# 정보보호론 Cryptography HW2 12215227 김나현

**Task 3.1 Private key**

**① p, q, e 10진수로 변환하기**

```c
#include <stdio.h>
#include <openssl/bn.h>

int main() {

        const char *p_str = "F7E75FDC469067FFDC4E847C51F452DF";
        const char *q_str = "E85CED54AF57E53E092113E62F436F4F";
        const char *e_str = "0D88C3";

        BIGNUM *p = BN_new();
        BIGNUM *q = BN_new();
        BIGNUM *e = BN_new();

        BN_hex2bn(&p, p_str);
        BN_hex2bn(&q, q_str);
        BN_hex2bn(&e, e_str);

        char *dec_p = BN_bn2dec(p);
        char *dec_q = BN_bn2dec(q);
        char *dec_e = BN_bn2dec(e);

        printf("p(decimal) : %s\n", dec_p);
        printf("q(decimal) : %s\n", dec_q);
        printf("e(decimal) : %s\n", dec_e);

        OPENSSL_free(dec_p);
        OPENSSL_free(dec_q);
        OPENSSL_free(dec_e);


        BN_free(p);
        BN_free(q);
        BN_free(e);

        return 0;
}
~
```

문제에서 주어진 p, q, e는 이미 매우 큰 수로 16진수 문자열이다. 큰 수에 대한 연산을 정확하게 수행하기 위해 BIGNUM을 이용하여 16진수 문자열을 큰 수(임의 정밀도 정수) 로 변환했다. 코드에 보이는 BN_hex2bn() 함수가 16진수를 BIGNUM 객체로 변환시킨다. BIGNUM 객체에 저장된 큰 수를 BN_bn2dec() 함수를 사용하여 10진수 문자열로 변환하여 사람이 이해하고 읽을 수 있는 형식으로 숫자를 출력하였다.

```
annynahyun2002@instance-20251118-052056:~$ vim task1.c
annynahyun2002@instance-20251118-052056:~$ gcc -o task1 task1.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task1
p(decimal) : 329520679814142392965336341297134588639
q(decimal) : 308863399973593539130925275387286220623
e(decimal) : 886979
```

**p, q, e값으로 위와 같이 변환 결과가 출력되었다.**

**② n = p * q**

```c
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
        char *number_str = BN_bn2hex(a);
        printf("%s %s\n", msg, number_str);
        OPENSSL_free(number_str);
}

int main() {
    BN_CTX *ctx = BN_CTX_new();

    const char *p_str = "F7E75FDC469067FFDC4E847C51F452DF";
    const char *q_str = "E85CED54AF57E53E092113E62F436F4F";
    const char *e_str = "0D88C3";

    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();

    BN_hex2bn(&p, p_str);
    BN_hex2bn(&q, q_str);
    BN_hex2bn(&e, e_str);

    char *dec_p = BN_bn2dec(p);
    char *dec_q = BN_bn2dec(q);
    char *dec_e = BN_bn2dec(e);

    printf("p(decimal) : %s\n", dec_p);
    printf("q(decimal) : %s\n", dec_q);
    printf("e(decimal) : %s\n", dec_e);

    BN_mul(n, p, q, ctx);
    printBN("n = p * q =", n);

    OPENSSL_free(dec_p);
    OPENSSL_free(dec_q);
    OPENSSL_free(dec_e);

    BN_free(p);
    BN_free(q);
    BN_free(e);
```

```
annynahyun2002@instance-20251118-052056:~$ vim task1.c
annynahyun2002@instance-20251118-052056:~$ gcc -o task1 task1.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task1
p(decimal) : 329520679814142392965336341297134588639
q(decimal) : 308863399973593539130925275387286220623
e(decimal) : 886979
n = p * q = E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
annynahyun2002@instance-20251118-052056:~$
```

**BIGNUM 구조체와 함께 출력에 사용할 메시지 문자열을 입력하고 메시지와 함께 해당 BIGNUM 값을 16진수 문자열 형태로 출력하는 printBN 함수를 작성했다. p * q를 계산한 n 값을 printBN 함수로 출력하였다.**

**③ (p-1)(q-1) 계산하기**

```
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
        char *number_str = BN_bn2hex(a);
        printf("%s %s\n", msg, number_str);
        OPENSSL_free(number_str);
}

int main() {
    BN_CTX *ctx = BN_CTX_new();

    const char *p_str = "F7E75FDC469067FFDC4E847C51F452DF";
    const char *q_str = "E85CED54AF57E53E092113E62F436F4F";
    const char *e_str = "0D88C3";

    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *p_q = BN_new();

    BN_hex2bn(&p, p_str);
    BN_hex2bn(&q, q_str);
    BN_hex2bn(&e, e_str);

    char *dec_p = BN_bn2dec(p);
    char *dec_q = BN_bn2dec(q);
    char *dec_e = BN_bn2dec(e);

    printf("p(decimal) : %s\n", dec_p);
    printf("q(decimal) : %s\n", dec_q);
    printf("e(decimal) : %s\n", dec_e);

    BN_mul(n, p, q, ctx);
    printBN("n = p * q =", n);

    BIGNUM *p_minus_1 = BN_dup(p);
    BIGNUM *q_minus_1 = BN_dup(q);

    BN_sub(p_minus_1, p_minus_1, BN_value_one());
    BN_sub(q_minus_1, q_minus_1, BN_value_one());

    BN_mul(p_q, p_minus_1, q_minus_1, ctx);
```

```
  printBN("(p-1)*(q-1) = ", p_q);

  OPENSSL_free(dec_p);
  OPENSSL_free(dec_q);
  OPENSSL_free(dec_e);

  BN_free(p);
  BN_free(q);
  BN_free(e);
  BN_free(p_minus_1);
  BN_free(q_minus_1);
  BN_CTX_free(ctx);

  return 0;
```

**BN_dup()를 이용해 p와 q의 복사본을 각각 p_minus_1과 q_minus_1에 저장한 후, BN_sub() 함수를 사용해 두 값에서 1을 뺀다. p_minus_1에는 p-1, q_minus_1에 는 q-1이 저장된다. BN_mul() 함수를 이용해 (p-1)과 (q-1)을 곱한 결과를 p_q에 저 장하고 printBN()을 호출해 그 결과를 16진수 형태로 출력한다. 모든 작업이 끝난 후에는 메모리 누수를 방지하기 위해 할당된 BIGNUM 객체와 문자열 메모리를 해제한다.**

```
annynahyun2002@instance-20251118-052056:~$ vim task1.c
annynahyun2002@instance-20251118-052056:~$ gcc -o task1 task1.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task1
p(decimal) : 32952067981414239296533634129713458863 9
q(decimal) : 30886339997359353913092527538728622062 3
e(decimal) : 886979
n = p * q = E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
 (p-1)*(q-1) =  E103ABD94892E3E74AFD724BF28E78348D52298BD687C44DEB3A81065A7981A4
annynahyun2002@instance-20251118-052056:~$
```

**(p-1)(q-1)의 값이 출력되었다.**

④ d 값 구하기

```c
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
        char *number_str = BN_bn2hex(a);
        printf("%s %s\n", msg, number_str);
        OPENSSL_free(number_str);
}

int main() {
    BN_CTX *ctx = BN_CTX_new();

    const char *p_str = "F7E75FDC469067FFDC4E847C51F452DF";
    const char *q_str = "E85CED54AF57E53E092113E62F436F4F";
    const char *e_str = "0D88C3";

    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *p_q = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *gcd = BN_new();

    BN_hex2bn(&p, p_str);
    BN_hex2bn(&q, q_str);
    BN_hex2bn(&e, e_str);

    char *dec_p = BN_bn2dec(p);
    char *dec_q = BN_bn2dec(q);
    char *dec_e = BN_bn2dec(e);

    printf("p(decimal) : %s\n", dec_p);
    printf("q(decimal) : %s\n", dec_q);
    printf("e(decimal) : %s\n", dec_e);

    BN_mul(n, p, q, ctx);
    printBN("n = p * q =", n);

    BIGNUM *p_minus_1 = BN_dup(p);
    BIGNUM *q_minus_1 = BN_dup(q);

    BN_sub(p_minus_1, p_minus_1, BN_value_one());
    BN_sub(q_minus_1, q_minus_1, BN_value_one());
```

```
BN_mul(p_q, p_minus_1, q_minus_1, ctx);

BN_gcd(gcd, e, p_q, ctx);
printBN("gcd: ", gcd);

BN_mod_inverse(d, e, p_q, ctx);

printBN("(p-1)*(q-1) = ", p_q);
printBN("d = ", d);

OPENSSL_free(dec_p);
OPENSSL_free(dec_q);
OPENSSL_free(dec_e);

BN_free(p);
BN_free(q);
BN_free(e);
BN_free(p_minus_1);
BN_free(q_minus_1);
BN_CTX_free(ctx);

return 0;
```

