

İSTANBUL TECHNICAL UNIVERSITY

INFORMATICS INSTITUTE

CYBERSECURITY ENGINEERING AND
CRYPTOGRAPHY

RSA Public-Key Encryption and Signature Lab

Detailed Report

Name: Halis Taha Şahin

ID: 707201006

E-Mail: sahinh20@itu.edu.tr

Semester: Fall 2020/2021

Fachsemester: 01

Date: December 8, 2020

Contents

1 Task 1: Deriving the Private Key

Given p, q and e. We must calculate d value from given values. To find the d value, we need to use the following equation.

$$d * e = 1 \bmod \phi(n) \quad (1)$$

$$d * e = 1 \bmod (p - 1)(q - 1) \quad (2)$$

p = F7E75FDC469067FFDC4E847C51F452DF

q = E85CED54AF57E53E092113E62F436F4F

e = 0D88C3

Firstly we must initialize values in c file.

```
BN_CTX *ctx = BN_CTX_new();

BIGNUM *p = BN_new(); // p value
BIGNUM *q = BN_new(); // q value
BIGNUM *e = BN_new(); // e value
BIGNUM *d = BN_new(); // d value
BIGNUM *p_minus = BN_new(); // (p-1) value
BIGNUM *q_minus = BN_new(); // (q-1) value
BIGNUM *f_of_fi = BN_new(); // fi value
BIGNUM *one = BN_new(); // define 1 for minus

// initialize p, q, e from PDF file
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e, "0D88C3");
BN_dec2bn(&one, "1");
```

After creating the values, we can continue our operations. First we will get the value of (p-1) and (q-1), and then we will find the inverse of the number e.

```

// Calculate (p-1) value
BN_sub(p_minus, p, one);

// Calculate (q-1) value
BN_sub(q_minus, q, one);

// fi = (p-1)*(q-1)
BN_mul(f_of_fi, p_minus, q_minus, ctx);

// d = e.modInverse(fi)
BN_mod_inverse(d, e, f_of_fi, ctx);

printBN("d = ", d);

```

We calculated d value. If we run the program, output must be like this.

```

halis@VM: /home/seed/rsa
halis@VM:/home/seed/rsa$ sudo gcc task1.c -o task1 -lcrypto
halis@VM:/home/seed/rsa$ ./task1
d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
halis@VM:/home/seed/rsa$ 

```

Figure 1: Task1 Output

$d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB$

2 Task 2: Encrypting a Message

We have a public key and a message that we need to encrypt. First we will encrypt the message using the value of e and n. Then we will check whether we have done the correct encryption by decrypting with the d value. Firstly, we must calculate the hex value of the message. ("A top secret!")

```

halis@VM:/home/seed/rsa$ python -c 'print("A top secret!".encode("hex"))'
4120746f702073656372657421
halis@VM:/home/seed/rsa$ 

```

Figure 2: Hex value of the message.

Hex value: 4120746f702073656372657421

After that we must initialize the variables.

```
// initialize m, e, n and d values from PDF file
BN_hex2bn(&m, "4120746f702073656372657421");      // "A top secret!"
BN_hex2bn(&e, "010001");
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
```

We use the public key to encrypt our message. We will then use the private key to decrypt it. Equation 3 will give us the cipher text and equation 4 will give us plain text.

$$c = m^e \bmod (n) \quad (3)$$

$$m = c^d \bmod (n) \quad (4)$$

```
// Encryption m^e mod n , , Compute c = m^e mod n
BN_mod_exp(c, m, e, n, ctx);
printBN("Encrypted Message = ", c);

// Decryption c^d mod n , Compute m_dec = c^d mod n
BN_mod_exp(m_dec, c, d, n, ctx);
printBN("Decrypted Message = ", m_dec);

printBN("Original Message = ", m);
```

If we run our program, the output will be as follows.

```
halis@VM:/home/seed/rsa$ sudo gcc task2.c -o task2 -lcrypto
halis@VM:/home/seed/rsa$ ./task2
Encrypted Message = 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
Decrypted Message = 4120746F702073656372657421
Original Message = 4120746F702073656372657421
halis@VM:/home/seed/rsa$
```

Figure 3: Output of task2.c

3 Task 3: Decrypting a Message

We have a cipher text. First, we have to decrypt it with the private key and then decode it.

