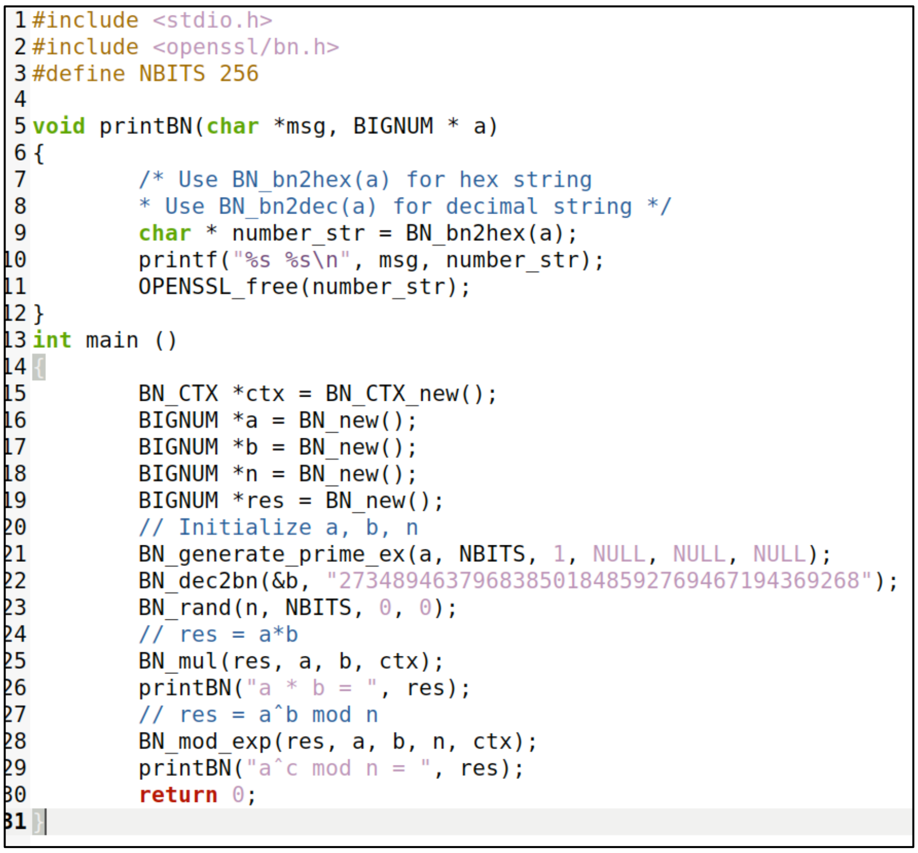
Assignment 4.6 Lab: RSA

National Taiwan Normal University CSIE Information Security

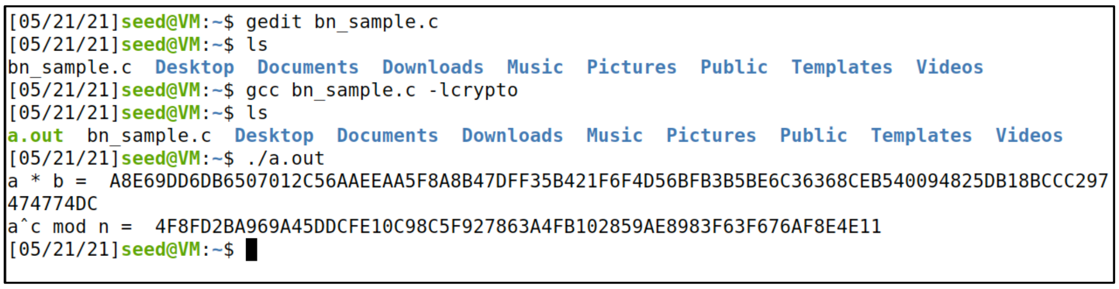
60947038S

2.2 A Complete Example:



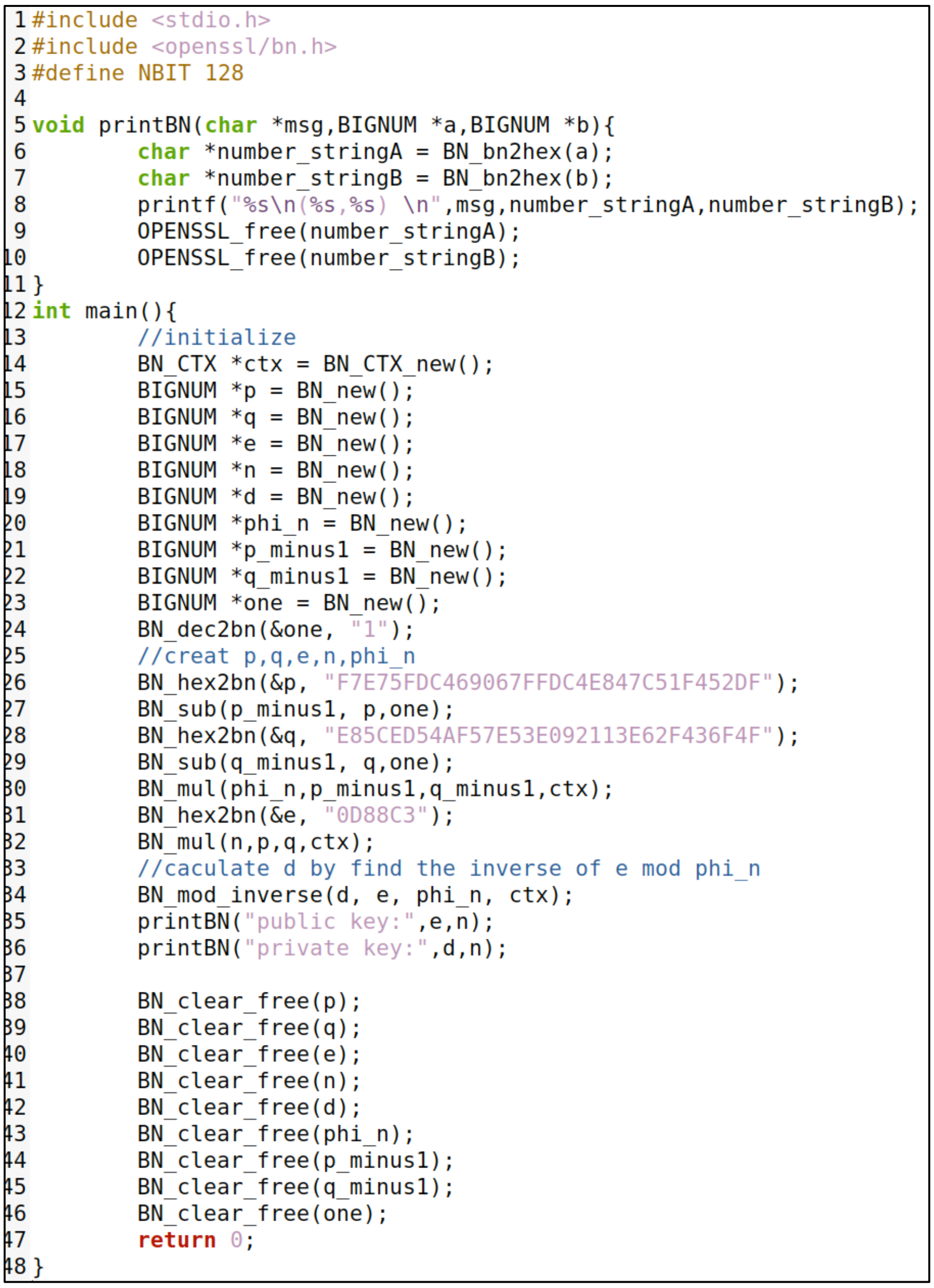
```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

void printBN(char *msg, BIGNUM * a)
{
        /* Use BN_bn2hex(a) for hex string
        * Use BN_bn2dec(a) for decimal string */
        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();
        BIGNUM *a = BN_new();
        BIGNUM *b = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *res = BN_new();
        // Initialize a, b, n
        BN_generate_prime_ex(a, NBITS, 1, NULL, NULL, NULL);
        BN_dec2bn(&b, "273489463796838501848592769467194369268");
        BN_rand(n, NBITS, 0, 0);
        // res = a*b
        BN_mul(res, a, b, ctx);
        printBN("a * b = ", res);
        // res = a^b mod n
        BN_mod_exp(res, a, b, n, ctx);
        printBN("a^c mod n = ", res);
        return 0;
}
```

