Project6 - 암호학

A3조 고동우 우경완 이지현 최성환

1. 함수설명

1) sel_shaFunc

사용할 sha2 함수를 선택해서 호출하는 함수이다. input으로 message, len, digest, sha2_ndx을 받고, sha2_ndx에 맞는 sha 함수를 호출한다. switch case문을 이용하였다.

2) sel_shaLen

sha2_ndx 함수의 output 길이를 return하는 함수이다. input으로 sha2_ndx를 받고 sha2_ndx에 맞는 output 길이를 return한다. switch case문을 이용하였다.

3) ec_add, ec_doubling

타원곡선 P-256 상에서 두 점을 더하는 함수이다. P + Q = O일 경우(0으로 나누어지는 케이스) 1을 리턴, 정상적으로 계산되면 0을 리턴한다. 타원곡선 P-256 y^2 = x^3 + ax + b 상의 두 점 P(X1, Y1), Q(X2, Y2)의 덧셈 R(X3, Y3)은 다음과 같이 계산한다.

P != Q 일 때: ec_add

X3 = ((Y2 - Y1)/(X2 - X1))^2 - X1 - X2
Y3 = (Y2 - Y1)/(X2 - X1) * (X1 - X3) - Y1

P == Q 일 때: ec_doubling

X3 = ((3(X1^2) + a)/(2 * Y1)) - 2*X1
Y3 = (3(X1^2) + a)/(2 * Y1) * (X1 - X3) - Y1

4) ec_mul

square multiplication 함수에 기반한 함수로, 타원곡선 P-256의 점 A를 d번 더한 값을 B에 저장한다.

1. i = 0, bits = d의 비트 개수이다.

2. i <= bits 인지 확인한다. TRUE이면 다음 단계를 실행하고, FALSE이면 계산을 멈추고 결과를 B에 저장한다.

3. d의 i번 비트가 1이면 B에 A를 더한다(ec_add(B, A, B)).

4. A를 2배하고 A에 저장한다(ec_doubling(A, A)).

5. i에 1을 더한다.

6. 2단계로 다시 돌아간다.

5) ecdsa_p256_init

p, n, G(x, y)를 프로젝트에서 주어진 고정된 값으로 초기화하는 함수이다.

6) ecdsa_p256_clear

p, n의 할당된 공간을 clear하는 함수이다.

7) ecdsa_p256_key

개인키 d를 random으로 생성하고 elliptic curve 곱셉 연산 Q=dG를 통해 공개키 Q를 생성하는 함수이다.

8) ecdsa_p256_sign

우선 메시지를 암호화해서 변수 e에 저장하고 해시 길이 h_len을 설정하는데 만약 해시 색인 값이 SHA384나 SHA512일 경우 해시 길이가 256비트를 넘어가기 때문에 최대값인 256으로 설정한다. 그 후 설정된 해시 길이만큼 해시값을 복사하고 d값이 변하는 것을 방지하기위해 이 값도 복사한다. (x1, y1) = kG를 계산할 때 nG값을 저장하기 위한 tempG를 선언하고 랜덤값 k를 생성하기 위한 시드를 설정하고 반복문으로 들어간다. k를 생성하고 (x1, y1)을 (0, 0)으로, temG를 G로 초기화 하고 (x1, y1) = kG를 계산한다. 그 후 r = x1 mod n을 계산하고 여기서 r = 0일경우 반복문의 처음으로 돌아가고, 아닐경우 s = k^-1 * (e + rd) mod n을 계산한 후 반복문의 조건으로 진입해서 s = 0인지 확인한다. (x1, y1), r, s가 계산되면 _r, _s에 r, s를 각각 변환해서 저장하고 메모리를 차지하는 변수를 삭제하고 암호화 결과값 0을 반환한다.

9) ecdsa_p256_verify

r과 s가 [1, n-1]사이에 있는지 파악하기 위해 mpz_cmp_ui와 mpz_cmp를 이용해 비교한다. 만약 위의 범위에 있지 않으면 ECDSA_SIG_INVALID를 return한다. 그 후 sel_shaFunc 함수를 사용하여 m을 해쉬한다. e의 길이가 n의 길이(256비트) 보다 길면 뒷부분을 자른다. sha_ndx가 256비트보다 긴 SHA384, SHA512일 경우 hlen을 SHA256 digit size로 바꾸어준다. u1, u2를 계산하기 위해 mpz_invert 함수로 s의 inverse를 구하고, mpz_mul 함수로 e*s^-1를 구한다. u2 계산은 앞에 계산했던 s^-1를 재사용하고 mpz_mul을 사용하여 r*s^-1를 계산한 후 mpz_mod로 u1, u2를 mod 연산 한다. 그 후 x1, y1을 계산한다. 먼저 u1G, u2G를 memset을 이용해 0으로 초기화한다. ec_mul 함수를 이용해 u1과 G, u2와 Q를 곱해준다. 만약 ec_add 함수로 u1G와 u2Q를 더한 결과가 1이라면 x1, y1이 무한대 점이라는 의미이므로 ECDSA_SIG_INVALID를 return한다. x1을 mod한 결과와 r을 mod한 결과가 같지 않다면 잘못된 서명이므로 ECD_SIG_MISMATCH를 return한다. 모든 계산이 끝난 후 모든 mpz 변수를 mpz_clears 함수를 이용하여 초기화한다. 위의 과정에서 중간에 return되지 않았다면 올바른 서명이므로 0을 return한다.

