mod17_ex01: Creating a CA

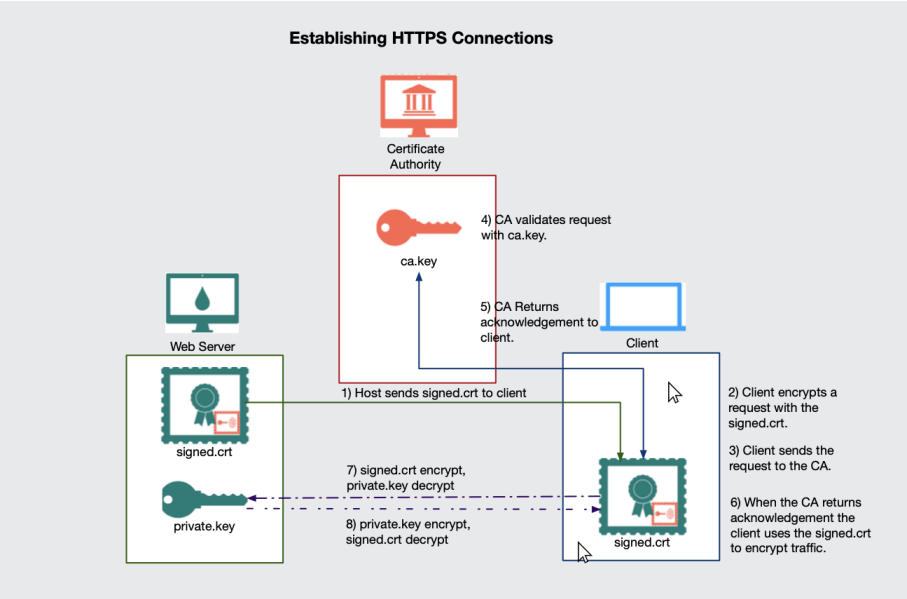
The purpose of this exercise is to create a simulated Certificate Authority and to sign a certificate signing request. This exericse presents the Transport Security Layer and introduces administrative skills for the openssl command.

Reference Information

The following documents provide information related to this exercise.

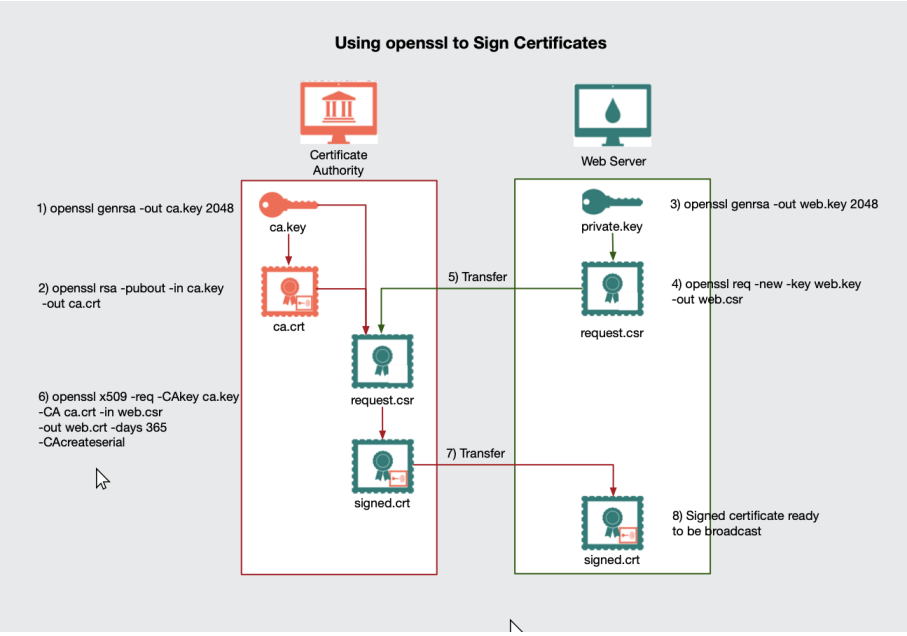
- TBD

1. Transport Layer Security (TLS) is used to establish HTTPS connectivity.



1. The web server sends a signed certificate to the client. The signed certificate is signed by a trusted Certificate Authority (CA).
2. The client reads the contact information for the CA from the signed certificate and uses this information to contact the CA.
3. The client encrypts a request using the signed certificate and sends this request to the CA.
4. The CA will use it's private key to decrypt this request.
5. If the request decrypts the CA sends back acknowledgement to the client.
6. The client will now establish secure network communications with the web server.
7. The client uses the signed certificate to encyrpt traffic and the web server uses the private key to decrypt.
8. In reverse, the web server will use the private key to encrypt and the client will use the signed certificate to decrypt.