告訴我們 lab 有一個 function 是專們針對大數運算，在逐一講述其功能後，要求我們實作一個$a * b$ 和 $a^c \bmod n$，並賦予程式碼以及執行指令。

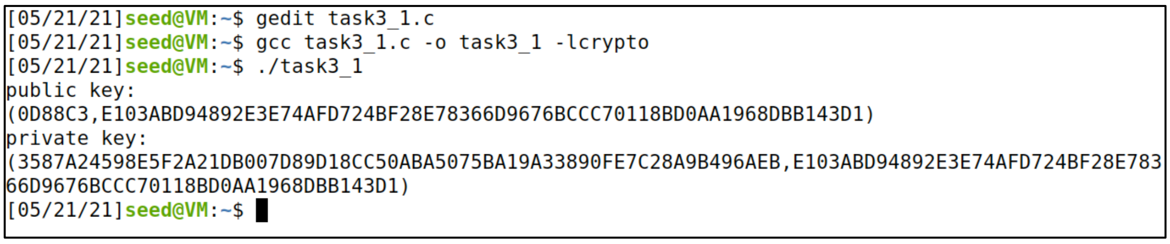


```
[05/21/21]seed@VM:~$ gedit bn_sample.c
[05/21/21]seed@VM:~$ ls
bn_sample.c  Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[05/21/21]seed@VM:~$ gcc bn_sample.c -lcrypto
[05/21/21]seed@VM:~$ ls
a.out  bn_sample.c  Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[05/21/21]seed@VM:~$ ./a.out
a * b =  A8E69DD6DB6507012C56AAEEAA5F8A8B47DFF35B421F6F4D56BFB3B5BE6C36368CEB540094825DB18BCCC297
474774DC
a^c mod n =  4F8FD2BA969A45DDCFE10C98C5F927863A4FB102859AE8983F63F676AF8E4E11
[05/21/21]seed@VM:~$ 
```

3.1 Task 1: Deriving the Private Key:

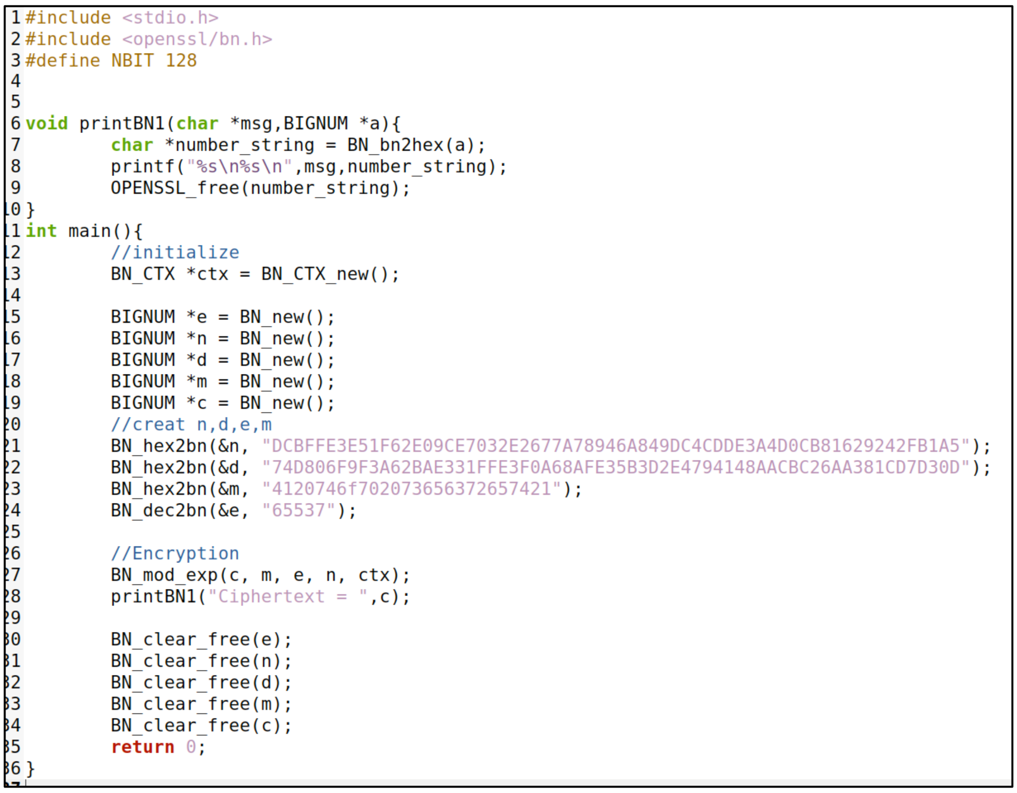


```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBIT 128

void printBN(char *msg,BIGNUM *a,BIGNUM *b){
        char *number_stringA = BN_bn2hex(a);
        char *number_stringB = BN_bn2hex(b);
        printf("%s\n(%s,%s) \n",msg,number_stringA,number_stringB);
        OPENSSL_free(number_stringA);
        OPENSSL_free(number_stringB);
}
int main(){
        //initialize
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *p = BN_new();
        BIGNUM *q = BN_new();
        BIGNUM *e = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *phi_n = BN_new();
        BIGNUM *p_minus1 = BN_new();
        BIGNUM *q_minus1 = BN_new();
        BIGNUM *one = BN_new();
        BN_dec2bn(&one, "1");
        //creat p,q,e,n,phi_n
        BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        BN_sub(p_minus1, p,one);
        BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
        BN_sub(q_minus1, q,one);
        BN_mul(phi_n,p_minus1,q_minus1,ctx);
        BN_hex2bn(&e, "0D88C3");
        BN_mul(n,p,q,ctx);
        //caculate d by find the inverse of e mod phi_n
        BN_mod_inverse(d, e, phi_n, ctx);
        printBN("public key:",e,n);
        printBN("private key:",d,n);

        BN_clear_free(p);
        BN_clear_free(q);
        BN_clear_free(e);
        BN_clear_free(n);
        BN_clear_free(d);
        BN_clear_free(phi_n);
        BN_clear_free(p_minus1);
        BN_clear_free(q_minus1);
        BN_clear_free(one);
        return 0;
}
```

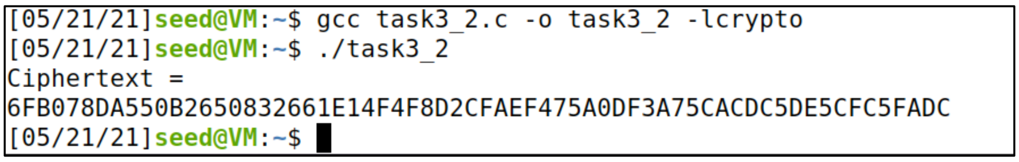


```
[05/21/21]seed@VM:~$ gedit task3_1.c
[05/21/21]seed@VM:~$ gcc task3_1.c -o task3_1 -lcrypto
[05/21/21]seed@VM:~$ ./task3_1
public key:
(0D88C3,E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1)
private key:
(3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB,E103ABD94892E3E74AFD724BF28E783
66D9676BCCC70118BD0AA1968DBB143D1)
[05/21/21]seed@VM:~$ 
```

