



Faculty of Engineering and Technology

Electrical and Computer Engineering Department

INFORMATION SECURITY AND COMPUTER NETWORK

LABORATORY ENCS5121

Report IV

Experiment # 5: RSA Public-Key Encryption and Signature

---

Student Name: Yara Darabumukho

Student Number: 1211269

Instructor: Dr. Ahmad Alsadeh

Teaching Assistant: Eng. Tariq Odeh

Section No: 1

Date: 10/11/202

## **Abstract**

In this experiment the main aim is to understand and apply the RSA Public-Key Encryption and Signature and learn more about the digital signature. This report contains the sixth task from the lab, Manually Verifying an X.509 Certificate.

## Table of Contents

Step I.....	1
Step II .....	2
N .....	2
E.....	2
Step III.....	3
Step IV .....	5
Step V .....	7
Appendix.....	8

## Table of Figures

Figure 1: Certificate downloading.....	1
Figure 2: Files. ....	1
Figure 3: N extraction. ....	2
Figure 4: E extraction.....	2
Figure 5: Command runs. ....	3
Figure 6: The signature.....	3
Figure 7: Signature temporary file. ....	4
Figure 8: The required format.....	4
Figure 9: Offset command run.....	5
Figure 10: Offset number.....	5
Figure 11: Hashed body result.....	6
Figure 12: Verify Signature. ....	7

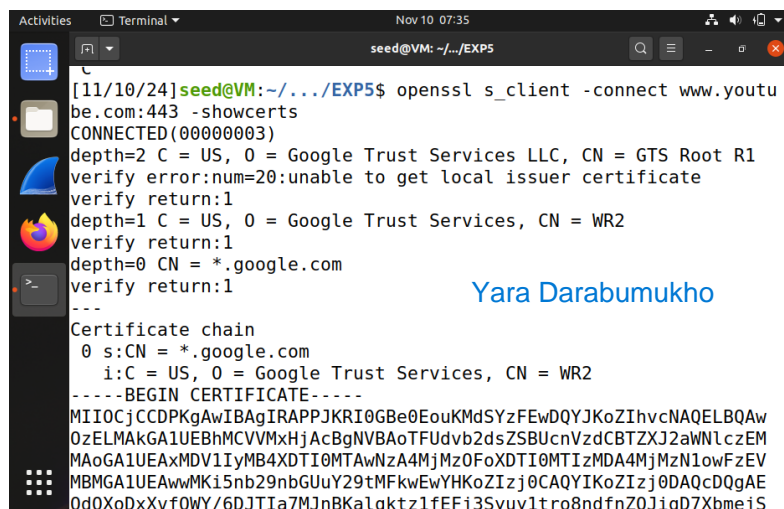
## Step I

*Download a certificate from a real web server.*

In this step the following command runs:

⇒ `openssl s_client -connect youtube.com:443 -showcerts`

which retrieve the server certificate.