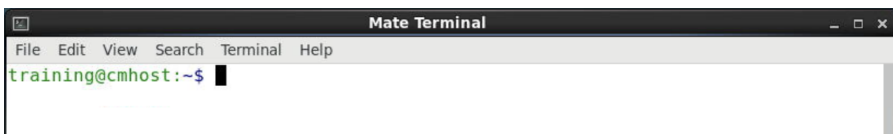
2. The openssl command manages TLS certificates.



The openssl command is used to generate private keys (.key), certificate signing requests (.csr), and to sign certificate signing requests (.csr) creating a signed public certificates (.crt).

1. The openssl command generates a private key (.key) for the CA.
2. The openssl command uses the private key to create a public key (.crt) for the CA.
3. The openssl command creates a private key (.key) for the web server.
4. The openssl command uses the web servers private key to create a signing request (.csr).
5. The signing request (.csr) is transferred to the CA.
6. The openssl command uses the CA private key (.key) and the CA public key (.crt) to sign the request (.csr).
7. The signed certificate (.crt) is transferred to the webserver.
8. The signed certificate (.crt) is ready to broadcast to any client.

3. Open a Mate terminal as the user training.



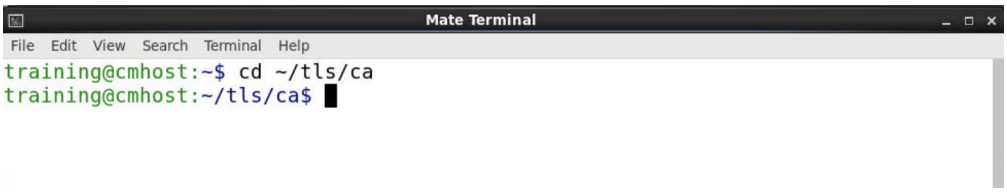
4. Create a keypair for the CA.

A key pair is an unpredictable (meaning large and random) number is used to generate an acceptable pair of keys suitable for use by an asymmetric key algorithm. The private key has the bulk of the encryption algorithm, typically containing either 1012 or 2024 characters of the encryption algorithm. The private key is identified with a file extension of .key. The public key contains fewer characters. The public key commonly has an extension of .crt. The two keys are referred to as a key pair.

4.1 Open a Mate terminal. Create four directories - ca, web, local, and client.

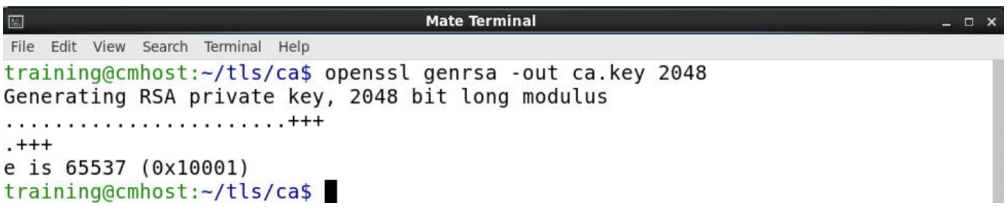

```
$ mkdir -p ~/tls/ca ~/tls/client ~/tls/local ~/tls/web
$ ls ~/tls
```

