Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

Lab 3: Cryptography

Virtual machine:

In this lab, you will be using the **Ubuntu 20** VM.

The credentials are:

Username: sec-lab
Password: untccdc

IMPORTANT NOTES (READ THIS BEFORE YOU START):

1. Before running the lab, customize the command prompt to show your EUID refer to the manual provided on the lab page.
2. If you want to create your answers file easier, use google docs. Upload the pdf question into drive, open it as a google doc modify the answers.
3. Save your screenshots then import them to pdf using the insert tap or after taking a shot simply by short key (ctrl + v).
4. Export your submission as pdf format. (File/download/PDF document).
5. Address all the questions (Q1, Q2, etc.) marked in bold. When a screenshot is requested, try to fit all the results in one image. If this is not possible, then attach multiple screenshots.
6. When a question is asked, e.g., “Who is an owner of the file?”, type your answer, do not simply provide a screenshot.

Introduction

The learning objective of this lab is for students to get familiar with the concepts in the secret key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms and their modes of operation. We will study tools and libraries for providing data confidentiality.

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

OpenSSL (https://www.openssl.org/) is toolkit for the Transport Layer Security (TLS) protocol, and also a general-purpose cryptographic library. Its latest full-featured version OpenSSL 1.1.1 is installed on the VM.

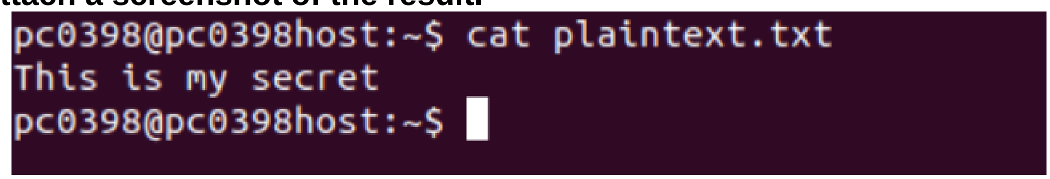
Section 1: Symmetric Encryption using OpenSSL

Encryption and decryption is performed using “openssl enc” and “openssl dec” commands, respectively. You may type “man openssl” to learn more.

1. In your home directory, create a text file *plaintext.txt* and write a sentence “This is my secret message” into it (make sure to close the file). To confirm, type:

   cat plaintext.txt

Q1: Attach a screenshot of the result.



2. Let us now encrypt this file using a password. Type:

   openssl enc -aes-256-ctr -pass pass:***euid*** -pbkdf2 -in plaintext.txt -out ciphertext.bin

   The first option requests to use the AES-256 cipher in the counter (CTR) mode. The second option defined a password to be used for encryption, and the next option requests to use the PBKDF2 algorithm for generating a key from the password.

   1. In this exercise, for simplicity, use your EUID as a password (for example, if your EUIDis “aa0001”, then the respective option will be written as “-pass pass:aa0001”). Note that in practice, such a password should never be used as it is very weak (i.e., too short and too easy to guess). The remaining options define the filenames for input

   (the plaintext) and output (the ciphertext).

   Note: If the “-pass” option was not used, then the utility would request the password to be entered manually (two times – the second one for confirmation).

3. Display the contents of the ciphertext file:

   hexdump -C ciphertext.bin

Q2: Attach a screenshot of the result.

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

```
pc0398@pc0398host:~$ cat plaintext.txt
This is my secret
pc0398@pc0398host:~$ openssl enc -aes-256-ctr -pass pass:pc0398 -pbkdf
2 -in plaintext.txt -out ciphertext.bin
pc0398@pc0398host:~$ hexdump -C ciphertext.bin
00000000  53 61 6c 74 65 64 5f 5f  fb 46 6a 65 60 d1 d9 9a  |Salted__.
Fje`...|
00000010  81 e5 3c db b1 a1 d5 12  17 9f 1b 61 4c ef 86 43  |..<......
..aL..C|
00000020  42 79                                             |By|
00000022
```

4. For decryption:

   openssl enc -aes-256-ctr -pass pass:***euid*** -pbkdf2 -d -in ciphertext.bin -out plaintext_dec.txt

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

Note: If the “-pass” option were not used, then the utility would request the password to be entered manually.

5. To confirm, type:

   cat plaintext_dec.txt

   Note that the original messages have been decrypted.

Q3:Attach a screenshot of the result.

```
pc0398@pc0398host:~$ cat plaintext.txt
This is my secret
pc0398@pc0398host:~$ openssl enc -aes-256-ctr -pass pass:pc0398 -pbkdf
2 -in plaintext.txt -out ciphertext.bin
pc0398@pc0398host:~$ hexdump -C ciphertext.bin
00000000  53 61 6c 74 65 64 5f 5f  fb 46 6a 65 60 d1 d9 9a  |Salted__.
Fje`...|
00000010  81 e5 3c db b1 a1 d5 12  17 9f 1b 61 4c ef 86 43  |..<......
..aL..C|
00000020  42 79                                             |By|
00000022
pc0398@pc0398host:~$ openssl enc -aes-256-ctr -pass pass:pc0398 -pbkdf
2 -d -in ciphertext.bin -out plaintext_dec.txt
pc0398@pc0398host:~$ cat plaintext_dec.txt
This is my secret
pc0398@pc0398host:~$
```

6. It is possible to encode the ciphertext using Base 64, in order to have it in the text format:

   openssl enc -aes-256-ctr -a -pass pass:***euid*** -pbkdf2 -in plaintext.txt -out ciphertext.txt

7. Type:

   cat ciphertext.txt

8. Verify that decryption works correctly:

   openssl enc -aes-256-ctr -d -a -pass pass:***euid*** -pbkdf2 -in ciphertext.txt -out plaintext_dec2.txt

   Note: If the option “-out” is omitted, the standard output is used.

```
pc0398@pc0398host:~$ openssl enc -aes-256-ctr -d -a -pass pass:pc0398
-pbkdf2 -in ciphertext.txt
This is my secret
```

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

Q4: Attach a screenshot of the result.

```
pc0398@pc0398host:~$ openssl enc -aes-256-ctr -a -pass pass:pc0398 -pb
kdf2 -in plaintext.txt -out ciphertext.txt
pc0398@pc0398host:~$ cat ciphertext.txt
U2FsdGVkX1/FeriTJPUu3UxqE1O0Fy2AUb/QHGc4Co/xig==
pc0398@pc0398host:~$ openssl enc -aes-256-ctr -d -a -pass pass:pc0398
-pbkdf2 -in ciphertext.txt -out plaintext_dec2.txt
pc0398@pc0398host:~$ cat plaintext_dec2.txt
This is my secret
pc0398@pc0398host:~$
```

9. Run the encryption again with the same password, but write the output into *ciphertext2.txt*:

   openssl enc -aes-256-ctr -a -pass pass:***euid*** -pbkdf2 -in plaintext.txt -out ciphertext2.txt

10. Note that the decryption works correctly again:

    openssl enc -aes-256-ctr -d -a -pass pass:***euid*** -pbkdf2 -in ciphertext2.txt

11. Now, note that the ciphertexts are different. To verify, type:

    cat ciphertext.txt ciphertext2.txt

Q5: Explain why the ciphertexts in *ciphertext.txt* and *ciphertext2.txt* are different, even though the same password was used for encryption.

```
pc0398@pc0398host:~$ openssl enc -aes-256-ctr -a -pass pass:pc0398 -pb
kdf2 -in plaintext.txt -out ciphertext2.txt
pc0398@pc0398host:~$ openssl enc -aes-256-ctr -d -a -pass pass:pc0398
-pbkdf2 -in ciphertext2.txt
This is my secret
pc0398@pc0398host:~$ cat ciphertext.txt ciphertext2.txt
U2FsdGVkX1/FeriTJPUu3UxqE1O0Fy2AUb/QHGc4Co/xig==
U2FsdGVkX1/nY+wMFRGcYjDBvx19OGgMJAYVr5K188bc0g==
pc0398@pc0398host:~$
```

When using openSSL Initialization vector (this is a randomly generate variable) is used to encrypt a file. Even if the same file is encrypted twice, the initialization vector will be unique/different every time.

