vrfy( const string );
46     static void init(){
47       pairing_init(prg, "ECBN254");
48       point_init(g2, prg->g2);
49       gen_g2();
50       field_init(f, "bn254_fp");
51     }
52     static void fin() {/* abbr. */}
53
54   private:
55     static EC_PAIRING prg;
56     static EC_POINT g2;
57     static Field f ;
58     EC_POINT v, h, s;
59     Element x;
60     static void gen_g2();
61 };
```

- 内部変数
  - x: 秘密鍵，v: 公開鍵
  - h: メッセージのハッシュ値
  - s: 署名
- g2の生成元作成
  - TEPLAのライブラリからは取得できないので関数を定義
- static 変数
  - 無駄の削減
  - g2の値を統一

Key Generation

Key Generation. For a particular user, pick random $x \xleftarrow{R} \mathbb{Z}_p$, and compute $v \leftarrow g_2^x$. The user's public key is $v \in G_2$. The user's secret key is $x \in \mathbb{Z}_p$.

```
137 // Key Generation
138 void Sig::key_gen(){
139   // *** generation of secret key 'x' (random value) ***
140   element_random(x);
141
142   // *** generation of public key 'v' ***
143   // v = g_2^x ⇔v = x * g_2
144   set_mpz_from_element(tmp, x); // tmp(mpz_t) ←x(Element)
145   point_mul(v, tmp, g2);   // v ←g_2^x
146 }
```

Signing

Signing. For a particular user, given the secret key x and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(M)$, where $h \in G_1$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G_1$.

```
148 // Signing
149 void Sig::sign( const string m ) {
150   // *** h ←H(M) ***
151   point_map_to_point(h, m.c_str(), m.size(), 192);
152
153   // *** generation of signature 's' ***
154   set_mpz_from_element(tmp, x); // tmp(mpz_t) ←x(Element)
155   point_mul(s, tmp, h); // s ←h^x
156 }
157
```

Verification

Verification. Given user's public key v, a message M, and a signature σ, compute $h \leftarrow H(M)$; accept if $e(\sigma, g_2) = e(h, v)$ holds.

```
158 // Verification
159 bool Sig::vrfy(const string m ) {
160   // *** initialization ***
161   Element t1, t2;
162   element_init(t1, prg->g3); element_init(t2, prg->g3);
163   bool rslt = false;
164
165   // *** h ←H(M) ***
166   point_map_to_point(h, m.c_str(), m.size(), 192);
167
168   // *** pairing computation ***
169   // *** e(s,g2), e(h,v)     ***
170   pairing_map(t1, s, g2, prg); pairing_map(t2, h, v,  prg);
171   rslt = element_cmp(t1, t2) == 0 ? true : false;
172
173   // *** finalization ***
174   element_clear(t1); element_clear(t2);
175
176   return rslt;
177 }
```

Sample Code

```
 7 int main()
 8 {
 9   Sig::init();
10   Sig hoge;
11   string m = "hogehoge";
12   bool r1, r2;
13
14   hoge.sign(m);
15   r1 = hoge.vrfy(m);
16   r2 = hoge.vrfy("hogehogf");
17   assert( true  == r1 );
18   assert( false == r2 );
19 }
```

アグリゲート署名クラス

□ アグリゲート署名クラス(AggSig Class)

● 複数署名の集約，およびこれの検証

Aggregation. For the aggregating subset of users $U \subseteq \mathbb{U}$, assign to each user an index i, ranging from 1 to $k = |U|$. Each user $u_i \in U$ provides a signature $\sigma_i \in G_1$ on a message $M_i \in \{0,1\}^*$ of his choice. The messages M_i must all be distinct. Compute $\sigma \leftarrow \prod_{i=1}^{k} \sigma_i$. The aggregate signature is $\sigma \in G_1$.

Aggregate Verification. We are given an aggregate signature $\sigma \in G_1$ for an aggregating subset of users U, indexed as before, and are given the original messages $M_i \in \{0,1\}^*$ and public keys $v_i \in G_2$ for all users $u_i \in U$. To verify the aggregate signature σ,

1. ensure that the messages M_i are all distinct, and reject otherwise; and
2. compute $h_i \leftarrow H(M_i)$ for $1 \leq i \leq k = |U|$, and accept if $e(\sigma, g_2) = \prod_{i=1}^{k} e(h_i, v_i)$ holds.

クラス構造