**주어진 e와 (p-1)(q-1) 값을 기반으로 d를 구한다. BN_gcd를 사용해 e와 (p-1)(q-1)의 최 대공약수를 계산하여 두 값이 서로소임을 확인한다. 이것은 (p-1)(q-1)이 올바르게 계산되었는지 검증하는 용도로도 사용된다. BN_mod_inverse 함수를 통해 (e * d) mod (p-1)(q-1) = 1을 만족하는 d 값을 구한다. BN_mod_inverse(d, e, p_q, ctx)는 e의 모듈로서의 역원을 계산해 d에 저장한다.**

**프로그램은 계산된 (p-1)(q-1)과 d 값을 출력하고 이는 RSA에서 비밀 키를 결정하는 중요한 과정이다. 최종적으로 모든 동적 메모리는 BN_free()와 BN_CTX_free()를 사용해 해제되어 메모리 누수를 방지한다.**

```
annynahyun2002@instance-20251118-052056:~$ vim task1.c
annynahyun2002@instance-20251118-052056:~$ gcc -o task1 task1.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task1
p(decimal) : 329520679814142392965336341297134588639
q(decimal) : 308863399973593539130925275387286220623
e(decimal) : 886979
n = p * q = E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
gcd:  01
 (p-1)*(q-1) =  E103ABD94892E3E74AFD724BF28E78348D52298BD687C44DEB3A81065A7981A4
d =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
annynahyun2002@instance-20251118-052056:~$
```

**gcd와 d값이 출력되었다. 공개키 암호화에 쓰일 개인 키와 공개 키의 쌍을 얻었다.**

**Task 3.2 메시지를 암호화하기**

```
annynahyun2002@instance-20251118-052056:~$ python3 -c 'print("A top secret!".encode("utf-8").hex())'
4120746f702073656372657421
annynahyun2002@instance-20251118-052056:~$
```

**평문 M을 주어진 명령문을 사용해서 구했다.**

```c
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();

    const char *n_str = "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5";
    const char *e_str = "010001";
    const char *M_str = "4120746F702073656372657421";
    const char *d_str = "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D";

    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *C = BN_new();

    BN_hex2bn(&n, n_str);
    BN_hex2bn(&e, e_str);
    BN_hex2bn(&M, M_str);
    BN_hex2bn(&d, d_str);

    BN_mod_exp(C, M, e, n, ctx);

    printBN("C = ", C);

    BN_free(n);
    BN_free(e);
    BN_free(M);
    BN_free(d);
    BN_CTX_free(ctx);

    return 0;
}
~
```

**BN_mod_exp 함수를 사용해 RSA 암호화 과정인 $M^e \bmod n$을 수행한다. 메시지 $M$, 공개 지수 $e$, 모듈러스 $n$을 이용해 암호문 $C$ 계산하였다. 또한 printBN 함수를 통해 결과를 출력했다.**

```
annynahyun2002@instance-20251118-052056:~$ gcc -o task2 task2.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task2
C =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
annynahyun2002@instance-20251118-052056:~$ 
```

**위의 사진을 통해 $C$ 값을 알 수 있다.**

### Task 3.3 암호문을 복호화 하기

```c
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();

    const char *n_str = "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5";
    const char *e_str = "010001";
    const char *d_str = "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D";
    const char *C_str = "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F";

    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *C = BN_new();

    BN_hex2bn(&n, n_str);
    BN_hex2bn(&e, e_str);
    BN_hex2bn(&d, d_str);
    BN_hex2bn(&C, C_str);

    BN_mod_exp(M, C, d, n, ctx);

    printBN("M = ", M);

    BN_free(n);
    BN_free(e);
    BN_free(M);
    BN_free(d);
    BN_CTX_free(ctx);

    return 0;
}
```

```
annynahyun2002@instance-20251118-052056:~$ gcc -o task3 task3.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task3
M =  50617373776F726420697320646565573
```

**BN_mod_exp를 사용해 $C^d \mod n$ 연산을 수행하여 암호문 $C$를 복호화하고 결과를 평문 $M$으로 출력한다.**

```
annynahyun2002@instance-20251118-052056:~$ python3 -c "print(bytes.fromhex('50617373776F72642069732064656573').decode('utf-8'))"
Password is dees
```

**출력된 평문을 ASCII 문자열로 변환한 결과 Password is dees 라는 결과가 출력되었다.**

### Task 3.4 전자 서명

```
annynahyun2002@instance-20251118-052056:~$ python3 -c 'print("I owe you $2000.".encode("utf-8").hex())'
49206f776520796f752024323030302e
```

**주어진 메시지를 16진수 문자열로 변환했다.**

```c
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();

    const char *n_str = "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5";
    const char *e_str = "010001";
    const char *M_str = "49206f776520796f752024323030302e";
    const char *d_str = "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D";

    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *C = BN_new();

    BN_hex2bn(&n, n_str);
    BN_hex2bn(&e, e_str);
    BN_hex2bn(&M, M_str);
    BN_hex2bn(&d, d_str);

    BN_mod_exp(C, M, e, n, ctx);

    printBN("C = ", C);

    BN_free(n);
    BN_free(e);
    BN_free(M);
    BN_free(d);
    BN_free(C);
    BN_CTX_free(ctx);

    return 0;
}
~
```

```
annynahyun2002@instance-20251118-052056:~$ gcc -o task4 task4.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task4
C =  3A759CBF53901AC41373EEC603955A8E6AF8D3BCD5E9F6DD62C873CBB675051E
```

**코드에 M 값을 넣어서 컴파일 했더니 C 값이 출력되었다.**

```
annynahyun2002@instance-20251118-052056:~$ python3 -c 'print("I owe you $3000.".encode("utf-8").hex())'
49206f776520796f752024333030302e
annynahyun2002@instance-20251118-052056:~$
```

**이번에는 문구를 $2000에서 $3000으로만 바꿔서 구한 M 값을 넣어 다시 컴파일 해보았다.**

```c
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();

    const char *n_str = "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5";
    const char *e_str = "010001";
    const char *M_str = "49206f776520796f75202433303030302e";
    const char *d_str = "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D";

    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *C = BN_new();

    BN_hex2bn(&n, n_str);
    BN_hex2bn(&e, e_str);
    BN_hex2bn(&M, M_str);
    BN_hex2bn(&d, d_str);

    BN_mod_exp(C, M, e, n, ctx);

    printBN("C = ", C);

    BN_free(n);
    BN_free(e);
    BN_free(M);
    BN_free(d);
    BN_free(C);
    BN_CTX_free(ctx);

    return 0;
}
~
~
```

```
annynahyun2002@instance-20251118-052056:~$ gcc -o task4 task4.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task4
C =  D06908047527906C724937169FA68CE0AC442FEB99D1880438D331A88F44B074
```

**C값을 비교해보자면**

```
C =  3A759CBF53901AC41373EEC603955A8E6AF8D3BCD5E9F6DD62C873CBB675051E
```

```
C =  D06908047527906C724937169FA68CE0AC442FEB99D1880438D331A88F44B074
```

**위가 I love you $2000이 적혀 있는 전자 문서에 대한 서명이다. 아래가 I love you $3000이 적혀 있는 전자 문서에 대한 서명이다. 두 문자열을 16진수 문자열로 변환했을 때의 값은 거의 유사했다. 하지만 암호화를 통해 최종적으로 출력된 결 과를 통해 알 수 있듯이 비슷한 문장을 암호화 했다는 것은 전혀 알 수 없을 만큼 다른 값이 출력되었다. 즉, 비슷한 평문에 대한 여러 암호문이 주어진다 하더라도 평문을 추측하는 일이 여전히 어렵다는 것을 알 수 있었다.**