Hint: Run decryption of both files again, not adding the “-p” option.

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

Section 2: Public Key Encryption and Digital Signatures Using OpenSSL

Let us focus on the RSA algorithm in this section.

1. First, let us generate an RSA private key (effectively, we will generate the public/private key pair). Type:

   openssl genrsa -out ***euid***.key 3072

   As usual, replace “***euid***” with your actual EUID.

   This command generates the RSA private key and outputs it to the file ***euid****.key*.
   The key is stored in the PEM format. Display it:

   cat ***euid***.key

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

```
pc0398@pc0398host:~$ openssl genrsa -out pc0398.key 3072
Generating RSA private key, 3072 bit long modulus (2 primes)
.........++++
.....++++
e is 65537 (0x010001)
pc0398@pc0398host:~$ cat pc0398.key
-----BEGIN RSA PRIVATE KEY-----
```


```
MIIG4gIBAAKCAYEAwj36AcuWyVCVP1s8sDdESrNYQPY/5XpGCVSOkvBXoYOUD+5z
UDSCNJxIqnecYEYFIa+lMTdA3evsNAGWoKHh8smRpMR/3WoBMi/tkhfoBVZc5KPX
hlg4uEr8NKr8OVTC12CajKMv/BXlrcXbsDSW5G4HoeX/rj6o87vIYwgSo3arZ/aG
P9a8H/cDb59oyl1BJIAuB2gzkmbLXCGI95DjdIKM/C0zyJHbXcZb5NPOPKyjLPGU
3spEAdZaJZYEGKr/2NMKudO7NMeiaiseFKUqFzy379jy/2LiYhy7x8p8gjS7v3/z
YdlGmBZ787SnDblyd/1QxaNeo+sbHxBzFqXXBfRGNggSHdmppR1OjT1lFd3J6gZv
K2uY41J9hapWJ8+CjPrekLHXdva3aznuB68vssS2hZikTSSst7idpbE1owA1+lvL
yzVkvemd1/UD7a/NsDHMc4x8jatWM7aKVzeljmKRLMJJB4OdYJ5mmZJS+KEBHgXb
6aBIjUuQA4mYijUzAgMBAAECggGATlo5suew11wOGBRxkpN4lLlrbCjo9WQxRCo7
qMISVzCAVTzWfw/vRdGU7k55Ycxc3Y3AkO282MgF66SIYhtqzUjJ2zD4cxQDj38N
0dSa3yxLJF+C3MpRduWofoZKX8X1InPxY7Gf9YvPrh0V1Sohg9tucLwaOutDs62X
03q6Kf4AfltPJAk2sZ4uYKRo3NhpIpQh/RLnWrLkbTGJ0nM7jQ/sVvzU2p8bYeRV
xSQ7GPZ6i/KXHpdwf10719V/qYOwJCHf7OVxZkKm9nTbr7IU6QyluGguwrEFfxj5
GYKjuYGw377fql/gikKkJlLYzK8aiulFpPkCCZDiepcJ60FhbrPbuzVZgh5pU9c4
lMoXf8PGeQjI76zA2U8nbFglNzy+jwuXanegzsWbLCBh3pxiL2CULxl0d8omtE9V
dlHrOPOK3Dd2uzgEDqvdKz5MjKMnXqFfDHLyiT/3CpLc9qTuBsnoUsVX/+Ab7ahq
6NcuF68ExMquEw8bndOSncD9JRsBAoHBAO/IvTqnuij0VaTtvHZXoH2BL+ThEoY+
```

```
wtfpFuEWJ9H33o42DnTwMzvxvVRfCkTNrvzKg1E/eyI3IQ44sGLZ2EgR4afSPcZb
mmtiewaenkmK86VwfvDoclN3FL6qfuXGhh+VpnuPdFp8ZHZOaI344dCHY9gsog5Q
6QO542Dl+yJMFKScTZT0H2tbI4XVUcGXaG9ZvswuWv2HQmVQ4snWvjXmFyN6yEOp
yqbvDTRAhv8oj92lsCsN7I8xChAodXfyYQKBwQDPYMAsHFU4aLBszya0cTTlNktz
dP/IM5OgrtHZOiYC4nja1Z4HN0rOzqboPldWYd/Q1+y7GD+r0Yf/Ao7QWfPRURzd
ryQGYcnKXHathefmDPnbixy7Dr8oeSGgOtG5jXFLBD3Ln9tmKtIlV9gRfjVKD7xt
YII2yJ+UBNShm+sHg3sWH19aVFXgBIs3ygQbcS0b4i+HghZKC4Q6wmQFv66QElnr
FquESRxh3oXj4q0nAyXzzUKA+LXPZGL+diReOBMCgcBnWkwiFBmRfX9Mz5EQMUTj
08MpDDpH0tV46ucJrdsMDaLH+VTR4payx8erjiuG7ID4RPHFG2+fl9OPpa+ha/Rl
XA+/nkcyMFmm35PXjpb7bAEI1ixg+lAM2c7/P83ijGdTRdW0fGjSd7t1ouY+C4CJ
uVG348UWFpnZGU6AfaB5cAGV0ik+gtroEyiEjyXmI7TICg7LQ0/E54Ae23ZlOUJT
SRHAITQgtha7TV6LnySQ5pLcHoaIiD/mmGSWAlpfkQECgcARLP49qQsV5PXXzUBz
aXzsDxcotOQKfNsZT+FpkHtdecxws4inKFk1z7/bNG6fZPPYpEmjJVsArBYLdPbH
IuwixgkSA1OEg8zmycm6jYCreR+oo1j51TrX61BxqO195M0hXXSQzOirCZjmJuRq
bn2TaMI+fOPMDcnJxKynxfYtZSJorqiDHsEOENJHCKBWdYU67fzpyEh4UYPeKgOG
xl166UoxQmtqRXPoVznXqgqVVJlsG/AKJo9IYhQg50AT3akCgcA8cWXre1UKwmbW
aX8rNMNMcPN7DfSGuWTl/wYTsJ5PCt00vwbDNF9ipCMlC5rDOhB0t4C8o27vl7Ju
vvyaoOGrTXE5x9iyeDNckWe1hERj93Jh1I9chtZHEU4TD9TAB02zCaVQAzEBorvP
Z7+5y+ZcHXbXWgjpXKme8af2Mh9TTBuNjMCw8/Nd60GwmDQuZ1USyVOP8NzztamU
/noCu61/tuSTe+LFypSzWckE6j19JRUxVhwZQrFm25kcw7jGCGU=
-----END RSA PRIVATE KEY-----
```


2. Next, we extract and display the public key:

   openssl rsa -in ***euid***.key -pubout -out ***euid***_pk.key

   cat ***euid***_pk.key

Q6: Attach a screenshot of the result.

```
pc0398@pc0398host:~$ openssl rsa -in pc0398.key -pubout -out pc0398_pk
.key
writing RSA key
pc0398@pc0398host:~$ cat pc0398_pk.key
-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAwj36AcuWyVCVP1s8sDdE
SrNYQPY/5XpGCVSOkvBXoYOUD+5zUDSCNJxIqnecYEYFIa+lMTdA3evsNAGWoKHh
8smRpMR/3WoBMi/tkhfoBVZc5KPXhlg4uEr8NKr8OVTC12CajKMv/BXlrcXbsDSW
5G4HoeX/rj6o87vIYwgSo3arZ/aGP9a8H/cDb59oyl1BJIAuB2gzkmbLXCGI95Dj
dIKM/C0zyJHbXcZb5NPOPKyjLPGU3spEAdZaJZYEGKr/2NMKudO7NMeiaiseFKUq
Fzy379jy/2LiYhy7x8p8gjS7v3/zYdlGmBZ787SnDblyd/1QxaNeo+sbHxBzFqXX
BfRGNggSHdmppR1OjT1lFd3J6gZvK2uY41J9hapWJ8+CjPrekLHXdva3aznuB68v
ssS2hZikTSSst7idpbE1owA1+lvLyzVkvemd1/UD7a/NsDHMc4x8jatWM7aKVzel
jmKRLMJJB4OdYJ5mmZJS+KEBHgXb6aBIjUuQA4mYijUzAgMBAAE=
-----END PUBLIC KEY-----
pc0398@pc0398host:~$
```