4.2 Change directory to ca.



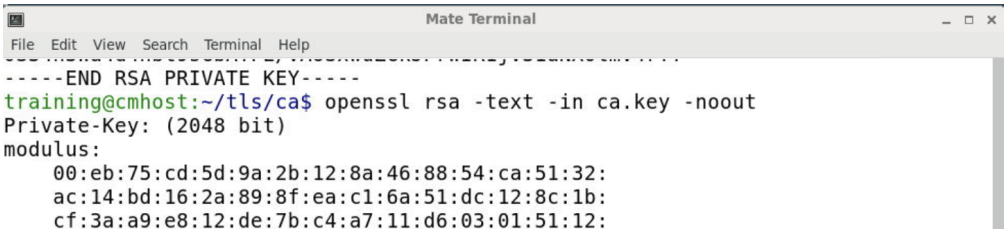

```
$ cd ~/tls/ca
```

4.3 Use the openssl command to create a private key for the CA.



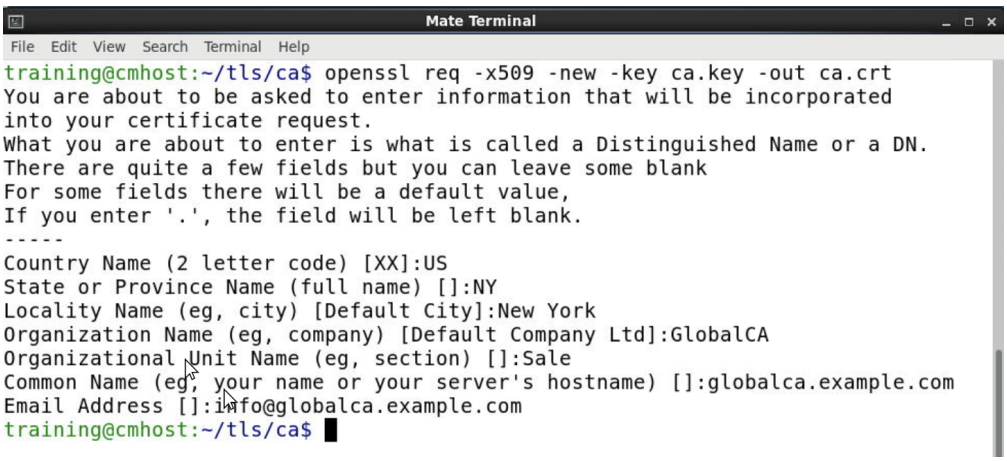

```
$ openssl genrsa -out ca.key 2048
```

4.4 Verify the private key. The output from both commands must match.



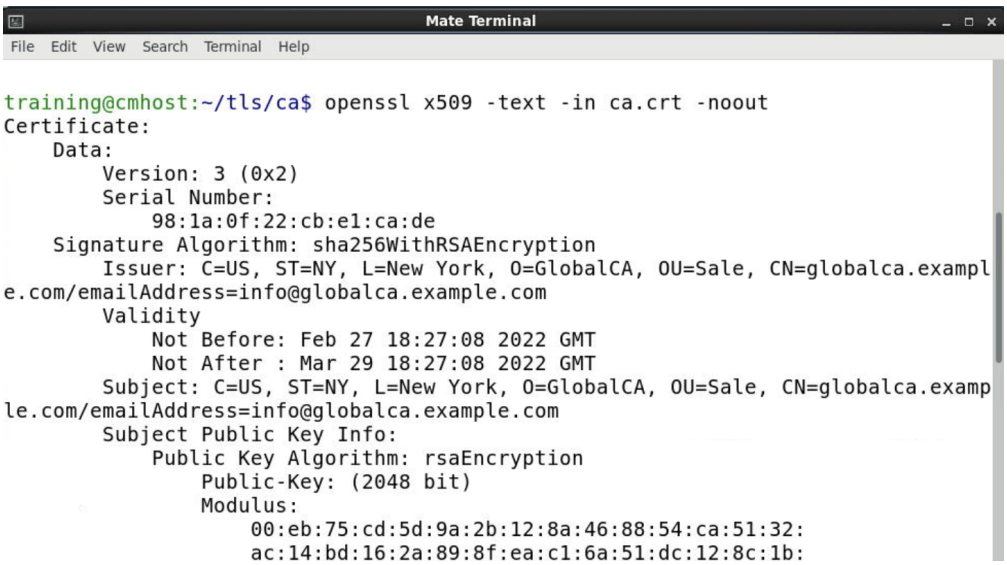

```
$ cat ca.key
            $ openssl rsa -check -in ca.key
            $ openssl rsa -text -in ca.key -noout
```

4.5 Use the CA's private key to create the CA's public certificate. Use the provided information. This information will be encoded into ca.crt.




```
$ openssl req -x509 -new -key ca.key -out ca.crt
```

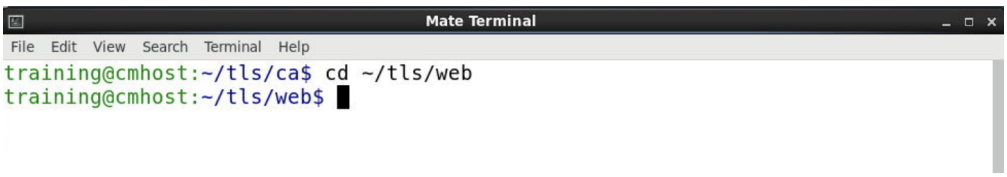
4.6 Verify the ca.crt certificate.




```
$ openssl x509 -text -in ca.crt -noout
```