**Task 3.5 서명 검증**

```c
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();

    const char *n_str = "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115";
    const char *e_str = "010001";
    const char *C_str = "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F";

    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *C = BN_new();

    BN_hex2bn(&n, n_str);
    BN_hex2bn(&e, e_str);
    BN_hex2bn(&C, C_str);

    BN_mod_exp(M, C, e, n, ctx);

    printBN("M = ", M);

    BN_free(n);
    BN_free(e);
    BN_free(C);
    BN_CTX_free(ctx);

    return 0;
}
```

**$C$는 전자서명 S를 가리킨다. 코드에 알맞은 값들을 넣어 컴파일 했다.**

```
annynahyun2002@instance-20251118-052056:~$ gcc -o task5 task5.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task5
M =  4C61756E63682061206D697373696C652E
annynahyun2002@instance-20251118-052056:~$ python3 -c 'print("Launch a missile.".encode("utf-8").hex())'
4c61756e63682061206d697373696c652e
annynahyun2002@instance-20251118-052056:~$
```

**공개키로 복호화된 M 값이 출력이 되었다. 그리고 "Launch a missile."에 대한 messege digest를 했더니 공개키로 복호화된 M 값과 동일한 것을 통해 메시지에 대한 서명이 Alice의 것임을 알 수있다.**

```c
#include <stdio.h>
#include <openssl/bn.h>

void printBN(char *msg, BIGNUM *a)
{
    char *number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main()
{
    BN_CTX *ctx = BN_CTX_new();

    const char *n_str = "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115";
    const char *e_str = "010001";
    const char *C_str = "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F";

    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *M = BN_new();
    BIGNUM *C = BN_new();

    BN_hex2bn(&n, n_str);
    BN_hex2bn(&e, e_str);
    BN_hex2bn(&C, C_str);

    BN_mod_exp(M, C, e, n, ctx);

    printBN("M = ", M);

    BN_free(n);
    BN_free(e);
    BN_free(C);
    BN_CTX_free(ctx);

    return 0;
}
```

**만약 서명이 손상되어 마지막 byte가 2F->3F로 바뀌었을 경우를 컴파일 했다.**

```
annynahyun2002@instance-20251118-052056:~$ gcc -o task5 task5.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task5
M =  91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
annynahyun2002@instance-20251118-052056:~$
```

