# REPORT

## 암호학 프로젝트 6 ECDSA(Elliptic Curve)


| | |
|---|---|
| 과목명 | 암호학 |
| 담당교수 | 오희국 교수님 |
| 학생이름 | 지현도 |
| 학과 | 컴퓨터학부 |
| 학번 | 2021004866 |
| 제출일 | 2023/12/11 |

HANYANG UNIVERSITY

*본인이작성한함수에대한설명:

1. int SHA2SIZE(int sha2_ndx)

```
49  // SHA2SIZE() - 사용할 sha2 함수의 길이를 반환
50  int SHA2SIZE(int sha2_ndx)
51  {
52      switch (sha2_ndx)
53      {
54          case SHA224:
55              return SHA224_DIGEST_SIZE;
56          case SHA256:
57              return SHA256_DIGEST_SIZE;
58          case SHA384:
59              return SHA384_DIGEST_SIZE;
60          case SHA512:
61              return SHA512_DIGEST_SIZE;
62          case SHA512_224:
63              return SHA224_DIGEST_SIZE;
64          case SHA512_256:
65              return SHA256_DIGEST_SIZE;
66      }
67      return 0;
68  }
```

기존에 헤더파일에 넣고 사용하던 형식을 위와같이 더 알아보기쉽게 함수형태로 변경하였습니다.

2. void sha(const unsigned char *data, unsigned len, unsigned char *digest, int sha2_ndx)

```
49  // SHA2SIZE() - 사용할 sha2 함수의 길이를 반환
50  int SHA2SIZE(int sha2_ndx)
51  {
52      switch (sha2_ndx)
53      {
54          case SHA224:
55              return SHA224_DIGEST_SIZE;
56          case SHA256:
57              return SHA256_DIGEST_SIZE;
58          case SHA384:
59              return SHA384_DIGEST_SIZE;
60          case SHA512:
61              return SHA512_DIGEST_SIZE;
62          case SHA512_224:
63              return SHA224_DIGEST_SIZE;
64          case SHA512_256:
65              return SHA256_DIGEST_SIZE;
66      }
67      return 0;
68  }
```

프로젝트에서 기본적으로 해시함수로 sha를 사용하므로

인자로받는 sha2_ndx 즉 사용될 sha종류에 따라 적절한 sha 시리즈를 사용하도록 함수를 설정해주었습니다.

3. int ecc_add(ecdsa_p256_t *P, ecdsa_p256_t *Q, ecdsa_p256_t *R)

```
// ecc상의 덧셈 계산, P!=Q 인 경우의 계산, 계산 결과 R을 반환해준다
int ecc_add(ecdsa_p256_t *P, ecdsa_p256_t *Q, ecdsa_p256_t *R)
{
    mpz_t x1, y1, x2, y2, x3, y3, tmp;
    mpz_inits(x1, y1, x2, y2, x3, y3, tmp, NULL);
    mpz_import(x1, ECDSA_P256/8, 1, 1, 1, 0, P->x);
    mpz_import(y1, ECDSA_P256/8, 1, 1, 1, 0, P->y);
    mpz_import(x2, ECDSA_P256/8, 1, 1, 1, 0, Q->x);
    mpz_import(y2, ECDSA_P256/8, 1, 1, 1, 0, Q->y);

   if(mpz_cmp_ui(x1,0)==0&&mpz_cmp_ui(y1,0)==0){
      // O + Q = Q
      mpz_set(x3,x2);
      mpz_set(y3,y2);
   }else if(mpz_cmp_ui(x2,0)==0&&mpz_cmp_ui(y2,0)==0){
      // P + O = P
      mpz_set(x3,x1);
      mpz_set(y3,y1);
   }else{
    // x3 생성
    mpz_sub(x3, y2, y1);   // x3 = y2 - y1
    mpz_sub(tmp, x2, x1);  // tmp = x2 - x1

    // P + Q = O일 경우 오류발생
    if(mpz_cmp_ui(tmp, 0) == 0){
      mpz_clears(x1, y1, x2, y2, x3, y3, tmp, NULL);
      return 1;
    }

    mpz_invert(tmp, tmp, p);  // inverse(x2 - x1) (mod n)
    mpz_mul(tmp, x3, tmp);    // tmp = (y2 - y1) / (x2 - x1)
    mpz_mul(x3, tmp, tmp);    // x3 = ((y2 - y1) / (x2 - x1))^2
    mpz_sub(x3, x3, x1);      // x3 = ((y2 - y1) / (x2 - x1))^2 - x1
    mpz_sub(x3, x3, x2);      // x3 = ((y2 - y1) / (x2 - x1))^2 - x1 - x2
    mpz_mod(x3, x3, p);       // x3 mod n