3.2 Task 2: Encrypting a Message:

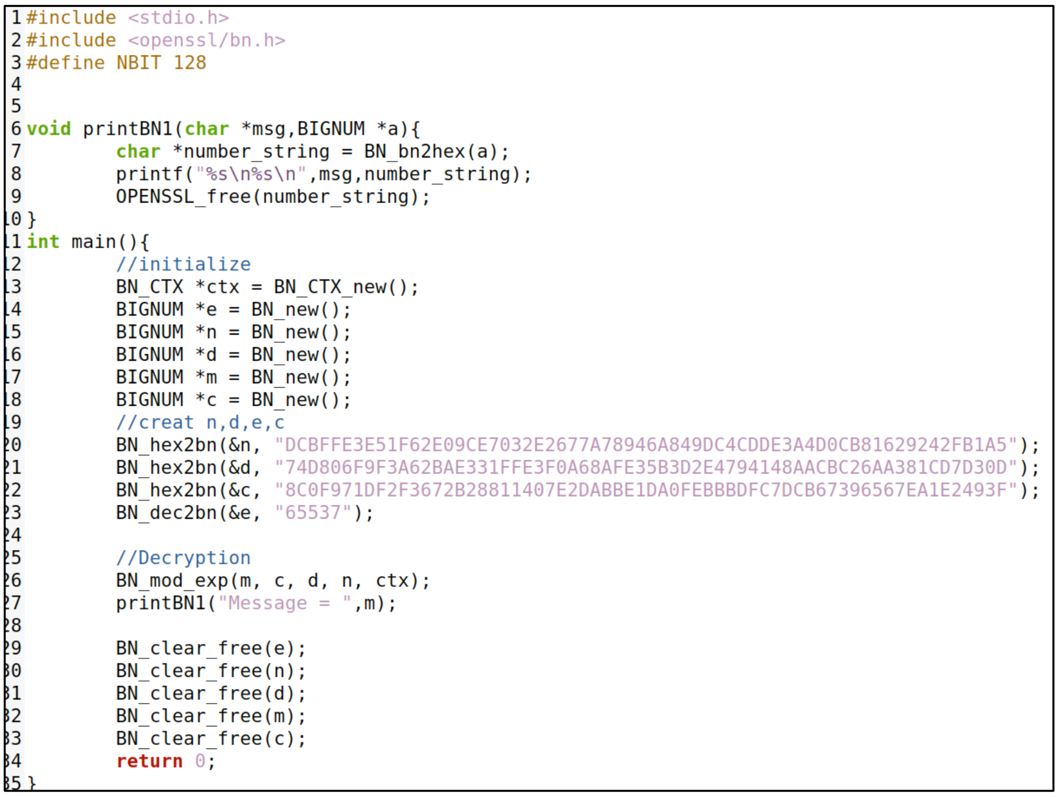


```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBIT 128


void printBN1(char *msg,BIGNUM *a){
        char *number_string = BN_bn2hex(a);
        printf("%s\n%s\n",msg,number_string);
        OPENSSL_free(number_string);
}
int main(){
        //initialize
        BN_CTX *ctx = BN_CTX_new();

        BIGNUM *e = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *m = BN_new();
        BIGNUM *c = BN_new();
        //creat n,d,e,m
        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        BN_hex2bn(&m, "4120746f702073656372657421");
        BN_dec2bn(&e, "65537");

        //Encryption
        BN_mod_exp(c, m, e, n, ctx);
        printBN1("Ciphertext = ",c);

        BN_clear_free(e);
        BN_clear_free(n);
        BN_clear_free(d);
        BN_clear_free(m);
        BN_clear_free(c);
        return 0;
}
```

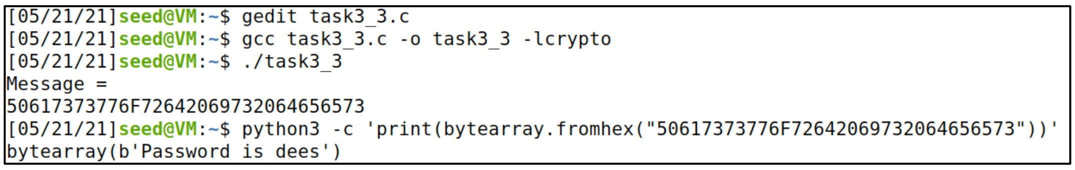


```
[05/21/21]seed@VM:~$ gcc task3_2.c -o task3_2 -lcrypto
[05/21/21]seed@VM:~$ ./task3_2
Ciphertext =
6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
[05/21/21]seed@VM:~$ 
```

3.3 Task 3: Decrypting a Message:

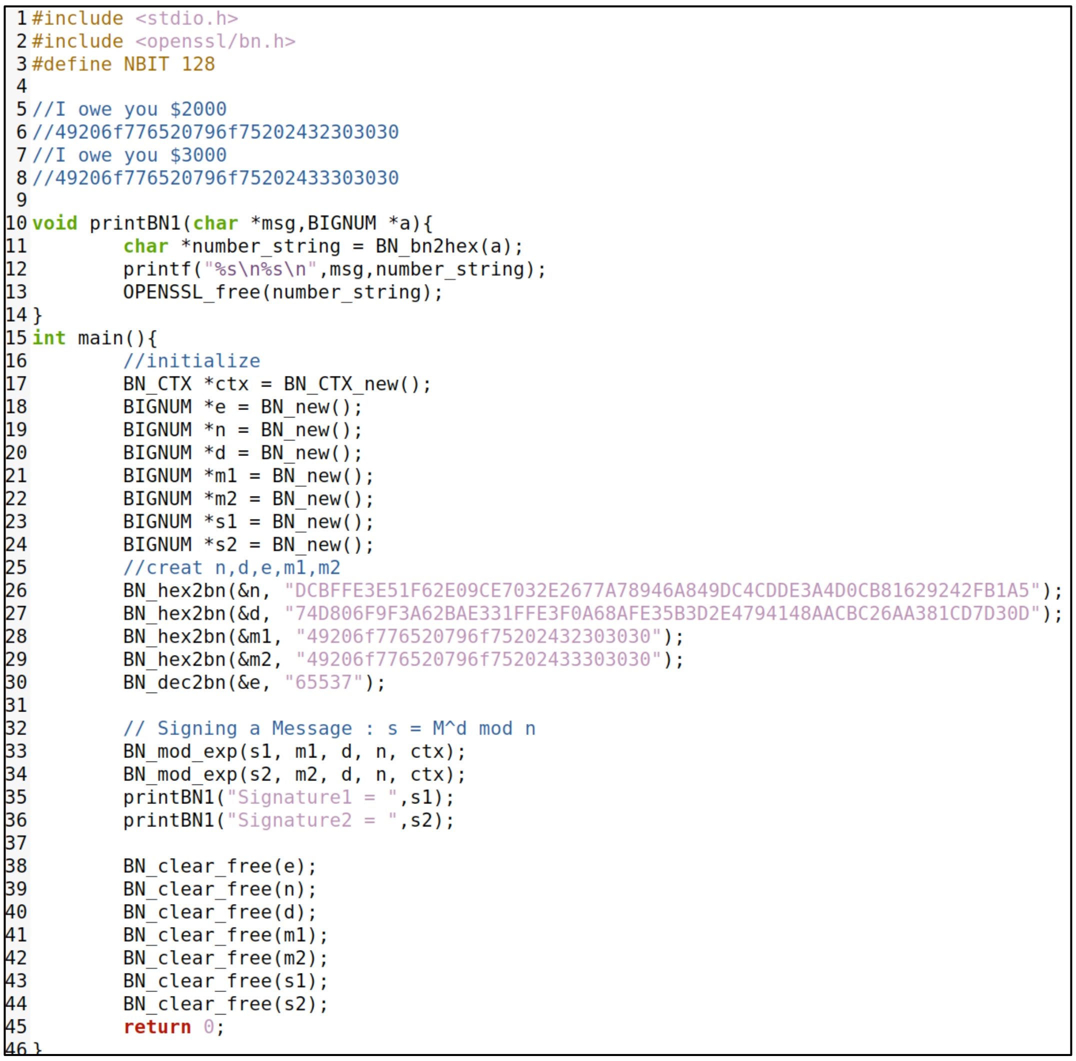


```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBIT 128


void printBN1(char *msg,BIGNUM *a){
        char *number_string = BN_bn2hex(a);
        printf("%s\n%s\n",msg,number_string);
        OPENSSL_free(number_string);
}
int main(){
        //initialize
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *e = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *m = BN_new();
        BIGNUM *c = BN_new();
        //creat n,d,e,c
        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        BN_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
        BN_dec2bn(&e, "65537");

        //Decryption
        BN_mod_exp(m, c, d, n, ctx);
        printBN1("Message = ",m);

        BN_clear_free(e);
        BN_clear_free(n);
        BN_clear_free(d);
        BN_clear_free(m);
        BN_clear_free(c);
        return 0;
}
```