C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBDFC7DCB67396567EA1E2493F

```
// Initialize the c, n and d values from PDF
BN_hex2bn(&c, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBDFC7DCB67396567EA1E2493F");
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

// Decryption c^d mod n , Compute m_dec = c^d mod n
BN_mod_exp(m_dec, c, d, n, ctx);
printBN("Decrypted Message = ", m_dec);
```

If we run our program, the output will be as follows.

```
halis@VM:/home/seed/rsa$ sudo gcc task3.c -o task3 -lcrypto
halis@VM:/home/seed/rsa$ ./task3
Decrypted Message = 50617373776F72642069732064656573
halis@VM:/home/seed/rsa$
```

Figure 4: Output of task3.c

We must decode the hex value to get string.

```
halis@VM:/home/seed/rsa$ python -c 'print("50617373776F72642069732064656573".decode("hex"))'
Password is dees
halis@VM:/home/seed/rsa$
```

Figure 5: Secret message.

Secret message: "Password is dees"

4 Task 4: Signing a Message

In this task we need to sign our message. First, we will convert our message to hexadecimal. Then we will sign our message with the private key. I will repeat the processes in two different messages and try to observe the results.

Encode the messages.

```
halis@VM:/home/seed/rsa$ python -c 'print("I owe you $2000.".encode("hex"))'  
49206f776520796f752024323030302e  
halis@VM:/home/seed/rsa$ python -c 'print("I owe you $3000.".encode("hex"))'  
49206f776520796f752024333030302e  
halis@VM:/home/seed/rsa$ █
```

Figure 6: Encoded messages.

As can be seen in Figure 6, there is only one character difference between hex values. We will sign both messages and examine the differences between signatures.

```
// initialize e, m, n from PDF file (Task 2)  
BN_hex2bn(&e, "010001");  
BN_hex2bn(&m, "49206f776520796f752024323030302e");  
BN_hex2bn(&m2, "49206f776520796f752024333030302e");  
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");  
  
// Encryption  $m^e \bmod n$ , Compute  $c = m^e \bmod n$   
BN_mod_exp(c, m, e, n, ctx);  
printBN("Signed Message (I owe you $2000.) = ", c);  
  
// Encryption  $m^e \bmod n$ , Compute  $c = m^e \bmod n$   
BN_mod_exp(c, m2, e, n, ctx);  
printBN("Signed Message (I owe you $3000.) = ", c);
```

If we run our program, the output will be as follows.

```
halis@VM:/home/seed/rsa$ sudo gcc task4.c -o task4 -lcrypto  
halis@VM:/home/seed/rsa$ ./task4  
Signed Message (I owe you $2000.) = 3A759CBF53901AC41373EEC603955A8E6AF8D3BCD5E9F6DD62C873CBB675051E  
Signed Message (I owe you $3000.) = D06908047527906C724937169FA68CE0AC442FEB99D1880438D331A88F44B074  
halis@VM:/home/seed/rsa$ █
```

Figure 7: Output of task4.c

A single character difference between the messages made the signatures completely different from each other. Therefore, we can say that the confidentiality is very good in the signing process.

5 Task 5: Verifying a Signature

First, let's get the hex value of our message.

```
halis@VM:/home/seed/rsa$ python -c 'print("Launch a missile.".encode("hex"))'  
4c61756e63682061206d697373696c652e  
halis@VM:/home/seed/rsa$ █
```

Figure 8: Hex value of a message.