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

3. For encrypting the *plaintext.txt*, type:

   openssl rsautl -encrypt -pubin -inkey ***euid***_pk.key -in plaintext.txt -out rsa_ciphertext.bin

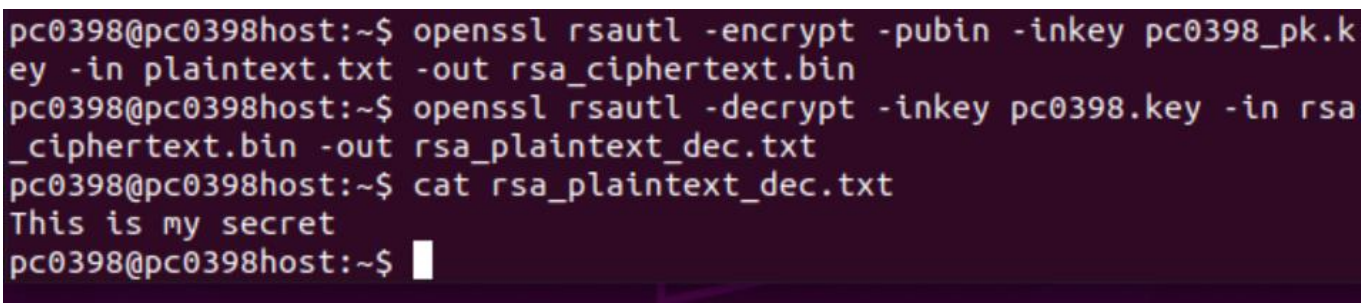
4. For decryption, type:

   openssl rsautl -decrypt -inkey ***euid***.key -in rsa_ciphertext.bin -out rsa_plaintext_dec.txt

   cat rsa_plaintext_dec.txt

Q7: Attach a screenshot of the result.

Note: The above method is suitable for encryption of short messages (up to about 1 kilobyte), for longer messages, hybrid encryption (KEM/DEM) should be used.



5. To digitally sign the file *plaintext.txt*, type:

   openssl dgst -sign ***euid***.key -out sig.bin plaintext.txt

   Note: As of the current version 1.1.1, OpenSSL signs messages directly when using the *rsautl* or *pkeyutl* commands. For this reason, it is simpler to deploy the *dgst* command, in order to hash and sign the message with one command.

   Display the signature:

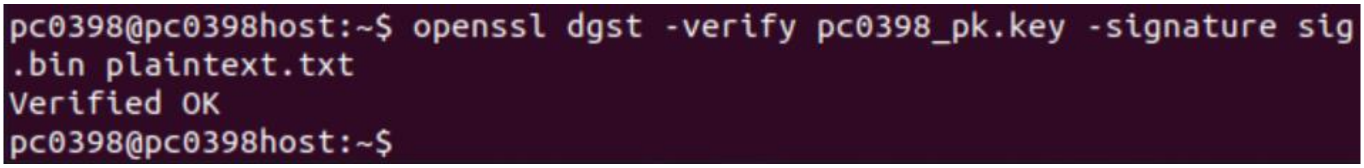
   hexdump sig.bin

```
pc0398@pc0398host:~$ openssl dgst -sign pc0398.key -out sig.bin plaint
ext.txt
pc0398@pc0398host:~$ hexdump sig.bin
0000000 f308 ad24 7ea1 9501 25f8 6a81 0848 9dac
0000010 8073 bb86 57b2 3a55 3ba1 583b 2b57 e489
0000020 ec89 16d4 589f dff6 b74e cbbf 46d1 c8fa
0000030 a737 97cb 2282 0477 4684 c909 536a e35d
0000040 675c e06b 1a50 499d e696 0992 dbfd eb88
0000050 6d55 032d 20df 4e4c 9bb1 d7dc 7610 5aa0
0000060 7627 a72b e279 3444 24a7 6558 2842 bf21
0000070 e678 ddc9 71ca 00ec 41cf 48e1 e791 deae
0000080 514e 2f2a f8cb e891 ba19 0728 077e de0a
0000090 c187 95a7 42c0 d2a6 4f07 8f74 05aa aab1
00000a0 832d 88d5 2854 7d38 9324 1cd3 ad1f c235
00000b0 ad83 d81e dc58 6e46 5843 07ff 9615 6266
00000c0 2eea e993 c6f8 9c24 101a 00b1 05ec 434d
00000d0 b61f 19cb 6c75 7d53 d4f9 f9d6 5de5 2b3e
00000e0 2aa4 925a 7e20 1389 037c 2f78 dea9 4a6c
00000f0 f66c ed7e 98bb d796 b632 855b f0d2 a0b6
0000100 f2ba e49f 639a a271 618d c6e8 f0b1 c04f
0000110 d96f feaa 95b5 4033 5e4c 9370 65ea 7c4e
0000120 ad8f a2ee 83e4 3dfc fbd8 95ca 95b0 b51a
0000130 679b 974e e197 8e94 d8a6 bdf9 1e86 3765
0000140 102d 450b a551 1b72 dad4 728a 9f6c ddce
0000150 9185 3491 0e92 7e86 4cc2 30da ed17 f654
0000160 04aa 6201 a3bb 24f3 ab15 9b4c f81c daad
0000170 2330 84dc a71f aeb1 b667 7bba 1d18 1f43
0000180
pc0398@pc0398host:~$
```

6. To verify the signature:

openssl dgst -verify ***euid***_pk.key -signature sig.bin plaintext.txt

Q8: Attach a screenshot of the result



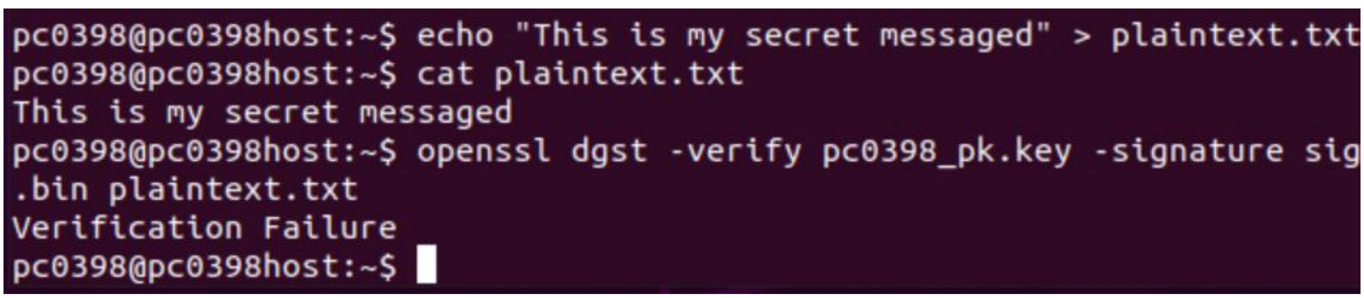
7. Any changes in the message will invalidate the signature.
   Let us replace the last letter in our message:

   echo "This is my secret messagd" > plaintext.txt

   cat plaintext.txt

8. Now, verification will fail:

   openssl dgst -verify ***euid***_pk.key -signature sig.bin plaintext.txt



Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

Section 3: Public Key Certificates Using OpenSSL

Let us now study the handling of X.509 public-key certificates using OpenSSL.
Suppose that we would like to create a certificate signing request (CSR) to the Certificate Authority for the RSA key that we generated earlier. The following command can be used (<u>do not type it yet</u>):

openssl req -key ***euid***.key -new -out ***euid***_domain.csr

Then, the utility will request some additional information, which is called a Distinguished Name (DN). An important field in the DN is the Common Name (CN) —it should be the exact domain name of the host for which the certificate will be used. Below is an example of the prompt:

Country Name (2 letter code): The two-letter country code where your company/organization is legally located. Example: US
State or Province Name (full name): Example: Texas
Locality Name (e.g., city): Example: Denton
Organization Name (e.g., company): University of North Texas
Organizational Unit Name (e.g., section): Department of Computer Science and Engineering (this field is optional)
Common Name (e.g. server FQDN): Fully Qualified Domain Name; Example: www.unt.edu
Email Address: Example: webmaster@unt.edu (this field is optional)

It is possible to enter all of the above information from the command line as described below.

