

RSA Public-Key Encryption and Signature Lab Report

1. Introduction

RSA (RivestShamirAdleman) is one of the first public-key cryptosystems and is widely used for secure communication. The RSA algorithm first generates two large random prime numbers, and then use them to generate public and private key pairs, which can be used to do encryption, decryption, digital signature generation, and digital signature verification. The RSA algorithm is built upon number theories, and it can be quite easily implemented with the support of libraries. Essentially, students will be implementing the RSA algorithm using the C program language.

The lab covers the following security-related topics:

- Public-key cryptography
- The RSA algorithm and key generation
- Big number calculation
- Encryption and Decryption using RSA
- Digital signature
- X.509 certificate

Lab environment: This lab requires the openssl library. I am using Ubuntu 16.04, which is already installed on.

2. Background

The RSA algorithm involves computations on large numbers. These computations cannot be directly conducted using simple arithmetic operators in programs, because those operators can only operate on primitive SEED Labs data types. In this lab, we will use the Big Number library provided by openssl. To use this library, we will define each big number as a BIGNUM type, and then use the APIs provided by the library for various operations, such as addition, multiplication, exponentiation, modular operations, etc.

First, I create a new folder named cs458lab2. Then I write all of the lab codes in this folder.

```
1. $ mkdir cs458Lab2
```

3. Task

3.1 Task 1: Deriving the Private Key

Let p , q , and e be three prime numbers. Let $n = p \cdot q$. We will use (e, n) as the public key. Please calculate the private key d .

First, I create a Task1.c and use the code showing in the lab instructions.

```
1. $ vi task1.c
2. #include <stdio.h>
3. #include <openssl/bn.h>
4. #define NBITS 256
5.
6. void printBN(char *msg, BIGNUM *a){
7. // Convert the BIGNUM to number string
8. char * number_str = BN_bn2hex(a);
9. // Print out the number string
10. printf("%s %s\n", msg, number_str);
11. // Free the dynamically allocated memory
12. OPENSSL_free(number_str);
13. }
14.
15. int main(){
16. BN_CTX *ctx = BN_CTX_new();
17. BIGNUM *p = BN_new();
18. BIGNUM *q = BN_new();
19. BIGNUM *e = BN_new();
20. BIGNUM *d = BN_new();
21. BIGNUM *res1 = BN_new();
22. BIGNUM *res2 = BN_new();
23. BIGNUM *res3 = BN_new();
24. BIGNUM *one = BN_new();
25. // initialize p q e
26. // Assign the first large prime
27. BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
28.
29. // Assign the second large prime
30. BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
31.
32. // Assign the Modulus
33. BN_hex2bn(&e, "0D88C3");
34. BN_dec2bn(&one, "1");
35. //res1 = p-1
36. BN_sub(res1, p, one);
37. //res2 = q-1
38. BN_sub(res2, q, one);
39. //res3=res1*res2
40. BN_mul(res3, res1, res2, ctx);
41. //res=a*b mod n
42. BN_mod_inverse(d, e, res3, ctx);
43. //print BN
44. printBN("d= ",d);
45. return 0;
46. }
47.
```

First print out a big number:

```

void printBN(char *msg, BIGNUM *a){
// Convert the BIGNUM to number string
char * number_str = BN_bn2dec(a);
// Print out the number string
printf("%s %s\n", msg, number_str);
// Free the dynamically allocated memory
OPENSSL_free(number_str);
}

```

Then in the main method, create a BN_CTX structure to holds BIGNUM temporary variables used by library functions. We need to create such a structure and pass it to the functions that require it. Then we initialize BIGNUM variables: p,q,e,d,res1.res2.res3.one.

There are a number of ways to assign a value to a BIGNUM variable(p,q,e.one)

```

int main(){
BN_CTX *ctx = BN_CTX_new();
BIGNUM *p = BN_new();
BIGNUM *q = BN_new();
BIGNUM *e = BN_new();
BIGNUM *d = BN_new();
BIGNUM *res1 = BN_new();
BIGNUM *res2 = BN_new();
BIGNUM *res3 = BN_new();
BIGNUM *one = BN_new();
// initialize p q e
// Assign the first large prime
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");

// Assign the second large prime
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");

// Assign the Modulus
BN_hex2bn(&e, "0D88C3");
BN_dec2bn(&one, "1");

```

Compute functions: Compute $res1 = p-1$, $res2 = q-1$, $res3 = res1 * res2$, $d * e \bmod res3 = 1$

```

BN_dec2bn(&one, "1");
//res1 = p-1
BN_sub(res1, p, one);
//res2 = q-1
BN_sub(res2, q, one);
//res3=res1*res2
BN_mul(res3, res1, res2, ctx);
//res=a*b mod n
BN_mod_inverse(d, e, res3, ctx);
//print BN
printBN("d= ",d);
return 0;
-- INSERT --
45,1

```

Finally, we can get `d` by following commands:

```
1. seed@VM:~/cs458Lab2$ vi task1.c[10/18/20]
2. seed@VM:~/cs458Lab2$ gcc -o task1 task1.c -lcrypto
3. seed@VM:~/cs458Lab2$ ./task1
4.
```

```
[10/18/20]seed@VM:~/cs458Lab2$ vi task1.c
[10/18/20]seed@VM:~/cs458Lab2$ gcc -o task1 task1.c -lc
rypto
[10/18/20]seed@VM:~/cs458Lab2$ ./task1
d= 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE
7C28A9B496AEB
```

Therefore, we get the private key

`d=3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB`

3.2 Task 2: Encrypting a Message

Let (e, n) be the public key, We need to convert this ASCII string to a hex string, and then convert the hex string to a BIGNUM using the hex-to-bn API `BN_hex2bn()`.

In this task, I create a `task2.c` using command

```
1. vi task2.c
```

Import library

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

Print method to make the result

```
//print a big number
void printBN(char *msg, BIGNUM *a){
// Convert the BIGNUM to number string
char * number_str = BN_bn2dec(a);
// Print out the number string
printf("%s %s\n", msg, number_str);
// Free the dynamically allocated memory
OPENSSL_free(number_str);
}
```

According to the lab, we know

```
1. $ python -c 'print("A top secret!".encode("hex"))'
2. 4120746f702073656372657421
```

Therefore, we can use the hexadecimal of `M`

```

int main(){
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *m = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();
    //Initialize
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849
DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&m, "4120746f702073656372657421");//A top
secret!
    BN_hex2bn(&d, " 74D806F9F3A62BAE331FFE3F0A68AFE35B3D
2E4794148AACBC26AA381CD7D30D");
    //encry = m^e mod n
    BN_mod_exp(enc, m, e, n, ctx);
    printBN("encrypt message = ", enc);

    //decry = enc^d mod n
    BN_mod_exp(dec, enc, d, n, ctx);
    printBN("decrypt message = ", dec);
    return 0;
}

```

Then, we run the code in the terminal, the result as shown

```

[10/18/20]seed@VM:~/cs458Lab2$ vi task2.c
[10/18/20]seed@VM:~/cs458Lab2$ gcc -o task2 task2.c -lc
rypto
[10/18/20]seed@VM:~/cs458Lab2$ ./task2
encrypt message = 6FB078DA550B2650832661E14F4F8D2CFAEF
475A0DF3A75CACDC5DE5CFC5FADC
decrypt message = 4120746F702073656372657421

```

We can see that the decrypted message and the original message are the same.