Initialize the values and decrypt signs with public key.

```
// Initialize m, e and n values from PDF file  
BN_hex2bn(&m, "4c61756e63682061206d697373696c652e");  
BN_hex2bn(&e, "010001");  
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");  
  
// Initialize true Sign  
BN_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");  
  
// Print original message.  
printBN("Message = ", m);  
  
// Decryption s^e mod n , Compute m_dec = s^e mod n  
BN_mod_exp(m_dec, c, e, n, ctx);  
printBN("Decrypted Sign = ", m_dec);  
  
// Corrupted Sign  
BN_hex2bn(&s, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");  
  
// Decryption s^e mod n , Compute m_dec2 = s^e mod n  
BN_mod_exp(m_dec2, c, e, n, ctx);  
printBN("Decrypted Sign (Corrupted)= ", m_dec2);
```

If we run our program, the output will be as follows.

```
halis@VM:/home/seed/rsa$ sudo gcc task5.c -lcrypto  
halis@VM:/home/seed/rsa$ ./task5  
Message = 4C61756E63682061206D697373696C652E  
Decrypted Sign = 4C61756E63682061206D697373696C652E  
Decrypted Sign (Corrupted)= 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294  
halis@VM:/home/seed/rsa$ █
```

Figure 9: Output of task5.c

After decrypting our original signature, we got our message correctly. However, we could not get the correct message in decrypting only 1 character corrupted signature.

6 Task 6: Manually Verifying an X.509 Certificate

In this task, we will manually verify an X.509 certificate using our program.

6.1 Step 1: Download a certificate from a real web server.

First I downloaded the certificates from the web server. Then I saved it in "c0.pem" and "c1.pem" files.

```
halis@VM:/home/seed/rsa/task6$ openssl s_client -connect www.example.org:443 -showcerts
CONNECTED(00000003)
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1
verify return:1
depth=0 C = US, ST = California, L = Los Angeles, O = Internet Corporation for Assigned Names and Numbers, CN = www.example.org
verify return:1
---
Certificate chain