1. Type:

   openssl req -key ***euid***.key -new -out ***euid***_domain.csr \

   -subj "/C=US/ST=Texas/L=Denton/O=UNT/OU=CSE/CN=www.***euid***.edu"

2. Let us verify the result:

   openssl req -text -in ***euid***_domain.csr -noout -verify

Q9: Attach a screenshot of the result.

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

```
pc0398@pc0398host:~$ openssl req -key pc0398.key -new -out pc0398_domain.csr \
> -subj "/C=US/ST=Texas/L=Denton/O=UNT/OU=CSE/CN=www.pc0398.edu"
pc0398@pc0398host:~$ openssl req -text -in pc0398_domain.csr -noout -verify
verify OK
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: C = US, ST = Texas, L = Denton, O = UNT, OU = CSE, CN = www.pc
0398.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (3072 bit)
                Modulus:
                    00:c2:3d:fa:01:cb:96:c9:50:95:3f:5b:3c:b0:37:
                    44:4a:b3:58:40:f6:3f:e5:7a:46:09:54:8e:92:f0:
                    57:a1:83:94:0f:ee:73:50:34:82:34:9c:48:aa:77:
                    9c:60:46:05:21:af:a5:31:37:40:dd:eb:ec:34:01:
                    96:a0:a1:e1:f2:c9:91:a4:c4:7f:dd:6a:01:32:2f:
                    ed:92:17:e8:05:56:5c:e4:a3:d7:86:58:38:b8:4a:
                    fc:34:aa:fc:39:54:c2:d7:60:9a:8c:a3:2f:fc:15:
                    e5:ad:c5:db:b0:34:96:e4:6e:07:a1:e5:ff:ae:3e:
                    a8:f3:bb:c8:63:08:12:a3:76:ab:67:f6:86:3f:d6:
                    bc:1f:f7:03:6f:9f:68:ca:5d:41:24:80:2e:07:68:
                    33:92:66:cb:5c:21:88:f7:90:e3:74:82:8c:fc:2d:
                    33:c8:91:db:5d:c6:5b:e4:d3:ce:3c:ac:a3:2c:f1:
                    94:de:ca:44:01:d6:5a:25:96:04:18:aa:ff:d8:d3:
                    0a:b9:d3:bb:34:c7:a2:6a:2b:1e:14:a5:2a:17:3c:
                    b7:ef:d8:f2:ff:62:e2:62:1c:bb:c7:ca:7c:82:34:
                    bb:bf:7f:f3:61:d9:46:98:16:7b:f3:b4:a7:0d:b9:
                    72:77:fd:50:c5:a3:5e:a3:eb:1b:1f:10:73:16:a5:
                    d7:05:f4:46:36:08:12:1d:d9:a9:a5:1d:4e:8d:3d:
                    65:15:dd:c9:ea:06:6f:2b:6b:98:e3:52:7d:85:aa:
                    56:27:cf:82:8c:fa:de:90:b1:d7:76:f6:b7:6b:39:
                    ee:07:af:2f:b2:c4:b6:85:98:a4:4d:24:ac:b7:b8:
                    9d:a5:b1:35:a3:00:35:fa:5b:cb:cb:35:64:bd:e9:
                    9d:d7:f5:03:ed:af:cd:b0:31:cc:73:8c:7c:8d:ab:
                    56:33:b6:8a:57:37:a5:8e:62:91:2c:c2:49:07:83:
                    9d:60:9e:66:99:92:52:f8:a1:01:1e:05:db:e9:a0:
                    48:8d:4b:90:03:89:98:8a:35:33
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
```

```
    Signature Algorithm: sha256WithRSAEncryption
         b4:c5:48:85:3e:3a:d6:e2:76:79:3b:72:6c:54:6e:88:71:77:
         dc:2f:75:f5:95:d5:58:a0:df:ad:9b:65:9c:76:25:2a:73:c6:
         ed:96:b0:42:d7:38:fd:5f:15:b9:41:54:f9:1f:3a:58:6f:c9:
         d9:4b:c0:6a:51:4c:73:2f:53:93:78:0a:b6:f9:fd:31:18:b8:
         7c:2b:12:07:d8:a4:b0:82:c2:58:29:41:f9:9f:04:d5:0b:d0:
         0d:66:a7:82:7b:16:da:18:dd:08:c2:39:21:b0:da:1b:1e:29:
         c9:8f:57:f4:b6:c6:5d:5c:e6:55:98:d7:df:83:4a:6c:36:86:
         84:59:e4:1f:2f:bf:e3:e1:97:8d:25:2b:f6:41:0d:1a:84:28:
         ac:23:d9:7b:8c:42:43:cd:a1:03:9a:f4:ea:84:ee:2b:52:d2:
         03:f8:e3:0c:66:70:99:15:ed:ab:74:b2:94:7b:cc:17:f5:d0:
         ae:6d:66:13:ee:c6:46:4e:4d:a4:3d:e2:e7:73:7d:e9:28:b3:
         37:cb:9c:13:69:84:40:70:41:7c:b5:8d:23:b0:ab:0b:24:31:
         4f:24:2b:57:22:68:7e:46:54:78:a6:78:99:e3:74:7c:75:b2:
         a3:f9:89:af:5b:99:d5:3b:9c:6f:11:d7:79:eb:25:ba:a8:b7:
         84:75:28:96:31:e4:a6:14:a4:fa:c3:8a:e9:ed:62:40:97:e0:
         b1:cc:0a:d9:f2:9e:5b:23:15:ce:d8:a4:ff:7c:a4:81:11:df:
         af:90:a1:19:a0:84:e6:78:8e:ec:21:e5:e6:d0:d5:41:71:f2:
         5f:8b:27:b4:21:0b:23:d5:a3:cf:d5:6f:f6:3f:8d:cc:92:7d:
         2e:51:ff:c7:bd:7c:94:60:c9:8e:95:e5:2e:d3:27:ca:9a:a1:
         6e:97:92:9d:71:dc:8e:21:ab:43:88:ec:02:34:37:44:49:11:
         ef:31:77:e7:f5:c8:77:77:47:61:13:03:ba:a1:4f:5e:0d:0f:
         c6:ef:80:0c:62:d9
pc0398@pc0398host:~$
```

Note: The CSR file "***euid***_domain.csr" will need to be sent to CA that will check the user information. If the check is successful, CA will issue the certificate file. We will omit this step in this lab. Instead, we will obtain and verify the certificate of the Google webserver. For that, we will use the *s_client* program (of the OpenSSL suite) which implements a generic SSL/TLS client.

