프로젝트 #6 (팀)

컴퓨터학부 암호학

2023년 11월 20일

문제

표준문서 NIST FIPS 186-4에 명시된 ECDSA (Elliptic Curve Digital Signature Algorithm) 전자서명 기법을 타원곡선 P-256 상에서 구현한다.

Curve P-256

타원곡선 P-256은 다음과 같이 정의한다.

$$y^2 = x^3 - 3x + b \pmod{p}$$

여기서 p는 길이가 256비트인 소수로 다음 값을 사용한다. 모든 수는 16진수로 표현하였다.

$$p = \text{FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF}$$

위 조건을 만족하는 타원곡선의 점들은 유한체를 이루는데, 이 과제에서 사용할 그룹의 기저점(base point)과 차수(order)는 다음과 같다.

$$\begin{aligned} n &= \text{FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551} \\ G_x &= \text{6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296} \\ G_y &= \text{4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5} \end{aligned}$$

소수 n은 그룹의 차수이고, G가 기저점이므로 $nG = O$를 만족한다. 여기서 O는 무한대 점으로 항등원이다.

ECDSA

서명자의 개인키가 d이고, 공개키가 $Q = dG$일 때, 메시지 m에 대한 ECDSA 전자서명 알고리즘은 다음과 같다.

- **서명 (Signature Generation)**
  1. $e = H(m)$. $H()$는 SHA-2 해시함수이다.
  2. e의 길이가 n의 길이(256비트)보다 길면 뒷 부분은 자른다. $bitlen(e) \le bitlen(n)$
  3. 비밀값 k를 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.

- **검증 (Signature Verification)**

  1. r과 s가 $[1, n-1]$ 사이에 있지 않으면 잘못된 서명이다.
  2. $e = H(m)$. $H()$는 서명에서 사용한 해시함수와 같다.
  3. e의 길이가 n의 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. $(x_1, y_1) = u_1 G + u_2 Q$. 만일 $(x_1, y_1) = O$이면 잘못된 서명이다.
  6. $r \equiv x_1 \pmod{n}$이면 올바른 서명이다.

GMP 함수

GNU GMP 라이브러리에는 크기가 2^{64}보다 큰 수를 계산하기 위한 여러 가지 함수가 있다. 이 과제는 길이가 256비트인 큰 수를 사용하여 계산한다. 과제를 수행하기 위해서는 이들 함수에 대한 지식이 필요하다. 함수의 수가 많기 때문에 다 이해하는 것은 시간이 많이 소요된다. 다행스럽게 ECDSA 계산에 꼭 필요한 함수의 수는 그렇게 많지 않다. 메뉴얼을 참조해서 다음에 열거한 함수의 사용법을 잘 숙지한다.

- 초기화/삭제: `mpz_init()`, `mpz_inits()`, `mpz_clear()`, `mpz_clears()`
- 값 설정: `mpz_set()`, `mpz_set_ui()`, `mpz_set_str()`, `mpz_get_str()`
- 산술연산1: `mpz_add()`, `mpz_add_ui()`, `mpz_sub()`, `mpz_sub_ui()`, `mpz_mul()`,
- 산술연산2: `mpz_mul_ui()`, `mpz_mod()`, `mpz_mod_ui()`, `mpz_powm()`, `mpz_powm_ui()`
- 비교연산: `mpz_cmp()`, `mpz_cmp_ui()`
- 비트연산1: `mpz_and()`, `mpz_ior()`, `mpz_xor()`, `mpz_com()`
- 비트연산2: `mpz_setbit()`, `mpz_clrbit()`, `mpz_combit()`, `mpz_tstbit()`
- 정수론: `mpz_probab_prime_p()`, `mpz_gcd()`, `mpz_lcm()`, `mpz_invert()`
- 입출력: `mpz_out_str()`, `mpz_inp_str()`
- 난수: `mpz_urandomb()`, `mpz_urandomm()`, `gmp_randinit_default()`
- 데이터 변환: `mpz_import()`, `mpz_export()`

함수 구현

ECDSA 전자서명 기법을 타원곡선 P-256 상에서 구현하는데 필요한 함수의 프로토타입을 아래에 열거하였다. 각 함수에 대한 요구사항은 다음과 같다.

- `void ecdsa_p256_init(void)` – 시스템 파라미터 p, n, G의 공간을 할당하고 값을 초기화한다.
- `void ecdsa_p256_clear(void)` – 할당된 파라미터 공간을 반납한다.
- `void ecdsa_p256_key(void *d, ecdsa_p256_t *Q)` – 사용자의 개인키와 공개키를 무작위로 생성한다.
- `int ecdsa_p256_sign(const void *m, size_t len, const void *d, void *r, void *s, int sha2_ndx)` – 길이가 `len` 바이트인 메시지 `m`을 개인키 `d`로 서명한 결과를 `r, s`에 저장한다. `sha2_ndx`는 사용할 SHA-2 해시함수 색인 값으로 `SHA224`, `SHA256`, `SHA384`, `SHA512`, `SHA512_224`, `SHA512_256` 중에서 선택한다. `r`과 `s`의 길이는 256비트이어야 한다. 성공하면 0, 그렇지 않으면 오류 코드를 넘겨준다.
- `int ecdsa_p256_verify(const void *m, size_t len, const ecdsa_p256_t *Q, const void *r, const void *s, int sha2_ndx)` – 길이가 `len` 바이트인 메시지 `m`에 대한 서명이 `(r,s)`가 맞는지 공개키 `Q`로 검증한다. 성공하면 0, 그렇지 않으면 오류 코드를 넘겨준다.