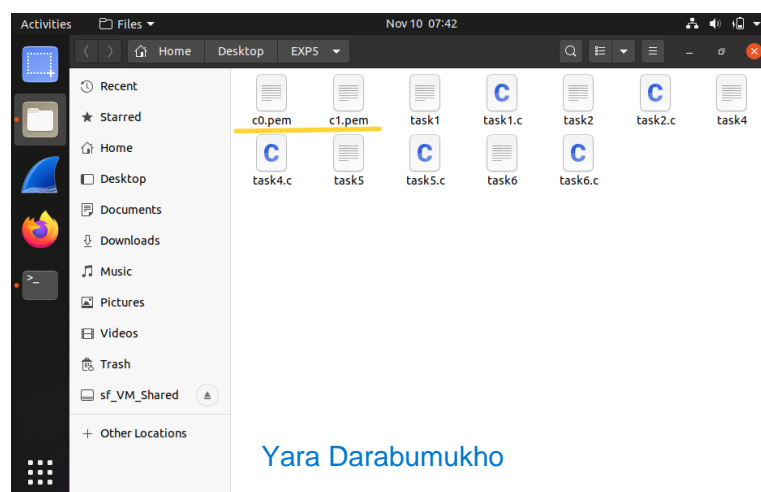


```
Activities Terminal Nov 10 07:35
seed@VM: ~/.../EXP5
[11/10/24]seed@VM:~/.../EXP5$ openssl s_client -connect www.youtu
be.com:443 -showcerts
CONNECTED(00000003)
depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=1 C = US, O = Google Trust Services, CN = WR2
verify return:1
depth=0 CN = *.google.com
verify return:1
-----
Certificate chain
 0 s:CN = *.google.com
  i:C = US, O = Google Trust Services, CN = WR2
-----BEGIN CERTIFICATE-----
MIIOCjCCDPKGAwIBAgIRAPPJKRI0GBBe0EouKMdSYzFEwDQYJKoZIhvcNAQELBQAw
OzELMAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvb2dsZSBucnVzdCBTZXJ2aWNlczEM
MAoGA1UEAxMDV1IyMB4XDTE0MTAwNzA4MjMzOFoXDTE0MTIzMDA4MjMzN1owFzEV
MBMGA1UEAwMKi5nb29nbGUuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE
0d0XoDxXvFOWY/6DJTIA7MJnBKalaktz1fEFi3Svuv1tro8ndfnZ0JiaD7Xbme1S
```

Yara Darabumukho

*Figure 1: Certificate downloading.*

This is the data shown by the pervious command, the first certificate “server’s certificate” copied in a file called `c0.pem`, and the second certificate “issuer’s certificate” copied in a file called `c1.pem`.



*Figure 2: Files.*

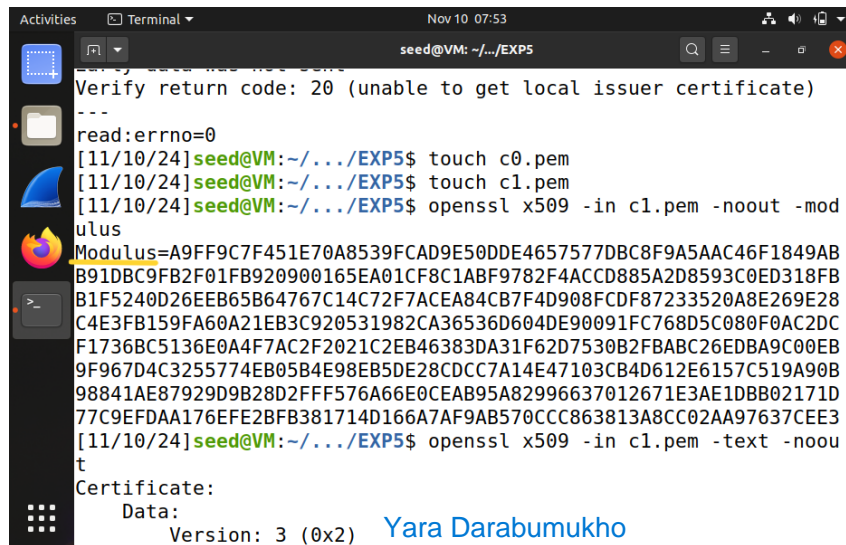
## Step II

*Extract the public key ( $e, n$ ) from the issuer's certificate.*

**N**

The value of N extracted by using the following command that applies at c1.pem file “issuer’s certificate”:

⇒ `openssl x509 -in c1.pem -noout -modulus`



```
Activities Terminal Nov 10 07:53 seed@VM: ~/.../EXP5
Verify return code: 20 (unable to get local issuer certificate)
---
read:errno=0
[11/10/24]seed@VM:~/.../EXP5$ touch c0.pem
[11/10/24]seed@VM:~/.../EXP5$ touch c1.pem
[11/10/24]seed@VM:~/.../EXP5$ openssl x509 -in c1.pem -noout -modulus
Modulus=A9FF9C7F451E70A8539FCAD9E50DDE4657577DBC8F9A5AAC46F1849AB
B91DBC9FB2F01FB920900165EA01CF8C1ABF9782F4ACCD885A2D8593C0ED318FB
B1F5240D26EEB65B64767C14C72F7ACEA84CB7F4D908FCDF87233520A8E269E28
C4E3FB159FA60A21EB3C920531982CA36536D604DE90091FC768D5C080F0AC2DC
F1736BC5136E0A4F7AC2F2021C2EB46383DA31F62D7530B2FBABC26EDBA9C00EB
9F967D4C3255774EB05B4E98EB5DE28CDDC7A14E47103CB4D612E6157C519A90B
98841AE87929D9B28D2FFF576A66E0CEAB95A82996637012671E3AE1DBB02171D
77C9EFDAA176EFE2BFB381714D166A7AF9AB570CCC863813A8CC02AA97637CEE3
[11/10/24]seed@VM:~/.../EXP5$ openssl x509 -in c1.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
```

*Figure 3: N extraction.*

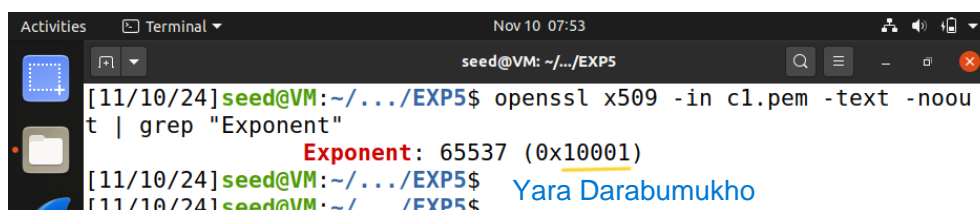
**E**

In the manual the value of E extracted by using the following command that applies at c1.pem file “issuer’s certificate”:

⇒ `openssl x509 -in c1.pem -text -noout`

Running the previous command shows a big amount of the data, so we can apply a grep command to find it directly insisted of find the key manually. So, this is the used command:

⇒ `openssl x509 -in c1.pem -text -noout | grep “Exponent”`



```
Activities Terminal Nov 10 07:53 seed@VM: ~/.../EXP5
[11/10/24]seed@VM:~/.../EXP5$ openssl x509 -in c1.pem -text -noout | grep "Exponent"
Exponent: 65537 (0x10001)
[11/10/24]seed@VM:~/.../EXP5$
```

*Figure 4: E extraction.*

### Step III

*Extract the signature from the server's certificate.*

In this step the following command runs:

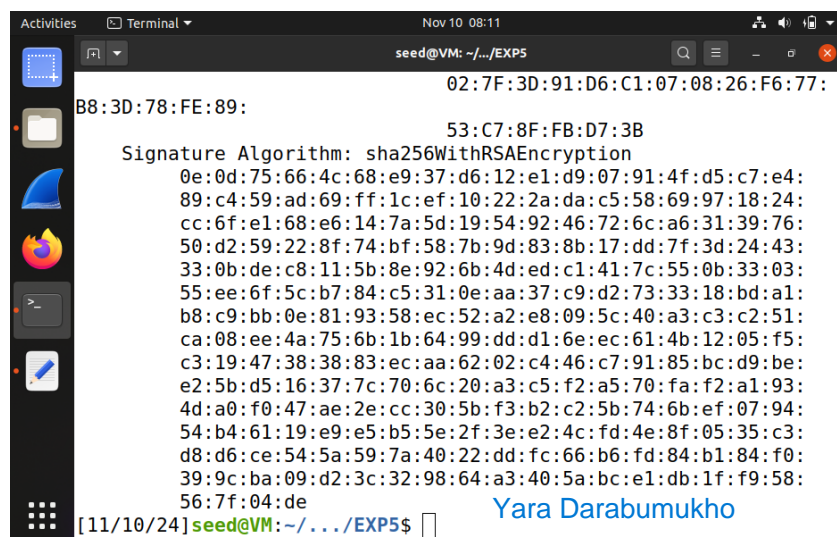
⇒ `openssl x509 -in c0.pem -text -noout`



```
Nov 10 08:11
seed@VM: ~/.../EXP5
[11/10/24]seed@VM:~/.../EXP5$ openssl x509 -in c0.pem -text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      f3:c9:29:12:34:18:17:b4:12:8b:8a:31:d4:98:cc:51
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = US, O = Google Trust Services, CN = WR2
    Validity
      Not Before: Oct  7 08:23:38 2024 GMT
      Not After : Dec 30 08:23:37 2024 GMT
    Subject: CN = *.google.com
    Subject Public Key Info:
      Public Key Algorithm: id-ecPublicKey
      Public-Key: (256 bit)
      pub:
        04:39:d3:97:a0:3c:57:bd:f4:16:63:fe:83:25:32:
        1a:ec:c2:67:04:a6:a5:aa:4b:73:d5:f1:05:8f:74:
        b2:bb:2d:6d:ae:8f:27:75:f9:d9:38:98:a0:0f:b5:
```