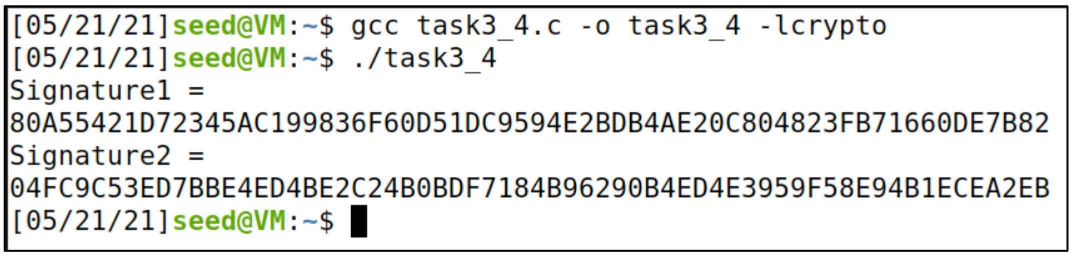


```
[05/21/21]seed@VM:~$ gedit task3_3.c
[05/21/21]seed@VM:~$ gcc task3_3.c -o task3_3 -lcrypto
[05/21/21]seed@VM:~$ ./task3_3
Message =
50617373776F72642069732064656573
[05/21/21]seed@VM:~$ python3 -c 'print(bytearray.fromhex("50617373776F72642069732064656573"))'
bytearray(b'Password is dees')
```

3.4 Task 4: Signing a Message:



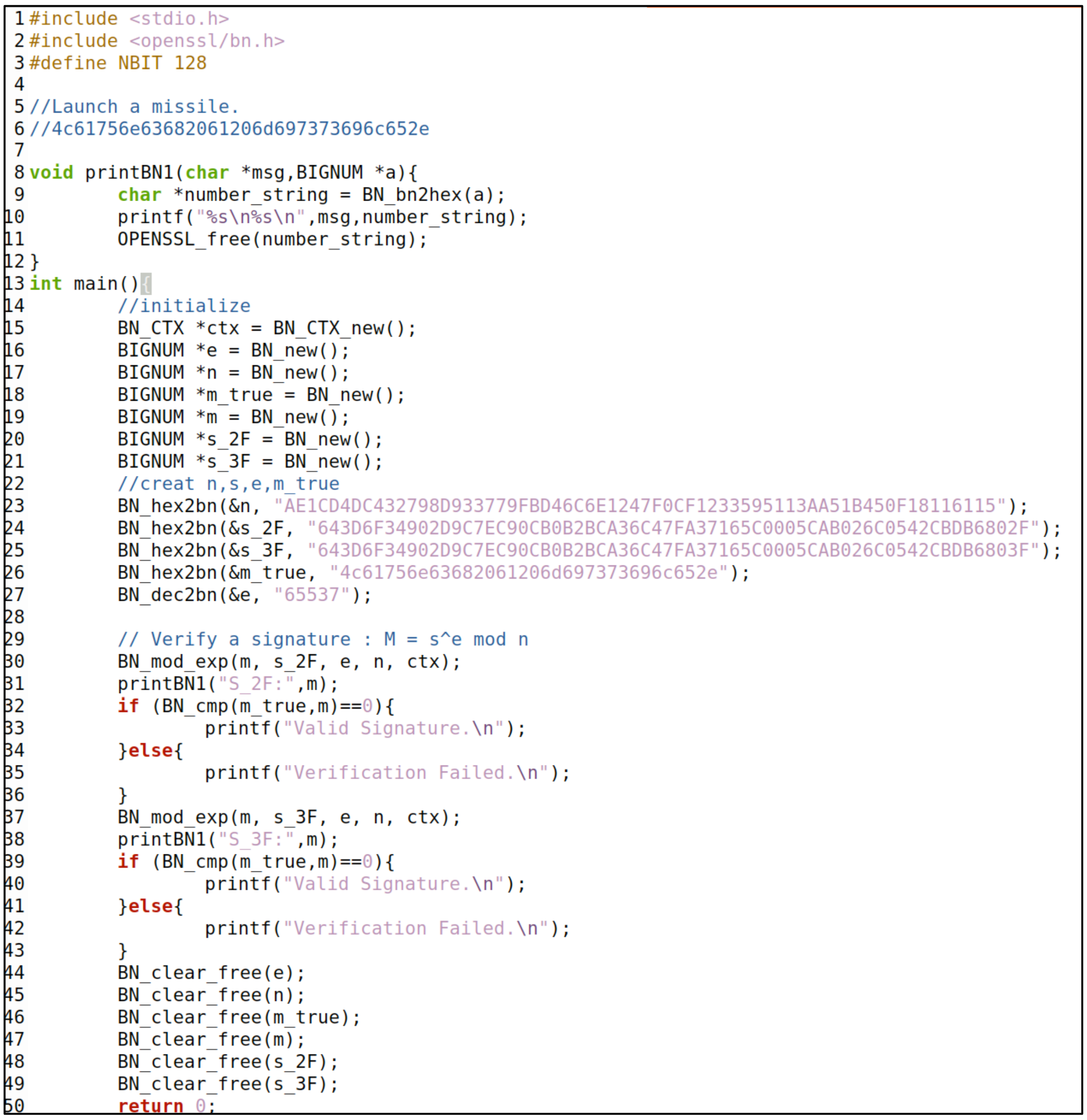
```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBIT 128

//I owe you $2000
//49206f776520796f75202432303030
//I owe you $3000
//49206f776520796f75202433303030

void printBN1(char *msg,BIGNUM *a){
        char *number_string = BN_bn2hex(a);
        printf("%s\n%s\n",msg,number_string);
        OPENSSL_free(number_string);
}
int main(){
        //initialize
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *e = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *m1 = BN_new();
        BIGNUM *m2 = BN_new();
        BIGNUM *s1 = BN_new();
        BIGNUM *s2 = BN_new();
        //creat n,d,e,m1,m2
        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
        BN_hex2bn(&m1, "49206f776520796f75202432303030");
        BN_hex2bn(&m2, "49206f776520796f75202433303030");
        BN_dec2bn(&e, "65537");

        // Signing a Message : s = M^d mod n
        BN_mod_exp(s1, m1, d, n, ctx);
        BN_mod_exp(s2, m2, d, n, ctx);
        printBN1("Signature1 = ",s1);
        printBN1("Signature2 = ",s2);

        BN_clear_free(e);
        BN_clear_free(n);
        BN_clear_free(d);
        BN_clear_free(m1);
        BN_clear_free(m2);
        BN_clear_free(s1);
        BN_clear_free(s2);
        return 0;
}
```



