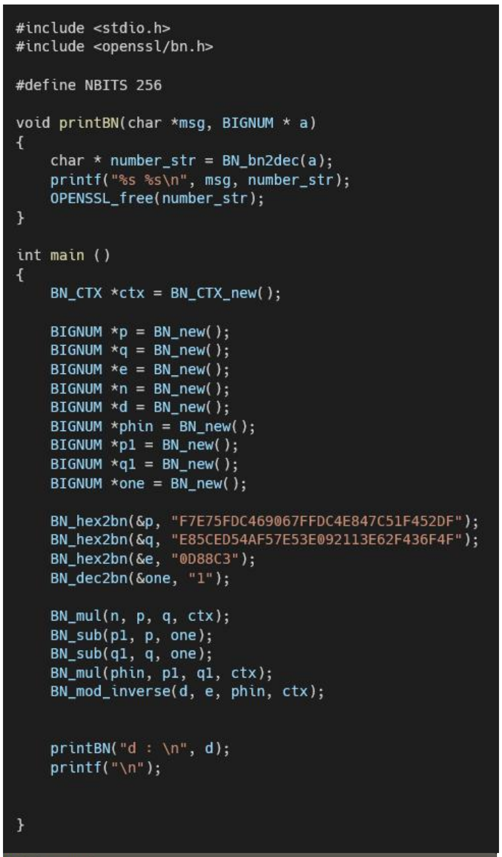
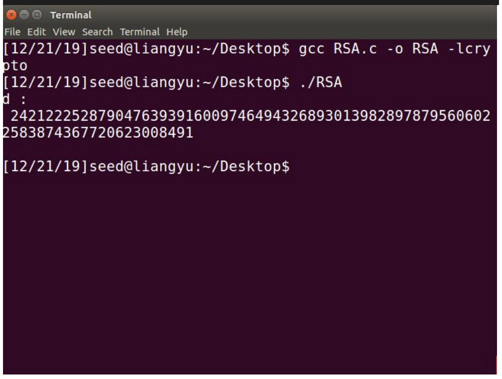
RSA Public-Key Encryption and Signature Lab

Liangyu W

Task 1: Deriving the Private Key

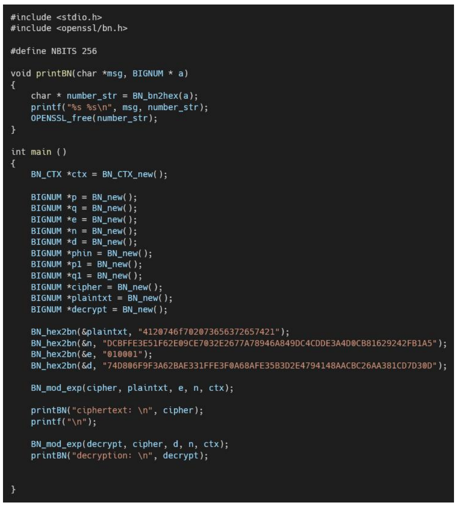


```c
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

void printBN(char *msg, BIGNUM * a)
{
    char * number_str = BN_bn2dec(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main ()
{
    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 *phin = BN_new();
    BIGNUM *p1 = BN_new();
    BIGNUM *q1 = BN_new();
    BIGNUM *one = BN_new();

    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");
    BN_dec2bn(&one, "1");

    BN_mul(n, p, q, ctx);
    BN_sub(p1, p, one);
    BN_sub(q1, q, one);
    BN_mul(phin, p1, q1, ctx);
    BN_mod_inverse(d, e, phin, ctx);


    printBN("d : \n", d);
    printf("\n");


 }
```

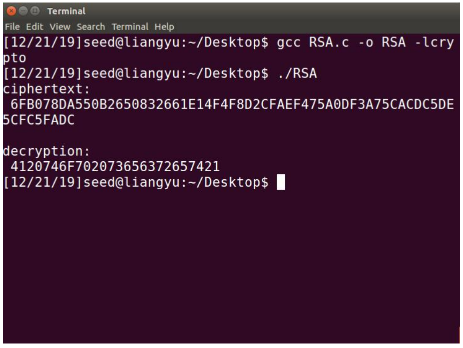


```
[12/21/19]seed@liangyu:~/Desktop$ gcc RSA.c -o RSA -lcry
pto
[12/21/19]seed@liangyu:~/Desktop$ ./RSA
d :
 24212225287904763939160097464943268930139828978795606
02
2583874367720623008491

[12/21/19]seed@liangyu:~/Desktop$
```

Task 2: Encrypting a Message

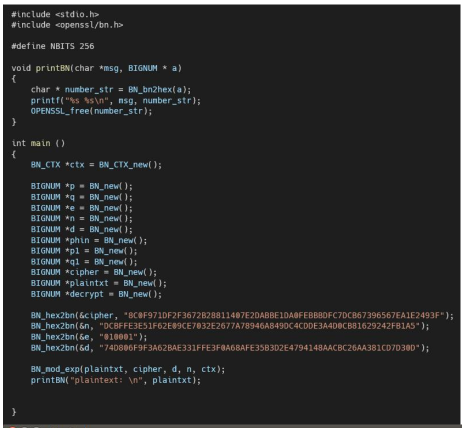


```
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

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();

    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *phin = BN_new();
    BIGNUM *p1 = BN_new();
    BIGNUM *q1 = BN_new();
    BIGNUM *cipher = BN_new();
    BIGNUM *plaintxt = BN_new();
    BIGNUM *decrypt = BN_new();

    BN_hex2bn(&plaintxt, "4120746f702073656372657421");
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    BN_mod_exp(cipher, plaintxt, e, n, ctx);

    printBN("ciphertext: \n", cipher);
    printf("\n");

    BN_mod_exp(decrypt, cipher, d, n, ctx);
    printBN("decryption: \n", decrypt);


}
```

