암호학

Project #1

소프트웨어학부
2017012251
윤영훈

1. source code 설명

① gcd

```c
int gcd(int a, int b)
{
    int temp = 0, result = 0;

    // a, b 둘중 하나가 0이 될때까지 반복
    while (a != 0 && b != 0){
        temp = a;
        a = b;
        b = temp % b;
    }

    if (a == 0)
        result = b;
    else
        result = a;

    return result;
}
```

유클리드 알고리즘 gcd(a,b) = gcd(b, a mod b)를 사용하여 a, b 둘 중 하나가 0이 될 때까지 while loop를 반복하였다. loop가 끝난 뒤 a가 0이면 b를, b가 0이면 a를 return했다.

② xgcd

```c
int xgcd(int a, int b, int *x, int *y)
{
    // d0, d1, d2 선언
    int d0 = a, d1 = b, d2;
    // x0, x1, x2 선언
    int x0 = 1, x1 = 0, x2;
    // y0, y1, y2 선언
    int y0 = 0, y1 = 1, y2;
    // q 선언
    int q;

    // d2의 값이 0이 될때까지 loop
    while (1){
        q = d0 / d1;
        d2 = d0 - q * d1;
        x2 = x0 - q * x1;
        y2 = y0 - q * y1;

        // d2의 값이 0일 경우 x, y에 x1, y1의 값을 넣고 d1을 return하며 종료
        if (d2 == 0){
            *x = x1;
            *y = y1;

            return d1;
        }

        // d2의 값이 0이 아닌 경우 다음 loop를 위해 변수들을 swap
        d0 = d1;
        d1 = d2;
        x0 = x1;
        x1 = x2;
        y0 = y1;
        y1 = y2;
    }
}
```

d : d0, d1, d2
x : x0, x1, x2
y : y0, y1, y2

d, x, y 각각 값의 최신화를 위해 3개의 변수를 추가로 할당. 그 후 q, d2, x2, y2의 값을 갱신을 반복하다 d2의 값이 0이 되는 경우 x, y에 각각 x1, y1의 값을 넣고 d1을 return하며 종료한다.

d2의 값이 0이 아닌 경우 다음 loop를 위해 변수들을 swap한다.

③ mul_inv

```c
int mul_inv(int a, int m)
{
    // d0, d1 선언
    int d0 = a, d1 = m;
    // x0, x1 선언
    int x0 = 1, x1 = 0;

    // q 선언 후 q를 대입한 d2, x2 선언
    int q = d0 / d1;
    int d2 = d0 - q * d1;
    int x2 = x0 - q * x1;

    // d2가 1 이하가 될때까지 loop
    while (d2 > 1){
        q = d0 / d1;
        d2 = d0 - q * d1;
        x2 = x0 - q * x1;

        // 다음 loop를 위해 변수들을 swap
        d0 = d1;
        d1 = d2;
        x0 = x1;
        x1 = x2;
    }

    // d2가 1인 경우 x2가 양수인 경우 x2를 그대로 return, 음수인 경우 m을 더해 양수 처리 후 return
    if (d2 == 1)
        return (x2 > 0 ? x2 : x2 + m);
    // d2가 1이 아닌 경우 역이 없으므로 0을 return
    else
        return 0;
}
```

앞선 확장 유클리드 알고리즘을 변형하였다. 다만 결과 도출에 필요가 없는 y에 대한 변수들을 제거했다. 확장 유클리드 알고리즘과의 차이점은 d2가 1 이하가 될 때 loop를 종료하며, 이후 d2의 값에 따라 return값이 다르다는 점이다.

<table>
<tr><th colspan="2">d2 == 1</th><th>else</th></tr>
<tr><td>x2 양수</td><td>return x2</td><td rowspan="2">return 0</td></tr>
<tr><td>x2 음수</td><td>return x2 + m
(음수를 양수로 처리)</td></tr>
</table>