Task2 code:

```

1. #include <stdio.h>
2. #include <openssl/bn.h>
3. #define NBITS 256
4.
5. //print a big number
6. void printBN(char *msg, BIGNUM *a){
7. // Convert the BIGNUM to number string
8. char * number_str = BN_bn2hex(a);
9. // Print out the number string
10. printf("%s %s\n", msg, number_str);
11. // Free the dynamically allocated memory
12. OPENSSL_free(number_str);
13. }
14.

```

```

15. int main(){
16.     BN_CTX *ctx = BN_CTX_new();
17.     BIGNUM *m = BN_new();
18.     BIGNUM *e = BN_new();
19.     BIGNUM *n = BN_new();
20.     BIGNUM *d = BN_new();
21.     BIGNUM *enc = BN_new();
22.     BIGNUM *dec = BN_new();
23.     //Initialize
24.     BN_hex2bn(&e, "010001");
25.     BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
26.     BN_hex2bn(&m, "4120746f702073656372657421");//A top secret!
27.     BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
28.     //encry = m^e mod n
29.     BN_mod_exp(enc, m, e, n, ctx);
30.     printBN("encrypt message = ", enc);
31.
32.     //decry = enc^d mod n
33.     BN_mod_exp(dec, enc, d, n, ctx);
34.     printBN("decrypt message = ", dec);
35.     return 0;
36. }

```

3.3 Task3 Decrypting a Message

The public/private keys used in this task are the same as the ones used in Task 2. Please decrypt the following ciphertext C, and convert it back to a plain ASCII string

Code is similar to task2:

```

1. #include <stdio.h>
2. #include <openssl/bn.h>
3. #define NBITS 256
4.
5. //print a big number
6. void printBN(char *msg, BIGNUM *a){
7.     // Convert the BIGNUM to number string
8.     char * number_str = BN_bn2hex(a);
9.     // Print out the number string
10.    printf("%s %s\n", msg, number_str);
11.    // Free the dynamically allocated memory
12.    OPENSSL_free(number_str);
13. }
14.
15. int main(){
16.     BN_CTX *ctx = BN_CTX_new();
17.     BIGNUM *n = BN_new();
18.     BIGNUM *d = BN_new();
19.     BIGNUM *c = BN_new();
20.     BIGNUM *dec = BN_new();
21.     //Initialize
22.     BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
23.     BN_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBDFC7DCB67396567EA1E2493F");
24.     BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
25.     //encry = m^e mod n
26.     BN_mod_exp(dec, c, d, n, ctx);
27.     printBN("encrypt message = ", dec);
28.
29.     return 0;

```

We decrypt the given cipher text, c using the formula: $c^d \bmod n$.

By decrypting, we get the hex value of the message.

We then use the python to decode the hex value:

```
[10/18/20]seed@VM:~/cs458Lab2$ gcc -o task3 task3.c -lc
crypto
[10/18/20]seed@VM:~/cs458Lab2$ ./task3
encrypt message = 50617373776F72642069732064656573
[10/18/20]seed@VM:~/cs458Lab2$ python -c 'print("A top
secret!".encode("hex"))'
4120746f702073656372657421
[10/18/20]seed@VM:~/cs458Lab2$
```

3.4 Task4 Signing a Message

First, we get the hex value of “I owe you \$2000.”

```
[10/18/20]seed@VM:~/cs458Lab2$ python -c 'print("I owe
you $2000.".encode("hex"))'
49206f776520796f752024323030302e
```

Value is 49206f776520796f752024323030302e

We run our code to produce the signature for the message

```
[10/18/20]seed@VM:~/cs458Lab2$ vi task4.c
[10/18/20]seed@VM:~/cs458Lab2$ gcc -o task4 task4.c -lc
crypto
[10/18/20]seed@VM:~/cs458Lab2$ ./task4
encrypt message = 55A4E7F17F04CCFE2766E1EB32ADDBA890BB
E92A6FBE2D785ED6E73CCB35E4CB
```

Then, we get the hex value of “I owe you \$3000.”

```
[10/18/20]seed@VM:~/cs458Lab2$ python -c 'print("I owe
you $3000.".encode("hex"))'
49206f776520796f752024333030302e
```

Value is 49206f776520796f752024333030302e

We run our code to produce the signature for the message:

```
[10/18/20]seed@VM:~/cs458Lab2$ vi task4.c
[10/18/20]seed@VM:~/cs458Lab2$ gcc -o task4 task4.c -lc
crypto
[10/18/20]seed@VM:~/cs458Lab2$ ./task4
encrypt message = BCC20FB7568E5D48E434C387C06A6025E90D
29D848AF9C3EBAC0135D99305822
```

We can observe that, though there is only one byte of difference in the message, their signatures differ completely.

Code is similar to task3

```
1. #include <stdio.h>
2. #include <openssl/bn.h>
3. #define NBITS 256
4.
5. //print a big number
6. void printBN(char *msg, BIGNUM *a){
7. // Convert the BIGNUM to number string
8. char * number_str = BN_bn2hex(a);
9. // Print out the number string
10. printf("%s %s\n", msg, number_str);
11. // Free the dynamically allocated memory
12. OPENSSL_free(number_str);
13. }
14.
15. int main(){
16. BN_CTX *ctx = BN_CTX_new();
17. BIGNUM *n = BN_new();
18. BIGNUM *d = BN_new();
19. BIGNUM *c = BN_new();
20. BIGNUM *dec = BN_new();
21. //Initialize
22. BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
23. BN_hex2bn(&c, "49206f776520796f752024323030302e");// HEX value of "I owe you $2000."
24. //BN_hex2bn(&c, "49206f776520796f752024333030302e");// HEX value of "I owe you $3000."
25. BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
26. //encry = m^e mod n
27. BN_mod_exp(dec, c, d, n, ctx);
28. printBN("encrypt message = ", dec);
29.
30. return 0;
31. }
```

3.5 Task5: Verifying a Signature

First, We get the hex value of the message M, “Launch a missile.” using python

```
Valid Signature:
[10/18/20]seed@VM:~/cs458Lab2$ python -c 'print("Launch
a missile.".encode("hex"))'
4c61756e63682061206d6973736c652e
```

We use the signature to compute the value of the message C.