  0 s:/C=US/ST=California/L=Los Angeles/O=Internet Corporation for Assigned Names and Numbers/CN=www.example.org
  i:/C=US/O=digiCert Inc/CN=DigiCert TLS RSA SHA256 2020 CA1
  -----BEGIN CERTIFICATE-----
MIIG1TCBbzgAwIBAgIQd74IsIVNBXOKsMzhyauyTANBgkqhkiG9w0BAQsFADBP
MSowCQYDVQQGEwJVUzEVMBMGAIUEChMRGlnaUNlcn0gSw5jMSkwJwYDVQQDEyBE
awdpQ2VydCBUTFMgULNBFIQTII1nIAYMDIwIEhMTAeFw0yMDExMjQwMDAwMDBa
Fw0yMTEyMjUyMzU5NTlaMIGOMQswCQYDVQoGEwJVUzETMBEgA1UECBMKQ2fsaNzv
cm5pYTEUMBIGA1UEBxMLTG9zIEFuZ2VsZXMuPDAGBgNVBAoTM0ludGVybmv0IENv
cnBvcmlF0aW9uIGZvcIBc3NpZ25lZCBOYW1lcBhbmQgTnVtYmVyczEYMBYGA1UE
AxMPd3d3Lm4YWh1wbGUub3JnMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIBCqKC
AQEuuvuzMoKCP80kx2zygucA5YinrFPEK5RQP1TX7PEYUAoB06i5hIAstIKfmFxt
W2sghER1lU5rdnxQcf3fEx3sY40tY6VSBPLPhrbKozHLrQ8ZN/rYTb+hgNUE7TN
A1mP78IEkxAj4qG5tLi43q41acbult7equGxokImh+UV5Ip0EZS0tKD4vu2ksZ
040etp0k8jWdVMA27W3EwgHHNevGbWjPC0Dn7RpW13r7hFyS5Tpleywdy1nB7
ad6kcZXzbEcaFZ7ZuerA6RkPGE+PsnZRb1oFjkYoXimsuvkvFhWeHQXCGC1cuDWS
rM3cpQv0zKH2v57d15+zGls4IwIDAQABo4IDA0TCCA2UwHwYDVR0jBBgwFoAUT2ui
6qjnhIx56rTaD5iyxZV2ufQwHQDVR00BBYEFcYa+0SxHKEztqbBtInnPvt0j0X
MIGB8gNVHREEejB4gg93d3cuZXhhbXbsZ5vcmeCC2V4Yw1wbGUuY29ttgleGft
cGxLLmVkdYILZXhhbXbsZ5uZXSC2V4Yw1wbGUub3Jngg93d3cuZXhhbXbsZ5j
b22CD3d3dy5leGftcGxLMvd3d3LmV4Yw1wbGUubmV0MA4GA1UdDwEB/wQE
AwIFoAdBgNVHSUEfjAUBggrBgEFB0cDAQYIKwYBBQUHwIwgyYsGA1UdHwSBgzcB
gDA+Odygo0y4aHR0cDovL2NybmZlgnaWnlcnQuY29tL0RpZ2lDZXJ0VExTU1NB
U0hBmjU2MjAyMENBMS5jcmwwPqA8obqGOGh0dHA6Ly9jcmw0LmRpZ2ljjZXj0LnNv
bs9EawdpQ2VydFRMU1JTQVNjOTI1NjIwMjBDTEuy3jMEwGA1UdIARFMEMnwyJ
YZIZIAy9bAEBCoWkAYIKwYBBQUHAgEWGh0dHBz0i8vd3d3LmRpZ2ljjZXj0LnNv
bs9DUFMcCAYGZ4EMAQICMH0GCC5GAQUBFwEBBHewbzAkBggrBgEFB0cwAYYYahR0
cDovL29jc3AuZGlnaWnlcnQuY29tMEcGCCsGAQUBFzAChjtodIRw0i8Y2fjZXj0
cy5kaWdpY2VydC5jb20vRGlnaUNlcnRUTFNSU0FTSEyNTYyMDIwQ0ExLmNydam
BqNVHRMBAf8EAjAAIIIBQYKKwYBBAAHw0IEAgSB9gSB8wDxAhcA9lyUL9f3MCIU
VBgIMJRwjjuNNEkxzv98MLyALzE7zZOMAAAF1+73ybgaABAMASDBGAiEApGuo0Eok
8QcyLe2c0X136HPBn+0iSgDFvprJtbYS3LECIQCNC6+F+Kx1LNDAej1bW729tiE4gi
```