3. Type:

   openssl s_client -connect google.com:443 </dev/null

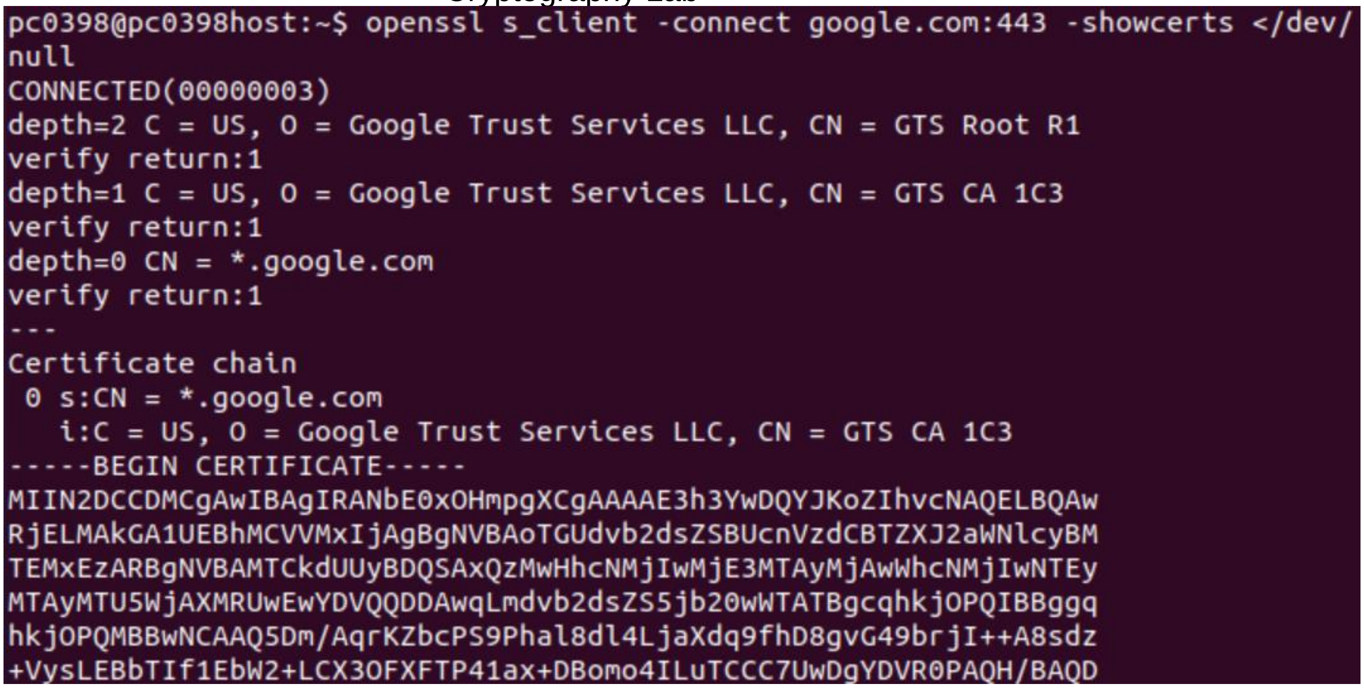
```
pc0398@pc0398host:~$ openssl s_client -connect google.com:443</dev/null
CONNECTED(00000003)
depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify return:1
depth=1 C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
verify return:1
depth=0 CN = *.google.com
verify return:1
---
Certificate chain
 0 s:CN = *.google.com
   i:C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
 1 s:C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
   i:C = US, O = Google Trust Services LLC, CN = GTS Root R1
 2 s:C = US, O = Google Trust Services LLC, CN = GTS Root R1
   i:C = BE, O = GlobalSign nv-sa, OU = Root CA, CN = GlobalSign Root CA
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIN2DCCDMCgAwIBAgIRANbE0xOHmpgXCgAAAAE3h3YwDQYJKoZIhvcNAQELBQAw
RjELMAkGA1UEBhMCVVMxIjAgBgNVBAoTGUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBM
```

   Note: The redirection from the null device immediately closes the *s_client* program, as in general it expects commands to establish the TLS connection.

4. In order to display the whole certificate chain, type:
   openssl s_client -connect google.com:443 -showcerts </dev/null

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab



5. Since the output of the previous command takes several screens to be display, making a picture of the last screen may not be very informative. The “more” command will be helpful in this case. Type:

   openssl s_client -connect google.com:443 -showcerts </dev/null | more

   Note: Scrolling is done by pressing “Space” to advance the whole screen down, or “Enter” to advance one line.

Q10: Attach <u>two</u> screenshots: The first and the last screen displayed as a result of the above command.

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

```
pc0398@pc0398host:~$ openssl s_client -connect google.com:443 -showcerts </dev/
null
CONNECTED(00000003)
depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify return:1
depth=1 C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
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 LLC, CN = GTS CA 1C3
-----BEGIN CERTIFICATE-----
MIIN2DCCDMCgAwIBAgIRANbE0xOHmpgXCgAAAAE3h3YwDQYJKoZIhvcNAQELBQAw
RjELMAkGA1UEBhMCVVMxIjAgBgNVBAoTGUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBM
TEMxEzARBgNVBAMTCkdUUyBDQSAxQzMwHhcNMjIwMjE3MTAyMjAwWhcNMjIwNTEy
MTAyMTU5WjAXMRUwEwYDVQQDDAwqLmdvb2dsZS5jb20wWTATBgcqhkjOPQIBBggq
hkjOPQMBBwNCAAQ5Dm/AqrKZbcPS9Phal8dl4LjaXdq9fhD8gvG49brjI++A8sdz
+VysLEBbTIf1EbW2+LCX3OFXFTP41ax+DBomo4ILuTCCC7UwDgYDVR0PAQH/BAQD
AgeAMBMGA1UdJQQMMAoGCCsGAQUFBwMBMAwGA1UdEwEB/wQCMAAwHQYDVR0OBBYE
FG/JTG67OE+UjcxhKQmr0BNFJtTPMB8GA1UdIwQYMBaAFIp0f6+Fze6VzT2c0OJG
FPNxNR0nMGoGCCsGAQUFBwEBBF4wXDAnBggrBgEFBQcwAYYbaHR0cDovL29jc3Au
cGtpLmdvb2cvZ3RzMWMzMDEGCCsGAQUFBzAChiVodHRwOi8vcGtpLmdvb2cvcmVw
by9jZXJ0cy9ndHMxYzMuZGVyMIIJaAYDVR0RBIIJXzCCCVuCDCouZ29vZ2xlLmNv
bYIWKi5hcHBlbmdpbmUuZ29vZ2xlLmNvbYIJKi5iZG4uZGV2ghIqLmNsb3VkLmdv
b2dsZS5jb22CGCouY3Jvd2Rzb3VyY2UuZ29vZ2xlLmNvbYIYKi5kYXRhY29tcHV0
ZS5nb29nbGUuY29tggsqLmdvb2dsZS5jYYILKi5nb29nbGUuY2yCDiouZ29vZ2xl
LmNvLmlugg4qLmdvb2dsZS5jby5qcIIOKi5nb29nbGUuY28udWuCDyouZ29vZ2xl
LmNvbS5hcoIPKi5nb29nbGUuY29tLmF1gg8qLmdvb2dsZS5jb20uYnKCDyouZ29v
Z2xlLmNvbS5jb4IPKi5nb29nbGUuY29tLm14gg8qLmdvb2dsZS5jb20udHKCDyou
Z29vZ2xlLmNvbS52boILKi5nb29nbGUuZGWCCyouZ29vZ2xlLmVzggsqLmdvb2ds
ZS5mcoILKi5nb29nbGUuaHWCCyouZ29vZ2xlLml0ggsqLmdvb2dsZS5ubIILKi5n
b29nbGUucGyCCyouZ29vZ2xlLnB0ghIqLmdvb2dsZWFkYXBpcy5jb22CDyouZ29v
Z2xlYXBpcy5jboIRKi5nb29nbGV2aWRlby5jb22CDCouZ3N0YXRpYy5jboIQKi5n
c3RhdGljLWNuLmNvbYIPZ29vZ2xlY25hcHBzLmNughEqLmdvb2dsZWNuYXBwcy5j
--More--DONE

...skipping 1 line
YXBwcy5jboIOKi5na2VjbmFwcHMuY26CEmdvb2dsZWRvd25sb2Fkcy5jboIUKi5n
b29nbGVkb3dubG9hZHMuY26CEHJlY2FwdGNoYS5uZXQuY26CEioucmVjYXB0Y2hh
Lm5ldC5jboIQcmVjYXB0Y2hhLWNuLm5ldIISKi5yZWNhcHRjaGEtY24ubmV0ggt3
aWRldmluZS5jboINKi53aWRldmluZS5jboIRYW1wcHJvamVjdC5vcmcuY26CEyou
YW1wcHJvamVjdC5vcmcuY26CEWFtcHByb2plY3QubmV0LmNughMqLmFtcHByb2pl
Y3QubmV0LmNughdnb29nbGUtYW5hbHl0aWNzLWNuLmNvbYIZKi5nb29nbGUtYW5h
bHl0aWNzLWNuLmNvbYIXZ29vZ2xlYWRzZXJ2aWNlcy1jbi5jb22CGSouZ29vZ2xl
YWRzZXJ2aWNlcy1jbi5jb22CEWdvb2dsZXZhZHMtY24uY29tghMqLmdvb2dsZXZh
ZHMtY24uY29tghFnb29nbGVhcGlzLWNuLmNvbYITKi5nb29nbGVhcGlzLWNuLmNv
bYIVZ29vZ2xlb3B0aW1pemUtY24uY29tghcqLmdvb2dsZW9wdGltaXplLWNuLmNv
bYISZG91YmxlY2xpY2stY24ubmV0ghQqLmRvdWJsZWNsaWNrLWNuLm5ldIIYKi5m
bHMuZG91YmxlY2xpY2stY24ubmV0ghYqLmcuZG91YmxlY2xpY2stY24ubmV0gg5k
```