```

```
105
106     // y3 생성
107      mpz_sub(y3, x1, x3);       // y3 = x1 - x3
108      mpz_mul(y3, tmp, y3);      // y3 = ((y2 - y1) / (x2 - x1))(x1 - x3)
109      mpz_sub(y3, y3, y1);       // y3 = ((y2 - y1) / (x2 - x1))(x1 - x3) - y1
110      mpz_mod(y3, y3, p);        // y3 mod n
111     }
112
113      mpz_export(R->x, NULL, 1, ECDSA_P256/8, 1, 0, x3);
114      mpz_export(R->y, NULL, 1, ECDSA_P256/8, 1, 0, y3);
115      mpz_clears(x1, y1, x2, y2, x3, y3, tmp, NULL);
116      return 0;
117 }
```

ECC상에서 P!=Q 일때의 P+Q의 계산의 결과를 반환해주는 함수이다. ECC상에서의 덧셈 즉 기하학적 계산을 하며 이는 강의노트와 프로젝트 계획서에 명시된대로 타원 곡선 P-256 즉 $y^2 = x^3 - 3x + b \pmod{p}$ 에서의 계산을 기준으로 한다.

4. int ecc_doubling(ecdsa_p256_t *P, ecdsa_p256_t *R)

```
119 // ecc상의 덧셈 계산, P=Q인 경우 즉 2P계산 결과를 R으로 반환해준다
120 int ecc_doubling(ecdsa_p256_t *P, ecdsa_p256_t *R)
121 {
122     mpz_t x1, y1, x3, y3, tmp;
123     mpz_inits(x1, y1, x3, y3, tmp, NULL);
124     mpz_import(x1, ECDSA_P256/8, 1, 1, 1, 0, P->x);
125     mpz_import(y1, ECDSA_P256/8, 1, 1, 1, 0, P->y);
126 
127     // x3 생성
128     mpz_mul(x3, x1, x1);       // x3 = x1^2
129     mpz_mul_ui(x3, x3, 3);     // x3 = 3x1^2
130     mpz_sub_ui(x3, x3, 3);     // x3 = 3x1^2 - 3
131     mpz_mul_ui(tmp, y1, 2);    // tmp = 2y1
132     mpz_invert(tmp, tmp, p);   // inverse(2y1)(mod n)
133     mpz_mul(tmp, x3, tmp);     // tmp = (3x1^2 - 3) / 2y1
134     mpz_mul(x3, tmp, tmp);     // x3 = ((3x1^2 - 3) / 2y1)^2
135     mpz_sub(x3, x3, x1);       // x3 = ((3x1^2 - 3) / 2y1)^2 - x1
136     mpz_sub(x3, x3, x1);       // x3 = ((3x1^2 - 3) / 2y1)^2 - 2x1
137     mpz_mod(x3, x3, p);
138 
139     // y3 생성
140     mpz_sub(y3, x1, x3);       // y3 = x1 - x3
141     mpz_mul(y3, tmp, y3);      // y3 = ((3x1^2 - 3) / 2y1)(x1 - x3)
142     mpz_sub(y3, y3, y1);       // y3 = ((3x1^2 - 3) / 2y1)(x1 - x3) - y1
143     mpz_mod(y3, y3, p);
144 
145     mpz_export(R->x, NULL, 1, ECDSA_P256/8, 1, 0, x3);
146     mpz_export(R->y, NULL, 1, ECDSA_P256/8, 1, 0, y3);
147     mpz_clears(x1, y1, x3, y3, tmp, NULL);
148     return 0;
149 }
```

ECC상에서 P=Q 일때의 P+Q의 계산의 결과를 반환해주는 함수이다. ECC상에서의 덧셈 즉 기하학적 계산을 하며 이는 강의노트와 프로젝트 계획서에 명시된대로 타원 곡선 P-256 즉 y^2 = x^3 – 3x + b (mod p) 에서의 계산을 기준으로 한다.