**출력된 M의 값이 Launch a missile.”에 대한 messege digest 값과 전혀 다른 값임을 알 수 있다. 따라서 서명이 유효하지 않게 되어버린다.**

## Task 3.6 인증서 검증

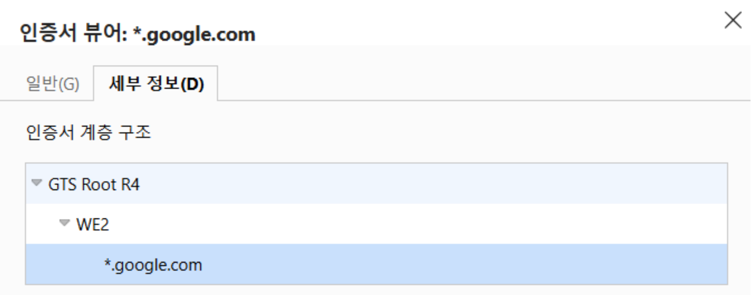
### Step 1.

```
annynahyun2002@instance-20251118-052056:~$  openssl s_client -connect www.google.com:443 -showcerts
Connecting to 64.233.181.106
CONNECTED(00000003)
depth=2 C=US, O=Google Trust Services LLC, CN=GTS Root R1
verify return:1
depth=1 C=US, O=Google Trust Services, CN=WR2
verify return:1
depth=0 CN=www.google.com
verify return:1
---
Certificate chain
 0 s:CN=www.google.com
   i:C=US, O=Google Trust Services, CN=WR2
   a:PKEY: EC, (prime256v1); sigalg: sha256WithRSAEncryption
   v:NotBefore: Oct 27 08:35:45 2025 GMT; NotAfter: Jan 19 08:35:44 2026 GMT
-----BEGIN CERTIFICATE-----
MIIEVjCCAz6gAwIBAgIQK4CoSsJ9L74K2cYV+NGbrDANBgkqhkiG9w0BAQsFADA7
MQswCQYDVQQGEwJVUzEeMBwGA1UEChMVR29vZ2xlIFRydXN0IFNlcnZpY2VzMQww
CgYDVQQDEwNXUjIwHhcNMjUxMDI3MDgzNTQ1WhcNMjYwMTE5MDgzNTQ0WjAZMRcw
FQYDVQQDEw53d3cuZ29vZ2xlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IA
BK4Gp/CaYtzmaPRrpEBoC7QZzUjgRd7mHSdUCTqWg3mU53V/Q96Tus1uR4T8xzMN
B1pQFIgXekW9HVEWhL0Aa36jggJBMIICPTAOBgNVHQ8BAf8EBAMCB4AwEwYDVR0l
BAwwCgYIKwYBBQUHAwEwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQUQNlhWVquExT5
MH6naM3bcQ+eO04wHwYDVR0jBBgwFoAU3hse7XkV1D43JMMhu+w0OW1CsjAwWAYI
KwYBBQUHAQEETDBKMCEGCCsGAQUFBzABhhVodHRwOi8vby5wa2kuZ29vZy93cjIw
JQYIKwYBBQUHMAKGGWh0dHA6Ly9pLnBraS5nb29nL3dyMi5jcnQwGQYDVR0RBBIw
EIIOd3d3Lmdvb2dsZS5jb20wEwYDVR0gBAwwCjAIBgZngQwBAgEwNgYDVR0fBC8w
LTAroCmgJ4YlaHR0cDovL2MucGtpLmdvb2cvd3IyLzc1cjRaeUEzdkEwLmNybDCC
AQQGCisGAQQB1nkCBAIEgfUEgfIA8AB2AA5XlLzzrqk+MxssmQez95Dfm8I9cTIl
3SGpJaxhxU4hAAABmiUGEj0AAAQDAEcwRQIhAJ1WwLwr4vsmo9oQP70urWKgoJZJ
UtyaM9RbkP4S+0qeAiANa8uFRuehigPskngJhXIIbtI/UCJsErzkYU5Ir0WffAB2
AA3h8jAr0w3BQGISCepVLvxHdHyx1+kw7w5CHrR+Tqo0AAABmiUGEkwAAAQDAEcw
RQIhAOLLAsnaYy2Y38/H6Q0RoGgevbwsaXxqaYbBZg8WCryiAiAX0kNxFUpcIt87
0NJKneiVNqH5nZTGPqOWSqR69yCVAjANBgkqhkiG9w0BAQsFAAOCAQEAEDLvfqrk
JKYpuS4F/5uvDCrpBlXxlOpxwTgDAXWag+MX+5j3bNNnNCeXI3LYD037QZGm0nQC
RA7OGqUaw0zDvnObK7HirMXu5Hf5W5c9FAYL+gItlxn+J/kKFUQ3kxLCNw5FKQE1
I42bKNoUYsCB4SobrQIAhL0sv7hOeuDGbiMvVsEHNOM0fEqUK+0hHsG0oEAfTHl5
KJ7H4j9r1RUteF1hMaNajbA+dmvyCPyLRsrwLwDUOJ3s31BGR240Eqyl23itvXUK
aSpTHRmhdKIcwM/NpED+UqEkjqAtxWqWEXrzMujLhti3nsaLDMEuVtnIYoVvWYOP
TAHs2/xUrmkqkg==
-----END CERTIFICATE-----
```

```
 1 s:C=US, O=Google Trust Services, CN=WR2
   i:C=US, O=Google Trust Services LLC, CN=GTS Root R1
   a:PKEY: RSA, 2048 (bit); sigalg: sha256WithRSAEncryption
   v:NotBefore: Dec 13 09:00:00 2023 GMT; NotAfter: Feb 20 14:00:00 2029 GMT
-----BEGIN CERTIFICATE-----
MIIFCzCCAvOgAwIBAgIQf/AFoHxM3tEArZ1mpRB7mDANBgkqhkiG9w0BAQsFADBH
MQswCQYDVQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZpY2VzIExM
QzEUMBIGA1UEAxMLR1RTIFJvb3QgUjEwHhcNMjMxMjEzMDkwMDAwWhcNMjkwMjIw
MTQwMDAwWjA7MQswCQYDVQQGEwJVUzEeMBwGA1UEChMVR29vZ2xlIFRydXN0IFNl
cnZpY2VzMQwwCgYDVQQDEwNXUjIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQCp/5x/RR5wqFOfytnlDd5GV1d9vI+aWqxG8YSau5HbyfsvAfuSCQAWXqAc
+MGr+XgvSszYhaLYWTwO0xj7sfUkDSbutltkdnwUxy96zqhMt/TZCPzfhyM1IKji
aeKMTj+xWfpgoh6zySBTGYLKNlNtYE3pAJH8do1cCA8Kwtzxc2vFE24KT3rC8gIc
LrRjg9ox9i11MLL7q8Ju26nADrn5Z9TDJVd06wW06Y613ijNzHoU5HEDy01hLmFX
xRmpC5iEGuh5KdmyjS//V2pm4M6rlagplmNwEmceOuHbsCFx13ye/aoXbv4r+zgX
FNFmp6+atXDMyGOBOozAKql2N87jAgMBAAGjgf4wgfswDgYDVR0PAQH/BAQDAgGG
MB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjASBgNVHRMBAf8ECDAGAQH/
AgEAMB0GA1UdDgQWBBTeGx7teRXUPjckwyG77DQ5bUKyMDAfBgNVHSMEGDAWgBTk
rysmcRorSCeFL1JmLO/wiRNxPjA0BggrBgEFBQcBAQQoMCYwJAYIKwYBBQUHMAKG
GGh0dHA6Ly9pLnBraS5nb29nL3IxLmNydDArBgNVHR8EJDAiMCCgHqAchhpodHRw
Oi8vYy5wa2kuZ29vZy9yL3IxLmNybDATBgNVHSAEDDAKMAgGBmeBDAECATANBgkq
hkiG9w0BAQsFAAOCAgEARXWL5R87RBOWGqtY8TXJbz3S0DNKhjO6V1FP7sQ02hYS
TL8Tnw3UVOlIecAwPJQl8hr0ujKUtjNyC4XuCRElNJThb0Lbgpt7fyqaqf9/qdLe
SiDLs/sDA7j4BwXaWZIvGEaYzq9yviQmsR4ATb0IrZNBRAq7x9UBhb+TV+PfdBJT
DhEl05vc3ssnbrPCuTNiOcLgNeFbpwkuGcuRKnZc8d/KI4RApW//mkHgte8y0YWu
ryUJ8GLFbsLIbjL9uNrizkqRSvOFVU6xddZIMy9vhNkSXJ/UcZhjJY1pXAprffJB
vei7j+Qi151lRehMCofa6WBmiA4fx+FOVsV2/7R6V2nyAiIJJkEd2nSi5SnzxJrl
Xdaqev3htytmOPvoKWa676ATL/hzfvDaQBEcXd2Ppvy+275W+DKcH0FBbX62xevG
iza3F4ydzxl6NJ8hk8R+dDXSqv1MbRT1ybB5W0k8878XSOjvmiYTDIfyc9acxVJr
Y/cykHipa+te1pOhv7wYPYtZ9orGBV5SGOJm4NrB3K1aJar0RfzxC3ikr7Dyc6Qw
qDTBU39CluVIQeuQRgwG3MuSxl7zRERDRilGoKb8uY45JzmxWuKxrfwT/478JuHU
/oTxUFqOl2stKnn7QGTq8z29W+GgBLCXSBxC9epaHM0myFH/FJlniXJfHeytWt0=
-----END CERTIFICATE-----
```

```
 2 s:C=US, O=Google Trust Services LLC, CN=GTS Root R1
   i:C=BE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign Root CA
   a:PKEY: RSA, 4096 (bit); sigalg: sha256WithRSAEncryption
   v:NotBefore: Jun 19 00:00:42 2020 GMT; NotAfter: Jan 28 00:00:42 2028 GMT
-----BEGIN CERTIFICATE-----
MIIFYjCCBEqgAwIBAgIQd70NbNs2+RrqIQ/E8FjTDTANBgkqhkiG9w0BAQsFADBX
MQswCQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBudi1zYTEQMA4GA1UE
CxMHUm9vdCBDQTEbMBkGA1UEAxMSR2xvYmFsU2lnbiBSb290IENBMB4XDTIwMDYx
OTAwMDA0MloXDTI4MDEyODAwMDA0MlowRzELMAkGA1UEBhMCVVMxIjAgBgNVBAoT
GUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBMTEMxFDASBgNVBAMTC0dUUyBSb290IFIx
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAthECix7joXebO9y/lD63
ladAPKH9gvl9MgaCcfb2jH/76Nu8ai6Xl6OMS/kr9rH5zoQdsfnFl97vufKj6bwS
iV6nqlKr+CMny6SxnGPb15l+8Ape62im9MZaRw1NEDPjTrETo8gYbEvs/AmQ351k
KSUjB6G00j0uYODP0gmHu81I8E3CwnqIiru6z1kZ1q+PsAewnjHxgsHA3y6mbWwZ
DrXYfiYaRQM9sHmklCitD38m5agI/pboPGiUU+6DOogrFZYJsuB6jC511pzrp1Zk
j5ZPaK49l8KEj8C8QMALXL32h7M1bKwYUH+E4EzNktMg6TO8UpmvMrUpsyUqtEj5
cuHKZPfmghCN6J3Cioj6OGaK/GP5Afl4/Xtcd/p2h/rs37EOeZVXtL0m79YB0esW
CruOC7XFxYpVq9Os6pFLKcwZpDIlTirxZUTQAs6qzkm06p98g7BAe+dDq6dso499
iYH6TKX/1Y7DzkvgtdizjkXPdsDtQCv9Uw+wp9U7DbGKogPeMa3Md+pvez7W35Ei
Eua++tgy/BBjFFFy3l3WFpO9KWgz7zpm7AeKJt8T11dleCfeXkkUAKIAf5qoIbap
sZWwpbkNFhHax2xIPEDgfg1azVY80ZcFuctL7TlLnMQ/0lUTbiSw1nH69MG6zO0b
9f6BQdgAmD06yK56mDcYBZUCAwEAAaOCATgwggE0MA4GA1UdDwEB/wQEAwIBhjAP
BgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBTkrysmcRorSCeFL1JmLO/wiRNxPjAf
BgNVHSMEGDAWgBRge2YaRQ2XyolQL30EzTSo//z9SzBgBggrBgEFBQcBAQRUMFIw
JQYIKwYBBQUHMAGGGWh0dHA6Ly9vY3NwLnBraS5nb29nL2dzcjEwKQYIKwYBBQUH
MAKGHWh0dHA6Ly9wa2kuZ29vZy9nc3IxL2dzcjEuY3J0MDIGA1UdHwQrMCkwJ6Al
oCOGIWh0dHA6Ly9jcmwucGtpLmdvb2cvZ3NyMS9nc3IxLmNybDA7BgNVHSAENDAy
MAgGBmeBDAECATAIBgZngQwBAgIwDQYLKwYBBAHWeQIFAwIwDQYLKwYBBAHWeQIF
AwMwDQYJKoZIhvcNAQELBQADggEBADSkHrEoo9C0dhemMXoh6dFSPsjbdBZBiLg9
NR3t5P+T4Vxfq7vqfM/b5A3Ri1fyJm9bvhdGaJQ3b2t6yMAYN/olUazsaL+yyEn9
WprKASOshIArAoyZl+tJaox118fessmXn1hIVw41oeQa1v1vg4Fv74zPl6/AhSrw
9U5pCZEt4Wi4wStz6dTZ/CLANx8LZh1J7QJVj2fhMtfTJr9w4z30Z209fOU0iOMy
+qduBmpvvYuR7hZL6Dupszfnw0Skfths18dG9ZKb59UhvmaSGZRVbNQpsg3BZlvi
d0lIKO2d1xozclOzgjXPYovJJIultzkMu34qQb9Sz/yilrbCgj8=
-----END CERTIFICATE-----
```