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

b3VibGVjbGljay5jboIQKi5kb3VibGVjbGljay5jboIUKi5mbHMuZG91YmxlY2xp
Y2suY26CEiouZy5kb3VibGVjbGljay5jboIRZGFydHNlYXJjaC1jbi5uZXSCEyou
ZGFydHNlYXJjaC1jbi5uZXSCHWdvb2dsZXRyYXZlbGFkc2VydmljZXMtY24uY29t
gh8qLmdvb2dsZXRyYXZlbGFkc2VydmljZXMtY24uY29tghhnb29nbGV0YWdzZXJ2
aWNlcy1jbi5jb22CGiouZ29vZ2xldGFnc2VydmljZXMtY24uY29tghdnb29nbGV0
YWdtYW5hZ2VyLWNuLmNvbYIZKi5nb29nbGV0YWdtYW5hZ2VyLWNuLmNvbYIYZ29v
Z2xlc3luZGljYXRpb24tY24uY29tghoqLmdvb2dsZXN5bmRpY2F0aW9uLWNuLmNv
bYIkKi5zYWZlZnJhbWUuZ29vZ2xlc3luZGljYXRpb24tY24uY29tghZhcHAtbWVh
c3VyZW1lbnQtY24uY29tghgqLmFwcC1tZWFzdXJlbWVudC1jbi5jb22CC2d2dDEt
Y24uY29tgg0qLmd2dDEtY24uY29tggtndnQyLWNuLmNvbYINKi5ndnQyLWNuLmNv
bYILMm1kbi1jbi5uZXSCDSouMm1kbi1jbi5uZXSCFGdvb2dsZWZsaWdodHMtY24u
bmV0ghYqLmdvb2dsZWZsaWdodHMtY24ubmV0ggxhZG1vYi1jbi5jb22CDiouYWRt
b2ItY24uY29tgg0qLmdzdGF0aWMuY29tghQqLm1ldHJpYy5nc3RhdGljLmNvbYIK
Ki5ndnQxLmNvbYIRKi5nY3BjZG4uZ3Z0MS5jb22CCiouZ3Z0Mi5jb22CDiouZ2Nw
Lmd2dDIuY29tghAqLnVybC5nb29nbGUuY29tghYqLnlvdXR1YmUtbm9jb29raWUu
Y29tggsqLnl0aW1nLmNvbYILYW5kcm9pZC5jb22CDSouYW5kcm9pZC5jb22CEyou
Zmxhc2guYW5kcm9pZC5jb22CBGcuY26CBiouZy5jboIEZy5jb4IGKi5nLmNvggZn
b28uZ2yCCnd3dy5nb28uZ2yCFGdvb2dsZS1hbmFseXRpY3MuY29tghYqLmdvb2ds
ZS1hbmFseXRpY3MuY29tggpnb29nbGUuY29tghJnb29nbGVjb21tZXJjZS5jb22C
FCouZ29vZ2xlY29tbWVyY2UuY29tgghnZ3BodC5jboIKKi5nZ3BodC5jboIKdXJj
aGluLmNvbYIMKi51cmNoaW4uY29tggh5b3V0dS5iZYILeW91dHViZS5jb22CDSou
eW91dHViZS5jb22CFHlvdXR1YmVlZHVjYXRpb24uY29tghYqLnlvdXR1YmVlZHVj
YXRpb24uY29tgg95b3V0dWJla2lkcy5jb22CESoueW91dHViZWtpZHMuY29tggV5
dC5iZYIHKi55dC5iZYIaYW5kcm9pZC5jbGllbnRzLmdvb2dsZS5jb22CG2RldmVs
b3Blci5hbmRyb2lkLmdvb2dsZS5jboIcZGV2ZWxvcGVycy5hbmRyb2lkLmdvb2ds
ZS5jboIYc291cmNlLmFuZHJvaWQuZ29vZ2xlLmNuMCEGA1UdIAQaMBgwCAYGZ4EM