5. void ecc_mul(ecdsa_p256_t *A, mpz_t d, ecdsa_P256_t *B)

```
151 // ecc상의 곱셈, square multiply 활용하여 A를 d번더하는 계산을 수행한다
152 void ecc_mul(ecdsa_p256_t *X, mpz_t d, ecdsa_p256_t *Y)
153 {
154     unsigned long int i = 0;
155     unsigned long int bin_bits = ECDSA_P256;
156 
157     // i = d 이고 i=d이기 전까지 계속 Y에 X를 더해준다
158     while(i <= bin_bits){
159         if(mpz_tstbit(d, i)==1) ecc_add(Y, X, Y);
160         // X+X 즉 X를 두배로만들고 저장(비트를 shift하는 square multiply 방식과 유사)
161         ecc_doubling(X, X);
162         i++;
163     }
164 }
```

ECC상에서 기하학적 곱셈계산 Y = dX를 계산하는 함수이다.

Square multiply 방식을 참고하여 먼저 구현해준 ecc_add와 ecc_doubling을 활용하여 구현해보았다.

6. void ecdsa_p256_clear(void)

```
186 /*
187  * Clear 256 bit ECDSA parameters
188  * 할당된 파라미터 공간을 반납한다.
189  */
190 void ecdsa_p256_clear(void)
191 {
192     mpz_clears(p, n, NULL);
193 }
```

할당된 파라미터 공간을 반납한다.

7. void ecdsa_p256_key(void *d, ecdsa_p256_t *Q)

```
/*
 * ecdsa_p256_key() - generates Q = dG
 * 사용자의 개인키와 공개키를 무작위로 생성한다.
 */

void ecdsa_p256_key(void *d, ecdsa_p256_t *Q)
{
   mpz_t temp_d;
   mpz_init(temp_d);

   // mpz_urandomm으로 랜덤값 temp_d 생성
   gmp_randstate_t state;
   gmp_randinit_default(state);
   gmp_randseed_ui(state, arc4random());
   mpz_urandomm(temp_d, state, n);

   // Q = d*G
   ecdsa_p256_t temp_G;  // G값을 저장하고 계산에 이용할 임시 변수 temp_G 생성
   memset(Q->x,0,ECDSA_P256/8);         // Q->x 초기화
   memset(Q->y,0,ECDSA_P256/8);         // Q->y 초기화
   memcpy(&temp_G.x,&G.x,ECDSA_P256/8); // G의 x값 복사
   memcpy(&temp_G.y,&G.y,ECDSA_P256/8); // G의 x값 복사
   ecc_mul(&temp_G, temp_d, Q);

   mpz_export(d, NULL, 1, ECDSA_P256/8, 1, 0, temp_d);

   mpz_clear(temp_d);
}
```

사용자의 개인키와 공개키를 무작위로 생성한다.