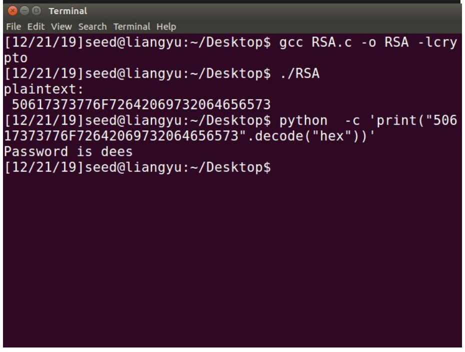


```
[12/21/19]seed@liangyu:~/Desktop$ gcc RSA.c -o RSA -lcry
pto
[12/21/19]seed@liangyu:~/Desktop$ ./RSA
ciphertext:
 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE
5CFC5FADC

decryption:
 4120746F702073656372657421
[12/21/19]seed@liangyu:~/Desktop$ 
```

Task 3: Decrypting a Message

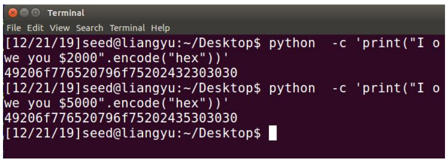


```
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

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();

    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *phin = BN_new();
    BIGNUM *p1 = BN_new();
    BIGNUM *q1 = BN_new();
    BIGNUM *cipher = BN_new();
    BIGNUM *plaintxt = BN_new();
    BIGNUM *decrypt = BN_new();

    BN_hex2bn(&cipher, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    BN_mod_exp(plaintxt, cipher, d, n, ctx);
    printBN("plaintext: \n", plaintxt);


}
```

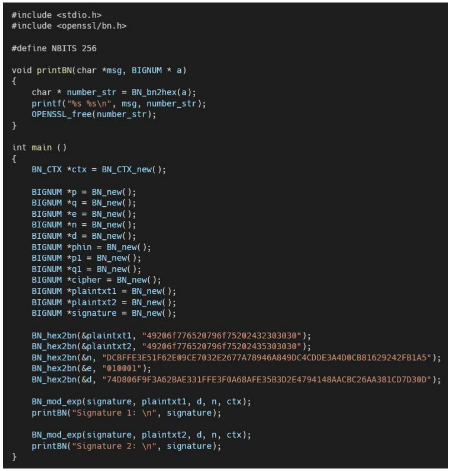


```
[12/21/19]seed@liangyu:~/Desktop$ gcc RSA.c -o RSA -lcry
pto
[12/21/19]seed@liangyu:~/Desktop$ ./RSA
plaintext:
 50617373776F72642069732064656573
[12/21/19]seed@liangyu:~/Desktop$ python  -c 'print("506
17373776F72642069732064656573".decode("hex"))'
Password is dees
[12/21/19]seed@liangyu:~/Desktop$
```

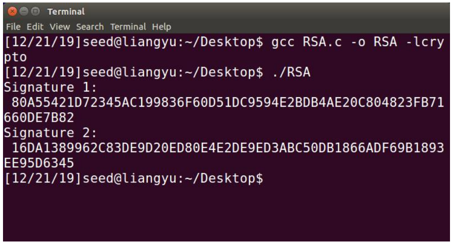
Task 4: Signing a Message



```
[12/21/19]seed@liangyu:~/Desktop$ python  -c 'print("I o
we you $2000".encode("hex"))'
49206f776520796f75202432303030
[12/21/19]seed@liangyu:~/Desktop$ python  -c 'print("I o
we you $5000".encode("hex"))'
49206f776520796f75202435303030
[12/21/19]seed@liangyu:~/Desktop$ 
```



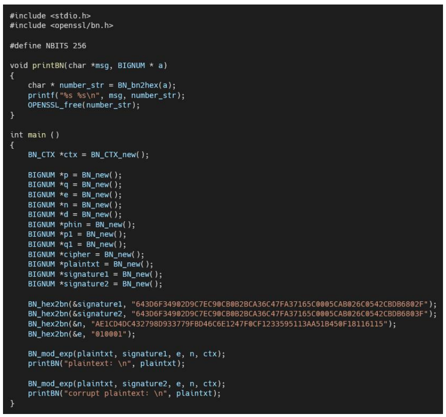
```
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

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();

    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *phin = BN_new();
    BIGNUM *p1 = BN_new();
    BIGNUM *q1 = BN_new();
    BIGNUM *cipher = BN_new();
    BIGNUM *plaintxt1 = BN_new();
    BIGNUM *plaintxt2 = BN_new();
    BIGNUM *signature = BN_new();

    BN_hex2bn(&plaintxt1, "49206f776520796f75202432303030");
    BN_hex2bn(&plaintxt2, "49206f776520796f75202435303030");
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    BN_mod_exp(signature, plaintxt1, d, n, ctx);
    printBN("Signature 1: \n", signature);

    BN_mod_exp(signature, plaintxt2, d, n, ctx);
    printBN("Signature 2: \n", signature);
}
```



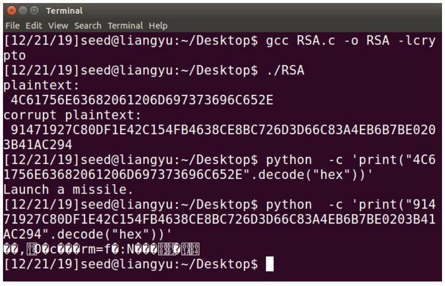
```
[12/21/19]seed@liangyu:~/Desktop$ gcc RSA.c -o RSA -lcry
pto
[12/21/19]seed@liangyu:~/Desktop$ ./RSA
Signature 1:
 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71
660DE7B82
Signature 2:
 16DA1389962C83DE9D20ED80E4E2DE9ED3ABC50DB1866ADF69B1893
EE95D6345
[12/21/19]seed@liangyu:~/Desktop$
```

Small change in the plain-text caused a major change in the signature, therefore RSA digital signature has the avalanche effect.