Figure 10: Download the certificates.

```

M: /home/seed/rsa/task6
halis@VM: /home/seed/rsa/task6$ cat c0.pem
-----BEGIN CERTIFICATE-----
MIIGIjCCBggwIBAgT0d74IsIVNBX0KsMzhya/vyTANBgkqhkiG9w0BAoqsFA0B
MQswCQYDVQoEwJvUeEWMBGA1UEChMMRG1nauUN1cn0gSw5jMSkwJyYDQ0DEvBE
aWdpQ2VvdCBUTMqU1NBfNlQTI1n1AyMD1wIEBMTAeFw0vMDExMj0wMDAwMDBa
Fw0yMjUyMjUyMjUyMjUyMjUyMjUyMjUyMjUyMjUyMjUyMjUyMjUyMjUyMjUy
cmppTUEMB1aUEBXMeTG9jIEFuZ2vS2XNxPdA6g9NVBA0Tm0LuUvbyhN0tEN
cnBvcmFoaw9uIGzv1cLB3NPz25LZCB0yW1lcybhbm0gTnvtYmVcyEMBYGA1UE
AxIMPD3d3Lm4Y1w1vbGuub33NM1B1jANBgkqhkiG9w0BAQEAAQ0A8M1BCgk
AQEAuvuzzokC80kx2vguca5YinrPEKSRP0IT7xquGokImhC+UY51pEs0tKd4v2ksZ
04qeotp0k8jdwAVMA27W3ewglvGWPJC0Dn7RqPw13r7hfys57plewjydn1b7
ad6kczX2bcaF7zuerA6RkPGE+Psn2Rb1cFJKy0xiuvvKFhWeHQXCG1cuDWS
rM3cpv0zKv5v7d15+g1s41wIDAQABo4DaTCCA2uWwYDVR0BBgwFaAUt2u1
6qiqihx56+TaD5iyxZV2ufoWtQYDVROB8t0FECYya-05xHKeztqBtImmPvt0j0X
MIGB8gNHREREeB4gg93d3cuZxhbXbzZ55vcmeC2V4Y1w1vbUuY29tggteLgFt
cgxLmVkdjILZhzbxBzZ55uXS5CC2V4Y1w1vbUu3Jng93d3cuZxhbXbzZ55j
b22C03d3dy5LeGftcG1mVkdjY1Pd3d3Lm4Y1w1vbUuMA4GA1udvEB/w0E
AwIfodBGNH5UEFUAIUbgrgBgfFB0cDAQYIKwBQQHUwIwgvsaG1uJduhsBsgzCB
gDA+oByg0YahR0cDovL29yDzG1j20wRGLnaUN1lRTUFNS10fTSEEYNTYMD1wQ8ExLmNv
dAM
BgNHRMBAfEaJAAMIIbbQKyKwYBBAHWeQ1Eag5B9g5B8w0xAhCA9jyluLS9fMCIC
VBg1MjRWjjuHg0y38mLYALzE7xzHdAFAR1+73YbgABA0SDwDAluqeu
80cyle2c0x136HPn+01sg0FpvJtbys3LE1QCN6f+KX1LND0EJE1bw729tIE4g1
1nDsg14/yayT1x0gB2Fzq5L+5qtFRFLFemTRWhsA3+9x6R9yhcsSyub2xw7K
AAABdfu92m0aaQDAEcWRQ1gaqwR+uJEv+bjokw3w4Fbsq0WzTcIKPDM0qLAZ
2qwIQDaf2xFrbwQpkgo1gezpQ992uWfLvvzMPfdntBd8vTtANBgkqhkiG9w0
A05fAA0CA0EApvoMPr4a3ob+GY49umtctUtgol4ZYXpbj+Eykdhzs++MFEdcc
MV404sAA5W06SL49W+6t1elturEz4TxEY7M54RFyJv0hLLNLCtxxcjhOHF617
qH9pKXX1pmfF1j914jtboazM3jBfcwH/zj+ku05IBYJ5y1xMm3Bcc+uZ560EB
XJXPK0xg1f3B6gwLlbR648/2/n7JuwlThsUT6nYnXmHs0rso0halgtuXcwOha
/sqUGki0xrj1Lh/dh4n6p9YJN6FitwAntb7xsV5FKaazVBXmw8issgH0huIr4Xkr
vUzLnf70ysJhvtaYrZ2MLxDG+NF18Bkkw==
-----END CERTIFICATE-----halis@VM:/home/seed/rsa/task6$ 