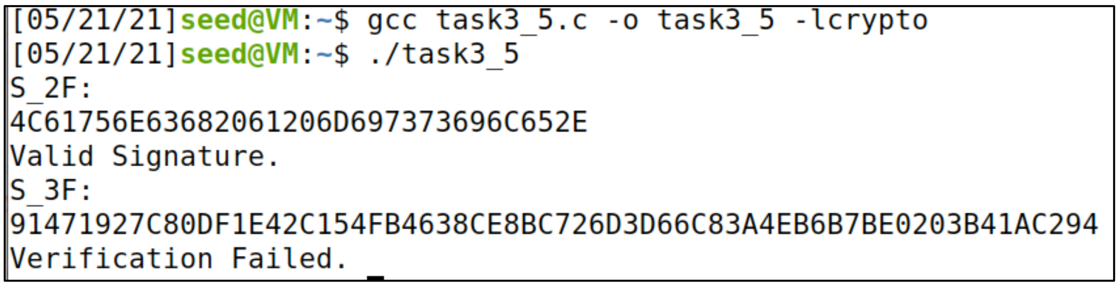
```
[05/21/21]seed@VM:~$ gcc task3_4.c -o task3_4 -lcrypto
[05/21/21]seed@VM:~$ ./task3_4
Signature1 =
80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82
Signature2 =
04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2EB
[05/21/21]seed@VM:~$ 
```

3.5 Task 5: Verifying a Signature:

驗證 Signature 很簡單，我們使用 BN_cmp(a,b)的 function，當 a 和 b 相等時則回傳 0 否則回傳-1，並且分別將內容印出來發現結果大不同。



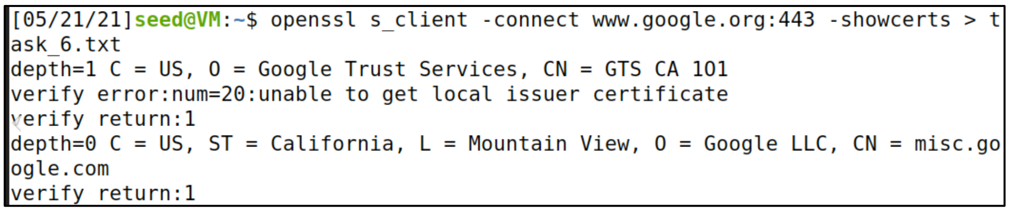
```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBIT 128

//Launch a missile.
//4c61756e63682061206d697373696c652e

void printBN1(char *msg,BIGNUM *a){
        char *number_string = BN_bn2hex(a);
        printf("%s\n%s\n",msg,number_string);
        OPENSSL_free(number_string);
}
int main(){
        //initialize
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *e = BN_new();
        BIGNUM *n = BN_new();
        BIGNUM *m_true = BN_new();
        BIGNUM *m = BN_new();
        BIGNUM *s_2F = BN_new();
        BIGNUM *s_3F = BN_new();
        //creat n,s,e,m_true
        BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
        BN_hex2bn(&s_2F, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
        BN_hex2bn(&s_3F, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
        BN_hex2bn(&m_true, "4c61756e63682061206d697373696c652e");
        BN_dec2bn(&e, "65537");

        // Verify a signature : M = s^e mod n
        BN_mod_exp(m, s_2F, e, n, ctx);
        printBN1("S_2F:",m);
        if (BN_cmp(m_true,m)==0){
                printf("Valid Signature.\n");
        }else{
                printf("Verification Failed.\n");
        }
        BN_mod_exp(m, s_3F, e, n, ctx);
        printBN1("S_3F:",m);
        if (BN_cmp(m_true,m)==0){
                printf("Valid Signature.\n");
        }else{
                printf("Verification Failed.\n");
        }
        BN_clear_free(e);
        BN_clear_free(n);
        BN_clear_free(m_true);
        BN_clear_free(m);
        BN_clear_free(s_2F);
        BN_clear_free(s_3F);
        return 0;
```



```
[05/21/21]seed@VM:~$ gcc task3_5.c -o task3_5 -lcrypto
[05/21/21]seed@VM:~$ ./task3_5
S_2F:
4C61756E63682061206D697373696C652E
Valid Signature.
S_3F:
91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
Verification Failed.
```

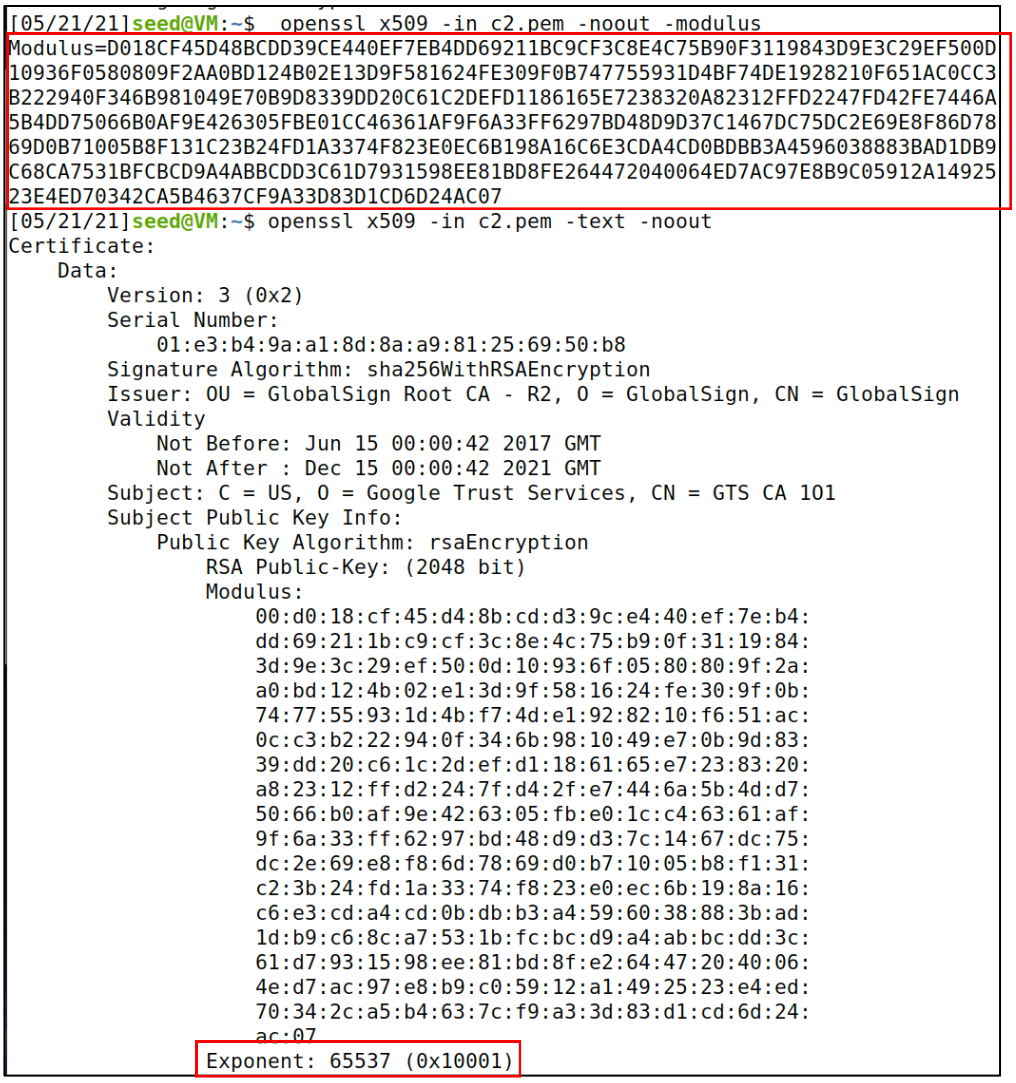
3.6 Task 6: Manually Verifying an X.509 Certificate:

Step1:下載由 Google Trust Services 簽發的 X.509 Certificate，並存入 task_6.txt，且分別將證書分成兩部份存放 c1.pem 和 c2.pem



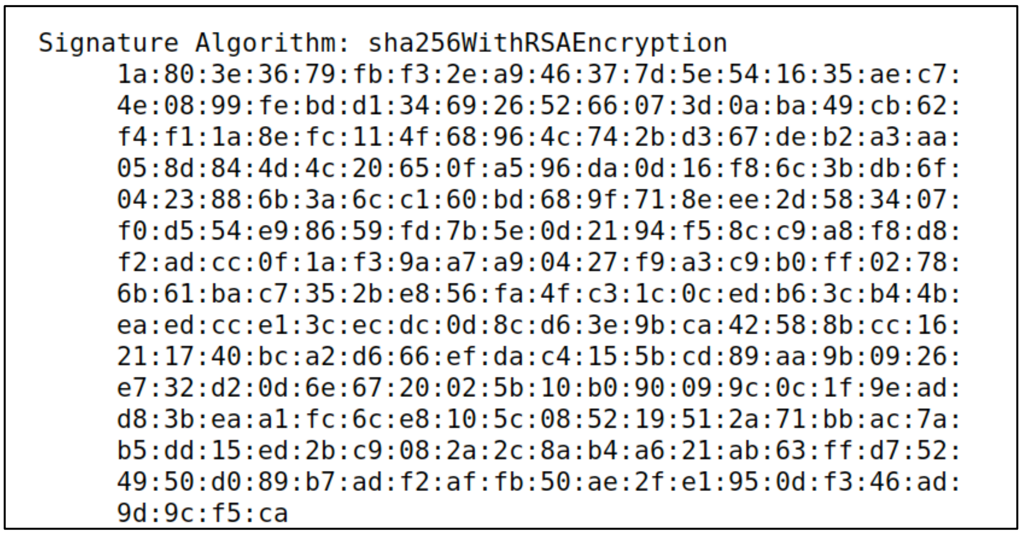
```
[05/21/21]seed@VM:~$ openssl s_client -connect www.google.org:443 -showcerts > t
ask_6.txt
depth=1 C = US, O = Google Trust Services, CN = GTS CA 101
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O = Google LLC, CN = misc.go
ogle.com
verify return:1
```

Step2:找出第二部分 CA 證書的 n 和 e，使用 lab 給予的指令可以輕鬆找到。



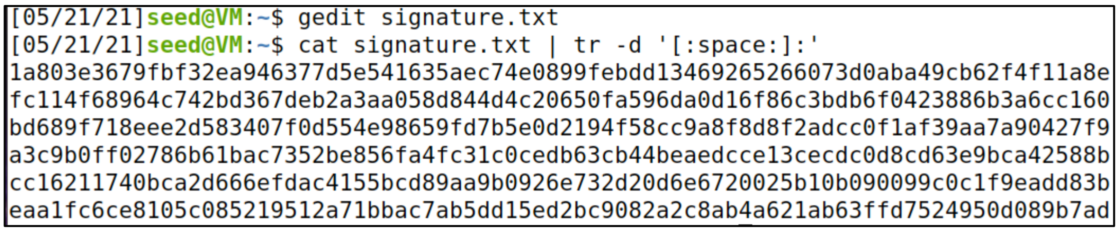
```
[05/21/21]seed@VM:~$  openssl x509 -in c2.pem -noout -modulus
Modulus=D018CF45D48BCDD39CE440EF7EB4DD69211BC9CF3C8E4C75B90F3119843D9E3C29EF500D
10936F0580809F2AA0BD124B02E13D9F581624FE309F0B747755931D4BF74DE1928210F651AC0CC3
B222940F346B981049E70B9D8339DD20C61C2DEFD1186165E7238320A82312FFD2247FD42FE7446A
5B4DD75066B0AF9E426305FBE01CC46361AF9F6A33FF6297BD48D9D37C1467DC75DC2E69E8F86D78
69D0B71005B8F131C23B24FD1A3374F823E0EC6B198A16C6E3CDA4CD0BDBB3A4596038883BAD1DB9
C68CA7531BFCBCD9A4ABBCDD3C61D7931598EE81BD8FE264472040064ED7AC97E8B9C05912A14925
23E4ED70342CA5B4637CF9A33D83D1CD6D24AC07
[05/21/21]seed@VM:~$ openssl x509 -in c2.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            01:e3:b4:9a:a1:8d:8a:a9:81:25:69:50:b8
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: OU = GlobalSign Root CA - R2, O = GlobalSign, CN = GlobalSign
        Validity
            Not Before: Jun 15 00:00:42 2017 GMT
            Not After : Dec 15 00:00:42 2021 GMT
        Subject: C = US, O = Google Trust Services, CN = GTS CA 1O1
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:d0:18:cf:45:d4:8b:cd:d3:9c:e4:40:ef:7e:b4:
                    dd:69:21:1b:c9:cf:3c:8e:4c:75:b9:0f:31:19:84:
                    3d:9e:3c:29:ef:50:0d:10:93:6f:05:80:80:9f:2a:
                    a0:bd:12:4b:02:e1:3d:9f:58:16:24:fe:30:9f:0b:
                    74:77:55:93:1d:4b:f7:4d:e1:92:82:10:f6:51:ac:
                    0c:c3:b2:22:94:0f:34:6b:98:10:49:e7:0b:9d:83:
                    39:dd:20:c6:1c:2d:ef:d1:18:61:65:e7:23:83:20:
                    a8:23:12:ff:d2:24:7f:d4:2f:e7:44:6a:5b:4d:d7:
                    50:66:b0:af:9e:42:63:05:fb:e0:1c:c4:63:61:af:
                    9f:6a:33:ff:62:97:bd:48:d9:d3:7c:14:67:dc:75:
                    dc:2e:69:e8:f8:6d:78:69:d0:b7:10:05:b8:f1:31:
                    c2:3b:24:fd:1a:33:74:f8:23:e0:ec:6b:19:8a:16:
                    c6:e3:cd:a4:cd:0b:db:b3:a4:59:60:38:88:3b:ad:
                    1d:b9:c6:8c:a7:53:1b:fc:bc:d9:a4:ab:bc:dd:3c:
                    61:d7:93:15:98:ee:81:bd:8f:e2:64:47:20:40:06:
                    4e:d7:ac:97:e8:b9:c0:59:12:a1:49:25:23:e4:ed:
                    70:34:2c:a5:b4:63:7c:f9:a3:3d:83:d1:cd:6d:24:
                    ac:07
                Exponent: 65537 (0x10001)
```