**www.google.co.kr의 인증서를 다운로드 받았다. 총 3개의 인증서를 확인할 수 있다.**




**GTS Root R4는 Root CA로 신뢰 체인의 최상위에 있으며, 모든 인증서 신뢰의 출발점 역할을 한다. WE2는 Intermediate CA(중간 인증서)로, Root CA와 최종 엔드 엔터티 인증서 사이의 중개자 역할을 한다. .google.com은 End-Entity 인증서(0번 인증서)로, 실제 구글 웹사이트 대한 서버 인증서이다. 이 세 인증서는 순서대로 연결되어 하나의 신뢰 체인을 구성하며, 각 상위 인증서의 서명을 통해 하위 인증서의 유효성이 검증된다.**

```
annynahyun2002@instance-20251118-052056:~$ nano c1.pem
annynahyun2002@instance-20251118-052056:~$ nano c2.pem
annynahyun2002@instance-20251118-052056:~$
```

```
  GNU nano 8.4                                                   c1.pem
```


```
-----BEGIN CERTIFICATE-----
MIIFCzCCAvOgAwIBAgIQf/AFoHxM3tEArZ1mpRB7mDANBgkqhkiG9w0BAQsFADBH
MQswCQYDVQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZpY2VzIExM
QzEUMBIGA1UEAxMLR1RTIFJvb3QgUjEwHhcNMjMxMjEzMDkwMDAwWhcNMjkwMjIw
MTQwMDAwWjA7MQswCQYDVQQGEwJVUzEeMBwGA1UEChMVR29vZ2xlIFRydXN0IFNl
cnZpY2VzMQwwCgYDVQQDEwNXUjIwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEK
AoIBAQCp/5x/RR5wqFOfytnlDd5GV1d9vI+aWqxG8YSau5HbyfsvAfuSCQAWXqAc
+MGr+XgvSszYhaLYWTwO0xj7sfUkDSbutltkdnwUxy96zqhMt/TZCPzfhyM1IKji
aeKMTj+xWfpgoh6zySBTGYLKNlNtYE3pAJH8do1cCA8Kwtzxc2vFE24KT3rC8gIc
LrRjg9ox9i11MLL7q8Ju26nADrn5Z9TDJVd06wW06Y613ijNzHoU5HEDy01hLmFX
xRmpC5iEGuh5KdmyjS//V2pm4M6rlagplmNwEmceOuHbsCFx13ye/aoXbv4r+zgX
FNFmp6+atXDMyGOBOozAKql2N87jAgMBAAGjgf4wgfswDgYDVR0PAQH/BAQDAgGG
MB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjASBgNVHRMBAf8ECDAGAQH/
AgEAMB0GA1UdDgQWBBTeGx7teRXUPjckwyG77DQ5bUKyMDAfBgNVHSMEGDAWgBTk
rysmcRorSCeFL1JmLO/wiRNxPjA0BggrBgEFBQcBAQQoMCYwJAYIKwYBBQUHMAKG
GGh0dHA6Ly9pLnBraS5nb29nL3IxLmNydDArBgNVHR8EJDAiMCCgHqAchhpodHRw
Oi8vYy5wa2kuZ29vZy9yL3IxLmNybDATBgNVHSAEDDAKMAgGBmeBDAECATANBgkq
hkiG9w0BAQsFAAOCAgEARXWL5R87RBOWGqtY8TXJbz3S0DNKhjO6V1FP7sQ02hYS
TL8Tnw3UVOlIecAwPJQl8hr0ujKUtjNyC4XuCRElNJThb0Lbgpt7fyqaqf9/qdLe
SiDLs/sDA7j4BwXaWZIvGEaYzq9yviQmsR4ATb0IrZNBRAq7x9UBhb+TV+PfdBJT
DhEl05vc3ssnbrPCuTNiOcLgNeFbpwkuGcuRKnZc8d/KI4RApW//mkHgte8y0YWu
ryUJ8GLFbsLIbjL9uNrizkqRSvOFVU6xddZIMy9vhNkSXJ/UcZhjJY1pXAprffJB
vei7j+Qi151lRehMCofa6WBmiA4fx+FOVsV2/7R6V2nyAiIJJkEd2nSi5SnzxJrl
Xdaqev3htytmOPvoKWa676ATL/hzfvDaQBEcXd2Ppvy+275W+DKcH0FBbX62xevG
iza3F4ydzxl6NJ8hk8R+dDXSqv1MbRT1ybB5W0k8878XSOjvmiYTDIfyc9acxVJr
Y/cykHipa+te1pOhv7wYPYtZ9orGBV5SGOJm4NrB3K1aJar0RfzxC3ikr7Dyc6Qw
qDTBU39CluVIQeuQRgwG3MuSxl7zRERDRilGoKb8uY45JzmxWuKxrfwT/478JuHU
/oTxUFqOl2stKnn7QGTq8z29W+GgBLCXSBxC9epaHM0myFH/FJlniXJfHeytWt0=
-----END CERTIFICATE-----
```

```
  GNU nano 8.4                                              c2.pem
-----BEGIN CERTIFICATE-----
MIIFYjCCBEqgAwIBAgIQd70NbNs2+RrqIQ/E8FjTDTANBgkqhkiG9w0BAQsFADBX
MQswCQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBudi1zYTEQMA4GA1UE
CxMHUm9vdCBDQTEbMBkGA1UEAxMSR2xvYmFsU2lnbiBSb290IENBMB4XDTIwMDYx
OTAwMDA0MloXDTI4MDEyODAwMDA0MlowRzELMAkGA1UEBhMCVVMxIjAgBgNVBAoT
GUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBMTEMxFDASBgNVBAMTC0dUUyBSb290IFIx
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAthECix7joXebO9y/lD63
ladAPKH9gvl9MgaCcfb2jH/76Nu8ai6Xl6OMS/kr9rH5zoQdsfnFl97vufKj6bwS
iV6nqlKr+CMny6SxnGPb15l+8Ape62im9MZaRw1NEDPjTrETo8gYbEvs/AmQ351k
KSUjB6G00j0uYODP0gmHu81I8E3CwnqIiru6z1kZ1q+PsAewnjHxgsHA3y6mbWwZ
DrXYfiYaRQM9sHmklCitD38m5agI/pboPGiUU+6DOogrFZYJsuB6jC511pzrp1Zk
j5ZPaK49l8KEj8C8QMALXL32h7M1bKwYUH+E4EzNktMg6TO8UpmvMrUpsyUqtEj5
cuHKZPfmghCN6J3Cioj6OGaK/GP5Afl4/Xtcd/p2h/rs37EOeZVXtL0m79YB0esW
CruOC7XFxYpVq9Os6pFLKcwZpDIlTirxZUTQAs6qzkm06p98g7BAe+dDq6dso499
iYH6TKX/1Y7DzkvgtdizjkXPdsDtQCv9Uw+wp9U7DbGKogPeMa3Md+pvez7W35Ei
Eua++tgy/BBjFFFy3l3WFpO9KWgz7zpm7AeKJt8T11dleCfeXkkUAKIAf5qoIbap
sZWwpbkNFhHax2xIPEDgfg1azVY80ZcFuctL7TlLnMQ/0lUTbiSw1nH69MG6zO0b
9f6BQdgAmD06yK56mDcYBZUCAwEAAaOCATgwggE0MA4GA1UdDwEB/wQEAwIBhjAP
BgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBTkrysmcRorSCeFL1JmLO/wiRNxPjAf
BgNVHSMEGDAWgBRge2YaRQ2XyolQL30EzTSo//z9SzBgBggrBgEFBQcBAQRUMFIw
JQYIKwYBBQUHMAGGGWh0dHA6Ly9vY3NwLnBraS5nb29nL2dzcjEwKQYIKwYBBQUH
MAKGHWh0dHA6Ly9wa2kuZ29vZy9nc3IxL2dzcjEuY3J0MDIGA1UdHwQrMCkwJ6Al
oCOGIWh0dHA6Ly9jcmwucGtpLmdvb2cvZ3NyMS9nc3IxLmNybDA7BgNVHSAENDAy
MAgGBmeBDAECATAIBgZngQwBAgIwDQYLKwYBBAHWeQIFAwIwDQYLKwYBBAHWeQIF
AwMwDQYJKoZIhvcNAQELBQADggEBADSkHrEoo9C0dhemMXoh6dFSPsjbdBZBiLg9
NR3t5P+T4Vxfq7vqfM/b5A3Ri1fyJm9bvhdGaJQ3b2t6yMAYN/olUazsaL+yyEn9
WprKASOshIArAoyZl+tJaox118fessmXn1hIVw41oeQa1v1vg4Fv74zPl6/AhSrw
9U5pCZEt4Wi4wStz6dTZ/CLANx8LZh1J7QJVj2fhMtfTJr9w4z30Z209fOU0iOMy
+qduBmpvvYuR7hZL6Dupszfnw0Skfths18dG9ZKb59UhvmaSGZRVbNQpsg3BZlvi
d0lIKO2d1xozclOzgjXPYovJJIultzkMu34qQb9Sz/yilrbCgj8=
-----END CERTIFICATE-----
```