```

```

[12/07/20]seed@VM:~/rsa/task6$ cat cl.pem
-----BEGIN CERTIFICATE-----
MIIE6jCAK9gkAwIBAgTOCjU1wPwK9F+K1wA/3SDAnBgkqhkiG9w0BAoqsFA0B
M0swCQYDVQoEwJvUeEWMBGA1UEChMMRG1nauUN1cn0gSw5jMSkwJyYDQ0DEvBE
d3c1ZGlnalnWlClnQuY29tMSAwfgYDQ0DExdawldp02VydCBhG9iYBwgNl0vdcBD
0TAefw0yM0DA5Mj0wMDAwMDBaFw0zMDA5MjMyMzU5NT1laME8xCjBjgnVBAmt1lVT
MRUwEVyDVQoKEwAwdp2VydC3jMxKTAnBqNVBAMTIERpz2LD2XJ0fFRMjyBS
U0EgUhBmJU21D1w1jAgotExM1B1jANBgkqhkiG9w0BQFEAAQ0A80AM1tBc0kC
AQEAw0zZu0wN1PNwvsn03DwUfMRNUrpmp83cUxKb-Uu3Ny5C1dt3+Pe0j6a
xQodgojIvBbhp9Yw1hlnDQnlthks4Vbl8X1f7suhYUdeSp5QWYQY9EX0e0nwDdn
g9/n0htnTCRpt80mRDt1f0Ju9x8pi1Mbpfy0IJVNwvTRYA1ueE/1+p1hJn1uW
raKlmxw0H2f6Vg0LbdbtN+12t1JLyrVmuZb9phjPv1j1hprPGU19W100gLB
AfRsyjK7t14nhyFk3TujNa3sNk+cr0U6JWvhqkjkkDk4775u+kfbn08lwVzV21r
eacrcge7XQPUDT1TAHK+jzQ0IDQA0B41BrjCCaoHgYDVR00BypEFLdroruqo
qSMeeq02g+YswVdn0B8G1UdIwQYMBaFAPeUDW0Uy7zCj4hsbw5eyPdFV
MA4GA1u0dwEBw/q0EawBjhjdbgNHvHSUfU)ABgq+BgEFB0cDAQYIKwBBSQJHwAvIw
EqYDVR0TA0H/BAgwB0zKwIBADB2B9g/BgEFB0cBA0RMGwJAYIKwBBSQJHwAgG
GGh0dHA6L9y93NwLmRpZ21jZXJ0LmNvbTBABgw/BgEFB0cDAQYIKwBBSQJHwAgG
Y2V7ydhMuZGlnaWnLcn0uV29t0RpZ22DZXJ0R2xvYmfu9vndENBLmNy0DB78gNV
HR8EdBMyDegenAzhjFodHRw1i0vY3jsMy5kawldp2VydC5j1b20vRGlnaunlcnR
bg91YwxSb29800euy3jshDegenAzhjFodHRw0i8.Y3j3.NC5kaWdp2Vyc5j1b20v
RGlnaunlcnRhb9jy1wXsB29800euy3j3.MDAGA1UdIAQPMCvBwVfZ4EMAEQewCAYG
Z4EMAQ1BMAgGBmeBDAECAjA1BgZng0wBAgMw0D0YJkZ1hvcNA0ELB0A0dgqEBAh
er3onPa679n/qw1bJhkrkW3Ex35JH/E6f7tDBpATho+vFscH90cnfjk+UFSxGkQn
05D5nk0LHEIqdinFOB8stchL44Gw+0w82zXHf08hVt1hbcpnj5h232sb0HIM
ULkuKQg/Yfk02zH6Lw/VEWwtIwvPqU7/wlhnnOKK24fXsuhe50g66sSmvKvhMnb
0qZgYorAKHKhjMo1wJk1KnpMzTfMlhoClw+dj20t1o779rxKtG14ZxuYR1H
as6xuuwAwapu3r9xZf+1ngkquqtglLozZxq8ox/fpf2kUcwa/d5KxtVtzhwoT0jzI
8ks5T1KE5aM2ke4f97o=
-----END CERTIFICATE-----[12/07/20]seed@VM:~/rsa/task6$ 