We then use the BN_cmp API in order to compare the two messages and conclude whether the signature is Alice’s or not:


```

[10/18/20]seed@VM:~/cs458Lab2$ vi task5.c
[10/18/20]seed@VM:~/cs458Lab2$ gcc -o task5 task5.c -lcrypto
[10/18/20]seed@VM:~/cs458Lab2$ ./task5
Original Message : 4C61756E63682061206D697373696C652E
Value of computed : 4C61756E63682061206D697373696C652E
Valid Signature!

```

From the result, we know the same message value, therefore, it's Alice's signature.

Code:

```

1. // task5.c
2. //
3. //
4. // Created by SHIQI LIU on 10/18/20.
5. //
6. #include <stdio.h>
7. #include <openssl/bn.h>
8. void printBN(char *msg, BIGNUM *a)
9. {
10.     char *number_str_a = BN_bn2hex(a);
11.     printf("%s %s\n", msg, number_str_a);
12.     OPENSSL_free(number_str_a);
13. }
14. int main()
15. {
16.     // init
17.     BN_CTX *ctx = BN_CTX_new();
18.     BIGNUM *n = BN_new();
19.     BIGNUM *e = BN_new();
20.     BIGNUM *M = BN_new();
21.     // BIGNUM *d = BN_new();
22.     BIGNUM *C = BN_new();
23.     BIGNUM *S = BN_new();
24.
25.     // assign values
26.     BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
27.     BN_dec2bn(&e, "65537");
28.     BN_hex2bn(&M, "4c61756e63682061206d697373696c652e"); //hex encode for " Launch a missile."
29.     BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
30.     // Get S^e mod: if S=M^d mod n, C=M
31.     BN_mod_exp(C, S, e, n, ctx);
32.     printBN("Original Message : ", M );
33.     printBN("Value of computed : ", C);
34.     // verify the signature
35.     if (BN_cmp(C, M) == 0)
36.     {
37.         printf("Valid Signature! \n");
38.     }
39.     else
40.     {
41.         printf("Verification fails! \n");
42.     }
43.
44.     return 0;
45. }

```

Suppose that the signature in is corrupted, such that the last byte of the signature changes from 2F to 3F,

S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F

We use the signature to compute the value of the message C.

We then use the BN_cmp API in order to compare the two messages and conclude whether the signature is Alice's or not:

```
[10/18/20]seed@VM:~/cs458Lab2$ vi task5.c
[10/18/20]seed@VM:~/cs458Lab2$ gcc -o task5 task5.c -lcrypto
[10/18/20]seed@VM:~/cs458Lab2$ ./task5
Original Message : 4C61756E63682061206D697373696C652E
Value of computed : 91471927C80DF1E42C154FB4638CE8BC72
6D3D66C83A4EB6B7BE0203B41AC294
Verification fails!
```

Therefore, we get the value of computed message is entirely different from original message, though only 1 byte of the signature is changed. It causes the verification fails.

3.6 Task6 Manually Verifying an X.509 Certificate

- Step 1

Download a certificate from a real web server. We use the www.google.com server in this document

```
1. [10/18/20]seed@VM:~/cs458Lab2$ openssl s_client -connect www.google.com:443 -showcerts
2. CONNECTED(00000003)
3. depth=2 OU = GlobalSign Root CA - R2, O = GlobalSign, CN = GlobalSign
4. verify return:1
5. depth=1 C = US, O = Google Trust Services, CN = GTS CA 101
6. verify return:1
7. depth=0 C = US, ST = California, L = Mountain View, O = Google LLC, CN = www.google.com

8. verify return:1
9. ---
10. Certificate chain
11.  0 s:/C=US/ST=California/L=Mountain View/O=Google LLC/CN=www.google.com
12.  1 i:/C=US/O=Google Trust Services/CN=GTS CA 101
13. -----BEGIN CERTIFICATE-----
14. MIIFkTCCBHmgAwIBAgIQB5q2sT+2NbIIAAAAFst1zANBgkqhkiG9w0BAQsFADBC
15. MQswCQYDVQQGEwJVUzEeMBwGA1UEChMVRR29vZ2x1IFRydXN0IFNlcnZpY2VzMRRMw
16. EQYDVQQDEwphVFmGq0EgMU8xMB4XDTEwMDkyMjE1MTYxOVoXDTEwMTIxNTE1MTYx
17. OVowADELMAkGA1UEBhMCVVMxEzARBgNVBAgTCKNhbg1mb3JuaWExfjAUBgNVBAcT
18. DU1vdW50YWluIFZpZXcxZzARBgNVBAoTCkdvd2dsZSBMTExmFzAVBgNVBAMTDnd3
19. dy5nb29nbGUuY29tMIIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA
20. nG1DZ2Tl8l5ruTFxLz5PmUKZjYFEihvXKsmiu2d0NTKv8RgWvMIItgvP/1IfIz+076jn3
21. meC71lP10LST+wI+vFPE9vEL/Zie5veUiJpXE0bUApeud+Rnlsw4UJ1x50PCBdSN
22. r41sNVCAAU98ibSGlC5n2Qr1/YYhs2qnqKKG8i8zNMbjDYyzOWyjfRvi8W7eU2Z
23. FpJvWc3D6WEACQrLATPcFNK0Jb3f5j7eohea9eMCs1B13e43iIbA2zHb8owE2Mu
24. F7GKhYiTr8OEFy4aq7meEeIFHDB/O5eO+6nSxV0p3dBsdxXu/BlEVruwnigCtMAN
25. LVS50EhwBuz1Wl+VWwIDAQABo4ICWzCCAlcwDgYDVROPAQH/BAQDAgWgMBMGA1Ud
26. JQqMMAoGCCsGAQUFBwMBMAwGA1UdEwEB/wQCMAAwHQYDVROBBYEFDDRC4ywBM9P
27. OAK0QWnLJl3HjmJMB8GA1UdIwQYMBaFAFJjR+G4Q68+b7GCFGJAboT9Cf0rMGgG
```

```
28. CCsGAQUFBwEBBFwwJArBggrBgEFBQcwAYYfaHR0cDovL29jc3AucGtpLmdvb2cv
29. Z3RzMW8xY29yZTAuBggrBgEFBQcwAoYfaHR0cDovL3BraS5nb29nL2dzcjIvR1RT
30. MU8xLmNyDAZBgNVHREEEjAQgg53d3cuZ29vZ2x1LmNvbTAhBgNVHSAEGjAYMAgG
31. BmeBDAECAjAMBgorBgEEAdZ5AgUDMDMGA1UdHwQSMCOWKKAmoCSGImh0dHA6Ly9j
32. cmwucGtpLmdvb2cvR1RTMU8xY29yZS5jcmwggEDBgorBgEEAdZ5AgQCBiH0BIHx
33. A08AdQDw1aRZ8gDRgkAQLS+TiI6tS/4dR+OZ4dA0prCoqo6ycwAAAXS2mge4AAAE
34. AwBGMEQCIBA000C/IxSaE2sVhS+dJtnXsh7fSYjeybH0nF0toRFCaiBnTcymqGeb
35. lwBe5U3nJyG3tngeH9YCFBdkeShmHdf6DgB2ALIEBcyLos2KIE6HZvkruYolIGdr
36. 2vpw57JJUy3vi5BeAAABdLaaADoAAQAQAEcwRQIGBFJa178fY3/4Pb95N5hh2JfR
37. 3EJ9AUbb0vff31qx/NsCIQDJTBFLs1QdubTe6S+Xc7EuQC9rh3YVLSYc8+dSRZuV
38. 3zANBqkqhkiG9w0BAQsFAA0CAQEafznjbmvP1GKbyj7RIT5L/x6dkPBCWp6u6toi
39. 1ak4chqHN7mkJkazcb+DGoSAkz7DWfvrVt6Kruh7Vq93Z90g9Nnp5ZiMvkHd5+JM
40. VqVq3SEK0x+Bd//cW7364zsqnCP97Dg1kvPZz/Rqkq04i9ajSGNxxkiMjkkFG4k10
41. tBUmXOmj0IPwa81iXT1tpV8TqV3uQj0FJ+WZXrYP33HSFGgEX04VJq6cAh1o5V4z
42. +3K0F1Si5pVAzIHjEbTB9RP2WK7XQ2VkmYfqcNEJYCENLDAREQkkGQMmJV014jHh
43. SRcKMTc/HXPKihjE7cPmhElEehHzWkJLsYwYZcIGOP+shpz1rw==
44. -----END CERTIFICATE-----
45. 1 s:/C=US/O=Google Trust Services/CN=GTS CA 101
46. i:/OU=GlobalSign Root CA - R2/O=GlobalSign/CN=GlobalSign
47. -----BEGIN CERTIFICATE-----
48. MIESjCcaZKgAwIBAgINAeO0mqGNIqmBJWlQuDANBgkqhkiG9w0BAQsFADBMMSAw
49. HgYDVQQLExdHbG9iYWxTaWduIFJvb3QgQ0EgLSBSMjETMBEGA1UECHMKR2xvYmFs
50. U21nbjETMBEGA1UEAxMKR2xvYmFsU21nbjAeFw0xNzA2MTUwMDAwNDJaFw0yMTEy
51. MTUwMDAwNDJaMEIxCzAJBgNVBAYTAlVTMR4wHAYDVQQKEgVhbnQgUgVHJ1c3Qg
52. U2VydmljZXMxEzARBGNVBAMTCKdUUYBDQSAxTzEwggEiMA0GCSqGSIb3DQEBAQUA
53. A4IBDwAwggEKAoIBAQQDQGM9F1IvN05zkQ09+tN1pIRvJzzyOTHW5DZEHd2ePCnv
54. UA0Qk28FgITCfKqC9EksC4T2fWBYk/jCFC3R3VZMds/dN4ZKCEPZRrAzDsikUDzRr
55. mBBJ5wudgzndIMYcLe/RGGfL5yODIKgjEv/SJH/UL+dEaltN11BmsK+eQmMF++Ac
56. xGNhr59qm/9il71I2dN8FGfcddwuaej4bXhp0LcQBbjxMcI7JP0aM3T4I+DsaxmK
57. FsbjzaTNC9uzpfLgOIg7rR25xoyNuxv8vNmKq7zdPGHXkxWY7oG9j+JkRyBABk7X
58. rJfoucBZEqFJJSPK7XA0LKW0Y3z5oz2D0c1tJKwHAgMBAAGjggEzMIIBLzA0BgNV
59. HQ8BAf8EBAMCAYYwHQYDVR0lBBYwFAIKwYBBQUHAWEGCCsGAQUFBwMCMBIGA1Ud
60. EwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFjJR+G4Q68+b7GCFGJAboOt9CF0rMB8G
61. A1UdIwQYMBAAfJviB1dnHB7AagbewbSaLd/cGYyUMDUGCCsGAQUFBwEBBCKwJzAl
62. BggrBgEFBQcwAYYzaHR0cDovL29jc3AucGtpLmdvb2cvZ3NyMjAyBgNVHR8EKzAp
63. MCegJaAJhiFodHRwOi8vY3JsLnBraS5nb29nL2dzcjIvZ3NyMi5jcmwvPwYDVR0g
64. BDgwNjA0BgZngQwBAGIwkjAoBggrBgEFBQcCARYcaHR0cHM6Ly9wa2kuZ29vZy9y
65. ZX8vc2l0b3J5LzANBgkqhkiG9w0BAQsFAA0CAQEAGoA+Nnn78y6prjd9XlQWNa7H
66. TgiZ/r3RNGkmUmYHPQq6Scti9PEajvwrT2iWTHQr02fesqOqBY2ETUwgZQ+lltoN
67. FvhsO9tvBCOIazpswWC9aJ9xju4tWDQH8NVU6YZZ/XteDSGU9YzJqPjY8q3MDxrz
68. mqepBCf5o8mw/wJ4a2G6xzUr6Fb6T8McDO22PLRL6u3M4Tzs3A2M1j6bykJYi8wW
69. IRdAvKLWZu/axBVbzYmqmwkm5zLSDW5nIAJbELCQCZwMH56t2Dvqofxs6BBcCFIZ
70. USpxu6x6td0V7SvJCCosirSmIatj/9dSSVDQibet8q/7UK4v4ZUN80atnZz1yg==
71. -----END CERTIFICATE-----
72. ---
73. Server certificate
74. subject=/C=US/ST=California/L=Mountain View/O=Google LLC/CN=www.google.com
75. issuer=/C=US/O=Google Trust Services/CN=GTS CA 101
76. ---
77. No client certificate CA names sent
78. Peer signing digest: SHA256
79. Server Temp Key: ECDH, P-256, 256 bits
80. ---
81. SSL handshake has read 3247 bytes and written 431 bytes
82. ---
83. New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
84. Server public key is 2048 bit
85. Secure Renegotiation IS supported
86. Compression: NONE
87. Expansion: NONE
88. No ALPN negotiated
```

```

89. SSL-Session:
90.   Protocol   : TLSv1.2
91.   Cipher     : ECDHE-RSA-AES128-GCM-SHA256
92.   Session-ID: BC337A8A0CE22588A9845CD955A25131346F558771B2B435CFB7373C3534E9E7
93.   Session-ID-ctx:
94.   Master-
      Key: 42EA2EB90D0B1F59618908405370AAB39B2C8E27CD06BB4C99E71435294027B3DAB36CCA97E370AA4A
      738095F45CDBC0
95.   Key-Arg    : None
96.   PSK identity: None
97.   PSK identity hint: None
98.   SRP username: None
99.   TLS session ticket lifetime hint: 100800 (seconds)
100.  TLS session ticket:
101.    0000 - 01 95 05 1c 6a 74 90 8c-f9 0e 6c 5a cc dc cd 1b ....jt....lZ....
102.    0010 - e0 a3 61 6d 1a 72 be 3c-e5 8a b1 55 2e d7 5a ab ..am.r.<...U..Z.
103.    0020 - a9 01 59 f0 e8 eb 0e a2-e5 34 a4 b6 be 05 ec 7c ..Y.....4....|
104.    0030 - 08 9e e2 70 94 9c e1 8b-47 8e 10 24 c8 e6 e6 5c ...p....G..$...\
105.    0040 - cc 83 71 a2 a2 1c c8 c3-db 1f df c9 15 23 3e e0 ..q.....#>.
106.    0050 - dc f9 73 33 46 83 27 d9-ab 92 40 0d 92 41 89 6f ..s3F.'...@..A.o
107.    0060 - 49 bb 2f 30 8c 8e fc fd-bc 3a 44 c5 67 3d c2 15 I./0.....:D.g=..
108.    0070 - 29 2c 86 39 53 66 a0 68-70 48 36 99 50 e7 09 ba ),.9Sf.hpH6.P...
109.    0080 - 39 ce 7f 5b d1 fb a1 7f-83 b9 6b be 36 b5 c1 a6 9..[.....k.6...
110.    0090 - a7 32 f0 85 d7 04 77 1b-e4 c4 03 77 d3 0b 27 92 .2....w....w..'.
111.    00a0 - 1d 4d a5 06 35 67 69 5e-8b 51 cd 64 2e 91 8f 79 .M..5gi^.Q.d...y
112.    00b0 - fa c8 5b 17 8f 63 f4 c5-89 e0 83 2d 16 2f 9e f7 ..[...c.....-./..
113.    00c0 - 33 20 5d 03 00 c6 f4 0d-ec b5 a2 d0 6a 49 a4 49 3 ].....jI.I
114.    00d0 - e5 6a a9 fc f7 05 34 d5-7d 63 af 4d 79 .j....4.}c.My
115.
116.   Start Time: 1603076169
117.   Timeout    : 300 (sec)
118.   Verify return code: 0 (ok)
119.   ---
120.   read:errno=0
121.   [10/18/20]seed@VM:~/cs458Lab2$

```

```

[10/18/20]seed@VM:~/cs458Lab2$ openssl s_client -connect
www.google.com:443 -showcerts
CONNECTED(00000003)
depth=2 OU = GlobalSign Root CA - R2, O = GlobalSign, C
N = GlobalSign
verify return:1
depth=1 C = US, O = Google Trust Services, CN = GTS CA
101
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O =
Google LLC, CN = www.google.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google LLC/C
N=www.google.com
 1 s:/C=US/O=Google Trust Services/CN=GTS CA 101
-----BEGIN CERTIFICATE-----
MIIFkTCCBHmgAwIBAgIQB5q2sT+2NbIIAAAAFst1zANBgkqhkiG9w0
BAQsFADBC

```

```

VqVq3SEK0x+Bd//cw7364zsqnCP97Dg1kvPZz/Rqkq04i9ajSGNxkiI
jkkFG4k10
tBuMX0mjoIPwa81iXT1tpV8TqV3uQj0FJ+WZXrYP33HSFGgEX04VJq6
cAh1o5V4z
+3K0F1Si5pVAzIHjEbTB9RP2WK7XQ2VkmyfqcnEJYCEnLDAReQkkGQM
mJV014jHh
SRcKMTc/HXPKihjE7cPmhElEehHzWkJLsYwYZcIGOP+shpz1rw==
-----END CERTIFICATE-----
1 s:/C=US/O=Google Trust Services/CN=GTS CA 101
i:/OU=GlobalSign Root CA - R2/O=GlobalSign/CN=Global
Sign
-----BEGIN CERTIFICATE-----
MIIESjCCAzKgAwIBAgINAe00mqGNIqmBJWlQuDANBgkqhkiG9w0BAQs
FADBMMSAw
HgYDVQQLExdHbG9iYWxTaWduIFJvb3QgQ0EgLSBSMjETMBEGA1UEChM
KR2xvYmFs
U2lnbjETMBEGA1UEAxMKR2xvYmFsU2lnbjAeFw0xNzA2MTUwMDAwNDJ
aFw0yMTEy
MTUwMDAwNDJhMEIxCzAJBgNVBAYTA1VTMR4wHAYDVQQKEjVHb29nbGU
gVHJ1c3Qg
U2VydmljZXMxEzARBgNVBAMTCkdUUyBDQSAXTzEwggEiMA0GCSqGSIb
BDQEBAQUA
IglZ/r3KNGkMUMYHPQqb5ct19PEajvwKIz1WIHQr0ZtesquqBYZEIUV
gZQ+lltoN
Fvhs09tvBC0IazpswWC9aJ9xju4tWDQH8NVU6YZZ/XteDSGU9YzJqPj
/8q3MDxrz
nqepBCf5o8mw/wJ4a2G6xzUr6Fb6T8McD022PLRL6u3M4Tzs3A2M1j6
bykJYi8wW
IRdAvKLWZu/axBVbzYmqmwkm5zLSDW5nIAJbELCQCZwMH56t2Dvqofx
s6BBcCFIZ
USpxu6x6td0V7SvJCCosirSmIatj/9dSSVDQibet8q/7UK4v4ZUN80a
tnZz1yg==
-----END CERTIFICATE-----
---
Server certificate
subject=/C=US/ST=California/L=Mountain View/O=Google LLC
C/CN=www.google.com
issuer=/C=US/O=Google Trust Services/CN=GTS CA 101
---
No client certificate CA names sent
Peer signing digest: SHA256
Server Temp Key: ECDH, P-256, 256 bits
---
SSL handshake has read 3247 bytes and written 431 bytes

```

We save these two certificates in the files c0.pem and c1.pem respectively

- Step2 Extract the public key (e, n) from the issuer's certificate.

We get the value of n using -modulus:

- [10/18/20]seed@VM:~/cs458Lab2\$ openssl x509 -in c1.pem -noout -modulus
- Modulus=D018CF45D48BCDD39CE440EF7EB4DD69211BC9CF3C8E4C75B90F3119843D9E3C29EF500D10936F0580809F2AA0BD124B02E13D9F581624FE309F0B747755931D4BF74DE1928210F651AC0CC3B222940F346B981049E70B9D8339DD20C61C2DEFD1186165E7238320A82312FFD2247FD42FE7446A5B4DD75066B0AF9E426305FBE01CC46361AF9F6A33FF6297BD48D9D37C1467DC75DC2E69E8F86D7869D0B71005B8F131C23B24FD1A3374F823E0EC6B198A16C6E3CDA4CD0BDBB3A4596038883BAD1DB9C68CA7531BFCBCD9A4ABBCDD3C61D7931598EE81BD8FE264472040064ED7AC97E8B9C05912A1492523E4ED70342CA5B4637CF9A33D83D1CD6D24AC07

```
[10/18/20]seed@VM:~/cs458Lab2$ vi c0.pem
[10/18/20]seed@VM:~/cs458Lab2$ vi c1.pem
[10/18/20]seed@VM:~/cs458Lab2$ openssl x509 -in c1.pem
-noout -modulus
Modulus=D018CF45D48BCDD39CE440EF7EB4DD69211BC9CF3C8E4C7
5B90F3119843D9E3C29EF500D10936F0580809F2AA0BD124B02E13D
9F581624FE309F0B747755931D4BF74DE1928210F651AC0CC3B2229
40F346B981049E70B9D8339DD20C61C2DEFD1186165E7238320A823
12FFD2247FD42FE7446A5B4DD75066B0AF9E426305FBE01CC46361A
F9F6A33FF6297BD48D9D37C1467DC75DC2E69E8F86D7869D0B71005
B8F131C23B24FD1A3374F823E0EC6B198A16C6E3CDA4CD0BDBB3A45
96038883BAD1DB9C68CA7531BFCBCD9A4ABBCDD3C61D7931598EE81
BD8FE264472040064ED7AC97E8B9C05912A1492523E4ED70342CA5B
4637CF9A33D83D1CD6D24AC07
```

Print all attributes of the certificate, and then find the exponent, which is public key e(line 35)

1. [10/18/20]seed@VM:~/cs458Lab2\$ openssl x509 -in c1.pem -text -noout
2. Certificate:
3. Data:
4. Version: 3 (0x2)
5. Serial Number:
6. 01:e3:b4:9a:a1:8d:8a:a9:81:25:69:50:b8
7. Signature Algorithm: sha256WithRSAEncryption
8. Issuer: OU=GlobalSign Root CA - R2, O=GlobalSign, CN=GlobalSign
9. Validity
10. Not Before: Jun 15 00:00:42 2017 GMT
11. Not After : Dec 15 00:00:42 2021 GMT
12. Subject: C=US, O=Google Trust Services, CN=GTS CA 101
13. Subject Public Key Info:
14. Public Key Algorithm: rsaEncryption
15. Public-Key: (2048 bit)
16. Modulus:
17. 00:d0:18:cf:45:d4:8b:cd:d3:9c:e4:40:ef:7e:b4:
18. dd:69:21:1b:c9:cf:3c:8e:4c:75:b9:0f:31:19:84:
19. 3d:9e:3c:29:ef:50:0d:10:93:6f:05:80:80:9f:2a:
20. a0:bd:12:4b:02:e1:3d:9f:58:16:24:fe:30:9f:0b:
21. 74:77:55:93:1d:4b:f7:4d:e1:92:82:10:f6:51:ac:
22. 0c:c3:b2:22:94:0f:34:6b:98:10:49:e7:0b:9d:83:
23. 39:dd:20:c6:1c:2d:ef:d1:18:61:65:e7:23:83:20:
24. a8:23:12:ff:d2:24:7f:d4:2f:e7:44:6a:5b:4d:d7:
25. 50:66:b0:af:9e:42:63:05:fb:e0:1c:c4:63:61:af:
26. 9f:6a:33:ff:62:97:bd:48:d9:d3:7c:14:67:dc:75:

```

27.          dc:2e:69:e8:f8:6d:78:69:d0:b7:10:05:b8:f1:31:
28.          c2:3b:24:fd:1a:33:74:f8:23:e0:ec:6b:19:8a:16:
29.          c6:e3:cd:a4:cd:0b:db:b3:a4:59:60:38:88:3b:ad:
30.          1d:b9:c6:8c:a7:53:1b:fc:bc:d9:a4:ab:bc:dd:3c:
31.          61:d7:93:15:98:ee:81:bd:8f:e2:64:47:20:40:06:
32.          4e:d7:ac:97:e8:b9:c0:59:12:a1:49:25:23:e4:ed:
33.          70:34:2c:a5:b4:63:7c:f9:a3:3d:83:d1:cd:6d:24:
34.          ac:07
35.          Exponent: 65537 (0x10001)

```

```

[10/18/20]seed@VM:~/cs458Lab2$ openssl x509 -in c1.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 101
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:d0:18:cf:45:d4:8b:cd:d3:9c:e4:40
:ef:7e:b4:

```

```

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 Extract the signature from the server's certificate:

We run the command: `openssl x509 -in c0.pem -text -noout` to extract the signature from the server's certificate, `c0.pem`. We put this signature into a file.

```
1. [10/18/20]seed@VM:~/cs458Lab2$ openssl x509 -in c0.pem -text -noout
2.      Signature Algorithm: sha256WithRSAEncryption
3.          7f:39:e3:6e:6b:cf:d4:62:9b:ca:3e:d1:21:3e:4b:ff:1e:9d:
4.          90:f0:42:5a:9e:ae:ea:da:22:d5:a9:38:72:1a:87:37:b9:a4:
5.          26:46:b3:71:bf:83:1a:84:80:93:3e:c3:59:fb:eb:56:de:8a:
6.          ae:e8:7b:56:af:77:67:dd:20:f4:d9:e9:e5:98:8c:be:41:dd:
7.          e7:e2:4c:56:a5:6a:dd:21:0a:d3:1f:81:77:ff:dc:5b:bd:fa:
8.          e3:3b:2a:9c:23:fd:ec:38:35:92:f3:d9:cf:f4:6a:92:ad:38:
9.          8b:d6:a3:48:63:71:92:23:23:92:41:46:e2:49:4e:b4:1b:8c:
10.         5c:e9:a3:a0:83:f0:6b:cd:62:5d:3d:6d:a5:5f:13:a9:5d:ee:
11.         42:3d:05:27:e5:99:5e:b6:0f:df:71:d2:14:68:04:5c:ee:15:
12.         26:ae:9c:02:1d:68:e5:5e:33:fb:72:8e:17:54:a2:e6:95:40:
13.         cc:81:e3:11:b4:c1:f5:13:f6:58:ae:d7:43:65:64:9b:27:ea:
14.         72:71:09:60:21:27:2c:30:11:79:09:24:19:03:26:25:53:b5:
15.         e2:31:e1:49:17:0a:31:37:3f:1d:73:ca:8a:18:c4:ed:c3:e6:
16.         84:49:44:7a:11:f3:5a:42:4b:b1:8c:18:65:c2:06:38:ff:ac:
17.         86:9c:f5:af
18. [10/18/20]seed@VM:~/cs458Lab2$
```