2. 컴파일 과정

```
root@LAPTOP-IFTR7CVN:/home/dongwoo/proj#6# make
gcc -Wall -O3 -c ecdsa.c
gcc -o test test.o ecdsa.o sha2.o -lgmp -lbsd
```

3. 실행 결과물 및 소감

```
root@LAPTOP-IFTR7CVN:/home/dongwoo/proj#6# ./test
d = d8c37457b25fcb7f92ec7143b08d7c1b54d856e5f07715e463db198b547fb44e
Qx = f2a0449016394e2562349c55d20cb053c1c11559543425d03e027e97d3a3bf66
Qy = 258abbdde134965959263229ab9e0345c3e9fabe9e639fbddae1428bd9c6439b
r = b7269b0842f3d466f12c5b5dd0e2e6b9c283994d5526c8b848330fe2dc965bef
s = 42b186b1b61a7fe767284e926b390aeb6ab61db5021df26eb45511cbdc41f06d
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 = 120.2051초
```

동우

elliptic curve 연산을 구현하는 데 오래걸렸다. 아무래도 mpz연산을 다루는 것이 처음이다보니 메뉴얼을 뒤져보고 이해하는 과정에서 시간이 좀 걸렸다. elliptic curve 덧셈 연산을 ppt에 있는 식대로 제대로 구현했는데 테스트벡터에 있는 값이 나오지 않아 애를 먹었다. 그래서 입력값들을 단순화하고 각 연산과정마다 출력을 다 해보면서 어디서부터 잘못되었는지 찾아보았다. 잘못된 점은 P, Q를 더하는 과정에서 둘 중 하나가 O일 경우 주어진 식대로 연산한 것이 문제였다. O가 elliptic curve 덧셈 연산에서 항등원 역할을 한다는 것을 생각해야 한다. 그래서 P가 O일 경우에는 결과값으로 O+Q=Q이고 Q가 O일 경우에는 결과값으로 P+O=P로 구하면 해결되는 문제였다. 이것을 해결하고 난 후에 나머지 함수 구현은 신속히 진행할 수 있었다. 이 프로젝트를 통해 elliptic curve 연산과 DSA에 대해 잘 이해할 수 있었고 뿐만 아니라 mpz연산을 익히는 데 많은 도움이 되었다.

경완

타원곡선이라는 다소 생소한 개념을 생소한 함수들을 이용하여 구현하니 초반에는 고전하였다. 과제 pdf와 수업자료를 참고하며 코드를 작성했음에도 불구하고 결과가 생각처럼 나오지 않아 진전이 느렸지만, 팀원들과 의논하고 서로가 작성한 코드를 비교하면서 조금씩 배우다보니 큰 문제없이 코드를 완성할 수 있었던 것 같다.

지현

gmp 라이브러리를 이용하여 알고리즘을 구현하게 되었는데, 처음에는 생소한 함수들이 많아 사용에 어려움을 겪었지만 반복해서 사용하다 보니 익숙해질 수 있었다. 이번 과제에서 가장 헷갈렸던 과정은 eliptic curve 좌표를 연산하는 부분이였다. 테스트 벡터를 이용하여 코드를 실행시키면 고정된 결과값이 나와야 하지만 실행시킬 때마다 다른 결과값이 나왔다. 많은 테스트 과정을 거친 결과, 변수를 초기화시키지 않아 결과값이 계속해서 변화했다는 것을 찾아낼 수 있었다. 알고리즘을 제대로 구현하는 것도 중요하지만, 가장 기본적인 변수 선언과 초기화 작업 또한 중요하다는 것을 알게 되었다.

성환

이번 과제에서는 어려움을 많이 겪었다. 변수 mpz_t부터 구조체 ecdsa_p256_t, 여러가지 함수들 등 모든 것이 처음이었고 과제를 해결하기 위해 mpz 매뉴얼을 계속 보고, 매뉴얼을 보고도 이해가 안된다면 인터넷에 검색해서 사용 예제 등을 찾아보았다. 또한 알고리즘대로 설계했을텐데 왜 결과값이 다르게 나오는지 고민하는 일도 많았다. 그래도 한번 감을 찾으니 그 다음은 꽤나 수월

하게 진행됐다. 이번 과제를 통해 ecdsa와 eliptic curve등에 대해 더 잘 알게되었다.