8. int ecdsa_p256_sign(const void *msg, size_t len, const void *d void *d, void *_r, void *_s, int sha2_ndx)

```
224 /*
225  * ecdsa_p256_sign(msg, len, d, r, s) - ECDSA Signature Generation
226  * 길이가 len 바이트인 메시지 m을 개인키 d로 서명한 결과를 r, s에 저장한다.
227  * sha2_ndx는 사용할 SHA-2 해시함수 색인 값으로 SHA224, SHA256, SHA384, SHA512,
228  * SHA512_224, SHA512_256 중에서 선택한다. r과 s의 길이는 256비트이어야 한다.
229  * 성공하면 0, 그렇지 않으면 오류 코드를 넘겨준다.
230  */
231 int ecdsa_p256_sign(const void *msg, size_t len, const void *d, void *_r, void *_s,
        int sha2_ndx)
232 {
233     unsigned char e[SHA512_DIGEST_SIZE];
234     int h_len;
235     mpz_t temp_e, temp_d, k, r, s;
236     ecdsa_p256_t signature;
237     gmp_randstate_t state;
238
239     // Step1. e = H(m). H()는 SHA-2 해시함수이다.
240     sha(msg, len, e, sha2_ndx);
241
242     // Step2. e의 길이가 n의 길이(256비트)보다 길면 뒷 부분은 자른다. bitlen(e) ≤ bitlen(n)
243     if (sha2_ndx == SHA384 || sha2_ndx == SHA512) h_len = SHA256_DIGEST_SIZE;
244     else h_len = SHA2SIZE(sha2_ndx); // 기존 비트 수 유지
245     mpz_inits(temp_e, temp_d, k, r, s, NULL);
246     mpz_import(temp_e, h_len, 1, 1, 1, 0, e);  // 해시 길이만큼 e를 잘라서 저장
247     mpz_import(temp_d, ECDSA_P256 / 8, 1, 1, 1, 0, d);
248
249     ecdsa_p256_t temp_G;  // G값을 저장하고 계산에 이용할 임시 변수 temp_G 선언
250     gmp_randinit_default(state);
251     gmp_randseed_ui(state, arc4random());
252     do
253     {
254         // Step3. 비밀값 k를 무작위로 선택한다. (0 < k < n)
255         mpz_urandomm(k, state, n);
256
257         // Step4. (x1, y1) = k*G
258         memset(signature.x, 0, ECDSA_P256 / 8);    // x1 초기화
259         memset(signature.y, 0, ECDSA_P256 / 8);    // y1 초기화
260         memcpy(&temp_G.x,&G.x,ECDSA_P256/8);        // G의 x값 복사
261         memcpy(&temp_G.y,&G.y,ECDSA_P256/8);        // G의 y값 복사
262
263         ecc_mul(&temp_G, k, &signature);   // (x1, y1) 생성
264
265         // Step5. r = x1 mod n
266         mpz_import(r, ECDSA_P256 / 8, 1, 1, 1, 0, signature.x);
267         mpz_mod(r, r, n);
268
269         // Step6. s = k^-1 * (e + rd) mod n
270         mpz_invert(k, k, n);     // k = k^-1
271         mpz_mul(temp_d, r, temp_d);      // temp_d = r*d
272         mpz_add(temp_d, temp_e, temp_d);     // temp_d = e + r*d
273         mpz_mul(s, k, temp_d);       // s = k^-1 * (e + rd)
274         mpz_mod(s, s, n);        // s = k^-1 * (e + rd) mod n
275
276     } while (mpz_cmp_ui(r, 0) == 0 || mpz_cmp_ui(s, 0) == 0);
```

```
278     mpz_export(_r, NULL, 1, ECDSA_P256 / 8, 1, 0, r);
279     mpz_export(_s, NULL, 1, ECDSA_P256 / 8, 1, 0, s);
280
281     mpz_clears(temp_e, temp_d, k, r, s, NULL);
282
283     return 0;
284 }
```

길이가 len바이트인 메세지 m을 개인키 d로 서명한 결과를 r,s 서명쌍으로 만들어준다. r과 s의 길이는 256비트이며 아래와같은 서명방식순서를 따른다.

- **서명 (Signature Generation)**
  1. $e = H(m)$. $H()$는 SHA-2 해시함수이다.
  2. $e$의 길이가 $n$의 길이(256비트)보다 길면 뒷 부분은 자른다. $bitlen(e) \leq bitlen(n)$
  3. 비밀값 $k$를 무작위로 선택한다. $(0 < k < n)$
  4. $(x_1, y_1) = kG$.
  5. $r = x_1 \bmod n$. 만일 $r = 0$이면 3번으로 다시 간다.
  6. $s = k^{-1}(e + rd) \bmod n$. 만일 $s = 0$이면 3번으로 다시 간다.
  7. $(r, s)$가 서명 값이다.