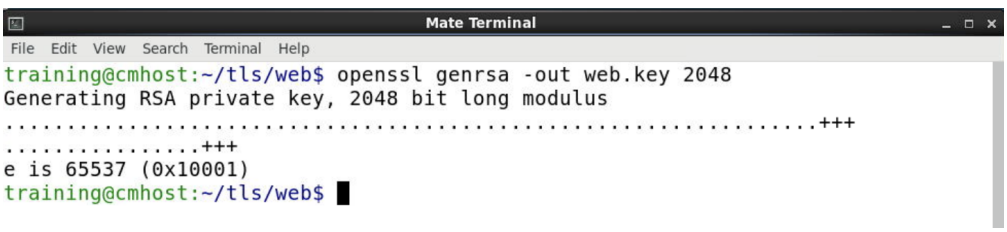
5. Create a private key for the web server.

5.1 Change directory to web server.




```
$ cd ~/tls/web
```

5.2 Create a private key for the web server.

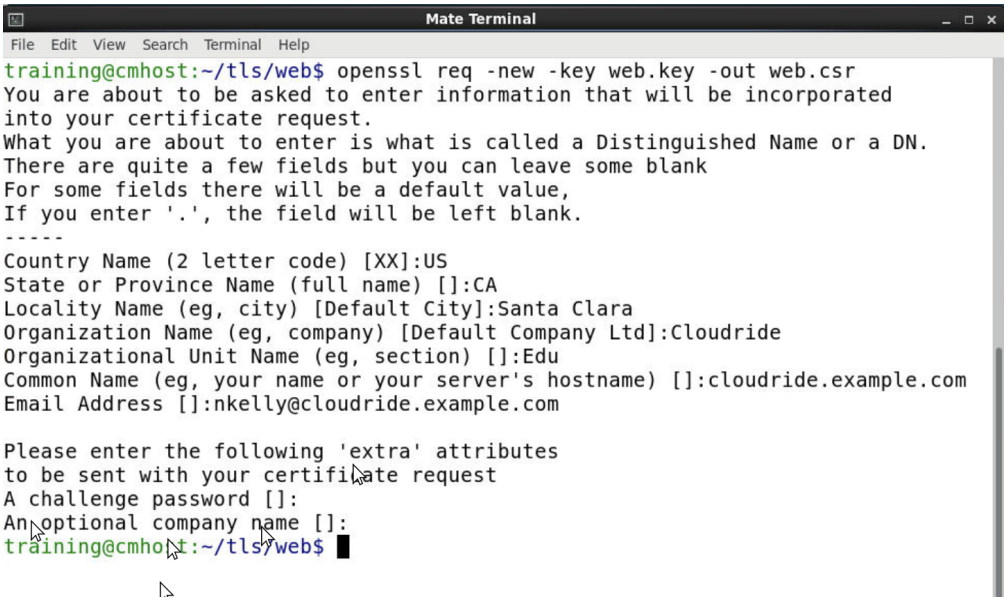



```
$ openssl genrsa -out web.key 2048
```