④ umul_inv

```c
uint64_t umul_inv(uint64_t a, uint64_t m)
{
    uint64_t d0 = a, d1 = m;
    uint64_t x0 = 1, x1 = 0;

    uint64_t q = d0 / d1;
    uint64_t d2 = d0 - q * d1;
    uint64_t x2 = x0 - q * x1;

    while (d2 > 1){
        q = d0 / d1;
        d2 = d0 - q * d1;
        x2 = x0 - q * x1;

        d0 = d1;
        d1 = d2;
        x0 = x1;
        x1 = x2;
    }

    // unsigned 64 비트 정수의 경우 음이 아닌 정수이므로 기존 mul_inv 함수의 경우 return문에서 x2 < 0인 케이스가 존재하지 않아 x2 + m이 실행되지 않음
    // 따라서 조건문과 그 값들을 unsigned 64 비트 정수 입력에 맞춰 수정
    if (d2 == 1)
        return (x2 > (uint64_t)1<<63 ? x2 + m : x2);
    //d2가 1이 아닌 경우 역이 없으므로 0을 return
    else
        return 0;
}
```

unsigned 64 비트 정수인 경우 음이 아닌 정수이므로 기존 mul_inv 함수를 그대로 사용할 경우 return문에서 x2 < 0 인 케이스가 존재하지 않아 x2 + m 이 실행되지 않음. 따라서 조건문과 그 값들을 맨 처음 비트가 1인 경우와 그렇지 않은 경우에 대해 케이스를 나눠 unsigned 64 비트 정수 입력에서도 정상적으로 동작하도록 수정.

test_umul_inv-1.c를 통해 umul_inv 함수 검증을 최종적으로 마쳤고 변수들의 자료형만 uint8_t에서 uint64_t 로 변경 후 euclid_gf8.c에 삽입하였다.

```
xion@xion-VirtualBox:~$ cd 바탕화면
xion@xion-VirtualBox:~/바탕화면$ cd project#1
xion@xion-VirtualBox:~/바탕화면/project#1$ gcc -o test_umul_inv test_umul_inv-1.c -lbsd
xion@xion-VirtualBox:~/바탕화면/project#1$ ./test_umul_inv
Passed...Congratulations!
xion@xion-VirtualBox:~/바탕화면/project#1$
```

umul_inv 함수 검증 결과

⑤ xtime & gf8_mul

```c
// gf8_mul, gf8_pow 함수 구현을 위한 xtime 함수 선언
uint8_t xtime(uint8_t x)
{
    return ((x<<1) ^ ((x>>7) & 1 ? 0x1B : 0));
}
```

```c
uint8_t gf8_mul(uint8_t a, uint8_t b)
{
    uint8_t r = 0;

    while (b>0){
        if (b&1) r = r ^ a;
        b = b >> 1;
        a = xtime(a);
    }

    return r;
}
```

강의노트 (lec 04 29p) 를 참고하여 a * b modulo m(x) 에 해당하는 gf8_mul() 구현. 이때 사용된 xtime() 함수 역시 강의노트 (lec 04 28p)를 참고하여 구현.