Step3:



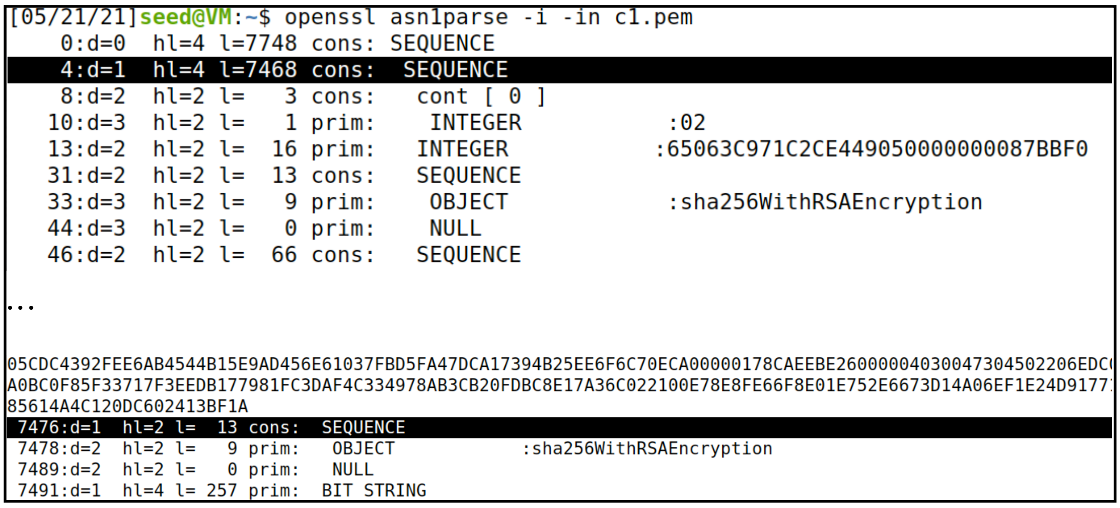
```
 Signature Algorithm: sha256WithRSAEncryption
      1a:80:3e:36:79:fb:f3:2e:a9:46:37:7d:5e:54:16:35:ae:c7:
      4e:08:99:fe:bd:d1:34:69:26:52:66:07:3d:0a:ba:49:cb:62:
      f4:f1:1a:8e:fc:11:4f:68:96:4c:74:2b:d3:67:de:b2:a3:aa:
      05:8d:84:4d:4c:20:65:0f:a5:96:da:0d:16:f8:6c:3b:db:6f:
      04:23:88:6b:3a:6c:c1:60:bd:68:9f:71:8e:ee:2d:58:34:07:
      f0:d5:54:e9:86:59:fd:7b:5e:0d:21:94:f5:8c:c9:a8:f8:d8:
      f2:ad:cc:0f:1a:f3:9a:a7:a9:04:27:f9:a3:c9:b0:ff:02:78:
      6b:61:ba:c7:35:2b:e8:56:fa:4f:c3:1c:0c:ed:b6:3c:b4:4b:
      ea:ed:cc:e1:3c:ec:dc:0d:8c:d6:3e:9b:ca:42:58:8b:cc:16:
      21:17:40:bc:a2:d6:66:ef:da:c4:15:5b:cd:89:aa:9b:09:26:
      e7:32:d2:0d:6e:67:20:02:5b:10:b0:90:09:9c:0c:1f:9e:ad:
      d8:3b:ea:a1:fc:6c:e8:10:5c:08:52:19:51:2a:71:bb:ac:7a:
      b5:dd:15:ed:2b:c9:08:2a:2c:8a:b4:a6:21:ab:63:ff:d7:52:
      49:50:d0:89:b7:ad:f2:af:fb:50:ae:2f:e1:95:0d:f3:46:ad:
      9d:9c:f5:ca
```

接著我們要把空白和 : 處理一下，因為我們等等要使用這個 signature



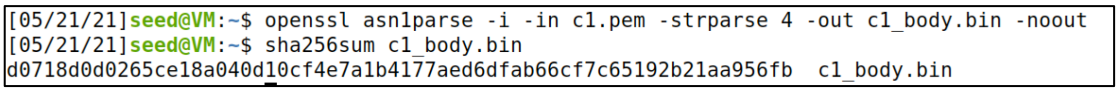
```
[05/21/21]seed@VM:~$ gedit signature.txt
[05/21/21]seed@VM:~$ cat signature.txt | tr -d '[:space:]:'
1a803e3679fbf32ea946377d5e541635aec74e0899febdd13469265266073d0aba49cb62f4f11a8e
fc114f68964c742bd367deb2a3aa058d844d4c20650fa596da0d16f86c3bdb6f0423886b3a6cc160
bd689f718eee2d583407f0d554e98659fd7b5e0d2194f58cc9a8f8d8f2adcc0f1af39aa7a90427f9
a3c9b0ff02786b61bac7352be856fa4fc31c0cedb63cb44beaedcce13cecdc0d8cd63e9bca42588b
cc16211740bca2d666efdac4155bcd89aa9b0926e732d20d6e6720025b10b090099c0c1f9eadd83b
eaa1fc6ce8105c085219512a71bbac7ab5dd15ed2bc9082a2c8ab4a621ab63ffd7524950d089b7ad
```

Step4:因為如果要驗證簽名我們必須知道在整個證書中哪一塊是簽名時用來作 hash 的部分，才能做驗證，透過 lab 給的 command 我們可以利用證書解析的方式來找到整個證書中簽名的那塊



```
[05/21/21]seed@VM:~$ openssl asn1parse -i -in c1.pem
    0:d=0  hl=4 l=7748 cons: SEQUENCE
    4:d=1  hl=4 l=7468 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           :65063C971C2CE449050000000087BBF0
   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=  66 cons:   SEQUENCE

...

05CDC4392FEE6AB4544B15E9AD456E61037FBD5FA47DCA17394B25EE6F6C70ECA00000178CAEEBE26000004030047304502206EDC
A0BC0F85F33717F3EEDB177981FC3DAF4C334978AB3CB20FDBC8E17A36C022100E78E8FE66F8E01E752E6673D14A06EF1E24D9177
85614A4C120DC602413BF1A
 7476:d=1  hl=2 l=  13 cons:  SEQUENCE
 7478:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 7489:d=2  hl=2 l=   0 prim:   NULL
 7491:d=1  hl=4 l= 257 prim:  BIT STRING
```