Task 5: Verifying a Signature



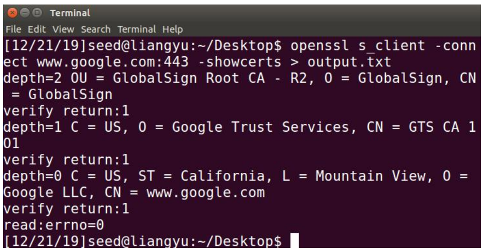
```
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

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();

    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *phin = BN_new();
    BIGNUM *p1 = BN_new();
    BIGNUM *q1 = BN_new();
    BIGNUM *cipher = BN_new();
    BIGNUM *plaintxt = BN_new();
    BIGNUM *signature1 = BN_new();
    BIGNUM *signature2 = BN_new();

    BN_hex2bn(&signature1, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    BN_hex2bn(&signature2, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_hex2bn(&e, "010001");

    BN_mod_exp(plaintxt, signature1, e, n, ctx);
    printBN("plaintext: \n", plaintxt);

    BN_mod_exp(plaintxt, signature2, e, n, ctx);
    printBN("corrupt plaintext: \n", plaintxt);
}
```



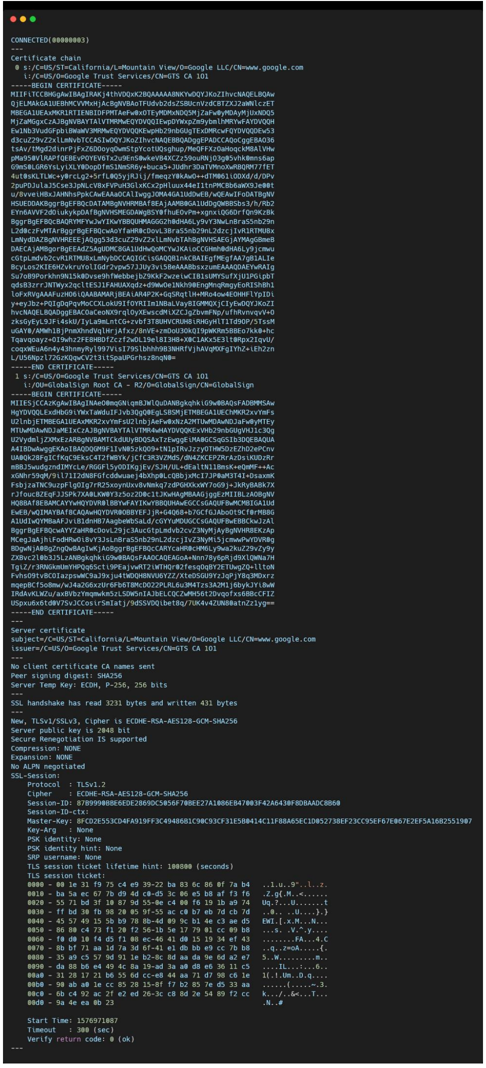
```
[12/21/19]seed@liangyu:~/Desktop$ gcc RSA.c -o RSA -lcry
pto
[12/21/19]seed@liangyu:~/Desktop$ ./RSA
plaintext:
 4C61756E63682061206D697373696C652E
corrupt plaintext:
 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE020
3B41AC294
[12/21/19]seed@liangyu:~/Desktop$ python  -c 'print("4C6
1756E63682061206D697373696C652E".decode("hex"))'
Launch a missile.
[12/21/19]seed@liangyu:~/Desktop$ python  -c 'print("914
71927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41
AC294".decode("hex"))'
��,▯D�c���rm=f�:N���▯▯�▯▯
[12/21/19]seed@liangyu:~/Desktop$ 
```

Even one bit of change in the signature will cause the verification process to fail.

Task 6: Manually Verifying an X.509 Certificate



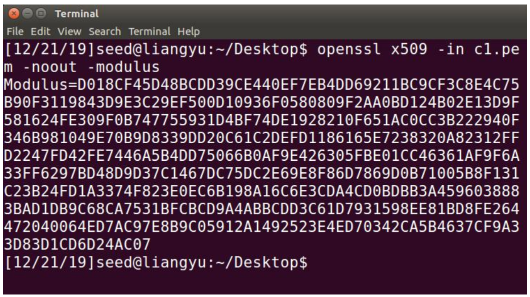
```
[12/21/19]seed@liangyu:~/Desktop$ openssl s_client -conn
ect www.google.com:443 -showcerts > output.txt
depth=2 OU = GlobalSign Root CA - R2, O = GlobalSign, CN
 = GlobalSign
verify return:1
depth=1 C = US, O = Google Trust Services, CN = GTS CA 1
O1
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O =
Google LLC, CN = www.google.com
verify return:1
read:errno=0
[12/21/19]seed@liangyu:~/Desktop$
```