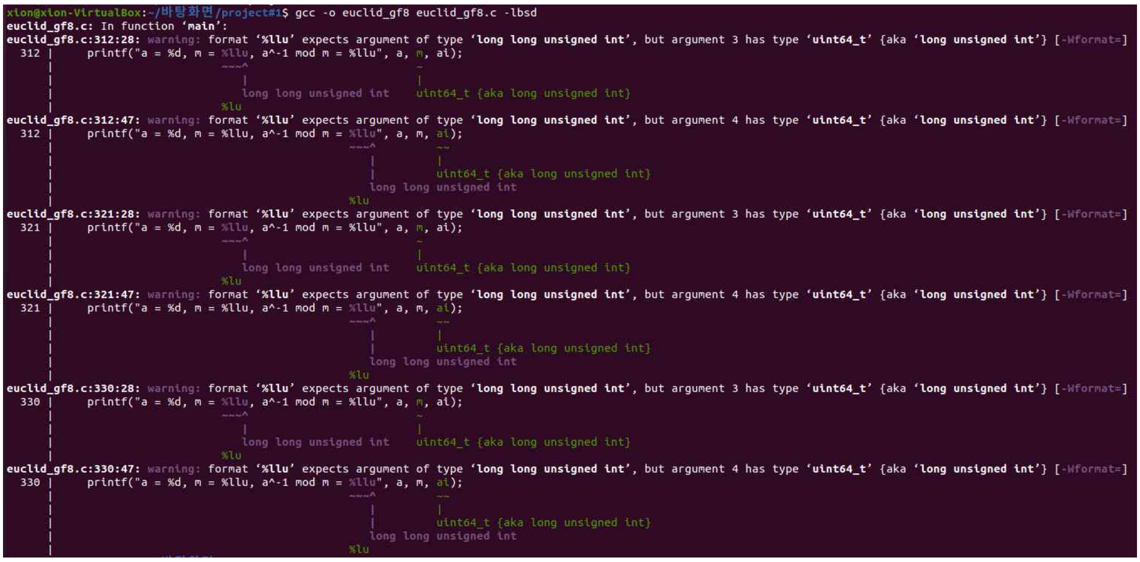
⑥ gf8_pow

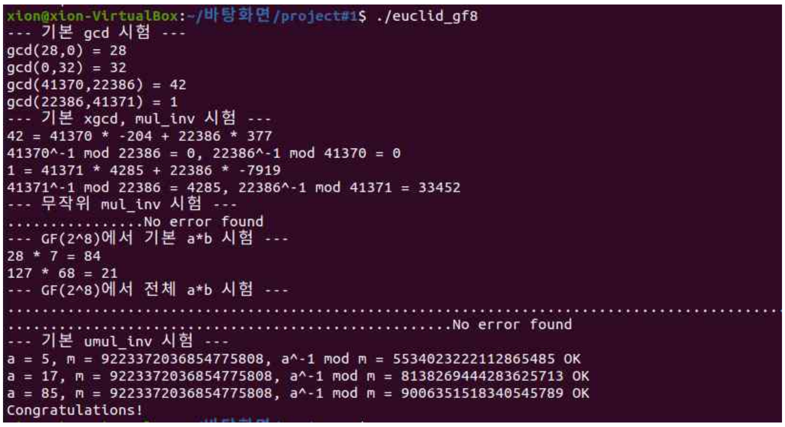
```c
uint8_t gf8_pow(uint8_t a, uint8_t b)
{
    uint8_t r = 1;

    while (b>0){
        if (b&1) r = gf8_mul(r, a);
        b = b >> 1;
        a = gf8_mul(a,a);
    }
    return r;
}
```

$a^b = a \times a \times a \times \ldots \times a$ 인 것을 감안, square multiplication을 변형하여 a의 값을 gf8_mul()을 통해 a * a modulo m(x)로 갱신 후 갱신된 a를 다시 gf8_mul()을 통해 r * a modulo m(x)를 적용하는 loop를 반복. 최종적으로 loop가 끝난 뒤 r을 return하여 gf8_pow()를 구현.

2. compile





test를 위한 main 함수 내부에서 자료형에 대한 issue가 발생했지만 main 함수를 수정하지 말라는 project 안내사항에 따라 무시하고 compile을 진행하고 실행하여 결과값이 정상적으로 출력되었다.

3. result

모든 값에 대한 결과값이 주어진 예상결과.txt의 결과값과 동일하게 출력되었다. 결과에 대한 자료는 터미널 명령어 '>'를 통해 euclid_gf8.txt 에 저장하여 별도로 제출한다.

4. feedback

xgcd(), mul_inv(), umul_inv()의 경우 d, x, y 변수에 대해 각각 두 가지의 변수 (d0, d1 / x0, x1 / y0, y1)들만 사용해서 구현할 수 있었다는 사실을 강의를 통해 확인했다. 하지만 직접 구현을 함에 있어서 세 가지의 변수 (d0, d1, d2 / x0, x1, x2 / y0, y1, y2)들을 사용하는 것이 본인에게 더 직관적으로 이해가 잘 되었기에 수정하지 않고 그대로 세 가지의 변수를 사용했다.

unsigned 64 비트 정수를 입력으로 받는 **umul_inv()**의 경우 unsigned / signed 자료형의 차이에 대해 잘 알고있다면 쉽게 접근할 수 있는 문제였지만 이 두 자료형의 차이점을 생각해야한다는 접근을 떠올리는 단계까지 많은 시간을 소모했다.

gf8_pow()의 경우 $a^b = a \times a \times a \times \ldots \times a$ 라는 것은 파악했으나 square multiplication을 변형하기보다 recursive한 접근법을 먼저 시도하여 결국 잘 구현되지 않았고 여기서 많은 시간을 소모했다.