6. Create a Certificate Signing Request (.csr).

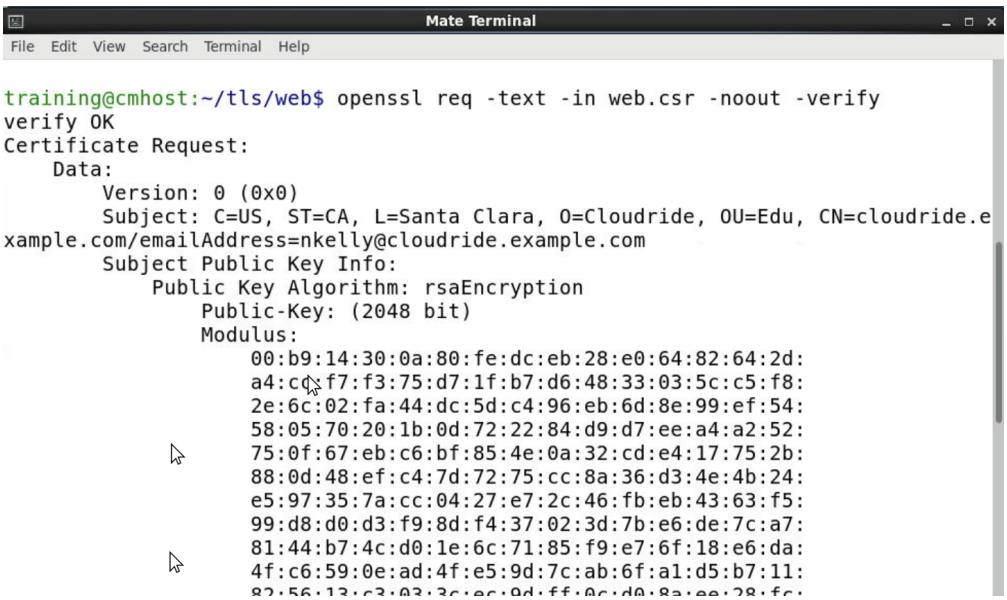
Note. There is syntax to create a .csr with a single command.

6.1 Use the private key (.key) to create a certificate signing request (.csr). Use the provided information.




```
$ openssl req -new -key web.key -out web.csr
```

6.2 Verify the content of the request.csr file. Scroll and review the contents of the request.csr. Scroll up to read the input information.




```
$ openssl req -text -in web.csr -noout -verify
```

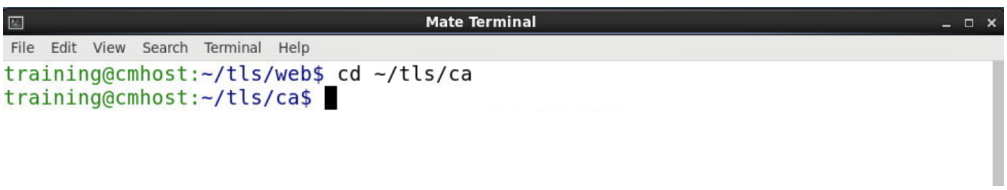
6.3 Transfer the certificate signing request (.csr) to the CA.


```
$ cp web.csr ~/tls/ca/web.csr
```

7. Sign a Certificate Signing Requests (.csr) creates a Signed Certificate (.cr).

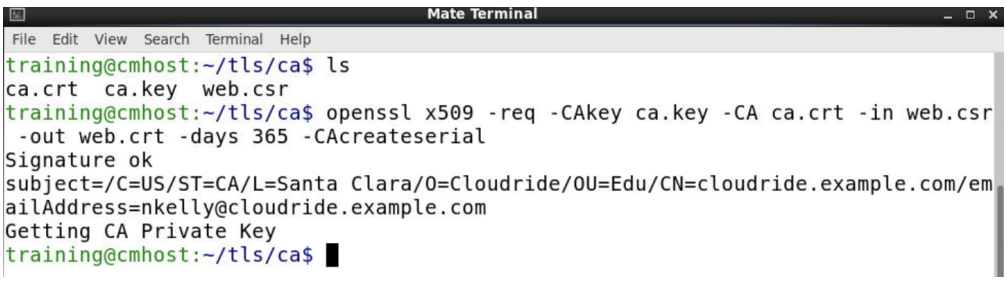
The certificate signing request (.csr) is submitted to the Certificate Authority (CA), this is generally through a web UI. The CA will sign the certificate signing request with its own private key (ca.key). The CA creates a new public key encrypted with its private key without compromising the certificate signing request (.crt) itself. The CA issues a command to encode additional information, such as the hostname or username, to create a new public key. This new public key is returned to the originator.

7.1 Change directory to the CA.




```
$ cd ~/tls/ca
```

7.2 Use the CA's private key and public certificate to sign the certificate signing request (.csr). The information from ca.crt will be written into web.crt.




```
$ openssl x509 -req -CAkey ca.key -CA ca.crt -in web.csr -out web.crt -days
365 -CAcreateserial
```