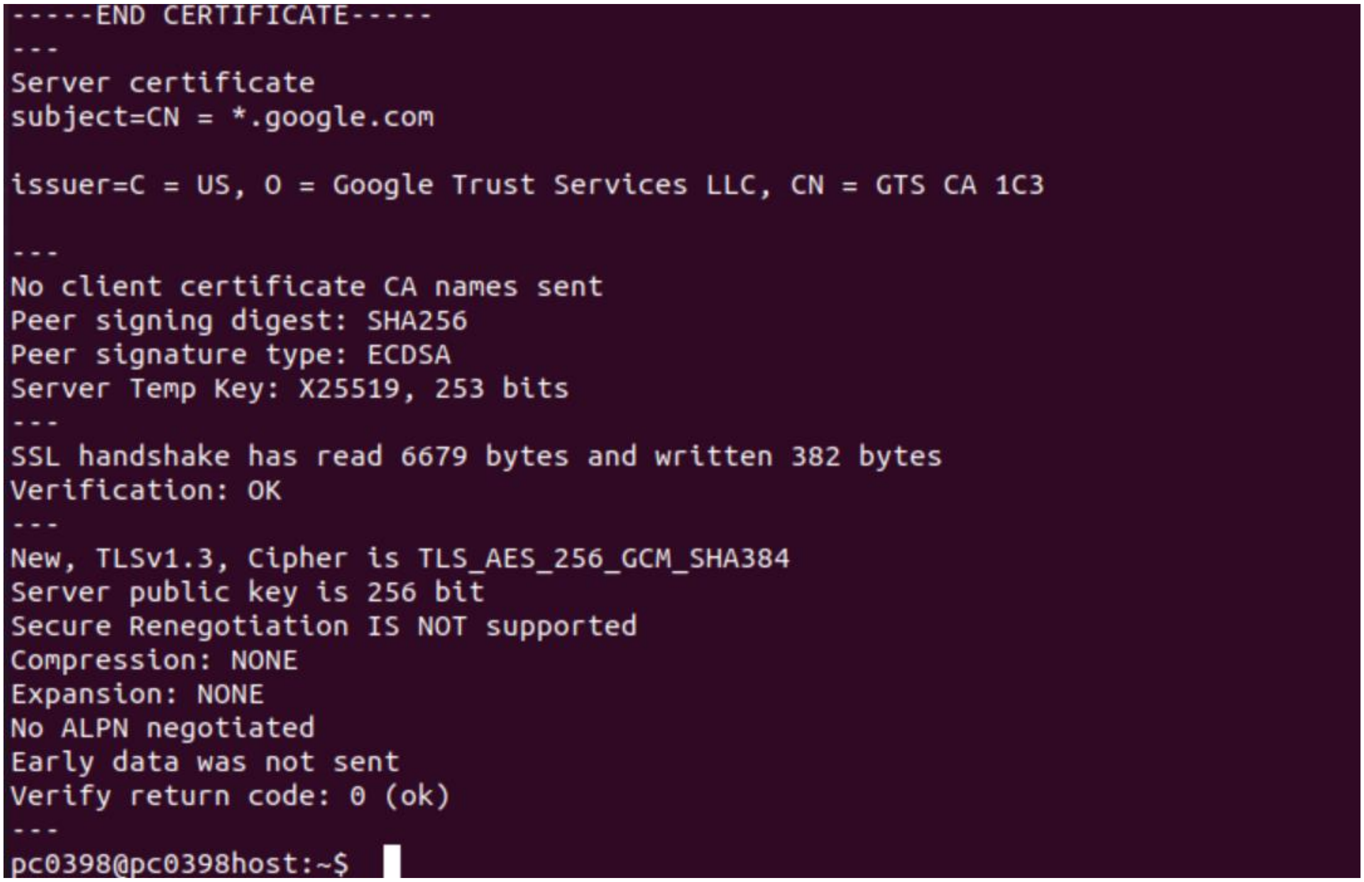
Cryptography Lab

AQIBMAwGCisGAQQB1nkCBQMwPAYDVR0fBDUwMzAxoC+gLYYraHR0cDovL2NybHMu
cGtpLmdvb2cvZ3RzMWMzL2ZWSnhiVi1LdG1rLmNybDCCAQUGCisGAQQB1nkCBAIE
gfYEgfMA8QB3ACl5vvCeOTkh8FZzn2Old+W+V32cYAr4+U1dJlwlXceEAAABfwdq
9rIAAAQDAEgwRgIhAIE0L7A9sMArrCjWhEqwVijFZbhUw06y6Diatb9rVKRaAiEA
sbuGyS4hbkvjqU7+4OsjOByFAZuLbWIKg+yaxXvfAbsAdgDfpV6raIJPH2yt7rhf
Tj5a6s2iEqRqXo47EsAgRFwqcwAAAX8Hava9AAAEAwBHMEUCIGN/uAfSygpZ8EWC
FrMFiADW7MbIcBapR9onXFGXeDf1AiEAzSnEMEP+kqAT9DCbCtVdHw38XOrSLAmi
+z6uqIP5oH8wDQYJKoZIhvcNAQELBQADggEBAGF6HW6K1/I7iYW3/2gE7zNJcbwe
zdtpqZoC6N77Ot+Jn2BS79kyGtgFjkAyTX4jPmpMZvUZX6RlXY93Xmji/0lS6cbF
NzZ1GDwQzzH25yELNzrUKwW3fUPt4xyS6BUinI3KC9F2ELPwccIjTdgMgNrYMHV3
Tn6f4P5lR4aFuwFYcz2d+P9/2cYNVD42Yy/3L6XxA1vD4edvdFDZoOay3Q6p0XOx
kKwiSywCkh7o9PtJVE5xCyeX5EvQJinodDdllgCJFwQ0qTEoBmrdiEhiQkheq5rl
oIupGoAjtQcI79DiGzygdH07nLandHQoNr9UL44XLzd5NeTncV+aam7k5Hk=
-----END CERTIFICATE-----
 1 s:C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
   i:C = US, O = Google Trust Services LLC, CN = GTS Root R1
-----BEGIN CERTIFICATE-----
MIIFljCCA36gAwIBAgINAgO8U1lrNMcY9QFQZjANBgkqhkiG9w0BAQsFADBHMQsw
CQYDVQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZpY2VzIExMQzEU
MBIGA1UEAxMLR1RTIFJvb3QgUjEwHhcNMjAwODEzMDAwMDQyWhcNMjcwOTMwMDAw
MDQyWjBGMQswCQYDVQQGEwJVUzEiMCAGA1UEChMZR29vZ2xlIFRydXN0IFNlcnZp
Y2VzIExMQzETMBEGA1UEAxMKR1RTIENBIDFDMzCCASIwDQYJKoZIhvcNAQEBBQAD
ggEPADCCAQoCggEBAPWI3+dijB43+DdCkH9sh9D7ZYIl/ejLa6T/belaI+KZ9hzp
kgOZE3wJCor6QtZeViSqejOEH9Hpabu5dOxXTGZok3c3VVP+ORBNtzS7XyV3NzsX
lOo85Z3VvMO0Q+sup0fvsEQRY9i0QYXdQTBIkxu/t/bgRQIh4JZCF8/ZK2VWNAcm