9. int ecdsa_p256_verify(const void *msg, size_t len, const ecdsa_p256_t *_Q, const void *_r, const void *_s, int sha2_ndx)

```
286 /*
287  * ecdsa_p256_verify(msg, len, Q, r, s) - ECDSA signature veryfication
288  * It returns 0 if valid, nonzero otherwise.
289  * 길이가 len 바이트인 메시지 m에 대한 서명이 (r,s)가 맞는지 공개키 Q로 검증한다.
290  * 성공하면 0, 그렇지 않으면 오류 코드를 넘겨준다.
291  */
292 int ecdsa_p256_verify(const void *msg, size_t len, const ecdsa_p256_t *_Q, const
        void *_r, const void *_s, int sha2_ndx)
293 {
294     // m의 길이가 hash function의 최대크기(2^61-1)보다 클 경우 오류 반환
295     if (len>0x1fffffffffffffff)
296         return ECDSA_MSG_TOO_LONG;
297 
298     unsigned char e[SHA512_DIGEST_SIZE];
299     int h_len;
300     mpz_t r, s, temp_e, temp, w, u1, u2, x1, v;
301     ecdsa_p256_t u1G, u2Q, XY;
302 
303     mpz_inits(r, s, temp_e, w, temp, u1, u2, x1, v, NULL);
304     mpz_import(r, ECDSA_P256/8, 1, 1, 1, 0, _r);
305     mpz_import(s, ECDSA_P256/8, 1, 1, 1, 0, _s);
306 
307     // Step1. r과 s가 [1,n-1] 사이에 있지 않으면 잘못된 서명이다.
308     mpz_sub_ui(temp, n, 1);
309     if (mpz_cmp_ui(r,1) < 0 || mpz_cmp(r,temp) > 0 || mpz_cmp_ui(s,1) < 0 ||
            mpz_cmp(s,temp) > 0) return ECDSA_SIG_INVALID;
310 
311     // Step2. e=H(m) H()는 서명에서 사용한 해시함수와 같다.
312     sha(msg, len, e, sha2_ndx);
313 
314     // Step3. e의 길이가 n의 길이(256비트)보다 길면 뒷 부분을 자른다. bitlen(e) <= bitlen(n)
315     if (sha2_ndx == SHA384 || sha2_ndx == SHA512) h_len = SHA256_DIGEST_SIZE;
316     else h_len=SHA2SIZE(sha2_ndx);
317     mpz_import(temp_e, h_len, 1, 1, 1, 0, e);  // 해시 길이만큼 e를 잘라서 저장
```

```
319     // Step4. u1 = es^-1 mod n, u2 = rs^-1 mod n
320     mpz_invert(w, s, n);  // w = s^-1 mod n
321     mpz_mul(u1, temp_e, w);   // u1 = e*s^-1
322     mpz_mul(u2, r, w);    // u2 = r*s^-1
323     mpz_mod(u1, u1, n);   // u1 = u1 mod n
324     mpz_mod(u2, u2, n);   // u2 = u2 mod n
325
326     // Step5. (x1, y1) = u1G + u2Q.만일 (x1, y1) = O이면 잘못된 서명이다.
327     ecdsa_p256_t temp_G;  // G값을 저장하고 계산에 이용할 임시 변수 temp_G 선언
328     ecdsa_p256_t temp_Q;  // Q값을 저장하고 계산에 이용할 임시 변수 temp_Q 선언
329     memcpy(&temp_G.x,&G.x,ECDSA_P256/8);    // G의 x값 복사
330     memcpy(&temp_G.y,&G.y,ECDSA_P256/8);    // G의 y값 복사
331     memcpy(&temp_Q.x,_Q->x,ECDSA_P256/8);   // Q의 x값 복사
332     memcpy(&temp_Q.y,_Q->y,ECDSA_P256/8);   // Q의 y값 복사
333
334     // u1G, u2Q를 0으로 초기화
335     memset(u1G.x,0,ECDSA_P256/8);
336     memset(u1G.y,0,ECDSA_P256/8);
337     memset(u2Q.x,0,ECDSA_P256/8);
338     memset(u2Q.y,0,ECDSA_P256/8);
339
340     ecc_mul(&temp_G, u1, &u1G);
341     ecc_mul(&temp_Q, u2, &u2Q);
342     if (ecc_add(&u1G, &u2Q, &XY)==1)
343         return ECDSA_SIG_INVALID;
344
345     // Step6. r = x1 (mod n)이면 올바른 서명이다.
346     mpz_import(x1, ECDSA_P256/8, 1, 1, 1, 0, XY.x);
347     mpz_mod(v, x1, n);   // v = x1 mod n
348
349     // v!=r 이면 전자서명 인증실패
350     if(mpz_cmp(v,r)!=0)
351         return ECDSA_SIG_MISMATCH;
352
353     mpz_clears(r, s, temp_e, temp, w, u1, u2, x1, v, NULL);
354     return 0;
355 }
356
```