7.3 Transfer the signed public certificate back to the web server.

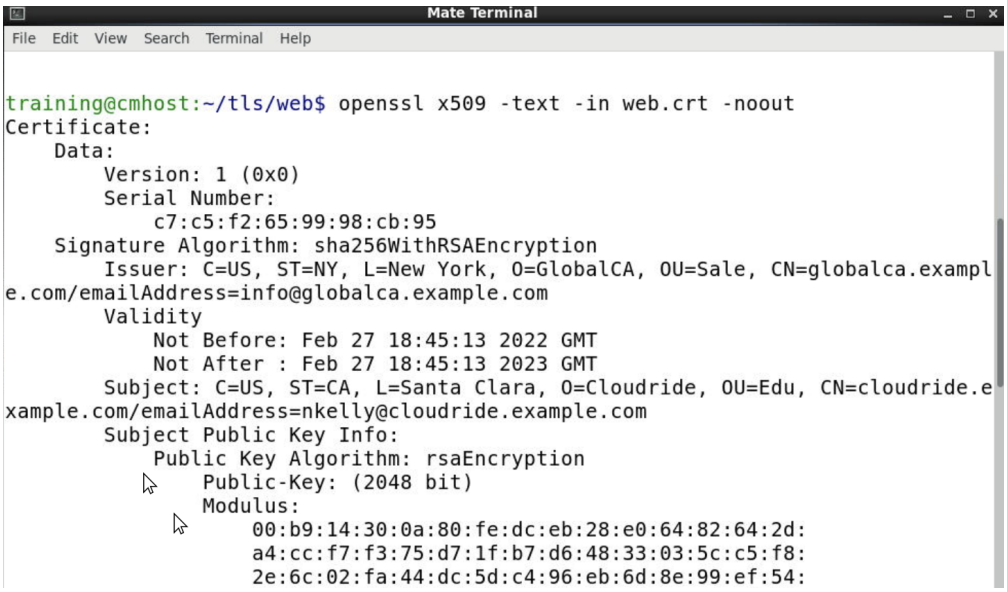

```
$ cp web.crt ~/tls/web/web.crt
```

8. Validate the signed public key.

8.1 Change directory to tls/web.

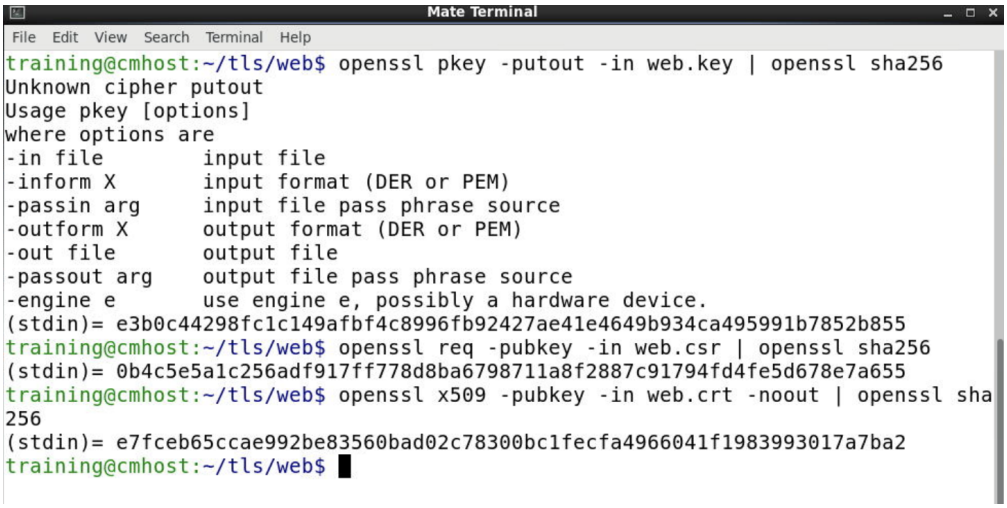

```
$ cd ~/tls/web
```

8.2 Review the Signed Certificate (.crt). Scroll up. Compare the information for the issuer and the subject.




```
$ openssl x509 -text -in web.crt -noout
```

8.3 Verify the keys match, extract the public key from each file and generate a hash output. All three files should share the same public key and the same hash value. You may also pipe to openssl md5.




```
$ openssl pkey -pubout -in web.key | openssl sha256
            $ openssl req -pubkey -in web.csr -noout | openssl sha256
            $ openssl x509 -pubkey -in web.crt -noout | openssl sha256
```

9. Self-Signed Certificates.

A standard work-around to avoid having to send a .csr to a CA is to create and use a self-signed certificate. Here the generating authority is the same as the signing authority. For Internet purposes this is considered distinctly unsecure. Self-signed certificates are only used for testing, training, and development purposes.