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab


```
BA2o/X3KLu/qSHw3TT8An4Pf73WELnlXXPxXbhqW//yMmqaZviXZf5YsBvcRKgKA
gOtjGDxQSYflispfGStZloEAoPtR28p3CwvJlk/vcEnHXG0g/Zm0tOLKLnf9LdwL
tmsTDIwZKxeWmLnwi/agJ7u2441Rj72ux5uxiZ0CAwEAAaOCAYAwggF8MA4GA1Ud
DwEB/wQEAwIBhjAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIwEgYDVR0T
AQH/BAgwBgEB/wIBADAdBgNVHQ4EFgQUinR/r4XN7pXNPZzQ4kYU83E1HScwHwYD
VR0jBBgwFoAU5K8rJnEaK0gnhS9SZizv8IkTcT4waAYIKwYBBQUHAQEEXDBaMCYG
CCsGAQUFBzABhhpodHRwOi8vb2NzcC5wa2kuZ29vZy9ndHNyMTAwBggrBgEFBQcw
AoYkaHR0cDovL3BraS5nb29nL3JlcG8vY2VydHMvZ3RzcjEuZGVyMDQGA1UdHwQt
MCswKaAnoCWGI2h0dHA6Ly9jcmwucGtpLmdvb2cvZ3RzcjEvZ3RzcjEuY3JsMFcG
A1UdIARQME4wOAYKKwYBBAHWeQIFAzAqMCgGCCsGAQUFBwIBFhxodHRwczovL3Br
aS5nb29nL3JlcG9zaXRvcnkvMAgGBmeBDAECATAIBgZngQwBAgIwDQYJKoZIhvcN
AQELBQADggIBAIl9rCBcDDy+mqhXlRu0rvqrpXJxtDaV/d9AEQNMwkYUuxQkq/BQ
cSLbrcRuf8/xam/IgxvYzolfh2yHuKkMo5uhYpSTld9brmYZCwKWnvy15xBpPnrL
RklfRuFBsdeYTWU0AIAaP0+fbH9JAIFTQaSSIYKCGvGjRFsqUBITTcFTNvNCCK9U
+o53UxtkOCcXCb1YyRt8OS1b887U7ZfbFAO/CVMkH8IMBHmYJvJh8VNS/UKMG2Yr
PxWhu//2m+OBmgEGcYk1KCTd4b3rGS3hSMs9WYNRtHTGnXzGsYZbr8w0xNPM1IER
lQCh9BIiAfq0g3GvjLeMcySsN1PCAJA/Ef5c7TaUEDu9Ka7ixzpiO2xj2YC/WXGs
Yye5TBeg2vZzFb8q3o/zpWwygTMD0IZRcZk0upONXbVRWPeyk+gB9lm+cZv9TSjO
z23HFtz30dZGm6fKa+l3D/2gthsjgx0QGtkJAITgRNOidSOzNIb2ILCkXhAd4FJG
AJ2xDx8hcFH1mt0G/FX0Kw4zd8NLQsLxdxP8c4CU6x+7Nz/OAipmsHMdMqUybDKw
juDEI/9bfU1lcKwrmz3O2+BtjjKAvpafkmO8l7tdufThcV4q5O8DIrGKZTqPwJNl
1IXNDw9bg1kWRxYtnCQ6yICmJhSFm/Y3m6xv+cXDBlHz4n/FsRC6UfTd
-----END CERTIFICATE-----
 2 s:C = US, O = Google Trust Services LLC, CN = GTS Root R1
   i:C = BE, O = GlobalSign nv-sa, OU = Root CA, CN = GlobalSign Root CA
-----BEGIN CERTIFICATE-----
MIIFYjCCBEqgAwIBAgIQd70NbNs2+RrqIQ/E8FjTDTANBgkqhkiG9w0BAQsFADBX
MQswCQYDVQQGEwJCRTEZMBcGA1UEChMQR2xvYmFsU2lnbiBudi1zYTEQMA4GA1UE
CxMHUm9vdCBDQTEbMBkGA1UEAxMSR2xvYmFsU2lnbiBSb290IENBMB4XDTIwMDYx
OTAwMDA0MloXDTI4MDEyODAwMDA0MlowRzELMAkGA1UEBhMCVVMxIjAgBgNVBAoT
GUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBMTEMxFDASBgNVBAMTC0dUUyBSb290IFIx
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAthECix7joXebO9y/lD63
ladAPKH9gvl9MgaCcfb2jH/76Nu8ai6Xl6OMS/kr9rH5zoQdsfnFl97vufKj6bwS
iV6nqlKr+CMny6SxnGPb15l+8Ape62im9MZaRw1NEDPjTrETo8gYbEvs/AmQ351k
KSUjB6G00j0uYODP0gmHu81I8E3CwnqIiru6z1kZ1q+PsAewnjHxgsHA3y6mbWwZ
DrXYfiYaRQM9sHmklCitD38m5agI/pboPGiUU+6DOogrFZYJsuB6jC511pzrp1Zk
j5ZPaK49l8KEj8C8QMALXL32h7M1bKwYUH+E4EzNktMg6TO8UpmvMrUpsyUqtEj5
cuHKZPfmghCN6J3Cioj6OGaK/GP5Afl4/Xtcd/p2h/rs37EOeZVXtL0m79YB0esW
CruOC7XFxYpVq9Os6pFLKcwZpDIlTirxZUTQAs6qzkm06p98g7BAe+dDq6dso499
iYH6TKX/1Y7DzkvgtdizjkXPdsDtQCv9Uw+wp9U7DbGKogPeMa3Md+pvez7W35Ei
Eua++tgy/BBjFFFy3l3WFpO9KWgz7zpm7AeKJt8T11dleCfeXkkUAKIAf5qoIbap
sZWwpbkNFhHax2xIPEDgfg1azVY80ZcFuctL7TlLnMQ/0lUTbiSw1nH69MG6zO0b
9f6BQdgAmD06yK56mDcYBZUCAwEAAaOCATgwggE0MA4GA1UdDwEB/wQEAwIBhjAP
BgNVHRMBAf8EBTADAQH/MB0GA1UdDgQWBBTkrysmcRorSCeFL1JmLO/wiRNxPjAf
BgNVHSMEGDAWgBRge2YaRQ2XyolQL30EzTSo//z9SzBgBggrBgEFBQcBAQRUMFIw
JQYIKwYBBQUHMAGGGWh0dHA6Ly9vY3NwLnBraS5nb29nL2dzcjEwKQYIKwYBBQUH
MAKGHWh0dHA6Ly9wa2kuZ29vZy9nc3IxL2dzcjEuY3J0MDIGA1UdHwQrMCkwJ6Al
oCOGIWh0dHA6Ly9jcmwucGtpLmdvb2cvZ3NyMS9nc3IxLmNybDA7BgNVHSAENDAy
MAgGBmeBDAECATAIBgZngQwBAgIwDQYLKwYBBAHWeQIFAwIwDQYLKwYBBAHWeQIF
AwMwDQYJKoZIhvcNAQELBQADggEBADSkHrEoo9C0dhemMXoh6dFSPsjbdBZBiLg9
NR3t5P+T4Vxfq7vqfM/b5A3Ri1fyJm9bvhdGaJQ3b2t6yMAYN/olUazsaL+yyEn9
WprKASOshIArAoyZl+tJaox118fessmXn1hIVw41oeQa1v1vg4Fv74zPl6/AhSrw
```



```
-----END CERTIFICATE-----
---
Server certificate
subject=CN = *.google.com

issuer=C = US, O = Google Trust Services LLC, CN = GTS CA 1C3

---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 6679 bytes and written 382 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 256 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
---
pc0398@pc0398host:~$
```

Section 4: SSH Authentication Using Public Keys

OpenSSH can use public key cryptography for authentication. We will use a freshly generated RSA key pair. Note that if you already have a generated key, you may use the *ssh-keygen* command with “-i” option and then specify the key file name.

Important note: In this lab, we will use the earlier generated key pair, only to demonstrate the conversion of key formats.

1. Type:

   ssh-keygen -t rsa

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

Note: You will see the following prompts—you may just press “Enter” for all of them—and the following messages will be displayed:

Enter file in which to save the key (/home/sec-lab/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/sec-lab/.ssh/id_rsa

Your public key has been saved in /home/sec-lab/.ssh/id_rsa.pub

The key fingerprint is:
SHA256:lsfwRd7HQmfPZ+rjQOGjLkyr+M67/xImKZRtVD31T9s sec-lab@vm1

(**Note:** The above value and the below randomart image will be different each time for each student. The “randomart” image is a visualization of the SHA256 hash value to make it easier to compare.)

The key's randomart image is:

```
+---[RSA 3072]---+
|      ... .o . o|
|     .   oo + =.|
|    +  .  .+ + O|
|   o o = o . B+|
|  . . .S + + ..E|
|   . o.+. o o  |
|    . = o. . o |
|    o  =.  o . |
|   .oB=.+o  .  |
+----[SHA256]----+
```

Q11: Attach a screenshot of the result.

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

```
pc0398@pc0398host:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/sec-lab/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/sec-lab/.ssh/id_rsa
Your public key has been saved in /home/sec-lab/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:GgEUJIqrwJiotwIbjf2CfMzbF7ufQSMa2/GY2tiPxXI sec-lab@seclab-Virt
ualBox
The key's randomart image is:
+---[RSA 3072]----+
|   .o=.          |
|.. . .           |
|o      .         |
|+o      .        |
|*=    ..oSo      |
|B o    =+O .     |
|=++. o.=oE       |
|+o.=o =o= o      |
| .oo.+.=++       |
+----[SHA256]-----+
pc0398@pc0398host:~$
```

2. Let us verify the result:

   ls ~/.ssh/

   You may expect to see two files: *id_rsa* and *id_rsa.pub*, which should contain the private and public keys, respectively.

Lab Group: 26
Course: CSCE 5550
Semester: Spring 2022

Cryptography Lab

3. Let us try to establish an SSH connection (for simplicity, we will connect to our own host):

   ssh localhost

   The SSH server will request a password. Press “Control+Z” to escape. Suppose that we want to allow trusted users to access our host without entering a password. Such a user needs to possess a private key corresponding to the public key communicated to the server in a trusted manner. Such public key are called “authorized keys”.

4. Designate your public key as the OpenSSH authorized key as follows:

   cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys

5. This should allow us to establish an SSH connection (to our own host) without the use of passwords:

   ssh localhost (if the prompt about adding to the known hosts appears, then accept it) (If successful, a welcome message will be displayed.)

   Close the SSH connection:exit

Q12: Attach a screenshot of the result.

As there was a connection error to local host I tried to re-install openssh-server and start ssh connection.

```
pc0398@pc0398host:~$ ls ~/.ssh/
authorized_keys  id_rsa  id_rsa.pub
pc0398@pc0398host:~$ ssh localhost
ssh: connect to host localhost port 22: Connection refused
pc0398@pc0398host:~$ cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
pc0398@pc0398host:~$ ssh localhost
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.13.0-30-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

223 updates can be applied immediately.
136 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Tue Mar  8 11:14:18 2022 from 127.0.0.1
pc0398@pc0398host:~$
```

References:

1. OpenSSL version 1.1.1 manual: https://www.openssl.org/docs/man1.1.1/
2. OpenSSL Cookbook, 3ed online: https://www.feistyduck.com/library/openssl-cookbook/online/
3. OpenSSL Quick Reference Guide: https://www.digicert.com/kb/ssl-support/openssl-quick-reference-guide.htm

4. M. Anicas: OpenSSL Essentials: Working with SSL Certificates, Private Keys and CSRs: https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs
5. Wikibook on OpenSSH/Cookbook/Public Key Authentication: https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication