

프로젝트 #4

소프트웨어학부 암호학

2021년 10월 14일

목표

키의 길이가 64 비트인 미니 RSA 알고리즘을 구현한다. 공개키 암호방식의 국제표준 알고리즘인 RSA는 현재 키의 길이가 최소 2048 비트가 되어야 안전하다. 이 과제에서는 실세계에서 활용하기에는 안전하지 않지만 RSA의 기본 원리를 이해하기에 충분하고, 구현이 까다롭지 않은 미니 RSA를 선택하였다.

RSA 키의 길이

RSA 키의 길이는 RSA 모듈러스 n 의 길이를 의미한다. 구체적으로는 n 값을 이진수로 표현하기 위한 최소 비트의 수를 뜻한다. 예를 들어, n 이 15이면 4 비트로 15를 표현할 수 있으므로 길이가 4 비트가 된다. 따라서 RSA 키의 길이가 64 비트이면 n 값이 $2^{63} \leq n < 2^{64}$ 사이에 있어야 한다는 뜻이다.

카마이클 함수 $\lambda(n)$

표준시스템에서는 RSA 키를 생성할 때 오일러 함수 $\phi(n)$ 대신에 계산량을 줄여주는 카마이클 함수 $\lambda(n)$ 을 사용한다. 미니 RSA도 $\lambda(n)$ 을 사용하여 키를 생성한다. $\lambda(n)$ 은 다음과 같이 정의된다.

$$\lambda(n) = \text{lcm}(p-1, q-1) = \frac{(p-1)(q-1)}{\text{gcd}(p-1, q-1)}$$

전역 함수

외부에서 보이는 전역 함수를 아래 열거한 프로토타입을 사용하여 구현한다. 각 함수에 대한 요구사항은 다음과 같다.

- `void mRSA_generate_key(uint64_t *e, uint64_t *d, uint64_t *n)` – 길이가 32 비트 내 외인 임의의 두 소수 p 와 q 를 생성한 다음, 키의 길이가 64 바이트인 RSA 공개키 (e, n) 과 개인키 (d, n) 을 생성한다. RSA 모듈러스 n 값은 $2^{63} \leq n < 2^{64}$ 을 만족해야 한다. 두 소수 p 와 q 의 길이가 비슷할수록 더 안전하다는 점을 참고한다.
- `int mRSA_cipher(uint64_t *m, uint64_t k, uint64_t n)` – $m \leftarrow m^k \bmod n$ 을 계산한다. 계산 중 오류가 발생하면 0이 아닌 값을 넘겨주고, 없으면 0을 넘겨준다. $m \geq n$ 이면 m 이 값의 범위를 넘었으므로 오류로 처리해야 한다.

지역 함수

내부에서만 사용하는 지역 함수는 별도로 지정하지 않고, 각자 필요에 맞게 작성한다. 다음에 열거한 함수와 프로토타입은 참고용이다.

- `static uint64_t gcd(uint64_t a, uint64_t b)` – $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$ 유클리드 알고리즘을 사용하여 음이 아닌 두 정수의 최대공약수를 계산한다. 단, 재귀함수 호출을 사용하지 말고 while 루프를 사용하여 구현한다.

- `static uint64_t mul_inv(uint64_t a, uint64_t m)` – 모듈로 m 에서 음이 아닌 정수 a 의 곱의 역인 $a^{-1} \bmod m$ 을 구한다. 만일 역이 존재하지 않는다면 0을 리턴해야 한다. 확장유클리드 알고리즘을 약간 변형하면 쉽게 구할 수 있다.
- `static uint64_t mod_add(uint64_t a, uint64_t b, uint64_t m)` – $a + b \bmod m$ 을 계산하여 넘겨준다. a 와 b 가 각각 m 보다 작다는 가정하에서 a 와 b 의 합이 m 보다 크거나 같으면 결과에서 m 을 빼줘야 한다. 이 경우 실제 계산은 $a - (m - b)$ 로 하는 것이 오버플로를 피할 수 있는 좋은 방법이다. 또한 $a + b \geq m$ 을 검사하는 과정에서 오버플로가 발생할 수 있으므로 b 를 오른쪽으로 넘겨 $a \geq m - b$ 를 검사하는 것이 오버플로를 피할 수 있는 현명한 방법이다.
- `static uint64_t mod_mul(uint64_t a, uint64_t b, uint64_t m)` – $ab \bmod m$ 을 계산하여 넘겨준다. 오버플로가 발생할 수 있기 때문에 프로그래밍 언어가 제공하는 곱셈으로는 계산이 올바르지 않을 수 있다. 앞에서 정의한 `mod_add()`가 오버플로를 고려했다는 점과 곱셈을 덧셈을 사용하여 빠르게 계산할 수 있는 “double addition” 알고리즘을 사용하면 문제를 해결할 수 있다. 그 알고리즘은 다음과 같다.

```

r = 0;
while (b > 0) {
    if (b & 1)
        r = mod_add(r, a, m);
    b = b >> 1;
    a = mod_add(a, a, m);
}
return r;

```

- `static uint64_t mod_pow(uint64_t a, uint64_t b, uint64_t m)` – $a^b \bmod m$ 을 계산하여 넘겨준다. 오버플로가 발생할 수 있기 때문에 이 역시 프로그래밍 언어가 제공하는 지수함수로는 계산이 올바르지 않을 수 있다. 앞에서 정의한 `mod_mul()`이 오버플로를 고려했다는 점과 지수연산을 곱셈을 사용하여 빠르게 계산할 수 있는 “square multiplication” 알고리즘을 사용하면 문제를 해결할 수 있다. 그 알고리즘은 다음과 같다.

```

r = 1;
while (b > 0) {
    if (b & 1)
        r = mod_mul(r, a, m);
    b = b >> 1;
    a = mod_mul(a, a, m);
}
return r;

```

- `static int miller_rabin(uint64_t n)` – 64 비트 정수 n 이 소수이면 **PRIME**을, 합성수이면 **COMPOSITE**를 넘겨준다. n 이 2^{64} 보다 작으므로 결정적 밀러라빈 알고리즘을 사용한다. 이 때 검증할 베이스 a 의 값은 다음에 열거된 것만으로 충분하다.

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37

골격 파일

구현에 필요한 골격파일 `mRSA.c`와 함께 헤더파일 `mRSA.h`, 프로그램을 검증할 수 있는 `test.c`, 그리고 `Makefile`을 제공한다. 이 가운데 `test.c`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 이름_학번_PROJ4.pdf로 제출한다. 여기에는 다음과 같은 것이 반드시 포함되어야 한다.

- 본인이 작성한 함수에 대한 설명
- 컴파일 과정을 보여주는 화면 캡처
- 실행 결과물의 주요 장면과 그에 대한 설명, 소감, 문제점 등
- 프로그램 소스파일 (`mRSA.c`, `mRSA.h`) 별도 제출
- 프로그램 실행 결과 (`mRSA.txt`) 별도 제출

평가

- **Correctness 50%:** 프로그램이 올바르게 동작하는 지를 보는 것입니다. 여기에는 컴파일 과정은 물론, 과제가 요구하는 기능이 문제없이 잘 작동한다는 것을 보여주어야 합니다.
- **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 practices**를 참조하기 바랍니다.

HK