오류 코드

ECDSA 실행 과정에서 발생하는 오류를 아래에 열거한 코드를 사용하여 식별한다.

- ECDSA_MSG_TOO_LONG – 입력 메시지가 너무 길어 한도를 초과함
- ECDSA_SIG_INVALID – 검증 과정에서 형식이나 값이 잘못된 서명
- ECDSA_SIG_MISMATCH – 검증 마지막 단계에서 값이 일치하지 않는 서명 불일치

ECDSA 테스트 벡터

다음은 타원곡선 P-256 상에서 SHA-384 해시함수를 사용해서 생성한 검증 벡터이다. 아래 벡터를 사용하여 프로그램이 올바르게 돌아가는지 확인한다.

```
Curve P-256:

y^2 = x^3 - 3x + b (mod p)

Group prime:

p = FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF

Group order:

n = FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551

Group base point:

Gx = 6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296
Gy = 4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5

Private key:

d = C9AFA9D845BA75166B5C215767B1D6934E50C3DB36E89B127B8A622B120F6721

Signature with SHA-384, message = "sample":

k = 09F634B188CEFD98E7EC88B1AA9852D734D0BC272F7D2A47DECC6EBEB375AAD4
x1 = 0EAFEA039B20E9B42309FB1D89E213057CBF973DC0CFC8F129EDDDC800EF7719
y1 = BB78F0E6EC1BC1F3DC0900D3C4F2955D1E27865BEE7AC17E57D465E06F981D86
e = 9A9083505BC92276AEC4BE312696EF7BF3BF603F4BBD381196A029F340585312
r = 0EAFEA039B20E9B42309FB1D89E213057CBF973DC0CFC8F129EDDDC800EF7719
s = 4861F0491E6998B9455193E34E7B0D284DDD7149A74B95B9261F13ABDE940954
```

골격 파일

구현이 필요한 골격파일 `ecdsa.skeleton.c`와 함께 헤더파일 `ecdsa.h`, 프로그램을 검증할 수 있는 `test.c`, SHA-2 오픈소스 `sha2.c`, `sha2.h` 그리고 `Makefile`을 제공한다. 이 가운데 `test.c`, `sha2.c`, `sha2.h`를 제외한 나머지 파일은 용도에 맞게 자유롭게 수정할 수 있다.

제출물

과제에서 요구하는 함수가 잘 설계되고 구현되었다는 것을 보여주는 자료를 보고서 형식으로 작성한 후 PDF로 변환하여 `PROJ6(팀원이름).pdf`로 제출한다. 다음과 같은 것이 반드시 포함되어야 한다.

- 작성한 함수에 대한 설명
- 컴파일 과정을 보여주는 화면 캡처
- 실행 결과물의 주요 장면과 그에 대한 설명, 소감, 문제점
- 프로그램 소스파일 (`ecdsa.c`, `ecdsa.h`) 별도 제출
- 프로그램 실행 결과 (`ecdsa.txt`) 별도 제출
- 팀원 평가표 (LMS를 통해 개별적으로 제출한다. 미제출자는 서류제출 미비로 취득한 점수에서 50% 감점한다.)

평가

- **Correctness 50%**: 프로그램이 올바르게 동작하는 지를 보는 것입니다. 여기에는 컴파일 과정은 물론, 과제가 요구하는 기능이 문제없이 잘 작동한다는 것을 보여주어야 합니다. 학생들이 제출한 `ecdsa.h`와 `ecdsa.c`는 Ubuntu 20.04 LTS 환경에서 컴파일하고 검증합니다. `Makefile`, `test.c`, `sha2.h`, `sha2.c`는 수정할 수 없습니다. 경고를 포함한 모든 오류는 큰 감점을 받습니다. macOS 사용자는 제출하시기 전에 우분투 환경에서 오류가 없는지 확인하시기 바랍니다.
- **Presentation 50%**: 자신의 생각과 작성한 프로그램을 다른 사람이 쉽게 이해할 수 있도록 프로그램 내에 적절한 주석을 다는 행위와 같이 자신의 결과를 잘 표현하는 것입니다. 뿐만 아니라, 프로그램의 가독성, 효율성, 확장성, 일관성, 모듈화 등도 여기에 해당합니다. 이 부분은 상당히 주관적이지만 그러면서도 중요한 부분입니다. 컴퓨터과학에서 중요하게 생각하는 best coding practices를 참조하기 바랍니다.
- **점수 비공개**: 과제 점수는 팀원 평가에 따라 각자 다릅니다. 평가자를 보호하기 위해 과제 점수와 팀원 평가 점수는 공개하지 않습니다.

HK