**다운로드 받은 인증서 중 첫번째는 복사하여 c1.pem에 저장하고, 두 번째 인증서는 복사하여 c2.pem에 저장했다.**

**c1.pem WR2 /Intermediate CA RSA**

**c2.pem GTS Root R4 Root/CA RSA**

**Step 2**

```
annynahyun2002@instance-20251118-052056:~$ openssl x509 -in c2.pem -noout -modulus
Modulus=B611028B1EE3A1779B3BDCBF943EB795A7403CA1FD82F97D32068271F6F68C7FFBE8DBBC6A2E9797A38C4BF92BF6B1F9CE841DB1
F9C597DEEFB9F2A3E9BC12895EA7AA52ABF82327CBA4B19C63DBD7997EF00A5EEB68A6F4C65A470D4D1033E34EB113A3C8186C4BECFC0990
DF9D6429252307A1B4D23D2E60E0CFD20987BBCD48F04DC2C27A888ABBBACF5919D6AF8FB007B09E31F182C1C0DF2EA66D6C190EB5D87E26
1A45033DB079A49428AD0F7F26E5A808FE96E83C689453EE833A882B159609B2E07A8C2E75D69CEBA756648F964F68AE3D97C2848FC0BC40
C00B5CBDF687B3356CAC18507F84E04CCD92D320E933BC5299AF32B529B3252AB448F972E1CA64F7E682108DE89DC28A88FA38668AFC63F9
01F978FD7B5C77FA7687FAECDFB10E799557B4BD26EFD601D1EB160ABB8E0BB5C5C58A55ABD3ACEA914B29CC19A432254E2AF16544D002CE
AACE49B4EA9F7C83B0407BE743ABA76CA38F7D8981FA4CA5FFD58EC3CE4BE0B5D8B38E45CF76C0ED402BFD530FB0A7D53B0DB18AA203DE31
ADCC77EA6F7B3ED6DF912212E6BEFAD832FC1063145172DE5DD61693BD296833EF3A66EC078A26DF13D757657827DE5E491400A2007F9AA8
21B6A9B195B0A5B90D1611DAC76C483C40E07E0D5ACD563CD19705B9CB4BED394B9CC43FD255136E24B0D671FAF4C1BACCED1BF5FE8141D8
00983D3AC8AE7A9837180595
```

**명령어를 입력하여 n을 구한다.**

```
Modulus:
    00:b6:11:02:8b:1e:e3:a1:77:9b:3b:dc:bf:94:3e:
    b7:95:a7:40:3c:a1:fd:82:f9:7d:32:06:82:71:f6:
    f6:8c:7f:fb:e8:db:bc:6a:2e:97:97:a3:8c:4b:f9:
    2b:f6:b1:f9:ce:84:1d:b1:f9:c5:97:de:ef:b9:f2:
    a3:e9:bc:12:89:5e:a7:aa:52:ab:f8:23:27:cb:a4:
    b1:9c:63:db:d7:99:7e:f0:0a:5e:eb:68:a6:f4:c6:
    5a:47:0d:4d:10:33:e3:4e:b1:13:a3:c8:18:6c:4b:
    ec:fc:09:90:df:9d:64:29:25:23:07:a1:b4:d2:3d:
    2e:60:e0:cf:d2:09:87:bb:cd:48:f0:4d:c2:c2:7a:
    88:8a:bb:ba:cf:59:19:d6:af:8f:b0:07:b0:9e:31:
    f1:82:c1:c0:df:2e:a6:6d:6c:19:0e:b5:d8:7e:26:
    1a:45:03:3d:b0:79:a4:94:28:ad:0f:7f:26:e5:a8:
    08:fe:96:e8:3c:68:94:53:ee:83:3a:88:2b:15:96:
    09:b2:e0:7a:8c:2e:75:d6:9c:eb:a7:56:64:8f:96:
    4f:68:ae:3d:97:c2:84:8f:c0:bc:40:c0:0b:5c:bd:
    f6:87:b3:35:6c:ac:18:50:7f:84:e0:4c:cd:92:d3:
    20:e9:33:bc:52:99:af:32:b5:29:b3:25:2a:b4:48:
    f9:72:e1:ca:64:f7:e6:82:10:8d:e8:9d:c2:8a:88:
    fa:38:66:8a:fc:63:f9:01:f9:78:fd:7b:5c:77:fa:
    76:87:fa:ec:df:b1:0e:79:95:57:b4:bd:26:ef:d6:
    01:d1:eb:16:0a:bb:8e:0b:b5:c5:c5:8a:55:ab:d3:
    ac:ea:91:4b:29:cc:19:a4:32:25:4e:2a:f1:65:44:
    d0:02:ce:aa:ce:49:b4:ea:9f:7c:83:b0:40:7b:e7:
    43:ab:a7:6c:a3:8f:7d:89:81:fa:4c:a5:ff:d5:8e:
    c3:ce:4b:e0:b5:d8:b3:8e:45:cf:76:c0:ed:40:2b:
    fd:53:0f:b0:a7:d5:3b:0d:b1:8a:a2:03:de:31:ad:
    cc:77:ea:6f:7b:3e:d6:df:91:22:12:e6:be:fa:d8:
    32:fc:10:63:14:51:72:de:5d:d6:16:93:bd:29:68:
    33:ef:3a:66:ec:07:8a:26:df:13:d7:57:65:78:27:
    de:5e:49:14:00:a2:00:7f:9a:a8:21:b6:a9:b1:95:
    b0:a5:b9:0d:16:11:da:c7:6c:48:3c:40:e0:7e:0d:
    5a:cd:56:3c:d1:97:05:b9:cb:4b:ed:39:4b:9c:c4:
    3f:d2:55:13:6e:24:b0:d6:71:fa:f4:c1:ba:cc:ed:
    1b:f5:fe:81:41:d8:00:98:3d:3a:c8:ae:7a:98:37:
    18:05:95
Exponent: 65537 (0x10001)
```

**명령어를 입력하여 e 값을 찾는다.**

**n:**

**B611028B1EE3A1779B3BDCBF943EB795A7403CA1FD82F97D32068271F6F68C7FFBE8DBBC6A2**

**E9797A38C4BF92BF6B1F9CE841DB1F9C597DEEFB9F2A3E9BC12895EA7AA52ABF82327CBA4B1**

**9C63DBD7997EF00A5EEB68A6F4C65A470D4D1033E34EB113A3C8186C4BECFC0990DF9D64292**

**52307A1B4D23D2E60E0CFD20987BBCD48F04DC2C27A888ABBBACF5919D6AF8FB007B09E31F**

**182C1C0DF2EA66D6C190EB5D87E261A45033DB079A49428AD0F7F26E5A808FE96E83C689453**

**EE833A882B159609B2E07A8C2E75D69CEBA756648F964F68AE3D97C2848FC0BC40C00B5CBDF**

**687B3356CAC18507F84E04CCD92D320E933BC5299AF32B529B3252AB448F972E1CA64F7E6821**

**08DE89DC28A88FA38668AFC63F901F978FD7B5C77FA7687FAECDFB10E799557B4BD26EFD601**

**D1EB160ABB8E0BB5C5C58A55ABD3ACEA914B29CC19A432254E2AF16544D002CEAACE49B4E**

**A9F7C83B0407BE743ABA76CA38F7D8981FA4CA5FFD58EC3CE4BE0B5D8B38E45CF76C0ED402B**

**FD530FB0A7D53B0DB18AA203DE31ADCC77EA6F7B3ED6DF912212E6BEFAD832FC1063145172**
**DE5DD61693BD296833EF3A66EC078A26DF13D757657827DE5E491400A2007F9AA821B6A9B1**
**95B0A5B90D1611DAC76C483C40E07E0D5ACD563CD19705B9CB4BED394B9CC43FD255136E24**
**B0D671FAF4C1BACCED1BF5FE8141D800983D3AC8AE7A9837180595**