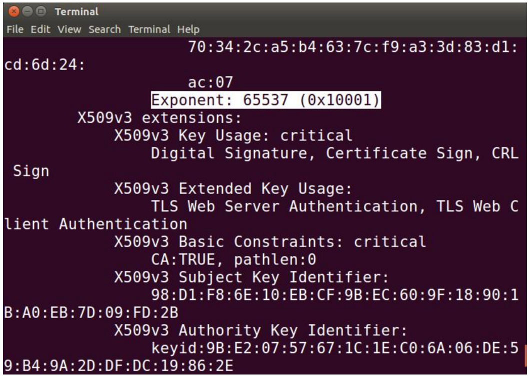
```
CONNECTED(00000003)
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google LLC/CN=www.google.com
   i:/C=US/O=Google Trust Services/CN=GTS CA 1O1
-----BEGIN CERTIFICATE-----
MIIFiTCCBHGgAwIBAgIRAKj4thVDQxK2BQAAAAA8NKYwDQYJKoZIhvcNAQELBQAw
QjELMAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczET
MBEGA1UEAxMKR1RTIENBIDFPMTAeFw0xOTEyMDMxNDQ5MjZaFw0yMDAyMjUxNDQ5
MjZaMGgxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQH
Ew1Nb3VudGFpbiBWaWV3MRMwEQYDVQQKEwpHb29nbGUgTExDMRcwFQYDVQQDEw53
d3cuZ29vZ2xlLmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAO36
tsAv/tMgd2dinrPjFxZ6DOoyqOwmStpYcotUQsghup/MeQFFXzOaHoqckM8AlVHw
pMa95OVlRAPfQEBEvPOYEV6Tx2u9EnS0wkeVB4XCZz59ouRNjO3g05vhk0mns6ap
G9mS0LGR6YsLyiXLY0DopDfmS1NmSR6y+buca5+JUdhr3DaTVMnoXwRBQRM77fET
4ut0sKLTLWc+y0rcLg2+5rfL0Q5yjRJij/fmeqzY0kAwO++dTM061iODXd/d/DPv
2puPDJulaJ5Cse3JpNLcV8xFVPuH3GlxKCx2pHluux44eI1tnPMCBb6aWX9Je00t
u/8vveiHBxJAHNhsPpkCAwEAAaOCAlIwggJOMA4GA1UdDwEB/wQEAwIFoDATBgNV
HSUEDDAKBggrBgEFBQcDATAMBgNVHRMBAf8EAjAAMB0GA1UdDgQWBBSbs3/h/Rb2
EYn6AVVF2dOiukykpDAfBgNVHSMEGDAWgBSY0fhuEOvPm+xgnxiQG6DrfQn9KzBk
BggrBgEFBQcBAQRYMFYwJwYIKwYBBQUHMAGGG2h0dHA6Ly9vY3NwLnBraS5nb29n
L2d0czFvMTArBggrBgEFBQcwAoYfaHR0cDovL3BraS5nb29nL2dzcjIvR1RTMU8x
LmNydDAZBgNVHREEEjAQgg53d3cuZ29vZ2xlLmNvbTAhBgNVHSAEGjAYMAgGBmeB
DAECAjAMBgorBgEEAdZ5AgUDMC8GA1UdHwQoMCYwJKAioCCGHmh0dHA6Ly9jcmwu
cGtpLmdvb2cvR1RTMU8xLmNybDCCAQIGCisGAQQB1nkCBAIEgfMEgfAA7gB1ALIe
BcyLos2KIE6HZvkruYolIGdr2vpw57JJUy3vi5BeAAABbsxzumEAAAQDAEYwRAIg
Su7oB9Porkhn9N15k0Dvse9hfWebbejbZ9KkF2wzeiwCIB1sUMYSufXjU1PGipbT
qdsB3zrrJNTWyx2qcltESJ1FAHUAXqdz+d9WwOe1Nkh90EngMnqRmgyEoRIShBh1
loFxRVgAAAFuzHO6iQAABAMARjBEAiAR4P2K+GqSRqtlH+MRo4ow4EOHHFlYpIDi
y+eyJbz+PQIgDqPqvMoCCXLokU9IfOYRIIm1NBaLVayBIGMMQXjCIyEwDQYJKoZI
hvcNAQELBQADggEBACOaCeoNX9rqlOyXEwscdMiXZCJgZbvmFNp/ufhRvnvqvV+O
zksGyEyL9JFi4skU/IyLa9mLntCG+zvbf3T8UHVCRUH8iRHGyHlT1Td9OP/5TssM
uGAY0/AMWh1BjPnmXhndVqlHrjAfxz/8nVE+zmDoU3OkQI9pWKRm5BBEo7kk0+hc
Tqavqoayz+OI9whz2FE8HBDfZczf2wDL19el8I3H8+X0C1AKx5E3lt0Rpx2IqvU/
coqxWEuA6n4y43hnmyRyl997VisI79Slbhhh9B3NHRfVjhAVqMXFgIYhZ+iEh2zn
L/U56Npzl72GzKQqwCV2t3itSpaUPGrhsz8nqN0=
-----END CERTIFICATE-----
 1 s:/C=US/O=Google Trust Services/CN=GTS CA 1O1
   i:/OU=GlobalSign Root CA - R2/O=GlobalSign/CN=GlobalSign
-----BEGIN CERTIFICATE-----
MIIESjCCAzKgAwIBAgINAeO0mqGNiqmBJWlQuDANBgkqhkiG9w0BAQsFADBMMSAw
HgYDVQQLExdHbG9iYWxTaWduIFJvb3QgQ0EgLSBSMjETMBEGA1UEChMKR2xvYmFs
U2lnbjETMBEGA1UEAxMKR2xvYmFsU2lnbjAeFw0xNzA2MTUwMDAwNDJaFw0yMTEy
MTUwMDAwNDJaMEIxCzAJBgNVBAYTAlVTMR4wHAYDVQQKExVHb29nbGUgVHJ1c3Qg
U2VydmljZXMxEzARBgNVBAMTCkdUUyBDQSAxTzEwggEiMA0GCSqGSIb3DQEBAQUA
A4IBDwAwggEKAoIBAQDQGM9F1IvN05zkQO9+tN1pIRvJzzyOTHW5DzEZhD2ePCnv
UA0Qk28FgICfKqC9EksC4T2fWBYk/jCfC3R3VZMdS/dN4ZKCEPZRrAzDsiKUDzRr
mBBJ5wudgzndIMYcLe/RGGFl5yODIKgjEv/SJH/UL+dEaltN11BmsK+eQmMF++Ac
xGNhr59qM/9il71I2dN8FGfcddwuaej4bXhp0LcQBbjxMcI7JP0aM3T4I+DsaxmK
FsbjzaTNC9uzpFlgOIg7rR25xoynUxv8vNmkq7zdPGHXkxWY7oG9j+JkRyBABk7X
rJfoucBZEqFJJSPk7XA0LKW0Y3z5oz2D0c1tJKwHAgMBAAGjggEzMIIBLzAOBgNV
HQ8BAf8EBAMCAYYwHQYDVR0lBBYwFAYIKwYBBQUHAwEGCCsGAQUFBwMCMBIGA1Ud
EwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFJjR+G4Q68+b7GCfGJAboOt9Cf0rMB8G
A1UdIwQYMBaAFJviB1dnHB7AagbeWbSaLd/cGYYuMDUGCCsGAQUFBwEBBCkwJzAl
BggrBgEFBQcwAYYZaHR0cDovL29jc3AucGtpLmdvb2cvZ3NyMjAyBgNVHR8EKzAp
MCegJaAjhiFodHRwOi8vY3JsLnBraS5nb29nL2dzcjIvZ3NyMi5jcmwwPwYDVR0g
BDgwNjA0BgZngQwBAgIwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly9wa2kuZ29vZy9y
ZXBvc2l0b3J5LzANBgkqhkiG9w0BAQsFAAOCAQEAGoA+Nnn78y6pRjd9XlQWNa7H
TgiZ/r3RNGkmUmYHPQq6Scti9PEajvwRT2iWTHQr02fesqOqBY2ETUwgZQ+lltoN
FvhsO9tvBCOIazpswWC9aJ9xju4tWDQH8NVU6YZZ/XteDSGU9YzJqPjY8q3MDxrz
mqepBCf5o8mw/wJ4a2G6xzUr6Fb6T8McDO22PLRL6u3M4Tzs3A2M1j6bykJYi8wW
IRdAvKLWZu/axBVbzYmqmwkm5zLSDW5nIAJbELCQCZwMH56t2DvqofxS6BBcCFIZ
USpxu6x6td0V7SvJCCosirSmIatj/9dSSVDQibet8q/7UK4v4ZUN80atnZz1yg==
-----END CERTIFICATE-----
---
Server certificate
subject=/C=US/ST=California/L=Mountain View/O=Google LLC/CN=www.google.com
issuer=/C=US/O=Google Trust Services/CN=GTS CA 1O1
---
No client certificate CA names sent
Peer signing digest: SHA256
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 3231 bytes and written 431 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID: 87B9990BBE6EDE2869DC5056F70BEE27A1086EB47003F42A6430F8DBAADC8B60
    Session-ID-ctx:
    Master-Key: 8FCD2E553CD4FA919FF3C49486B1C90C93CF31E5B0414C11F88A65EC1D052738EF23CC95EF67E067E2EF5A16B2551907
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 100800 (seconds)
    TLS session ticket:
    0000 - 00 1e 31 f9 75 c4 e9 39-22 ba 83 6c 86 0f 7a b4   ..1.u..9"..l..z.
    0010 - ba 5a ec 67 7b d9 4d c0-d5 3c 06 e5 b8 af f3 f6   .Z.g{.M..<......
    0020 - 55 71 bd 3f 10 87 9d 55-0e c4 00 f6 19 1b a9 74   Uq.?...U.......t
    0030 - ff bd 30 fb 98 20 05 9f-55 ac c0 b7 eb 7d cb 7d   ..0.. ..U....}.}
    0040 - 45 57 49 15 5b b9 78 8b-4d 09 9c b1 4e c3 ae d5   EWI.[.x.M...N...
    0050 - 86 80 c4 73 f1 20 f2 56-1b 5e 17 79 01 cc 09 b8   ...s. .V.^.y....
    0060 - f0 d0 10 f4 d5 f1 08 ec-46 41 d0 15 19 34 ef 43   ........FA...4.C
    0070 - 8b bf 71 aa 1d 7a 3d 6f-41 e1 db bb e9 cc 7b b8   ..q..z=oA.....{.
    0080 - 35 a9 c5 57 9d 91 1e b2-8c 8d aa da 9e 6d a2 e7   5..W.........m..
    0090 - da 88 b6 e4 49 4c 8a 19-ad 3a a0 d8 e6 36 11 c5   ....IL...:...6..
    00a0 - 31 28 17 21 b6 55 6d cc-e8 44 aa 71 d7 98 c6 1e   1(.!.Um..D.q....
    00b0 - 90 ab a0 1e cc 85 28 15-8f f7 b2 85 7e d5 33 aa   ......(.....~.3.
    00c0 - 6b c4 92 ac 2f e2 ed 26-3c c8 8d 2e 54 89 f2 cc   k.../..&<...T...
    00d0 - 9a 4e ea 0b 23                                    .N..#

    Start Time: 1576971087
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```

Create two files c1.pem and c2.pem from the above two certificates.

Extract the public key (e, n) from the issuer’s certificate:

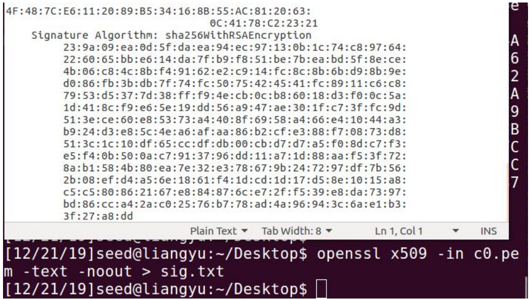


```
[12/21/19]seed@liangyu:~/Desktop$ openssl x509 -in c1.pe
m -noout -modulus
Modulus=D018CF45D48BCDD39CE440EF7EB4DD69211BC9CF3C8E4C75
B90F3119843D9E3C29EF500D10936F0580809F2AA0BD124B02E13D9F
581624FE309F0B747755931D4BF74DE1928210F651AC0CC3B222940F
346B981049E70B9D8339DD20C61C2DEFD1186165E7238320A82312FF
D2247FD42FE7446A5B4DD75066B0AF9E426305FBE01CC46361AF9F6A
33FF6297BD48D9D37C1467DC75DC2E69E8F86D7869D0B71005B8F131
C23B24FD1A3374F823E0EC6B198A16C6E3CDA4CD0BDBB3A459603888
3BAD1DB9C68CA7531BFCBCD9A4ABBCDD3C61D7931598EE81BD8FE264
472040064ED7AC97E8B9C05912A1492523E4ED70342CA5B4637CF9A3
3D83D1CD6D24AC07
[12/21/19]seed@liangyu:~/Desktop$
```



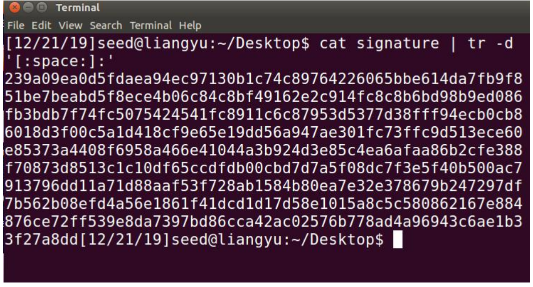
```
                        70:34:2c:a5:b4:63:7c:f9:a3:3d:83:d1:
cd:6d:24:
                        ac:07
                   Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL
 Sign
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web C
lient Authentication
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Subject Key Identifier:
                98:D1:F8:6E:10:EB:CF:9B:EC:60:9F:18:90:1
B:A0:EB:7D:09:FD:2B
            X509v3 Authority Key Identifier:
                keyid:9B:E2:07:57:67:1C:1E:C0:6A:06:DE:5
9:B4:9A:2D:DF:DC:19:86:2E
```

Extract the signature from the server's certificate:



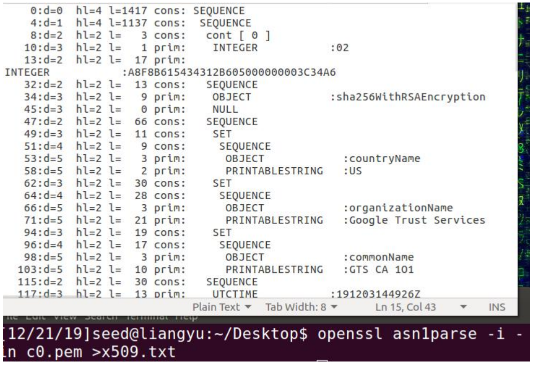
```
4F:48:7C:E6:11:20:89:B5:34:16:8B:55:AC:81:20:63:
                                0C:41:78:C2:23:21
    Signature Algorithm: sha256WithRSAEncryption
         23:9a:09:ea:0d:5f:da:ea:94:ec:97:13:0b:1c:74:c8:97:64:
         22:60:65:bb:e6:14:da:7f:b9:f8:51:be:7b:ea:bd:5f:8e:ce:
         4b:06:c8:4c:8b:f4:91:62:e2:c9:14:fc:8c:8b:6b:d9:8b:9e:
         d0:86:fb:3b:db:7f:74:fc:50:75:42:45:41:fc:89:11:c6:c8:
         79:53:d5:37:7d:38:ff:f9:4e:cb:0c:b8:60:18:d3:f0:0c:5a:
         1d:41:8c:f9:e6:5e:19:dd:56:a9:47:ae:30:1f:c7:3f:fc:9d:
         51:3e:ce:60:e8:53:73:a4:40:8f:69:58:a4:66:e4:10:44:a3:
         b9:24:d3:e8:5c:4e:a6:af:aa:86:b2:cf:e3:88:f7:08:73:d8:
         51:3c:1c:10:df:65:cc:df:db:00:cb:d7:d7:a5:f0:8d:c7:f3:
         e5:f4:0b:50:0a:c7:91:37:96:dd:11:a7:1d:88:aa:f5:3f:72:
         8a:b1:58:4b:80:ea:7e:32:e3:78:67:9b:24:72:97:df:7b:56:
         2b:08:ef:d4:a5:6e:18:61:f4:1d:cd:1d:17:d5:8e:10:15:a8:
         c5:c5:80:86:21:67:e8:84:87:6c:e7:2f:f5:39:e8:da:73:97:
         bd:86:cc:a4:2a:c0:25:76:b7:78:ad:4a:96:94:3c:6a:e1:b3:
         3f:27:a8:dd
```

```
[12/21/19]seed@liangyu:~/Desktop$ openssl x509 -in c0.pe
m -text -noout > sig.txt
[12/21/19]seed@liangyu:~/Desktop$
```



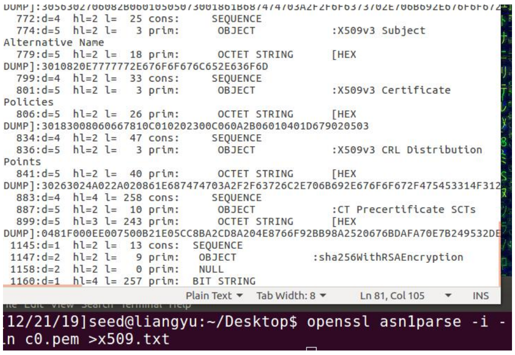
```
[12/21/19]seed@liangyu:~/Desktop$ cat signature | tr -d
'[:space:]:'
239a09ea0d5fdaea94ec97130b1c74c89764226065bbe614da7fb9f8
51be7beabd5f8ece4b06c84c8bf49162e2c914fc8c8b6bd98b9ed086
fb3bdb7f74fc5075424541fc8911c6c87953d5377d38fff94ecb0cb8
6018d3f00c5a1d418cf9e65e19dd56a947ae301fc73ffc9d513ece60
e85373a4408f6958a466e41044a3b924d3e85c4ea6afaa86b2cfe388
f70873d8513c1c10df65ccdfdb00cbd7d7a5f08dc7f3e5f40b500ac7
913796dd11a71d88aaf53f728ab1584b80ea7e32e378679b247297df
7b562b08efd4a56e1861f41dcd1d17d58e1015a8c5c580862167e884
876ce72ff539e8da7397bd86cca42ac02576b778ad4a96943c6ae1b3
3f27a8dd[12/21/19]seed@liangyu:~/Desktop$
```

Extract the body of the server's certificate:



```
    0:d=0  hl=4 l=1417 cons: SEQUENCE
    4:d=1  hl=4 l=1137 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=  17 prim:
INTEGER           :A8F8B615434312B605000000003C34A6
   32:d=2  hl=2 l=  13 cons:   SEQUENCE
   34:d=3  hl=2 l=   9 prim:    OBJECT            :sha256WithRSAEncryption
   45:d=3  hl=2 l=   0 prim:    NULL
   47:d=2  hl=2 l=  66 cons:   SEQUENCE
   49:d=3  hl=2 l=  11 cons:    SET
   51:d=4  hl=2 l=   9 cons:     SEQUENCE
   53:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   58:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
   62:d=3  hl=2 l=  30 cons:    SET
   64:d=4  hl=2 l=  28 cons:     SEQUENCE
   66:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   71:d=5  hl=2 l=  21 prim:      PRINTABLESTRING   :Google Trust Services
   94:d=3  hl=2 l=  19 cons:    SET
   96:d=4  hl=2 l=  17 cons:     SEQUENCE
   98:d=5  hl=2 l=   3 prim:      OBJECT            :commonName
  103:d=5  hl=2 l=  10 prim:      PRINTABLESTRING   :GTS CA 1O1
  115:d=2  hl=2 l=  30 cons:   SEQUENCE
  117:d=3  hl=2 l=  13 prim:    UTCTIME           :191203144926Z
```

```
[12/21/19]seed@liangyu:~/Desktop$ openssl asn1parse -i -
in c0.pem >x509.txt
```

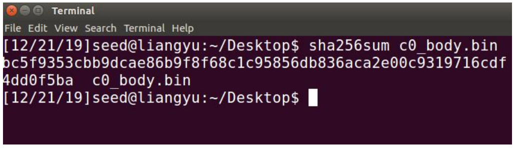


```
DUMP]:3056302706082B06010505073001861B687474703A2F2F6F6373702E706B692E676F6F672
  772:d=4  hl=2 l=  25 cons:     SEQUENCE
  774:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Subject
Alternative Name
  779:d=5  hl=2 l=  18 prim:      OCTET STRING      [HEX
DUMP]:3010820E7777772E676F6F676C652E636F6D
  799:d=4  hl=2 l=  33 cons:     SEQUENCE
  801:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Certificate
Policies
  806:d=5  hl=2 l=  26 prim:      OCTET STRING      [HEX
DUMP]:30183008060667810C010202300C060A2B06010401D679020503
  834:d=4  hl=2 l=  47 cons:     SEQUENCE
  836:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 CRL Distribution
Points
  841:d=5  hl=2 l=  40 prim:      OCTET STRING      [HEX
DUMP]:30263024A022A020861E687474703A2F2F63726C2E706B692E676F6F672F475453314F312
  883:d=4  hl=4 l= 258 cons:     SEQUENCE
  887:d=5  hl=2 l=  10 prim:      OBJECT            :CT Precertificate SCTs
  899:d=5  hl=3 l= 243 prim:      OCTET STRING      [HEX
DUMP]:0481F000EE007500B21E05CC8BA2CD8A204E8766F92BB98A2520676BDAFA70E7B249532DE
 1145:d=1  hl=2 l=  13 cons:  SEQUENCE
 1147:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 1158:d=2  hl=2 l=   0 prim:   NULL
 1160:d=1  hl=4 l= 257 prim:  BIT STRING
```

```
[12/21/19]seed@liangyu:~/Desktop$ openssl asn1parse -i -
in c0.pem >x509.txt
```

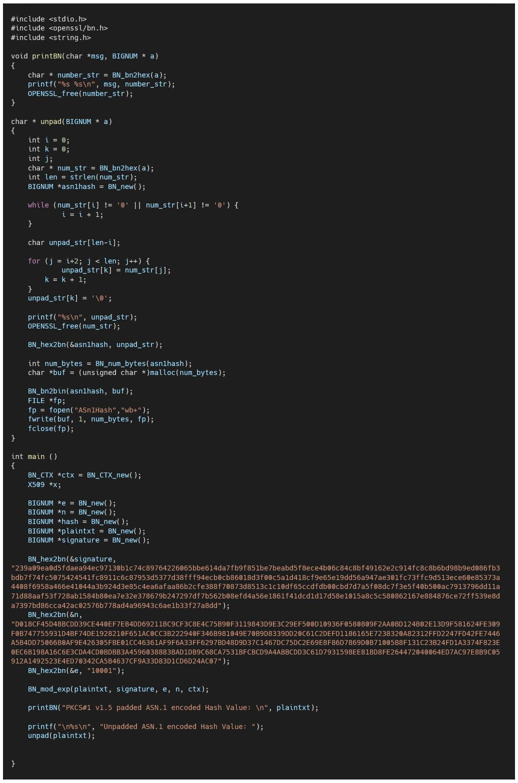
The certificate body is from offset 4 to 1137, while the signature block is from 1145 to the end of the file.

We generate the SHA256 hash value From certificate body:

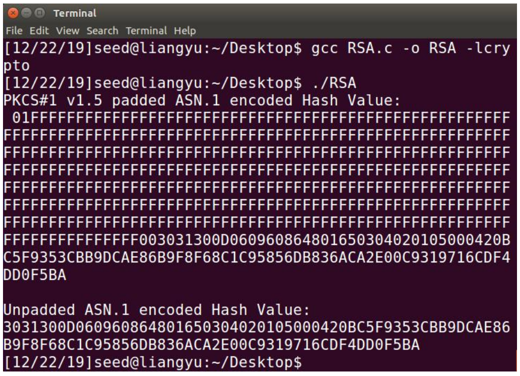


```
[12/21/19]seed@liangyu:~/Desktop$ sha256sum c0_body.bin
bc5f9353cbb9dcae86b9f8f68c1c95856db836aca2e00c9319716cdf
4dd0f5ba  c0_body.bin
[12/21/19]seed@liangyu:~/Desktop$
```

Verify the signature:

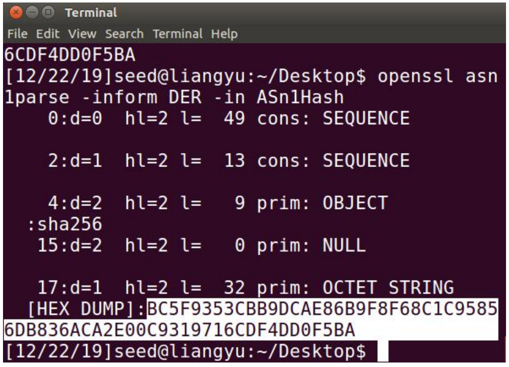


```c
#include <stdio.h>
#include <openssl/bn.h>
#include <string.h>

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

char * unpad(BIGNUM * a)
{
    int i = 0;
    int k = 0;
    int j;
    char * num_str = BN_bn2hex(a);
    int len = strlen(num_str);
    BIGNUM *asn1hash = BN_new();

    while (num_str[i] != '0' || num_str[i+1] != '0') {
            i = i + 1;
    }

    char unpad_str[len-i];

    for (j = i+2; j < len; j++) {
            unpad_str[k] = num_str[j];
        k = k + 1;
    }
    unpad_str[k] = '\0';

    printf("%s\n", unpad_str);
    OPENSSL_free(num_str);

    BN_hex2bn(&asn1hash, unpad_str);

    int num_bytes = BN_num_bytes(asn1hash);
    char *buf = (unsigned char *)malloc(num_bytes);

    BN_bn2bin(asn1hash, buf);
    FILE *fp;
    fp = fopen("ASn1Hash","wb+");
    fwrite(buf, 1, num_bytes, fp);
    fclose(fp);
}

int main ()
{
    BN_CTX *ctx = BN_CTX_new();
    X509 *x;

    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *hash = BN_new();
    BIGNUM *plaintxt = BN_new();
    BIGNUM *signature = BN_new();

    BN_hex2bn(&signature,
"239a09ea0d5fdaea94ec97130b1c74c89764226065bbe614da7fb9f851be7beabd5f8ece4b06c84c8bf49162e2c914fc8c8b6bd98b9ed086fb3
bdb7f74fc5075424541fc8911c6c87953d5377d38fff94ecb0cb86018d3f00c5a1d418cf9e65e19dd56a947ae301fc73ffc9d513ece60e85373a
4408f6958a466e41044a3b924d3e85c4ea6afaa86b2cfe388f70873d8513c1c10df65ccdfdb00cbd7d7a5f08dc7f3e5f40b500ac7913796dd11a
71d88aaf53f728ab1584b80ea7e32e378679b247297df7b562b08efd4a56e1861f41dcd1d17d58e1015a8c5c580862167e884876ce72ff539e8d
a7397bd86cca42ac02576b778ad4a96943c6ae1b33f27a8dd");
    BN_hex2bn(&n,
"D018CF45D48BCDD39CE440EF7EB4DD69211BC9CF3C8E4C75B90F3119843D9E3C29EF500D10936F0580809F2AA0BD124B02E13D9F581624FE309
F0B747755931D4BF74DE1928210F651AC0CC3B222940F346B981049E70B9D8339DD20C61C2DEFD1186165E7238320A82312FFD2247FD42FE7446
A5B4DD75066B0AF9E426305FBE01CC46361AF9F6A33FF6297BD48D9D37C1467DC75DC2E69E8F86D7869D0B71005B8F131C23B24FD1A3374F823E
0EC6B198A16C6E3CDA4CD0BDBB3A4596038883BAD1DB9C68CA7531BFCBCD9A4ABBCDD3C61D7931598EE81BD8FE264472040064ED7AC97E8B9C05
912A1492523E4ED70342CA5B4637CF9A33D83D1CD6D24AC07");
    BN_hex2bn(&e, "10001");

    BN_mod_exp(plaintxt, signature, e, n, ctx);

    printBN("PKCS#1 v1.5 padded ASN.1 encoded Hash Value: \n", plaintxt);

    printf("\n%s\n", "Unpadded ASN.1 encoded Hash Value: ");
    unpad(plaintxt);


}
```



```
[12/22/19]seed@liangyu:~/Desktop$ gcc RSA.c -o RSA -lcry
pto
[12/22/19]seed@liangyu:~/Desktop$ ./RSA
PKCS#1 v1.5 padded ASN.1 encoded Hash Value:
 01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFF003031300D060960864801650304020105000420B
C5F9353CBB9DCAE86B9F8F68C1C95856DB836ACA2E00C9319716CDF4
DD0F5BA

Unpadded ASN.1 encoded Hash Value:
3031300D060960864801650304020105000420BC5F9353CBB9DCAE86
B9F8F68C1C95856DB836ACA2E00C9319716CDF4DD0F5BA
[12/22/19]seed@liangyu:~/Desktop$
```

The decrypted hash value is encoded in ASN.1 format and then padded with PKCS#1 v1.5 padding characters. Our C program un-pads the PKCS#1 v1.5 padding and dumps the ASN.1 encoded hash value into a binary file ASn1Hash.



```
6CDF4DD0F5BA
[12/22/19]seed@liangyu:~/Desktop$ openssl asn
1parse -inform DER -in ASn1Hash
    0:d=0  hl=2 l=  49 cons: SEQUENCE

    2:d=1  hl=2 l=  13 cons: SEQUENCE

    4:d=2  hl=2 l=   9 prim: OBJECT
  :sha256
   15:d=2  hl=2 l=   0 prim: NULL

   17:d=1  hl=2 l=  32 prim: OCTET STRING
  [HEX DUMP]:BC5F9353CBB9DCAE86B9F8F68C1C9585
6DB836ACA2E00C9319716CDF4DD0F5BA
[12/22/19]seed@liangyu:~/Desktop$ 
```

Use openssl asn1parse to decode the ASn1Hash file, and the hash value obtained is the same SHA256 hash value that was previously generated from the certificate body.