길이가 len 바이트인 메세지 m에 대한 서명이 (r,s) 가 맞는지 공개키 Q로 검증한다. 검증 방식은 아래의 순서를 따른다.

- **검증 (Signature Verification)**
  1. $r$과 $s$가 $[1, n-1]$ 사이에 있지 않으면 잘못된 서명이다.
  2. $e = H(m)$. $H()$는 서명에서 사용한 해시함수와 같다.
  3. $e$의 길이가 $n$의 길이(256비트)보다 길면 뒷 부분은 자른다. $bitlen(e) \leq bitlen(n)$
  4. $u_1 = es^{-1} \bmod n, u_2 = rs^{-1} \bmod n$.
  5. $(x_1, y_1) = u_1G + u_2Q$. 만일 $(x_1, y_1) = O$이면 잘못된 서명이다.
  6. $r \equiv x_1 \pmod{n}$이면 올바른 서명이다.

*컴파일 과정

```
jihyeondo@Jiui-MacBookAir proj#6 % make
gcc -Wall -O3 -c ecdsa.c
gcc -o test test.o ecdsa.o sha2.o -lgmp
jihyeondo@Jiui-MacBookAir proj#6 %
```

오류없이 정상적으로 컴파일되는 모습이다.

*실행 결과물

```
jihyeondo@Jiui-MacBookAir proj#6 % ./test
d = 20255cacf547068372b87eccd010ff44971600074f30f14db946dc2ca2d3ee0d
Qx = f7b42c01d931d03884fedb24db7aa82a08f752303b1f400de5dac87e17d0bfe7
Qy = 87fef91edd454643e4ca5afc06e0cf3eea40e36ceee9cf928b053139358346fb
r = 1e62153a7ef352a95bed830003d80d8f6f6bcfb224a43bf260a1ae61c0fa659b
s = 2cc11cda1f78a087b47e04934cbcd25f9c3129821aab1c704a5c6de99da41418
Valid signature ...PASSED
Signature verification error = 3 ...PASSED
Signature verification error = 3 ...PASSED
---
Signature verification error = 1 ...PASSED
---
Signature verification error = 2 ...PASSED
Signature verification error = 2 ...PASSED
---
Valid signature ...PASSED
Valid signature ...PASSED
---
Random Testing.....................................................................
.............. ...PASSED
CPU 사용시간  = 281.2502초
jihyeondo@Jiui-MacBookAir proj#6 %
```

테스트코드의 모든 테스트를 오류없이 통과한 모습입니다.

모두 정상적으로 PASSED된걸로 보아 테스트코드의 모든 상황에 정상적으로 실행되는 것으로 보입니다.

*프로젝트 소감 및 부족한 점

수업에서 배운 Elliptic Curve 를 구현해보았는데, 아무래도 처음 써보는 mpz 함수들로 기하학적 계산을 구현하는 과정이 처음이다보니 개인적으로 공부하는데 시간이 조금 소모되었던 것 같습니다. 특히 기하학적 계산에서 항등원인 경우의 예외처리에서 애를 조금 먹었습니다. 이번 프로젝트로 Elliptic Curve의 연산에 대해 더욱 자세하게 알게 되었고 아직 완벽하게 적재적소에 적용하지는 못하는 것 같지만 mpz 라이브

러리의 사용법을 익힐 수 있는 유익한 시간이였던 것 같습니다.