```

Figure 11: Save the files.

6.2 Step 2: Extract the public key (e, n) from the issuer's certificate.

I obtained the public key value from the certificate provided by the issuer.

```

halis@VM:/home/seed/rsa/task6$ openssl x509 -in cl.pem -noout -modulus
Modulus=C14BB3654770BCDD4F58DBEC9CEDC366E51F311354AD4A66461F2C0AEC6407E52EDCDCB90A20
52416610604F571349F0E8376783DFE7D34B674C2251A6DF0E9910ED57517426E27DC7CA622E131B7F23
499AECC767D6E33EF5E3D6125E44F1BF71427D58840380B18101FAF9CA32BBB48E278727C52B74D4A8D6
D321300793EA99F5
halis@VM:/home/seed/rsa/task6$ 

```

Figure 12: n value

```

halis@VM:/home/seed/rsa/task6$ openssl x509 -in cl.pem -text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      0a:35:08:d5:5c:29:2b:01:7d:f8:ad:65:c0:0f:f7:e4
    Signature Algorithm: sha256WithRSAEncryption
      Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert Global Ro
A
  Validity
    Not Before: Sep 24 00:00:00 2020 GMT
    Not After : Sep 23 23:59:59 2030 GMT
  Subject: C=US, O=DigiCert Inc, CN=DigiCert TLS RSA SHA256 2020 CA1
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
        Modulus:
          00:c1:4b:b3:65:47:70:bc:dd:4f:58:db:ec:9c:ed:
          c3:66:e5:1f:31:13:54:ad:4a:66:46:1f:2c:0a:ec:
          64:07:e5:2e:dc:dc:b9:0a:20:ed:df:e3:c4:d0:9e:
          9a:a9:7a:1d:82:88:e5:11:56:db:1e:9f:58:c2:51:
          e7:2c:34:0d:2e:d2:92:e1:56:cb:f1:79:5f:b3:bb:
          87:ca:25:03:7b:9a:52:41:66:10:60:4f:57:13:49:
          f0:e8:37:67:83:df:e7:d3:4b:67:4c:22:51:a6:df:
          0e:99:10:ed:57:51:74:26:e2:7d:c7:ca:62:2e:13:
          1b:7f:23:88:25:53:6f:c1:34:58:00:8b:84:ff:f8:
          be:a7:58:49:22:7b:96:ad:a2:88:9b:15:bc:a0:7c:
          df:e9:51:a8:d5:b0:ed:37:e2:36:b4:82:4b:62:b5:
          49:9a:ec:c7:67:d6:e3:3e:f5:e3:d6:12:5e:44:f1:
          bf:71:42:7d:58:84:03:80:b1:81:01:fa:f9:ca:32:
          bb:b4:8e:27:87:27:c5:2b:74:d4:a8:d6:97:de:c3:
          64:f9:ca:ce:53:a2:56:bc:78:17:8e:49:03:29:ae:
          fb:49:4f:a4:15:b9:ce:f2:5c:19:57:6d:6b:79:a7:
          2b:a2:27:20:13:b5:d0:3d:40:d3:21:30:07:93:ea:
          99:f5
        Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Subject Key Identifier:
      B7:6B:A2:EA:A8:AA:84:8C:79:EA:B4:DA:0F:98:B2:C5:95:76:B9:F4

```

Figure 13: e value

6.3 Step 3: Extract the signature from the server's certificate.

I obtained the signature value from the certificate provided by the server.

Then I removed the punctuation marks in the signature.

```

Signature Algorithm: sha256WithRSAEncryption
a7:2a:10:30:5c:b8:6b:7a:1b:f8:66:38:f6:e9:a0:0a:d5:13:
82:82:f8:65:89:57:a5:b8:eb:13:29:1d:84:6c:ec:fb:e3:05:
11:d7:1e:31:5e:0e:e2:c0:00:e5:6d:06:48:be:3d:55:6f:ba:
b7:11:35:b6:ea:c4:cf:84:f1:30:4c:bb:33:9e:11:17:2b:c9:
d2:19:4b:2c:d0:ad:5f:17:23:84:e1:df:17:a2:3b:a8:7f:69:
29:7c:48:a6:61:5f:26:3f:75:e2:3b:5b:a3:36:b3:1c:cd:e3:
04:57:30:1f:fc:c9:fa:4b:8e:48:80:58:27:9c:a2:c7:c3:26:
dc:17:02:fa:e6:6c:ea:81:01:5c:92:8f:d3:18:08:17:70:7a:
c2:a3:4b:6c:3a:fa:e3:cf:f6:fe:7e:c9:56:e5:a5:4e:1b:14:
4f:a9:98:9d:79:b1:1e:c3:ab:b1:0d:15:85:a9:46:b6:e5:c2:
58:e8:5a:fe:c8:14:28:68:90:c6:b8:c8:94:7f:e1:0f:89:fa:
a7:d6:09:37:a1:62:b7:00:27:b5:be:f1:b1:5e:45:28:06:b3:
54:15:e6:c3:c8:ac:82:01:ce:86:e2:2b:e1:7a:e4:bd:4c:cb:
9c:5e:d0:62:c2:61:bd:8b:5a:62:b6:76:30:bc:46:0f:e3:45:
23:c0:64:5f
halis@VM:/home/seed/rsa/task6$ █