*Figure 5: Command runs.*

The command displays a large amount of data, and at the end of it the signature is found as shown in the following figure:



```
Nov 10 08:11
seed@VM: ~/.../EXP5
02:7F:3D:91:D6:C1:07:08:26:F6:77:
B8:3D:78:FE:89:
53:C7:8F:FB:D7:3B
Signature Algorithm: sha256WithRSAEncryption
0e:0d:75:66:4c:68:e9:37:d6:12:e1:d9:07:91:4f:d5:c7:e4:
89:c4:59:ad:69:ff:1c:ef:10:22:2a:da:c5:58:69:97:18:24:
cc:6f:e1:68:e6:14:7a:5d:19:54:92:46:72:6c:a6:31:39:76:
50:d2:59:22:8f:74:bf:58:7b:9d:83:8b:17:dd:7f:3d:24:43:
33:0b:de:c8:11:5b:8e:92:6b:4d:ed:c1:41:7c:55:0b:33:03:
55:ee:6f:5c:b7:84:c5:31:0e:aa:37:c9:d2:73:33:18:bd:a1:
b8:c9:bb:0e:81:93:58:ec:52:a2:e8:09:5c:40:a3:c3:c2:51:
ca:08:ee:4a:75:6b:1b:64:99:dd:d1:6e:ec:61:4b:12:05:f5:
c3:19:47:38:38:83:ec:aa:62:02:c4:46:c7:91:85:bc:d9:be:
e2:5b:d5:16:37:7c:70:6c:20:a3:c5:f2:a5:70:fa:f2:a1:93:
4d:a0:f0:47:ae:2e:cc:30:5b:f3:b2:c2:5b:74:6b:ef:07:94:
54:b4:61:19:e9:e5:b5:5e:2f:3e:e2:4c:fd:4e:8f:05:35:c3:
d8:d6:ce:54:5a:59:7a:40:22:dd:fc:66:b6:fd:84:b1:84:f0:
39:9c:ba:09:d2:3c:32:98:64:a3:40:5a:bc:e1:db:1f:f9:58:
56:7f:04:de
[11/10/24]seed@VM:~/.../EXP5$
```

*Figure 6: The signature.*

The signature should be in specific format without any spaces or colons. So, the signature copes in a file called sig.txt, then apply the following command to remove them:

⇒ `cat sig.txt | tr -d '[:space:]:'`

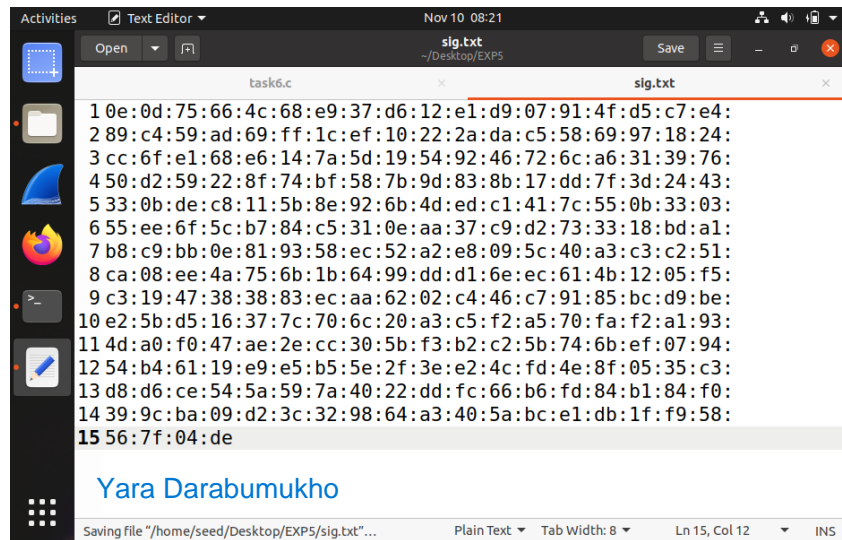


Figure 7: Signature temporary file.

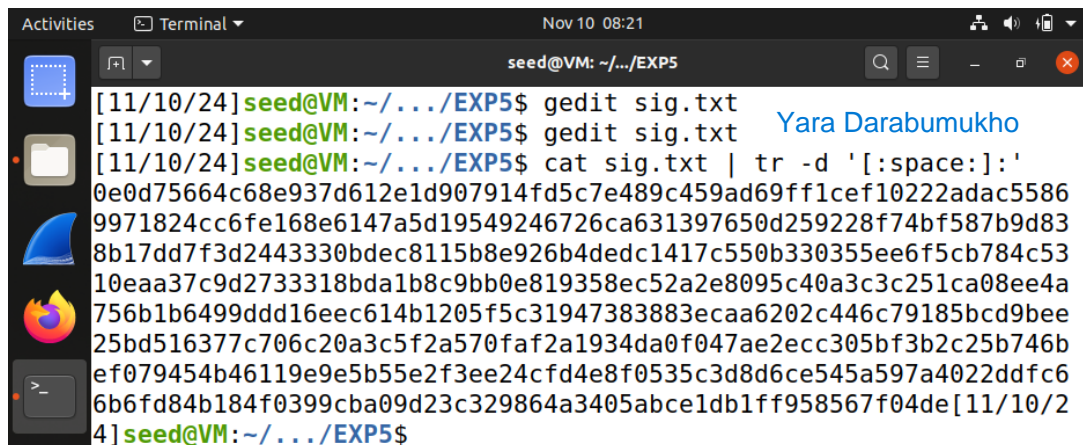


Figure 8: The required format.

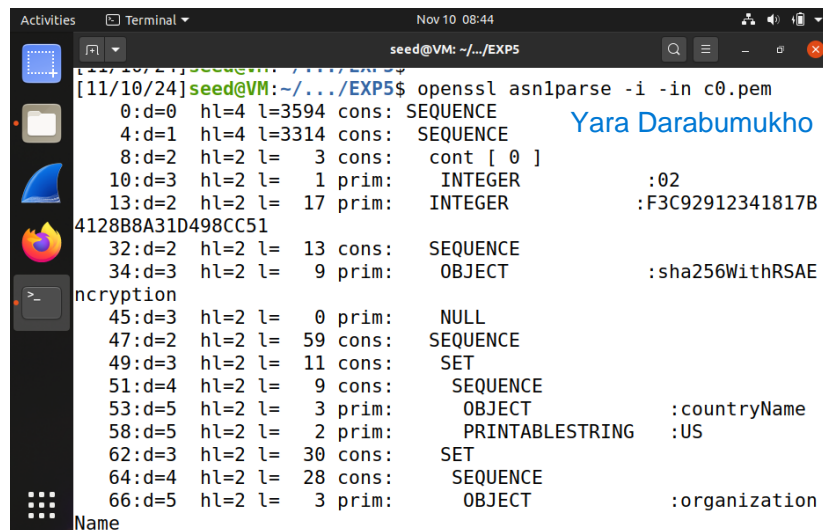


## Step IV

*Extract the body of the server's certificate.*

As a **first step** the offset should be determined using the following command:

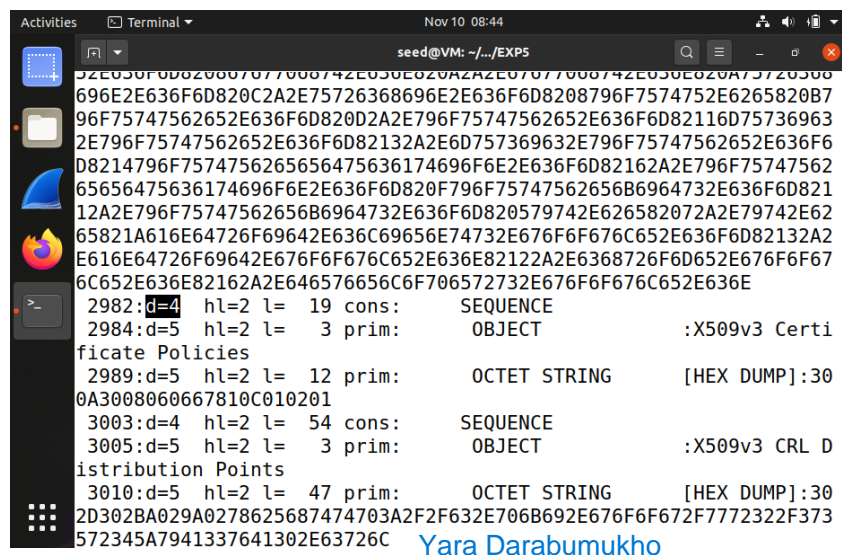
⇒ `openssl asn1parse -i -in c0.pem`



```
Nov 10 08:44
seed@VM: ~/.../EXP5
[11/10/24] seed@VM: ~/.../EXP5$ openssl asn1parse -i -in c0.pem
0:d=0  hl=4  l=3594 cons: SEQUENCE
4:d=1  hl=4  l=3314 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           :F3C92912341817B
4128B8A31D498CC51
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= 59 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            :organization
Name
```

Figure 9: Offset command run.

As shown in the previous figure running the command displays a large amount of data, the number of the offset found almost in the end of the data as shown in the next figure:



```
Nov 10 08:44
seed@VM: ~/.../EXP5
52E030F0D020007077008742E030E020A2E07077008742E030E020A73720308
696E2E636F6D820C2A2E75726368696E2E636F6D8208796F7574752E6265820B7
96F75747562652E636F6D820D2A2E796F75747562652E636F6D82116D75736963
2E796F75747562652E636F6D82132A2E6D757369632E796F75747562652E636F6
D8214796F7574756265656475636174696F6E2E636F6D82162A2E796F75747562
65656475636174696F6E2E636F6D820F796F75747562656B6964732E636F6D821
12A2E796F75747562656B6964732E636F6D820579742E626582072A2E79742E62
65821A616E64726F69642E636C69656E74732E676F6F676C652E636F6D82132A2
E616E64726F69642E676F6F676C652E636E82122A2E6368726F6D652E676F6F7
6C652E636E82162A2E646576656C6F706572732E676F6F676C652E636E
2982:d=4  hl=2  l= 19 cons: SEQUENCE
2984:d=5  hl=2  l= 3 prim: OBJECT            :X509v3 Certi
ficate Policies
2989:d=5  hl=2  l= 12 prim: OCTET STRING      [HEX DUMP]:30
0A3008060667810C010201
3003:d=4  hl=2  l= 54 cons: SEQUENCE
3005:d=5  hl=2  l= 3 prim: OBJECT            :X509v3 CRL D
istribution Points
3010:d=5  hl=2  l= 47 prim: OCTET STRING      [HEX DUMP]:30
2D302BA029A0278625687474703A2F2F632E706B692E676F6F7772322F373
572345A7941337641302E63726C
Yara Darabumukho
```

Figure 10: Offset number.

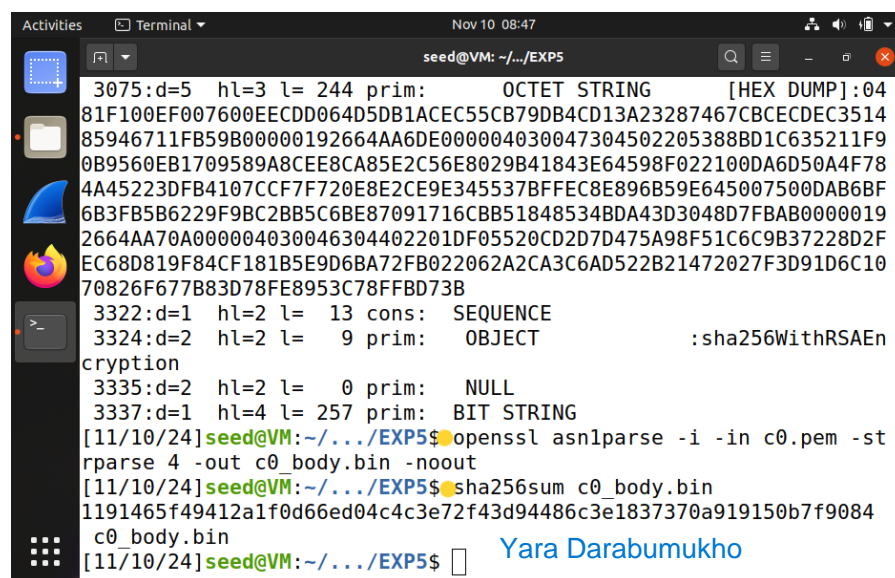
For X509 certificates the offset is always equal to 4.

In the [second step](#) the following commands run:

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

The first command gets the body of the certificate which stored in the `c0_body.bin` file, and then in the second command the hash calculated for that body.

- ⇒ The result of the hash function will be used in the comparison next step.



```
Activities Terminal Nov 10 08:47
seed@VM: ~/.../EXP5
3075:d=5 hl=3 l= 244 prim: OCTET STRING [HEX DUMP]:04
81F100EF007600EECD064D5DB1ACEC55CB79DB4CD13A23287467CBCECDEC3514
85946711FB59B00000192664AA6DE000004030047304502205388BD1C635211F9
0B9560EB1709589A8CEE8CA85E2C56E8029B41843E64598F022100DA6D50A4F78
4A45223DFB4107CCF7F720E8E2CE9E345537BFFEC8E896B59E645007500DAB6BF
6B3FB5B6229F9BC2BB5C6BE87091716CBB51848534BDA43D3048D7FBAB0000019
2664AA70A000004030046304402201DF05520CD2D7D475A98F51C6C9B37228D2F
EC68D819F84CF181B5E9D6BA72FB022062A2CA3C6AD522B21472027F3D91D6C10
70826F677B83D78FE8953C78FFBD73B
3322:d=1 hl=2 l= 13 cons: SEQUENCE
3324:d=2 hl=2 l= 9 prim: OBJECT :sha256WithRSAEn
ryption
3335:d=2 hl=2 l= 0 prim: NULL
3337:d=1 hl=4 l= 257 prim: BIT STRING
[11/10/24]seed@VM:~/.../EXP5$ openssl asn1parse -i -in c0.pem -st
rparse 4 -out c0_body.bin -noout
[11/10/24]seed@VM:~/.../EXP5$ sha256sum c0_body.bin
1191465f49412a1f0d66ed04c4c3e72f43d94486c3e1837370a919150b7f9084
c0_body.bin
[11/10/24]seed@VM:~/.../EXP5$ Yara Darabumukho
```

Figure 11: Hashed body result.

### Step V

*Verify the signature.*

In this step the value of the computed hash using the public key (e, n) and the signature, which we modify and calculate from the certificate. It compared with the certificate value that generated in the previous step.

⇒ **Note:**

The computed hash generated using a code that will be attached at the end of the report in the appendix part.

ActivitiesTerminalNov 08:50seed@VM: ~/.../EXPS

cryption  
3335:d=2 hl=2 l= 0 prim: NULL Yara Darabumukho  
3337:d=1 hl=4 l= 257 prim: BIT STRING  
[11/10/24] seed@VM:~/.../EXPS\$ openssl asnlpase -i -in c0.pem -st  
rpase 4 -out c0\_body.bin -noout  
[11/10/24] seed@VM:~/.../EXPS\$ sha256sum c0\_body.bin  
l191465f49412a1f0d66ed04c4c3e72f43d94486c3e1837370a919150b7f9084  
c0\_body.bin  
[11/10/24] seed@VM:~/.../EXPS\$ gcc task6.c -o task6 -lcrypto  
[11/10/24] seed@VM:~/.../EXPS\$ ./task6  
Message = 01FFF  
FF  
FF  
FF  
FF  
FF  
FF  
FF  
FF  
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003031300D0609608648016503040201050004  
20l191465f49412A1F0D66ED04C4C3E72F43D94486C3E1837370A919150B7F908  
4  
[11/10/24] seed@VM:~/.../EXPS\$

Figure 12: Verify Signature.

So, as shown in the previous figure the server certificate's signature and the computed hash using the CA's public key same as each other. So, the certificate is valid, authentic, and was issued by the trusted CA. if they don't match each other, then the certificate is not valid, which could imply tampering or an untrusted source.

## Appendix

```
#include <stdio.h>
```

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

```
#define NBITS 256
```

```
void printBN (char *msg, BIGNUM *a) // BN to Hex, then print the hex value
```

```
{  
    char *num_string = BN_bn2hex(a); // bn => decimal  
    printf ("%s %s\n", msg, num_string);  
    OPENSSL_free(num_string);  
}
```

```
int main()
```

```
{  
    BN_CTX *ctx = BN_CTX_new();  
    BIGNUM *s = BN_new();  
    BIGNUM *n = BN_new();  
    BIGNUM *e = BN_new();  
    BIGNUM *msg = BN_new();  
    // n, e, and s that extract from the certificate in steps number 2 and 3  
    BN_hex2bn(&n,  
"A9FF9C7F451E70A8539FCAD9E50DDE4657577DBC8F9A5AAC46F1849ABB91DBC9  
FB2F01FB920900165EA01CF8C1ABF9782F4ACCD885A2D8593C0ED318FBB1F5240D  
26EEB65B64767C14C72F7ACEA84CB7F4D908FCDF87233520A8E269E28C4E3FB159F  
A60A21EB3C920531982CA36536D604DE90091FC768D5C080F0AC2DCF1736BC5136E  
0A4F7AC2F2021C2EB46383DA31F62D7530B2FBABC26EDBA9C00EB9F967D4C32557  
74EB05B4E98EB5DE28CDCC7A14E47103CB4D612E6157C519A90B98841AE87929D9  
B28D2FFF576A66E0CEAB95A82996637012671E3AE1DBB02171D77C9EFDAA176EFE  
2BFB381714D166A7AF9AB570CCC863813A8CC02AA97637CEE3");  
    BN_hex2bn(&e, "10001");  
    BN_hex2bn(&s,  
"0e0d75664c68e937d612e1d907914fd5c7e489c459ad69ff1cef10222adac55869971824cc6fe  
168e6147a5d19549246726ca631397650d259228f74bf587b9d838b17dd7f3d2443330bdec81  
15b8e926b4dedc1417c550b330355ee6f5cb784c5310eaa37c9d2733318bda1b8c9bb0e819358  
ec52a2e8095c40a3c3c251ca08ee4a756b1b6499ddd16eec614b1205f5c31947383883ecaa620
```

```
2c446c79185bcd9bee25bd516377c706c20a3c5f2a570faf2a1934da0f047ae2ecc305bf3b2c25b
746bef079454b46119e9e5b55e2f3ee24cfd4e8f0535c3d8d6ce545a597a4022ddfc66b6fd84b1
84f0399cba09d23c329864a3405abce1db1ff958567f04de");
```

```
BN_mod_exp(msg, s, e, n, ctx); //msg = s^e mod n
printBN("Message = ", msg);
return 0;
```

```
}
```