**e : 10001**

**Step 3**

**명령어를 입력하여 Signature를 구했다.**

```
Signature Algorithm: sha256WithRSAEncryption
Signature Value:
    45:75:8b:e5:1f:3b:44:13:96:1a:ab:58:f1:35:c9:6f:3d:d2:
    d0:33:4a:86:33:ba:57:51:4f:ee:c4:34:da:16:12:4c:bf:13:
    9f:0d:d4:54:e9:48:79:c0:30:3c:94:25:f2:1a:f4:ba:32:94:
    b6:33:72:0b:85:ee:09:11:25:34:94:e1:6f:42:db:82:9b:7b:
    7f:2a:9a:a9:ff:7f:a9:d2:de:4a:20:cb:b3:fb:03:03:b8:f8:
    07:05:da:59:92:2f:18:46:98:ce:af:72:be:24:26:b1:1e:00:
    4d:bd:08:ad:93:41:44:0a:bb:c7:d5:01:85:bf:93:57:e3:df:
    74:12:53:0e:11:25:d3:9b:dc:de:cb:27:6e:b3:c2:b9:33:62:
    39:c2:e0:35:e1:5b:a7:09:2e:19:cb:91:2a:76:5c:f1:df:ca:
    23:84:40:a5:6f:ff:9a:41:e0:b5:ef:32:d1:85:ae:af:25:09:
    f0:62:c5:6e:c2:c8:6e:32:fd:b8:da:e2:ce:4a:91:4a:f3:85:
    55:4e:b1:75:d6:48:33:2f:6f:84:d9:12:5c:9f:d4:71:98:63:
    25:8d:69:5c:0a:6b:7d:f2:41:bd:e8:bb:8f:e4:22:d7:9d:65:
    45:e8:4c:0a:87:da:e9:60:66:88:0e:1f:c7:e1:4e:56:c5:76:
    ff:b4:7a:57:69:f2:02:22:09:26:41:1d:da:74:a2:e5:29:f3:
    c4:9a:e5:5d:d6:aa:7a:fd:e1:b7:2b:66:38:fb:e8:29:66:ba:
    ef:a0:13:2f:f8:73:7e:f0:da:40:11:1c:5d:dd:8f:a6:fc:be:
    db:be:56:f8:32:9c:1f:41:41:6d:7e:b6:c5:eb:c6:8b:36:b7:
    17:8c:9d:cf:19:7a:34:9f:21:93:c4:7e:74:35:d2:aa:fd:4c:
    6d:14:f5:c9:b0:79:5b:49:3c:f3:bf:17:48:e8:ef:9a:26:13:
    0c:87:f2:73:d6:9c:c5:52:6b:63:f7:32:90:78:a9:6b:eb:5e:
    d6:93:a1:bf:bc:18:3d:8b:59:f6:8a:c6:05:5e:52:18:e2:66:
    e0:da:c1:dc:ad:5a:25:aa:f4:45:fc:f1:0b:78:a4:af:b0:f2:
    73:a4:30:a8:34:c1:53:7f:42:96:e5:48:41:eb:90:46:0c:06:
    dc:cb:92:c6:5e:f3:44:44:43:46:29:46:a0:a6:fc:b9:8e:39:
    27:39:b1:5a:e2:b1:ad:fc:13:ff:8e:fc:26:e1:d4:fe:84:f1:
    50:5a:8e:97:6b:2d:2a:79:fb:40:64:ea:f3:3d:bd:5b:e1:a0:
    04:b0:97:48:1c:42:f5:ea:5a:1c:cd:26:c8:51:ff:14:99:67:
    89:72:5f:1d:ec:ad:5a:dd
```

```
annynahyun2002@instance-20251118-052056:~$ cat signature | tr -d '[:space:]:'
45758be51f3b4413961aab58f135c96f3dd2d0334a8633ba57514feec434da16124cbf139f0dd454e94879c0303c9425f21af4ba3294b633
720b85ee0911253494e16f42db829b7b7f2a9aa9ff7fa9d2de4a20cbb3fb0303b8f80705da59922f184698ceaf72be2426b11e004dbd08ad
9341440abbc7d50185bf9357e3df7412530e1125d39bdcdecb276eb3c2b9336239c2e035e15ba7092e19cb912a765cf1dfca238440a56fff
9a41e0b5ef32d185aeaf2509f062c56ec2c86e32fdb8dae2ce4a914af385554eb175d648332f6f84d9125c9fd4719863258d695c0a6b7df2
41bde8bb8fe422d79d6545e84c0a87dae96066880e1fc7e14e56c576ffb47a5769f202220926411dda74a2e529f3c49ae55dd6aa7afde1b7
2b6638fbe82966baefa0132ff8737ef0da40111c5ddd8fa6fcbedbbe56f8329c1f41416d7eb6c5ebc68b36b7178c9dcf197a349f2193c47e
7435d2aafd4c6d14f5c9b0795b493cf3bf1748e8ef9a26130c87f273d69cc5526b63f7329078a96beb5ed693a1bfbc183d8b59f68ac6055e
5218e266e0dac1dcad5a25aaf445fcf10b78a4afb0f273a430a834c1537f4296e54841eb90460c06dccb92c65ef3444443462946a0a6fcb9
8e392739b15ae2b1adfc13ff8efc26e1d4fe84f1505a8e976b2d2a79fb4064eaf33dbd5be1a004b097481c42f5ea5a1ccd26c851ff149967
89725f1decad5addannynahyun2002@instance-20251118-052056:~$ openssl asn1parse -i -in c1.pem
```

**Signature**


45758be51f3b4413961aab58f135c96f3dd2d0334a8633ba57514feec434da16124cbf139f0dd454
e94879c0303c9425f21af4ba3294b633720b85ee0911253494e16f42db829b7b7f2a9aa9ff7fa9d2
de4a20cbb3fb0303b8f80705da59922f184698ceaf72be2426b11e004dbd08ad9341440abbc7d50
185bf9357e3df7412530e1125d39bdcdecb276eb3c2b9336239c2e035e15ba7092e19cb912a765c
f1dfca238440a56fff9a41e0b5ef32d185aeaf2509f062c56ec2c86e32fdb8dae2ce4a914af385554e
b175d648332f6f84d9125c9fd4719863258d695c0a6b7df241bde8bb8fe422d79d6545e84c0a87d
ae96066880e1fc7e14e56c576ffb47a5769f202220926411dda74a2e529f3c49ae55dd6aa7afde1b
72b6638fbe82966baefa0132ff8737ef0da40111c5ddd8fa6fcbedbbe56f8329c1f41416d7eb6c5eb
c68b36b7178c9dcf197a349f2193c47e7435d2aafd4c6d14f5c9b0795b493cf3bf1748e8ef9a26130
c87f273d69cc5526b63f7329078a96beb5ed693a1bfbc183d8b59f68ac6055e5218e266e0dac1dca
d5a25aaf445fcf10b78a4afb0f273a430a834c1537f4296e54841eb90460c06dccb92c65ef3444443
462946a0a6fcb98e392739b15ae2b1adfc13ff8efc26e1d4fe84f1505a8e976b2d2a79fb4064eaf33
dbd5be1a004b097481c42f5ea5a1ccd26c851ff14996789725f1decad5add