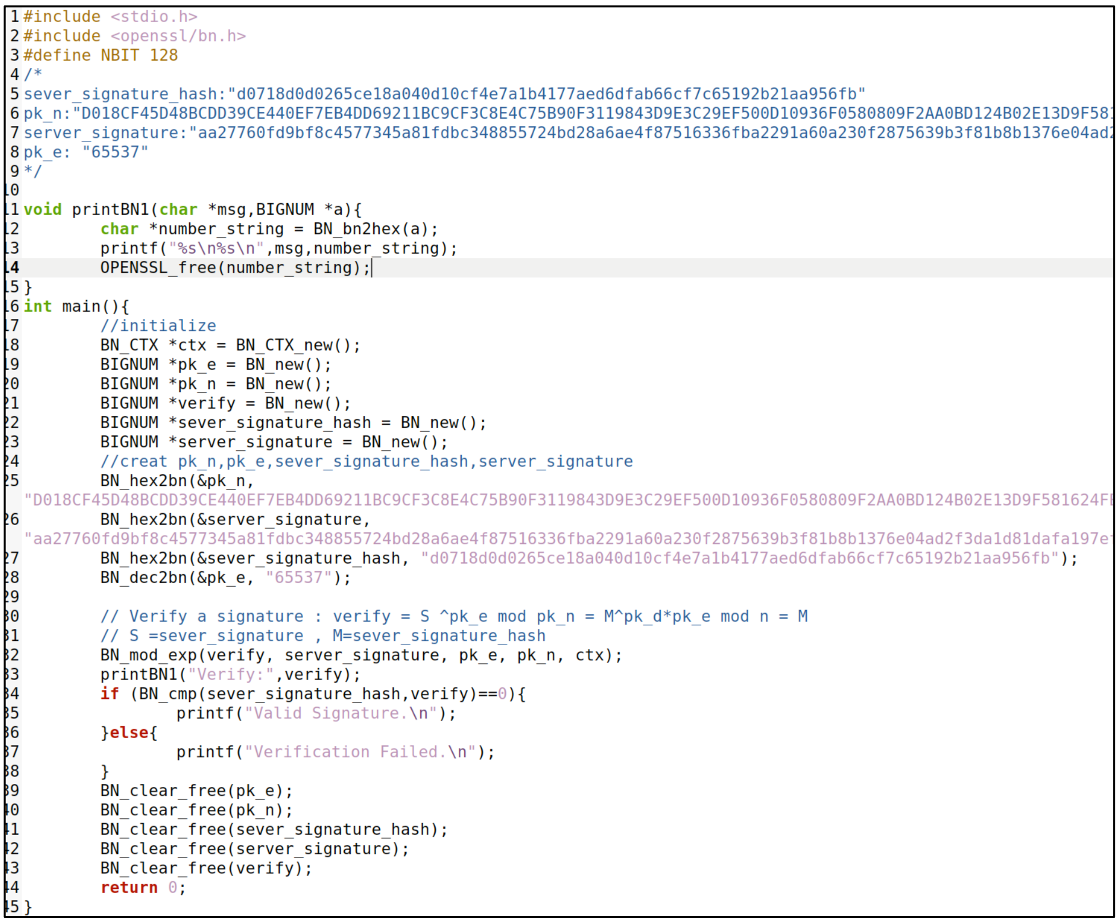
certificate body 是從 4(都是從 4 開始)到 7476，而簽名的部分則是從 7476 到最後，所以我們利用 lab 給的 command 可以把這兩塊一併擷取下來並且將其透過 sha256 hash 之後所得到的結果。



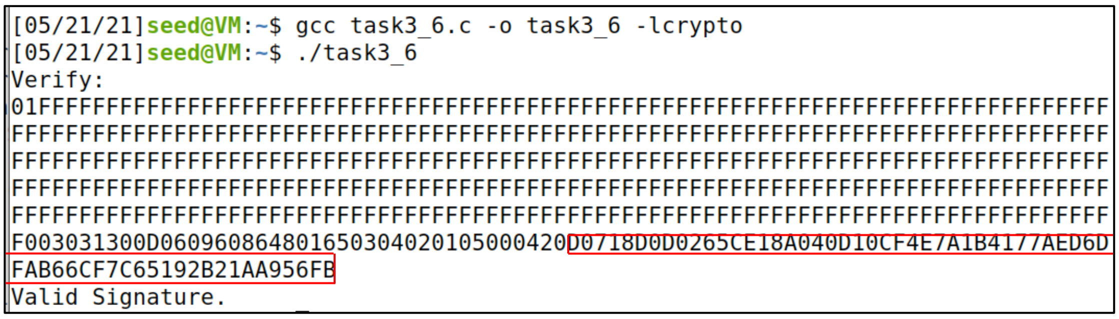
```
[05/21/21]seed@VM:~$ openssl asn1parse -i -in c1.pem -strparse 4 -out c1_body.bin -noout
[05/21/21]seed@VM:~$ sha256sum c1_body.bin
d0718d0d0265ce18a040d10cf4e7a1b4177aed6dfab66cf7c65192b21aa956fb  c1_body.bin
```

Step5:

根據以上步驟我們得到 sever_signature_hash、server_signature、pk_e、pk_n，分別用 M、S、e、n 表示，計算$S^e \ mod \ n = M^{de} \ mod \ n = M$，則我們可以得到驗證，但計算完此結果會夾帶一些憑證的資訊而且每個人前面所夾帶的都相同，所以如果在我的程式碼中 sever_signature_hash 加上前面那段則可以出現 valid signature 的結果，但我們也可以透過比對後半段的部分來發現我們的驗證是正確的。



```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBIT 128
/*
sever_signature_hash:"d0718d0d0265ce18a040d10cf4e7a1b4177aed6dfab66cf7c65192b21aa956fb"
pk_n:"D018CF45D48BCDD39CE440EF7EB4DD69211BC9CF3C8E4C75B90F3119843D9E3C29EF500D10936F0580809F2AA0BD124B02E13D9F58
server_signature:"aa27760fd9bf8c4577345a81fdbc348855724bd28a6ae4f87516336fba2291a60a230f2875639b3f81b8b1376e04ad
pk_e: "65537"
*/

void printBN1(char *msg,BIGNUM *a){
        char *number_string = BN_bn2hex(a);
        printf("%s\n%s\n",msg,number_string);
        OPENSSL_free(number_string);
}
int main(){
        //initialize
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *pk_e = BN_new();
        BIGNUM *pk_n = BN_new();
        BIGNUM *verify = BN_new();
        BIGNUM *sever_signature_hash = BN_new();
        BIGNUM *server_signature = BN_new();
        //creat pk_n,pk_e,sever_signature_hash,server_signature
        BN_hex2bn(&pk_n,
"D018CF45D48BCDD39CE440EF7EB4DD69211BC9CF3C8E4C75B90F3119843D9E3C29EF500D10936F0580809F2AA0BD124B02E13D9F581624F
        BN_hex2bn(&server_signature,
"aa27760fd9bf8c4577345a81fdbc348855724bd28a6ae4f87516336fba2291a60a230f2875639b3f81b8b1376e04ad2f3da1d81dafa197e
        BN_hex2bn(&sever_signature_hash, "d0718d0d0265ce18a040d10cf4e7a1b4177aed6dfab66cf7c65192b21aa956fb");
        BN_dec2bn(&pk_e, "65537");

        // Verify a signature : verify = S ^pk_e mod pk_n = M^pk_d*pk_e mod n = M
        // S =sever_signature , M=sever_signature_hash
        BN_mod_exp(verify, server_signature, pk_e, pk_n, ctx);
        printBN1("Verify:",verify);
        if (BN_cmp(sever_signature_hash,verify)==0){
                printf("Valid Signature.\n");
        }else{
                printf("Verification Failed.\n");
        }
        BN_clear_free(pk_e);
        BN_clear_free(pk_n);
        BN_clear_free(sever_signature_hash);
        BN_clear_free(server_signature);
        BN_clear_free(verify);
        return 0;
}
```



```
[05/21/21]seed@VM:~$ gcc task3_6.c -o task3_6 -lcrypto
[05/21/21]seed@VM:~$ ./task3_6
Verify:
01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
F003031300D060960864801650304020105000420D0718D0D0265CE18A040D10CF4E7A1B4177AED6D
FAB66CF7C65192B21AA956FB
Valid Signature.
```