```

Figure 14: Server's signature

```

halis@VM:/home/seed/rsa/task6$ cat signature | tr -d '[:space:]:' 
a72a10305cb86b7a1bf86638f6e9a00ad5138282f8658957a5b8eb13291d846cecfbe30511d71e315e0e
e2c000e56d0648be3d556fbab71135b6eac4cf84f1304ccb339e11172bc9d2194b2cd0ad5f172384e1df
17a23ba87f69297c48a6615f263f75e23b5ba336b31ccde30457301ffcc9fa4b8e488058279ca2c7c326
dc1702fae66cea81015c928fd3180817707ac2a34b6c3afae3cff6fe7ec956e5a54e1b144fa9989d79b1
1ec3abb10d1585a946b6e5c258e85afec814286890c6b8c8947fe10f89faa7d60937a162b70027b5bef1
b15e452806b35415e6c3c8ac8201ce86e22be17ae4bd4ccb9c5ed062c261bd8b5a62b67630bc460fe345
23c0645fhalis@VM:/home/seed/rsa/task6$ █

```

Figure 15: Signature

6.4 Step 4: Extract the body of the server's certificate.

Then I got the body part in the server's certificate with the following command.

In the last step I got the hash value for verification

```

$ openssl asn1parse -i -in c0.pem
$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
$ sha256sum c0_body.bin

```

```

halis@VM:/home/seed/rsa/task6$ sha256sum c0_body.bin
373e90af567edd6dab8b435927ff7cc2db6e8a17337c6eb9aabea0e4907b0eff  c0_body.bin
halis@VM:/home/seed/rsa/task6$ █

```

Figure 16: Hash of server's certificate's body

Hash: 373e90af567edd6dab8b435927ff7cc2db6e8a17337c6eb9aabea0e4907b0eff

6.5 Step 5: Verify the signature.

At the last task, we will decrypt the signature value using the values we obtained and check it with the hash value we obtained. If we can get the same hash value, we will have successfully completed our task.

```
// n value
BN_hex2bn(&n, "C14BB3654770BCDD4F58DBEC9CEDC366E51F311354AD4A6646\
1F2C0AEC6407E52EDDCB90A20EDDFE3C4D09E9AA97A1D8288E51156DB1E9F58C2\
51E72C340D2ED292E156CBF1795FB3BB87CA25037B9A52416610604F571349F0E8\
376783DFE7D34B674C2251A6DF0E9910ED57517426E27DC7CA622E131B7F238825\
536FC13458008B84FFF8BEA75849227B96ADA2889B15BCA07CDFE951A8D5B0ED37\
E236B4824B62B5499AECC767D6E33EF5E3D6125E44F1BF71427D58840380B18101\
FAF9CA32BBB48E278727C52B74D4A8D697DEC364F9CACE53A256BC78178E490329\
AEFB494FA415B9CEF25C19576D6B79A72BA2272013B5D03D40D321300793EA99F5");

// e value - 65537
BN_hex2bn(&e, "10001");

// Signature
BN_hex2bn(&s, "a72a10305cb86b7a1bf86638f6e9a00ad5138282f8658957a5\
b8eb13291d846cecfbe30511d71e315e0ee2c000e56d0648be3d556fbab71135b6\
eac4cf84f1304ccb339e11172bc9d2194b2cd0ad5f172384e1df17a23ba87f69297\
c48a6615f263f75e23b5ba336b31ccde30457301ffcc9fa4b8e488058279ca2c7c32\
6dc1702fae66cea81015c928fd3180817707ac2a34b6c3afae3cff6fe7ec956e5a54\
e1b144fa9989d79b11ec3abb10d1585a946b6e5c258e85afec814286890c6b8c8947\
fe10f89faa7d60937a162b70027b5bef1b15e452806b35415e6c3c8ac8201ce86e22\
be17ae4bd4ccb9c5ed062c261bd8b5a62b67630bc460fe34523c0645f");

// Decryption s^e mod n , Compute m_dec = c^e mod n
BN_mod_exp(s_dec, s, e, n, ctx)
printBN("Decrypted Sign = ", s_dec);

// Last 32 bit must be same with this hash
printf("the pre-computed hash was: ");
printf("373e90af567edd6dab8b435927ff7cc2db6e8a17337c6eb9aabea0e4907b0eff\n");
```

If we run our program, the output will be as follows.

Figure 17: Output of task6.c : Result

As can be seen in Figure 17, when we deciphered our signature, the last 32 bits were the same as the hash value we obtained. Therefore, we manually verified the certificate.