**Step 4**

**인증서를 구문 분석하기 위해 openssl asn1parse -i -in c1.pem 명령어를 입력했다.**

```
89725f1decad5addannynahyun2002@instance-20251118-052056:~$ openssl asn1parse -i -in c1.pem
    0:d=0  hl=4 l=1291 cons: SEQUENCE
    4:d=1  hl=4 l= 755 cons:  SEQUENCE
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  16 prim:   INTEGER           :7FF005A07C4CDED100AD9D66A5107B98
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
   44:d=3  hl=2 l=   0 prim:    NULL
   46:d=2  hl=2 l=  71 cons:   SEQUENCE
   48:d=3  hl=2 l=  11 cons:    SET
   50:d=4  hl=2 l=   9 cons:     SEQUENCE
   52:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   57:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
   61:d=3  hl=2 l=  34 cons:    SET
   63:d=4  hl=2 l=  32 cons:     SEQUENCE
   65:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   70:d=5  hl=2 l=  25 prim:      PRINTABLESTRING   :Google Trust Services LLC
   97:d=3  hl=2 l=  20 cons:    SET
   99:d=4  hl=2 l=  18 cons:     SEQUENCE
  101:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
  106:d=5  hl=2 l=  11 prim:      PRINTABLESTRING   :GTS Root R1
  119:d=2  hl=2 l=  30 cons:   SEQUENCE
  121:d=3  hl=2 l=  13 prim:    UTCTIME           :231213090000Z
  136:d=3  hl=2 l=  13 prim:    UTCTIME           :290220140000Z
  151:d=2  hl=2 l=  59 cons:   SEQUENCE
  153:d=3  hl=2 l=  11 cons:    SET
  155:d=4  hl=2 l=   9 cons:     SEQUENCE
  157:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
  162:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
  166:d=3  hl=2 l=  30 cons:    SET
  168:d=4  hl=2 l=  28 cons:     SEQUENCE
  170:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
  175:d=5  hl=2 l=  21 prim:      PRINTABLESTRING   :Google Trust Services
  198:d=3  hl=2 l=  12 cons:    SET
  200:d=4  hl=2 l=  10 cons:     SEQUENCE
  202:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
  207:d=5  hl=2 l=   3 prim:      PRINTABLESTRING   :WR2
  212:d=2  hl=4 l= 290 cons:   SEQUENCE
  216:d=3  hl=2 l=  13 cons:    SEQUENCE
  218:d=4  hl=2 l=   9 prim:     OBJECT            :rsaEncryption
  229:d=4  hl=2 l=   0 prim:     NULL
  231:d=3  hl=4 l= 271 prim:    BIT STRING
  506:d=2  hl=3 l= 254 cons:   cont [ 3 ]
  509:d=3  hl=3 l= 251 cons:    SEQUENCE
  512:d=4  hl=2 l=  14 cons:     SEQUENCE
  514:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Key Usage
```

```
annynahyun2002@instance-20251118-052056:~$ openssl asn1parse -i -in c1.pem -strparse 4 -out c1_body.bin -noout
annynahyun2002@instance-20251118-052056:~$ ls
c0.pem  c1_body.bin  task1    task2    task3    task4    task5
c1.pem  signature    task1.c  task2.c  task3.c  task4.c  task5.c
```

```
annynahyun2002@instance-20251118-052056:~$ sha256sum c1_body.bin
b0a28a138130bed21b3631a36a0236eb4ec474a4951e8c1866dcaf0fd88fae49  c1_body.bin
annynahyun2002@instance-20251118-052056:~$
```

**$ sha256sum c1_body.bin 명령어를 통해 인증서 본문의 Hash 값을 계산한 결과 위와 같이 출력되었다.**

**Step 5.**

```
int main()
{
    BN_CTX *ctx = BN_CTX_new();

    const char *n_str = "B611028B1EE3A1779B3BDCBF943EB795A7403CA1FD82F97D32068271F6F68C7FFBE8DBBC6A2E9797A38C4BF
92BF6B1F9CE841DB1F9C597DEEFB9F2A3E9BC12895EA7AA52ABF82327CBA4B19C63DBD7997EF00A5EEB68A6F4C65A470D4D1033E34EB113A
3C8186C4BECFC0990DF9D6429252307A1B4D23D2E60E0CFD20987BBCD48F04DC2C27A888ABBBACF5919D6AF8FB007B09E31F182C1C0DF2EA
66D6C190EB5D87E261A45033DB079A49428AD0F7F26E5A808FE96E83C689453EE833A882B159609B2E07A8C2E75D69CEBA756648F964F68A
E3D97C2848FC0BC40C00B5CBDF687B3356CAC18507F84E04CCD92D320E933BC5299AF32B529B3252AB448F972E1CA64F7E682108DE89DC28
A88FA38668AFC63F901F978FD7B5C77FA7687FAECDFB10E799557B4BD26EFD601D1EB160ABB8E0BB5C5C58A55ABD3ACEA914B29CC19A4322
54E2AF16544D002CEAACE49B4EA9F7C83B0407BE743ABA76CA38F7D8981FA4CA5FFD58EC3CE4BE0B5D8B38E45CF76C0ED402BFD530FB0A7D
53B0DB18AA203DE31ADCC77EA6F7B3ED6DF912212E6BEFAD832FC1063145172DE5DD61693BD296833EF3A66EC078A26DF13D757657827DE5
E491400A2007F9AA821B6A9B195B0A5B90D1611DAC76C483C40E07E0D5ACD563CD19705B9CB4BED394B9CC43FD255136E24B0D671FAF4C1B
ACCED1BF5FE8141D800983D3AC8AE7A9837180595";
    const char *e_str = "010001";
    const char *C_str = "45758be51f3b4413961aab58f135c96f3dd2d0334a8633ba57514feec434da16124cbf139f0dd454e94879c
0303c9425f21af4ba3294b633720b85ee0911253494e16f42db829b7b7f2a9aa9ff7fa9d2de4a20cbb3fb0303b8f80705da59922f184698c
eaf72be2426b11e004dbd08ad9341440abbc7d50185bf9357e3df7412530e1125d39bdcdecb276eb3c2b9336239c2e035e15ba7092e19cb9
12a765cf1dfca238440a56fff9a41e0b5ef32d185aeaf2509f062c56ec2c86e32fdb8dae2ce4a914af385554eb175d648332f6f84d9125c9
fd4719863258d695c0a6b7df241bde8bb8fe422d79d6545e84c0a87dae96066880e1fc7e14e56c576ffb47a5769f202220926411dda74a2e
529f3c49ae55dd6aa7afde1b72b6638fbe82966baefa0132ff8737ef0da40111c5ddd8fa6fcbedbbe56f8329c1f41416d7eb6c5ebc68b36b
7178c9dcf197a349f2193c47e7435d2aafd4c6d14f5c9b0795b493cf3bf1748e8ef9a26130c87f273d69cc5526b63f7329078a96beb5ed69
3a1bfbc183d8b59f68ac6055e5218e266e0dac1dcad5a25aaf445fcf10b78a4afb0f273a430a834c1537f4296e54841eb90460c06dccb92c
65ef3444443462946a0a6fcb98e392739b15ae2b1adfc13ff8efc26e1d4fe84f1505a8e976b2d2a79fb4064eaf33dbd5be1a004b097481c4
2f5ea5a1ccd26c851ff14996789725f1decad5add";
```

Task5.c에서 사용했던 코드에서 *n_str, *e_str, *C_str 값을 변경하여 task6.c의 코드를 작성 했다.

```
annynahyun2002@instance-20251118-052056:~$ gcc -o task6 task6.c -lcrypto
annynahyun2002@instance-20251118-052056:~$ ./task6
M =  01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D060960864801650304020105000420B0A28A138130BED21B3631A36A0236EB4EC474A4951E8
C1866DCAF0FD88FAE49
annynahyun2002@instance-20251118-052056:~$
```

```
annynahyun2002@instance-20251118-052056:~$ sha256sum c1_body.bin
b0a28a138130bed21b3631a36a0236eb4ec474a4951e8c1866dcaf0fd88fae49  c1_body.bin
```

복호화된 서명 값 $M$(2048비트)은 0x01 || FF...FF || 0x00 || ASN.1 SEQUENCE || Hash로 구성된 PKCS#1 v1.5 서명 구조를 가지고 있었다. 이 구조의 마지막 32바이트를 추출해 확인한 결과, 해당 값은 내가 OpenSSL로 추출한 TBS(TBS Certificate) 데이터인 c1_body.bin의 SHA-256 해시값(b0a2...fae49)과 정확히 일치하였다. 따라서 WR2 인증서의 서명은 올바른 Google Root CA의 개인키로 생성된 유효한 서명임을 확인 할 수 있었다.