```
221 class AggSig {
222   public:
223     Sig* agg( Sig* , int );
224     bool vrfy( string*, Sig* , int );
225   private:
226     Sig asig;
227     EC_PAIRING prg;
228 };
```

- 内部変数
  - asig: 集約した署名
- 内部関数
  - agg 関数
    - 署名を集約する
  - vrfy 関数
    - 集約した署名を検証

Aggregation

Aggregation. For the aggregating subset of users $U \subseteq \mathbb{U}$, assign to each user an index i, ranging from 1 to $k = |U|$. Each user $u_i \in U$ provides a signature $\sigma_i \in G_1$ on a message $M_i \in \{0,1\}^*$ of his choice. The messages M_i must all be distinct. Compute $\sigma \leftarrow \prod_{i=1}^{k} \sigma_i$. The aggregate signature is $\sigma \in G_1$.

```
241 // Aggregation
242 Sig* AggSig::agg(Sig *sigs, int size)
243 {
244   asig.set_sig_inf(); // init identity element
245
246   if (size==0) {
247     cout << "sigunare is empty. Can't aggregate opt. \n";
248     return &asig;
249   }
250   while (size--, size>=0) {
251     asig.sig_add(&asig, sigs+size); // asig = asig + sigs[size]
252   }
253   return &asig;
254 }
```

Verification

compute $h_i \leftarrow H(M_i)$ for $1 \leq i \leq k = |U|$, and accept if $e(\sigma, g_2) = \prod_{i=1}^{k} e(h_i, v_i)$ holds.

```
256 // Aggregation Verification
257 bool AggSig::vrfy(string *msgs, Sig *sigs, int size) {
258   // *** initialization ***
259   ▒ ▒ // *** abbr. *** //
260
261   // *** h ←H(M) ***
262   for (int i=0; i<size; i++) {
263   ▒ point_map_to_point(hashs[i], msgs[i].c_str(), msgs[i].size(), 192);
264   }
265
266   // *** pairing computation ***
267   // *** Π e(hi,vi) ***
268   for (int i=0; i<size; i++) {
269   ▒ pairing_map(t1, hashs[i], sigs[i].get_v(), prg); // t1 = e(hi,vi)
270   ▒ element_mul (t2, t1, t2); // t2 = t1 * t2
271   }
272   // *** e(s,g2) ***
273   pairing_map(t1, asig.get_sig(), asig.get_g2(), prg);
274
275   // *** finalization ***
276   ▒ ▒ // *** abbr. *** //
277
278   rslt = element_cmp(t1, t2) == 0 ? true : false;
279   return rslt;
280 }
```

Sample Code


```
 7 int main() {
 8   Sig::init();
 9   AggSig fuga;
10   const int size = 2;
11   string msgs[size] = { "fuga1" , "fuga2" }, msgs1[size] = { "fuga1" , "fuga3" };
12   Sig* sigs = new Sig[size];
13
14   for (int i=0; i< size; i++) {
15     sigs[i].sign(msgs[i]);
16     r1 = sigs[i].vrfy(msgs[i]);
17     assert( true == r1);
18   }
19
20   fuga.agg( sigs, size);
21   r2 = fuga.vrfy(msgs, sigs, size);
22   assert( true == r2 );
23
24   // *** abbr. (sigs[i].sign(msg1[i]) operation
25   r2 = fuga.vrfy(msgs1, sigs, size);
26   assert( false == r2 );
27
28   delete [] sigs; Sig::fin();
29   return 0;
30 }
```