```
      Signature Algorithm: sha256WithRSAEncryption
      7f:39:e3:6e:6b:cf:d4:62:9b:ca:3e:d1:21:3e:4b:ff:1e:9d:
f:1e:9d:
      90:f0:42:5a:9e:ae:ea:da:22:d5:a9:38:72:1a:87:3
7:b9:a4:
      26:46:b3:71:bf:83:1a:84:80:93:3e:c3:59:fb:eb:5
6:de:8a:
      ae:e8:7b:56:af:77:67:dd:20:f4:d9:e9:e5:98:8c:b
e:41:dd:
      e7:e2:4c:56:a5:6a:dd:21:0a:d3:1f:81:77:ff:dc:5
b:bd:fa:
      e3:3b:2a:9c:23:fd:ec:38:35:92:f3:d9:cf:f4:6a:9
2:ad:38:
      8b:d6:a3:48:63:71:92:23:23:92:41:46:e2:49:4e:b
4:1b:8c:
      5c:e9:a3:a0:83:f0:6b:cd:62:5d:3d:6d:a5:5f:13:a
9:5d:ee:
      42:3d:05:27:e5:99:5e:b6:0f:df:71:d2:14:68:04:5
c:ee:15:
      26:ae:9c:02:1d:68:e5:5e:33:fb:72:8e:17:54:a2:e
6:95:40:
      cc:81:e3:11:b4:c1:f5:13:f6:58:ae:d7:43:65:64:9
```

We put this signature into a file, then remove all the colons and spaces from the signature:

```
1. [10/18/20]seed@VM:~/cs458Lab2$ vi signature
2. [10/18/20]seed@VM:~/cs458Lab2$ cat signature | tr -d '[:space:]:'
3. ee87b56af7767dd20f4d9e9e5988cbe41dde7e24c56a56add210ad31f8177ffdc5bbdfae33b2a9c2
3fdec383592f3d9cff46a92ad388bd6a3486371922323924146e2494eb41b8c5ce9a3a083f06bcd6
25d3d6da55f13a95dee423d0527e5995eb60fdf71d21468045cee1526ae9c021d68e55e33fb728e1
754a2e69540[10/18/20]seed@VM:~/cs458Lab2$
```



```
[10/18/20]seed@VM:~/cs458Lab2$ vi signature
[10/18/20]seed@VM:~/cs458Lab2$ cat signature | tr -d '[:space:]':
ee87b56af7767dd20f4d9e9e5988cbe41dde7e24c56a56add210ad3
1f8177ffdc5bbdfae33b2a9c23fdec383592f3d9cff46a92ad388bd
6a3486371922323924146e2494eb41b8c5ce9a3a083f06bcd625d3d
6da55f13a95dee423d0527e5995eb60fdf71d21468045cee1526ae9
c021d68e55e33fb728e1754a2e69540[10/18/20]seed@VM:~/cs45
8Lab2$
```

- Step4 : Extract the body of the server's certificate

```
[10/18/20]seed@VM:~/cs458Lab2$ openssl asn1parse -i -in c0.pem
• 0:d=0 hl=4 l=1425 cons: SEQUENCE
• 4:d=1 hl=4 l=1145 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 :079AB6B13FB635B20800000000
5B2DD7
• 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
• 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= 30 cons: SET
• 63:d=4 hl=2 l= 28 cons: SEQUENCE
• 65:d=5 hl=2 l= 3 prim: OBJECT :organizationName
• 70:d=5 hl=2 l= 21 prim: PRINTABLESTRING :Google Trust Services
• 93:d=3 hl=2 l= 19 cons: SET
• 95:d=4 hl=2 l= 17 cons: SEQUENCE
• 97:d=5 hl=2 l= 3 prim: OBJECT :commonName
• 102:d=5 hl=2 l= 10 prim: PRINTABLESTRING :GTS CA 101
• 114:d=2 hl=2 l= 30 cons: SEQUENCE
• 116:d=3 hl=2 l= 13 prim: UTCTIME :200922151619Z
• 131:d=3 hl=2 l= 13 prim: UTCTIME :201215151619Z
• 146:d=2 hl=2 l= 104 cons: SEQUENCE
• 148:d=3 hl=2 l= 11 cons: SET
• 150:d=4 hl=2 l= 9 cons: SEQUENCE
• 152:d=5 hl=2 l= 3 prim: OBJECT :countryName
• 157:d=5 hl=2 l= 2 prim: PRINTABLESTRING :US
• 161:d=3 hl=2 l= 19 cons: SET
• 163:d=4 hl=2 l= 17 cons: SEQUENCE
• 165:d=5 hl=2 l= 3 prim: OBJECT :stateOrProvinceName
• 170:d=5 hl=2 l= 10 prim: PRINTABLESTRING :California
• 182:d=3 hl=2 l= 22 cons: SET
• 184:d=4 hl=2 l= 20 cons: SEQUENCE
• 186:d=5 hl=2 l= 3 prim: OBJECT :localityName
• 191:d=5 hl=2 l= 13 prim: PRINTABLESTRING :Mountain View
• 206:d=3 hl=2 l= 19 cons: SET
```

•	208:d=4	hl=2 l=	17 cons:	SEQUENCE	
•	210:d=5	hl=2 l=	3 prim:	OBJECT	:organizationName
•	215:d=5	hl=2 l=	10 prim:	PRINTABLESTRING	:Google LLC
•	227:d=3	hl=2 l=	23 cons:	SET	
•	229:d=4	hl=2 l=	21 cons:	SEQUENCE	
•	231:d=5	hl=2 l=	3 prim:	OBJECT	:commonName
•	236:d=5	hl=2 l=	14 prim:	PRINTABLESTRING	:www.google.com
•	252:d=2	hl=4 l=	290 cons:	SEQUENCE	
•	256:d=3	hl=2 l=	13 cons:	SEQUENCE	
•	258:d=4	hl=2 l=	9 prim:	OBJECT	:rsaEncryption
•	269:d=4	hl=2 l=	0 prim:	NULL	
•	271:d=3	hl=4 l=	271 prim:	BIT STRING	
•	546:d=2	hl=4 l=	603 cons:	cont [3]	
•	550:d=3	hl=4 l=	599 cons:	SEQUENCE	
•	554:d=4	hl=2 l=	14 cons:	SEQUENCE	
•	556:d=5	hl=2 l=	3 prim:	OBJECT	:X509v3 Key Usage
•	561:d=5	hl=2 l=	1 prim:	BOOLEAN	:255
•	564:d=5	hl=2 l=	4 prim:	OCTET STRING	[HEX DUMP]:030205A0
•	570:d=4	hl=2 l=	19 cons:	SEQUENCE	
•	572:d=5	hl=2 l=	3 prim:	OBJECT	:X509v3 Extended Key Usage
•	577:d=5	hl=2 l=	12 prim:	OCTET STRING	[HEX DUMP]:300A06082B06010505070301
•	591:d=4	hl=2 l=	12 cons:	SEQUENCE	
•	593:d=5	hl=2 l=	3 prim:	OBJECT	:X509v3 Basic Constraints
•	598:d=5	hl=2 l=	1 prim:	BOOLEAN	:255
•	601:d=5	hl=2 l=	2 prim:	OCTET STRING	[HEX DUMP]:3000
•	605:d=4	hl=2 l=	29 cons:	SEQUENCE	
•	607:d=5	hl=2 l=	3 prim:	OBJECT	:X509v3 Subject Key Identifier
•	612:d=5	hl=2 l=	22 prim:	OCTET STRING	[HEX DUMP]:041430D10B8CB004CF4F3809344169CB2658371E3989
•	636:d=4	hl=2 l=	31 cons:	SEQUENCE	
•	638:d=5	hl=2 l=	3 prim:	OBJECT	:X509v3 Authority Key Identifier
•	643:d=5	hl=2 l=	24 prim:	OCTET STRING	[HEX DUMP]:3016801498D1F86E10EBCF9BEC609F18901BA0EB7D09FD2B
•	669:d=4	hl=2 l=	104 cons:	SEQUENCE	
•	671:d=5	hl=2 l=	8 prim:	OBJECT	:Authority Information Access
•	681:d=5	hl=2 l=	92 prim:	OCTET STRING	[HEX DUMP]:305A302B06082B06010505073001861F687474703A2F2F6F6373702E706B692E676F6F672F677473316F31636F7265302B06082B06010505073002861F687474703A2F2F706B692E676F6F672F677372322F475453314F312E637274
•	775:d=4	hl=2 l=	25 cons:	SEQUENCE	
•	777:d=5	hl=2 l=	3 prim:	OBJECT	:X509v3 Subject Alternative Name
•	782:d=5	hl=2 l=	18 prim:	OCTET STRING	[HEX DUMP]:3010820E7777772E676F6F676C652E636F6D
•	802:d=4	hl=2 l=	33 cons:	SEQUENCE	
•	804:d=5	hl=2 l=	3 prim:	OBJECT	:X509v3 Certificate Policies
•	809:d=5	hl=2 l=	26 prim:	OCTET STRING	[HEX DUMP]:30183008060667810C010202300C060A2B06010401D679020503
•	837:d=4	hl=2 l=	51 cons:	SEQUENCE	
•	839:d=5	hl=2 l=	3 prim:	OBJECT	:X509v3 CRL Distribution Points

- 844:d=5 hl=2 l= 44 prim: OCTET STRING [HEX DUMP]:302A3028A026A0248622687474703A2F2F63726C2E706B692E676F672F475453314F31636F72652E63726C
- 890:d=4 hl=4 l= 259 cons: SEQUENCE
- 894:d=5 hl=2 l= 10 prim: OBJECT :CT Precertificate SCTs
- 906:d=5 hl=3 l= 244 prim: OCTET STRING [HEX DUMP]:0481F100EF007500F095A459F200D18240102D2F93888EAD4BFE1D47E399E1D034A6B0A8AA8EB27300000174B69A0138000004030046304402201034D340BF23149A136B15852F9D26D9D7B21EDF4988DEC9B1CE9C53ADA111420220674DCCA6A8679B97005EE54DE72721B7B6781E1FD6027C17647928661DD7FA0E007600B21E05CC8BA2CD8A204E8766F92BB98A2520676BDFA70E7B249532DEF8B905E00000174B69A003A0000040300473045022004525AD7BF1F637FF83DBF79379861D897D1DC427D0146DBD2F7DFDF5AB1FCDB022100C94C114BB3541DB9B4DEE92F9773B12E402F6B8776152D261CF3E752459B95DF
- 1153:d=1 hl=2 l= 13 cons: SEQUENCE
- 1155:d=2 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption
- 1166:d=2 hl=2 l= 0 prim: NULL
- 1168:d=1 hl=4 l= 257 prim: BIT STRING

```

775:d=4 hl=2 l= 25 cons: SEQUENCE
777:d=5 hl=2 l= 3 prim: OBJECT :X5
09v3 Subject Alternative Name
782:d=5 hl=2 l= 18 prim: OCTET STRING [HE
X DUMP]:3010820E7777772E676F6F676C652E636F6D
802:d=4 hl=2 l= 33 cons: SEQUENCE
804:d=5 hl=2 l= 3 prim: OBJECT :X5
09v3 Certificate Policies
809:d=5 hl=2 l= 26 prim: OCTET STRING [HE
X DUMP]:30183008060667810C010202300C060A2B06010401D6790
20503
837:d=4 hl=2 l= 51 cons: SEQUENCE
839:d=5 hl=2 l= 3 prim: OBJECT :X5
09v3 CRL Distribution Points
844:d=5 hl=2 l= 44 prim: OCTET STRING [HE
X DUMP]:302A3028A026A0248622687474703A2F2F63726C2E706B6
92E676F6F672F475453314F31636F72652E63726C
890:d=4 hl=4 l= 259 cons: SEQUENCE
894:d=5 hl=2 l= 10 prim: OBJECT :CT
Precertificate SCTs
906:d=5 hl=3 l= 244 prim: OCTET STRING [HE
X DUMP]:0481F100EF007500F095A459F200D18240102D2F93888EA

```

In this we cannot determine the end of the body. So we use -strparse to get the field from the offset 4, which will give us the body of the certificate, excluding the signature block.

1. [10/18/20]seed@VM:~/cs458Lab2\$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
2. [10/18/20]seed@VM:~/cs458Lab2\$ sha256sum c0_body.bin
3. fe5623d9b8e79ffa12b3b6471ae96c3ebc8ee5b6144305320c8c7de06443269a c0_body.bin

```

1007d 1 ne 1 e 257 prim: B11 STRING
[10/18/20]seed@VM:~/cs458Lab2$ openssl asn1parse -i -in
c0.pem -strparse 4 -out c0_body.bin -noout
[10/18/20]seed@VM:~/cs458Lab2$ sha256sum c0_body.bin
fe5623d9b8e79ffa12b3b6471ae96c3ebc8ee5b6144305320c8c7de
06443269a  c0_body.bin
[10/18/20]seed@VM:~/cs458Lab2$

```

- Step5: Verify the signature.

Code is similar to 3.5, We use the values obtained from the previous steps, get the signature and verify the signature obtained with the original signature

```

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

//print a big number
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 *S = BN_new();//signature from server step3
    BIGNUM *M = BN_new();//signature certificate
    BIGNUM *n = BN_new();//modulus
    BIGNUM *e = BN_new();//exponent
    BIGNUM *M1 = BN_new();
    //Initialize
    BN_hex2bn(&e,"01001");
@
"task6.c" 44L, 1731C

```

```

[10/19/20]seed@VM:~/cs458Lab2$ vi task6.c
[10/19/20]seed@VM:~/cs458Lab2$ gcc -o task6 task6.c -lc
rypto
[10/19/20]seed@VM:~/cs458Lab2$ ./task6
Valid Signature!
[10/19/20]seed@VM:~/cs458Lab2$

```

We can notice that the original message and the hash value of the computed message is the same. Hence we can conclude that the www.google.com certificate is verified to be right.