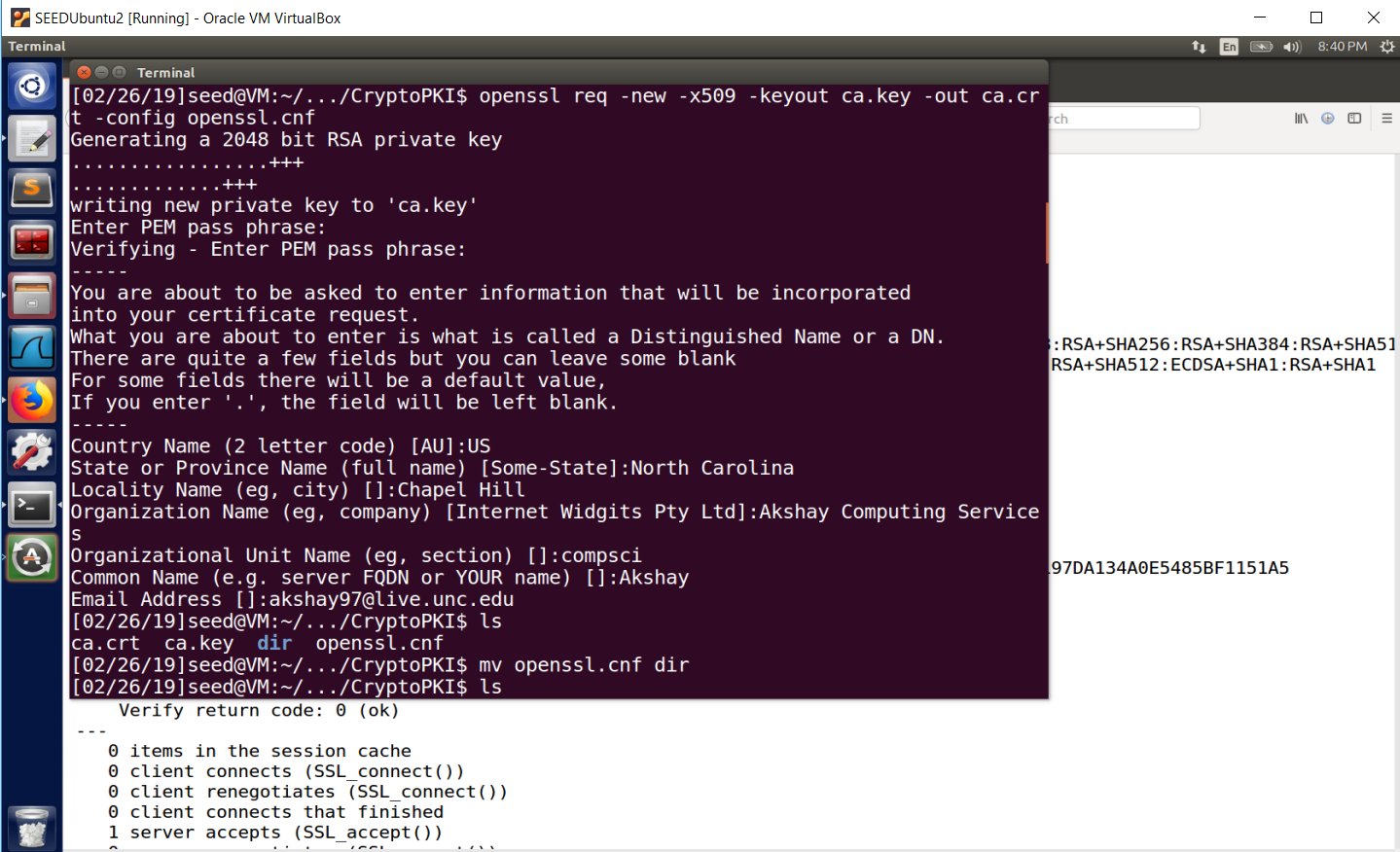
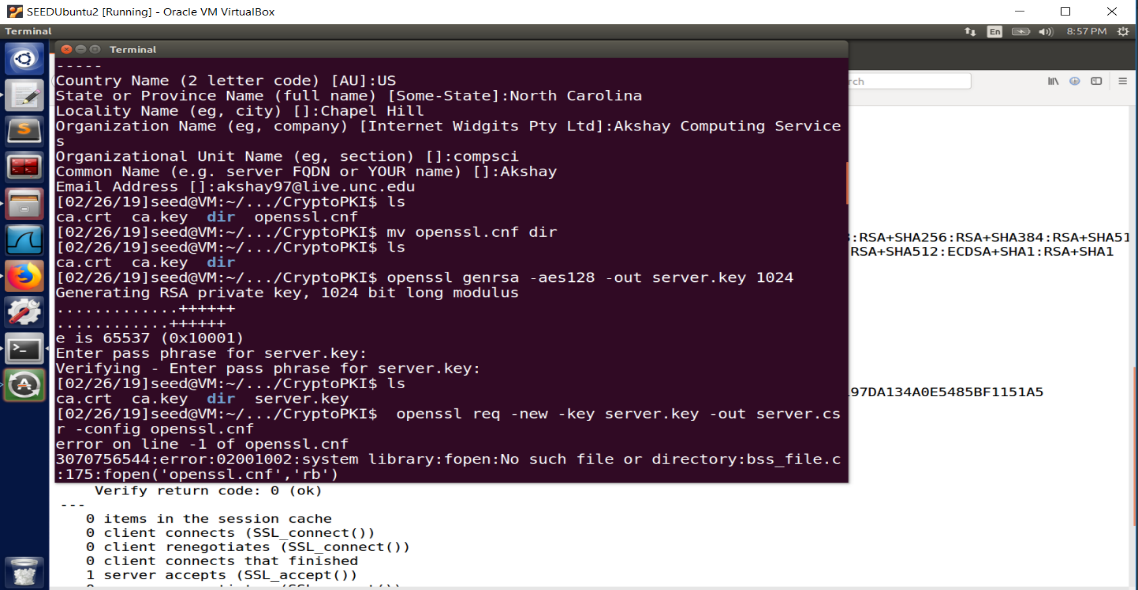
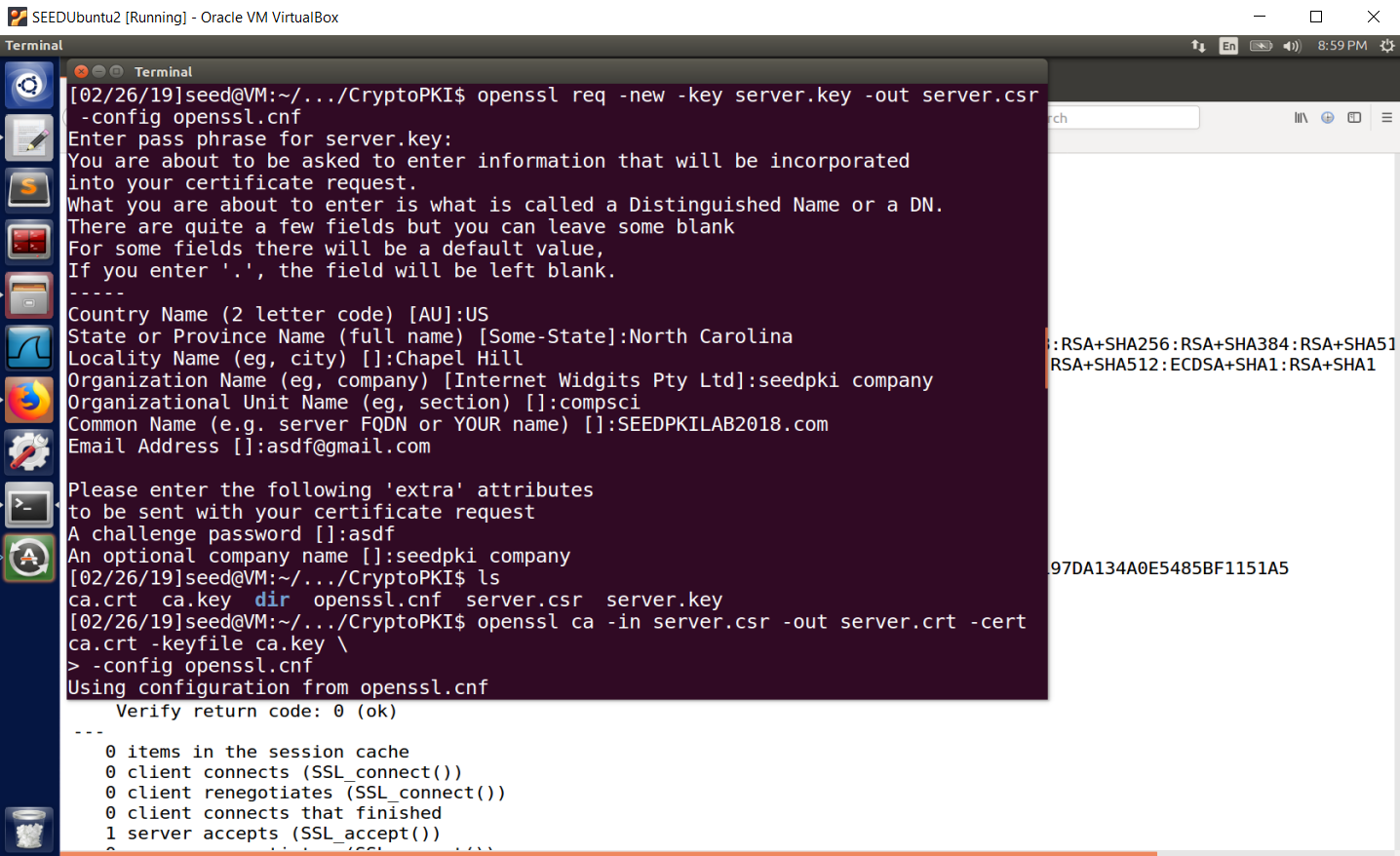
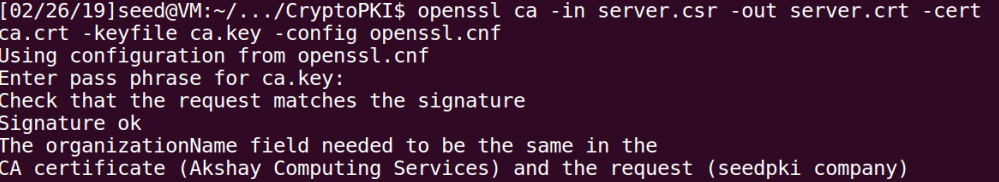
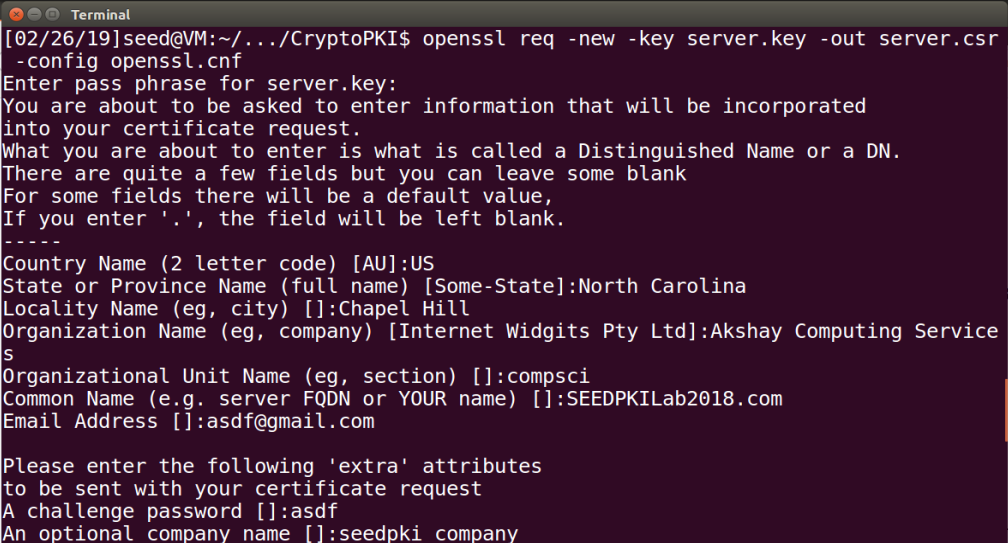
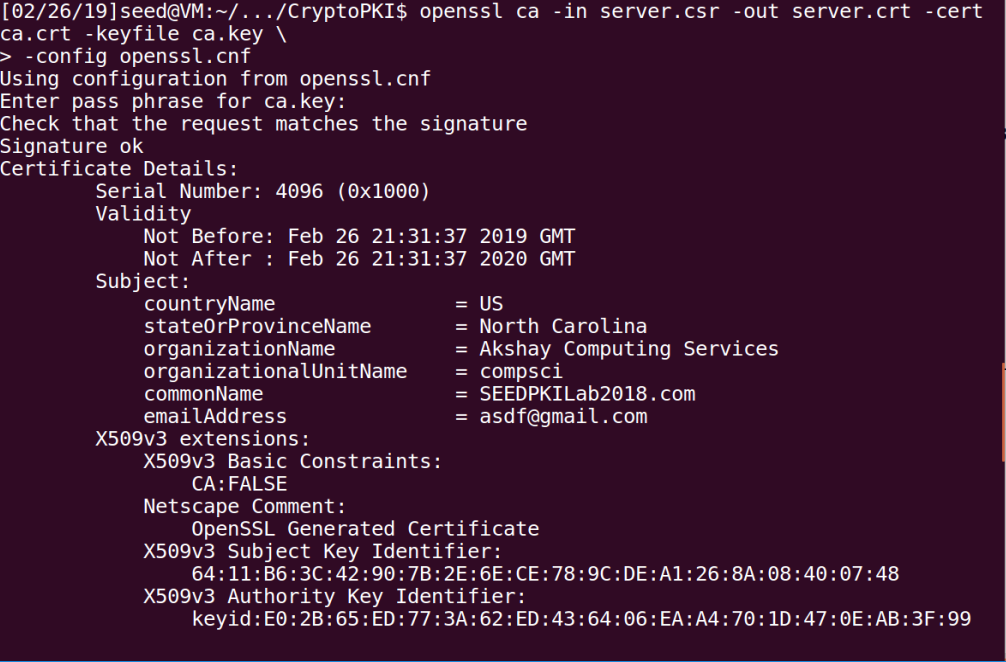
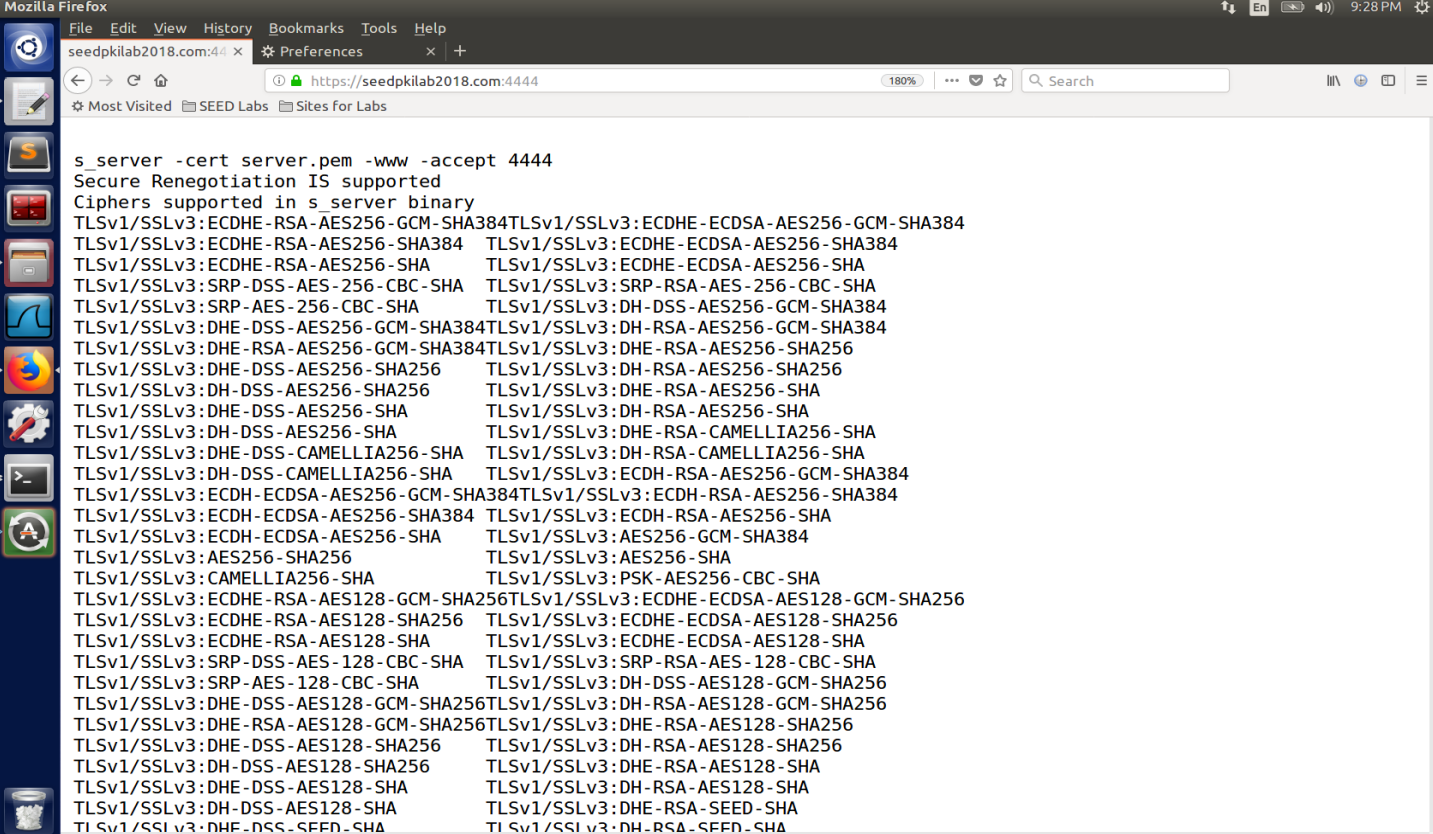
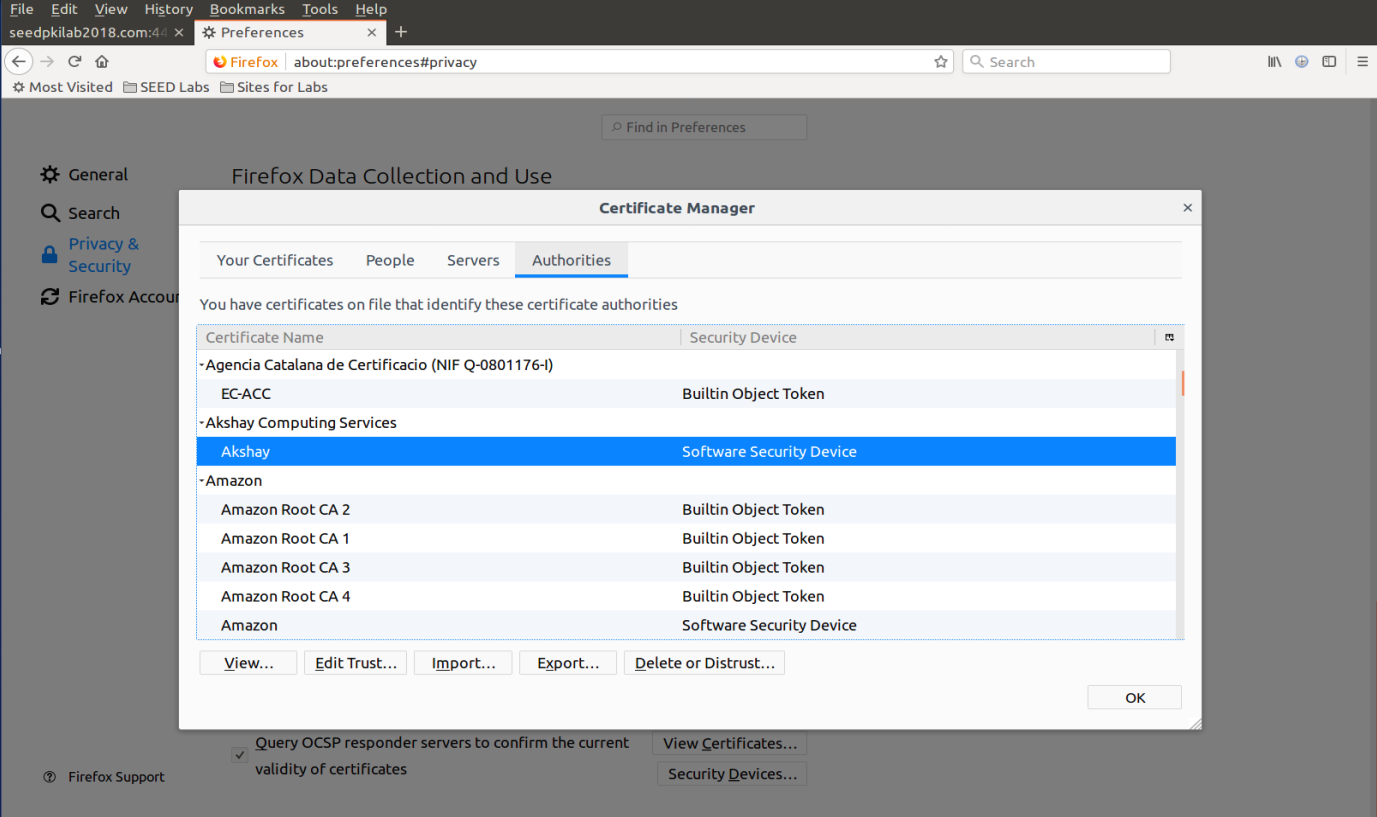
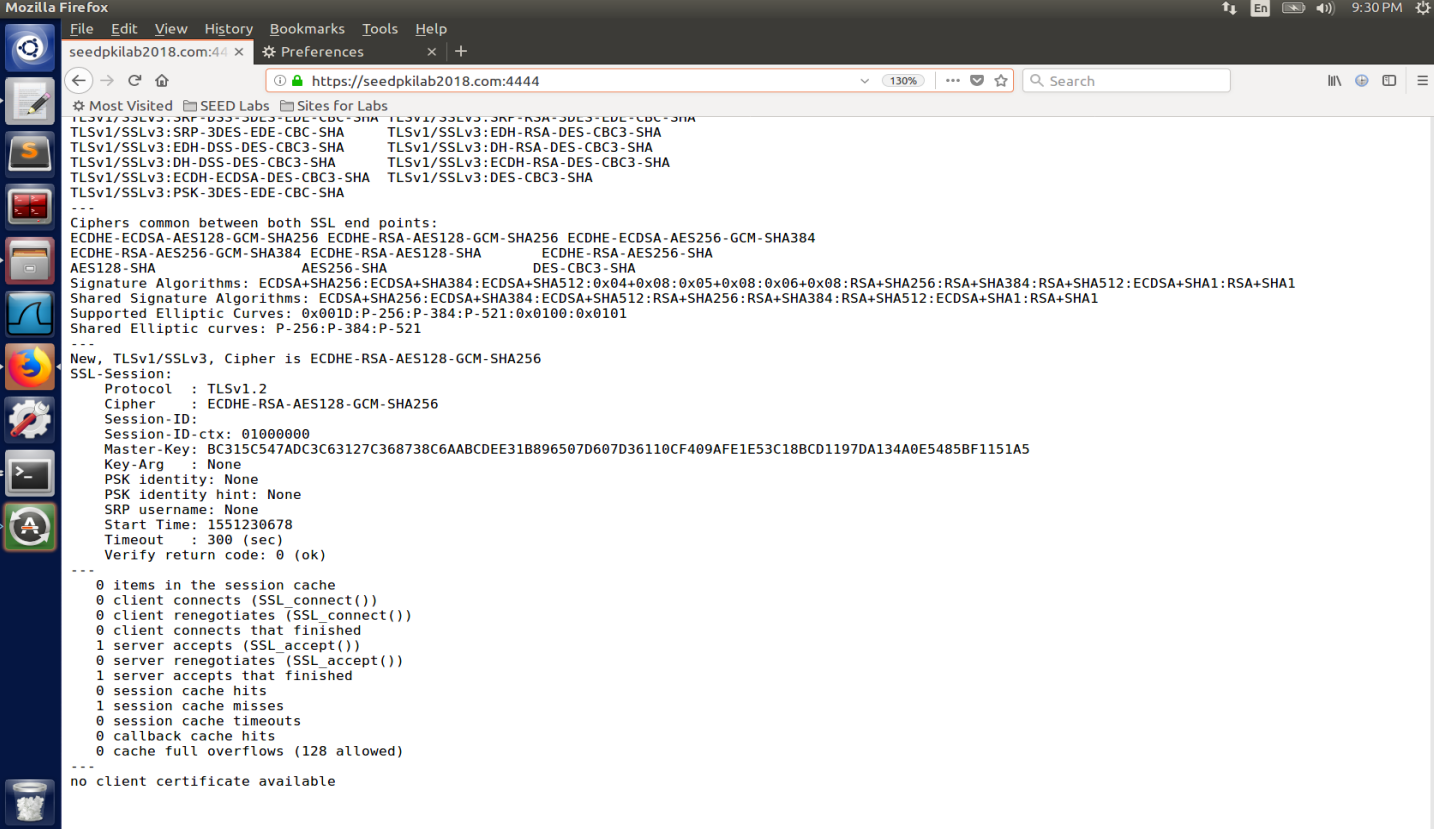
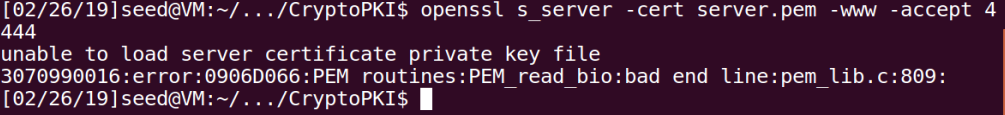
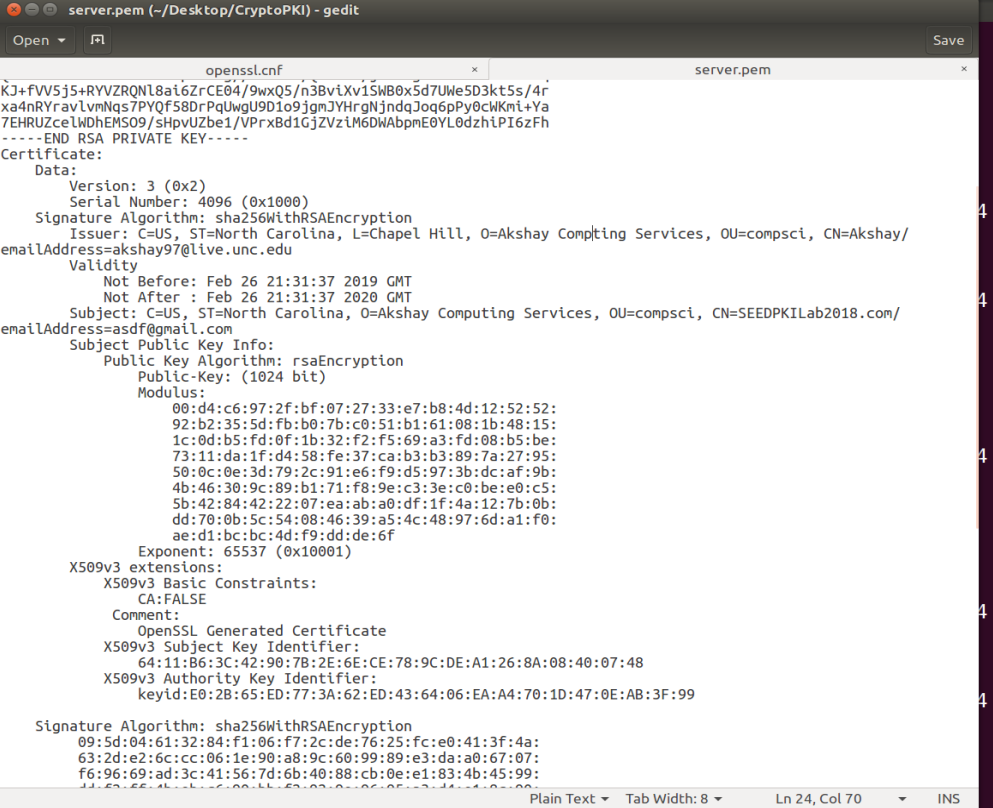
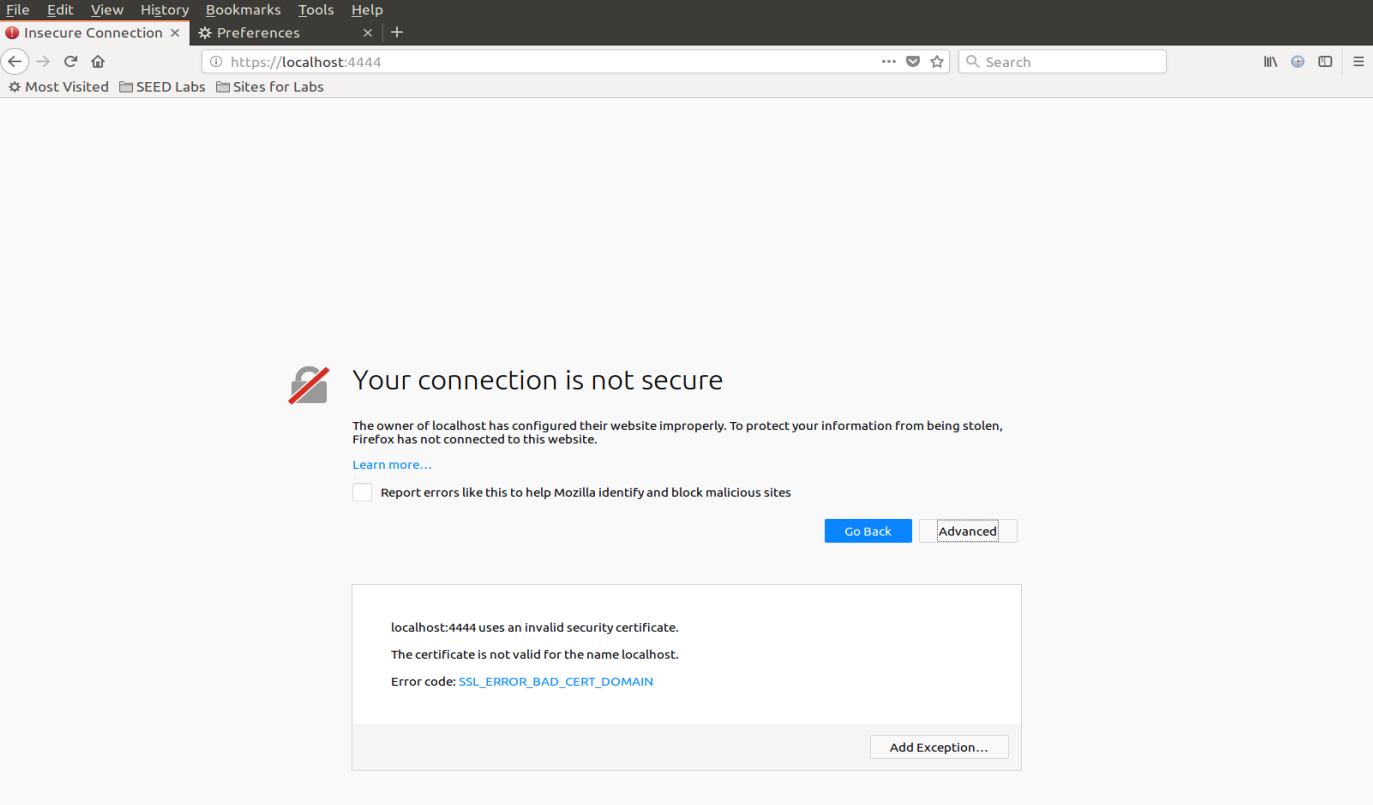
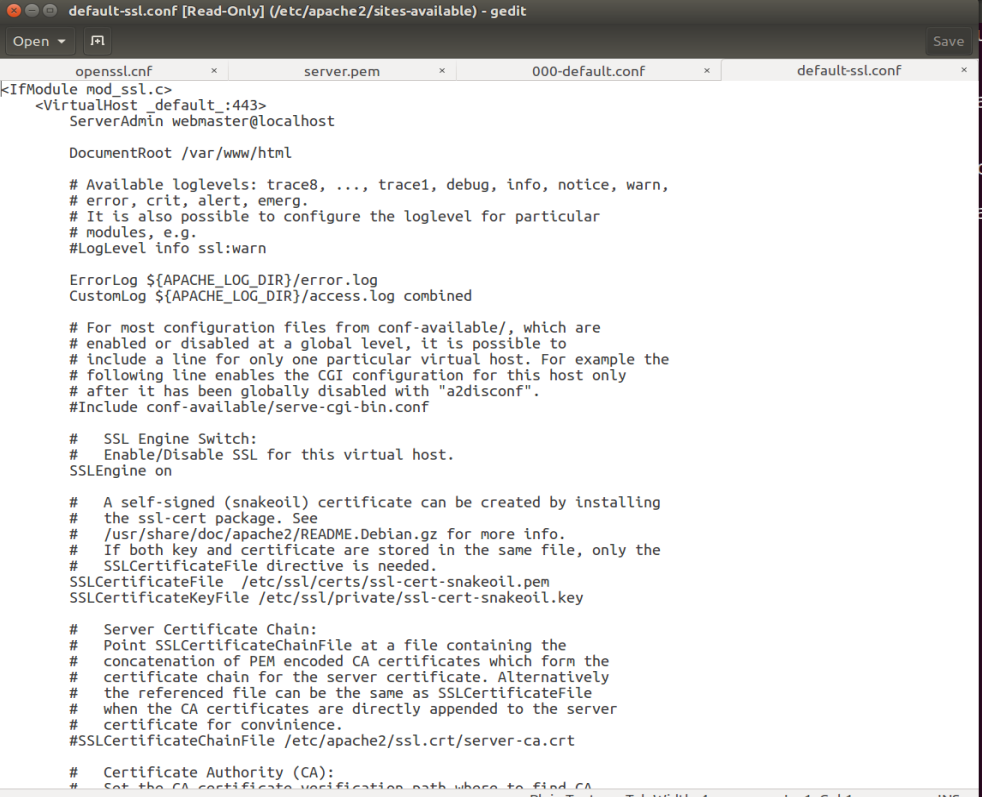
**Public Key Infrastructure (PKI) Lab**

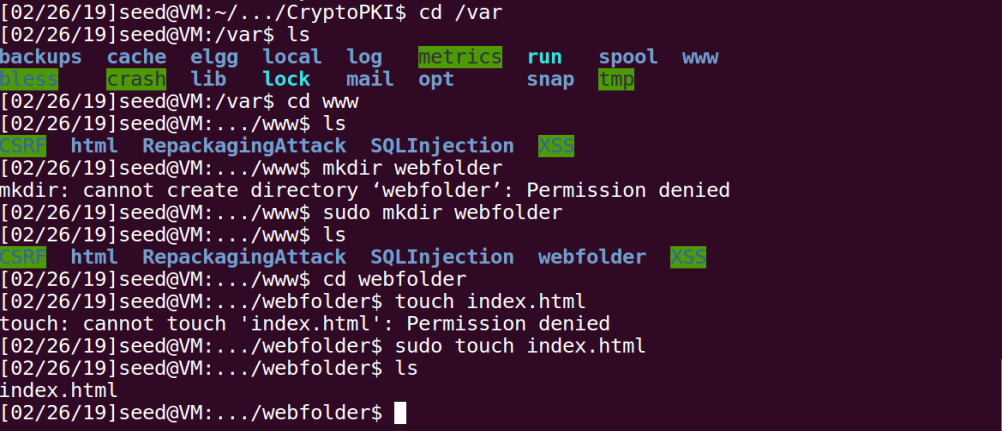
* Task #1: Becoming a certificate authority (CA)
  + After moving openssl.cnf to current directory and creating all prerequisite subdirectories we can become a certificate authority with the line
  + openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
  + We have Akshay Desai, US, North Carolina, Chapel Hill, Akshay Computing Services, and common name Akshay with my unc email as attributes for this certificate.
  + The resulting files ca.key has our private key and ca.crt has our public key certificate.
* Task #2: Creating a certificate for SEEDPKILab2018.com
  + Now for a company to receive a digital signature from us (a root CA) on their digital certificate, they need to have their own public and private key.
  + We can generate an RSA key pair for this fake company with the following line
  + openssl genrsa -aes128 -out server.key 1024

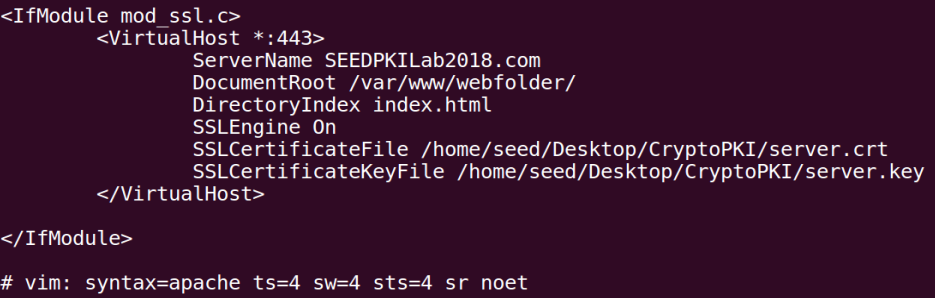


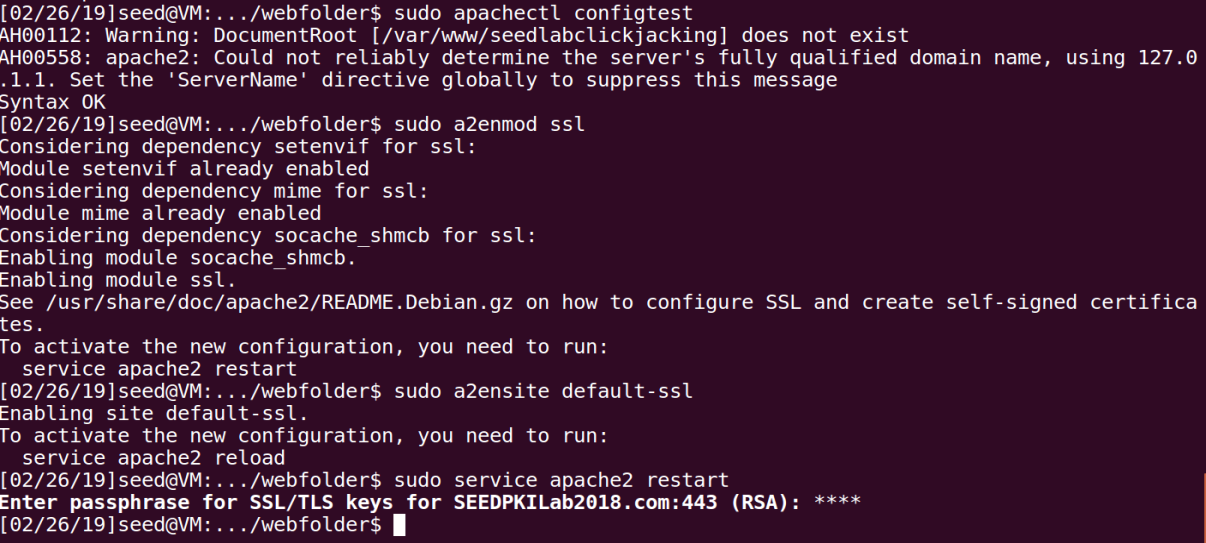
* + Then we want to generate a certificate signing request as this company, we can use the following line
  + openssl req -new -key server.key -out server.csr -config openssl.cnf
  + All we need to do is match the attributes according to the policy or else you get an error.
  + After fixing and redo-ing, the csr is created with common name SEEDPKILab2018.com.
  + Now we can create a certificate with the CA’s signature using the CA’s private and public key. We are using the following command to turn the certificate signing request into an actual certificate with our ca.crt and ca.key
  + openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \ -config openssl.cnf
* Task #3: Deploying certificate in an HTTPS web server
  + First we configure our DNS to recognize the website we want to visit, SEEDPKILab2018.com by going to /etc/hosts and appending this to 127.0.0.1.
  + We want to now launch a web server with the certificate generated and we can do this with openssl.
  + # Combine the secret key and certificate into one file
  + % cp server.key server.pem
  + % cat server.crt >> server.pem
  + # Launch the web server using server.pem
  + % openssl s\_server -cert server.pem -www -accept 4444
  + I am listening on port 4444. I can now access <https://SEEDPKILab2018.com:4444/>
  + But we receive an error until we allow firefox to recognize our CA certificate by going into Firefox settings and importing our ca.crt into the list of certificates. Then we can see our certificate in the list of accepted certificates.
  + Now we can visit the site and we are shown this
  + It seems that Firefox is allowing us to visit our web server listening on port 4444 as a result of the website’s certificate being signed by a root CA that Firefox allows because we imported it. The page also provides us with ciphers in the s\_server binary and then ciphers common between both SSL endpoints. It looks like some of those are created by the SHA cryptographic algorithm in sizes of 128, 256, and 512 bytes. Also includes SHA1 hash algorithm. Then it gives us a new TLSv1/SSLv3 along with the SSL session and says no client certificate available.
  + 1. Modify a single byte of server.pem, and restart the server, and reload the URL. What do you observe? Make sure you restore the original server.pem afterward. Note: the server may not be able to restart if certain places of server.pem is corrupted; in that case, choose another place to modify.
  + If we modify the first letter in the RSA private key we get an error like
  + Placing it back and removing the letter ‘u’ from organization in issuer and removing “Netscape” from “Netscape Comment” allows us to connect.

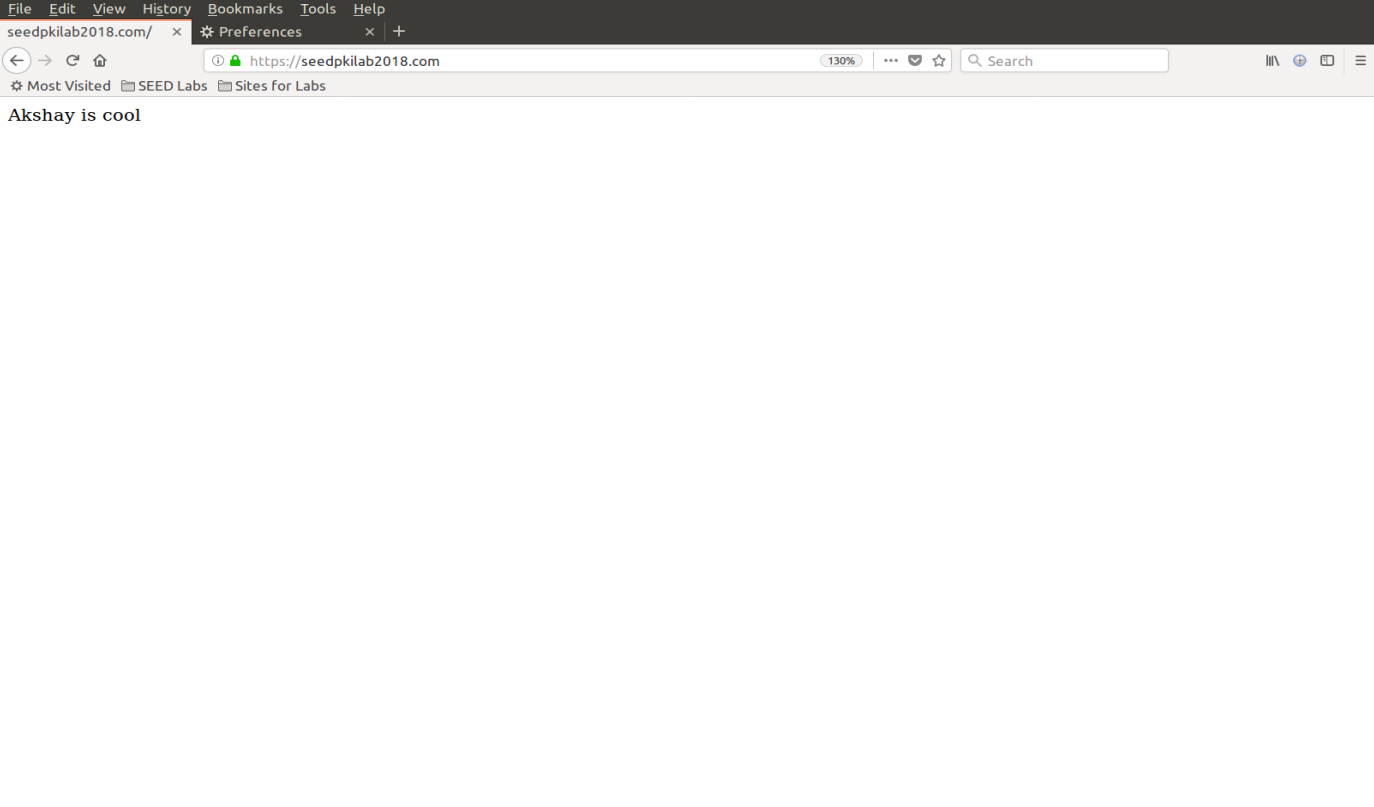
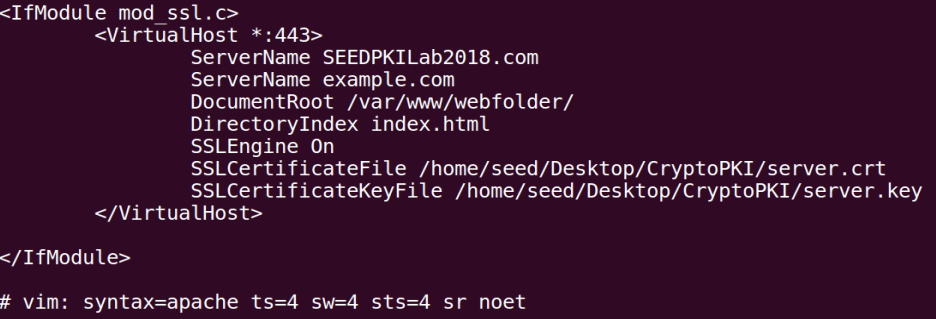
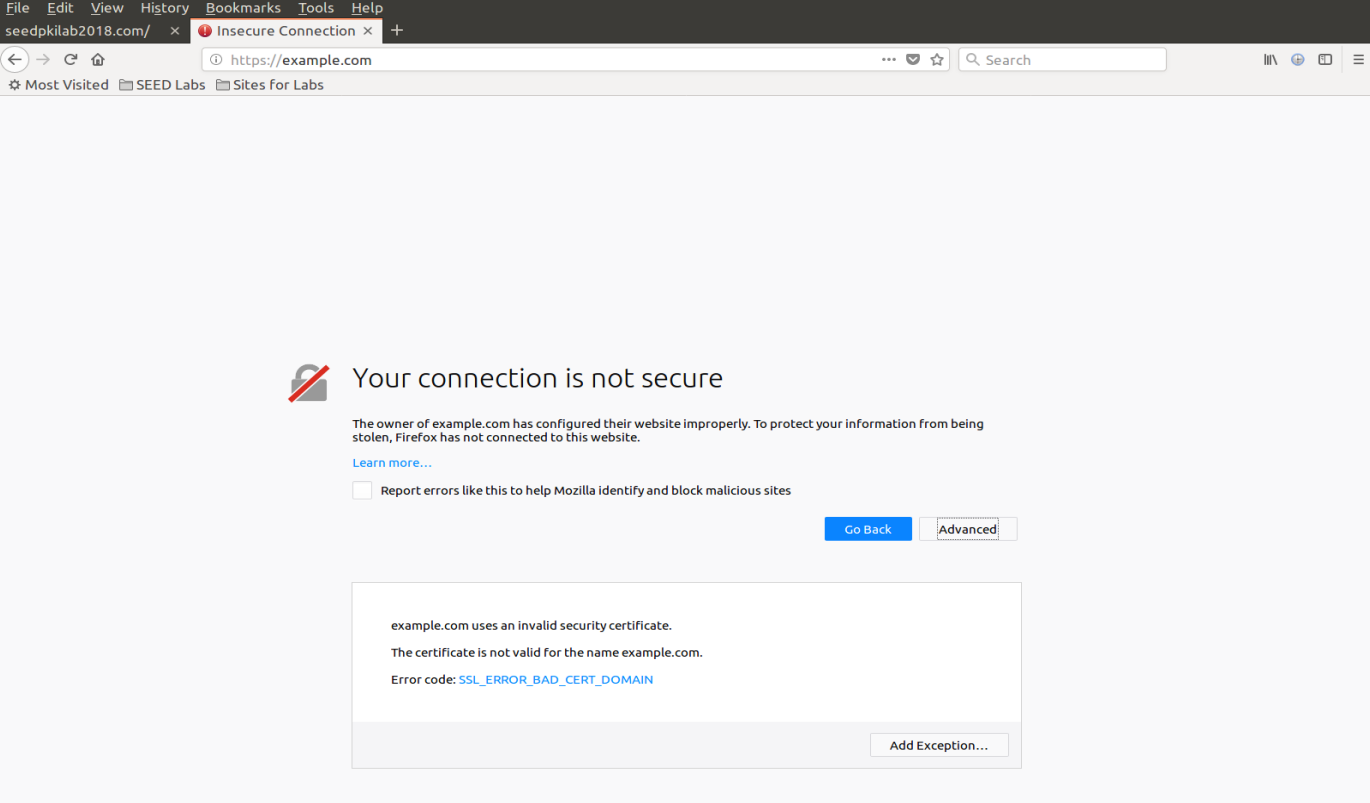
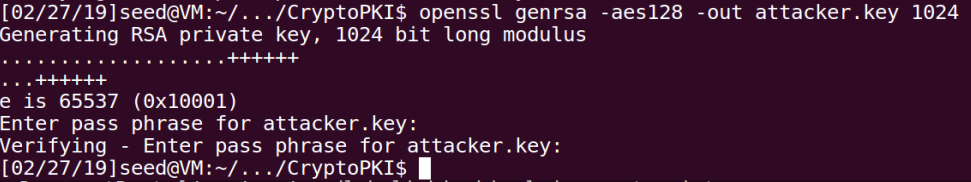
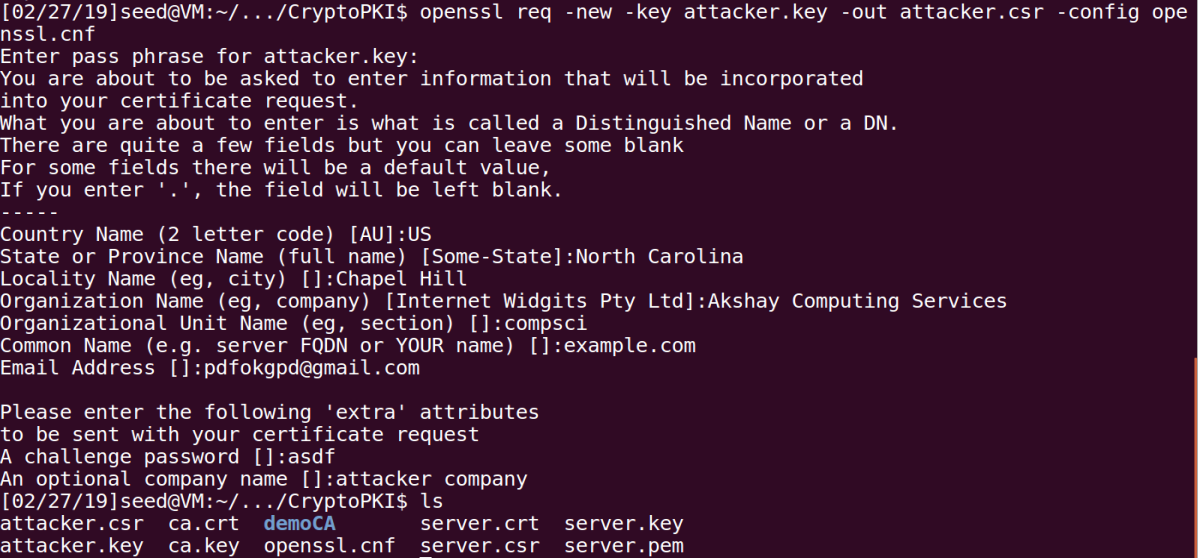
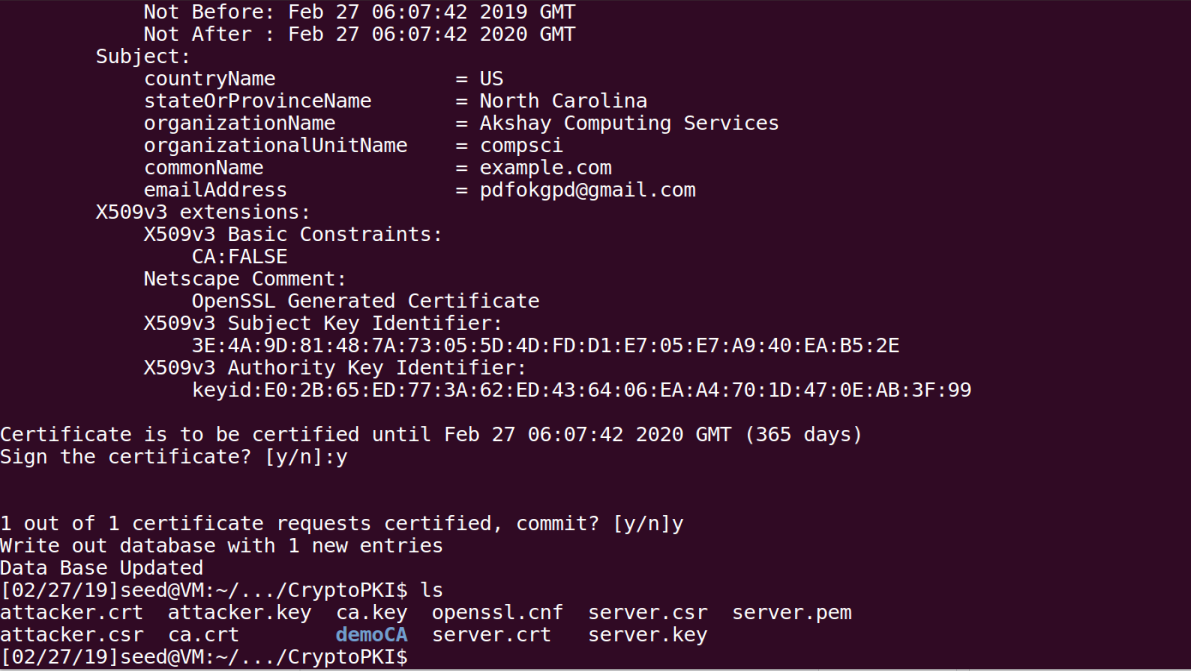
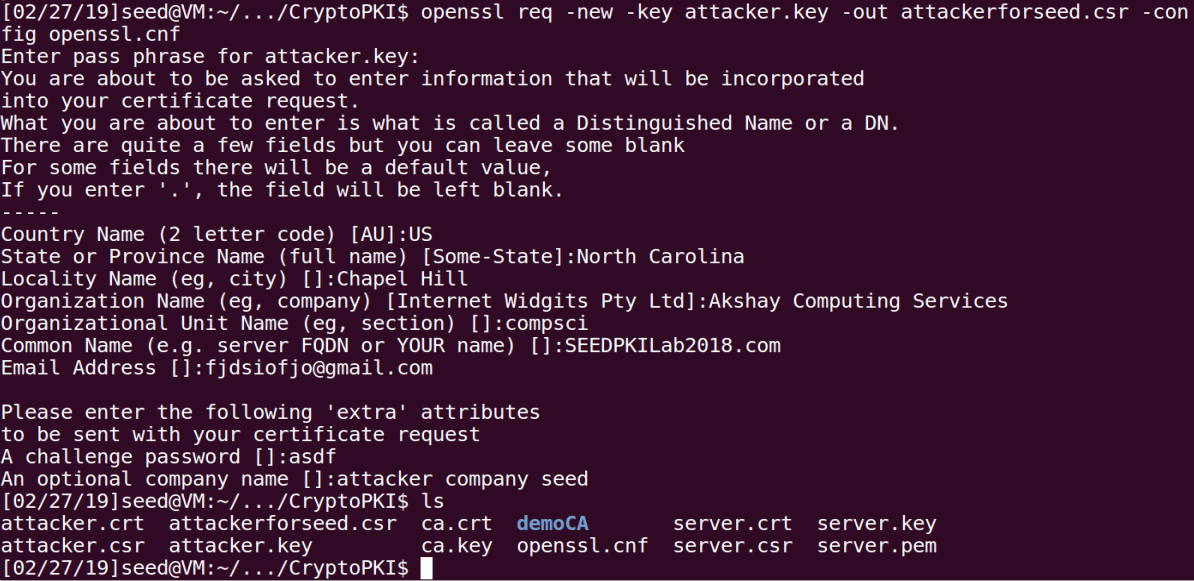
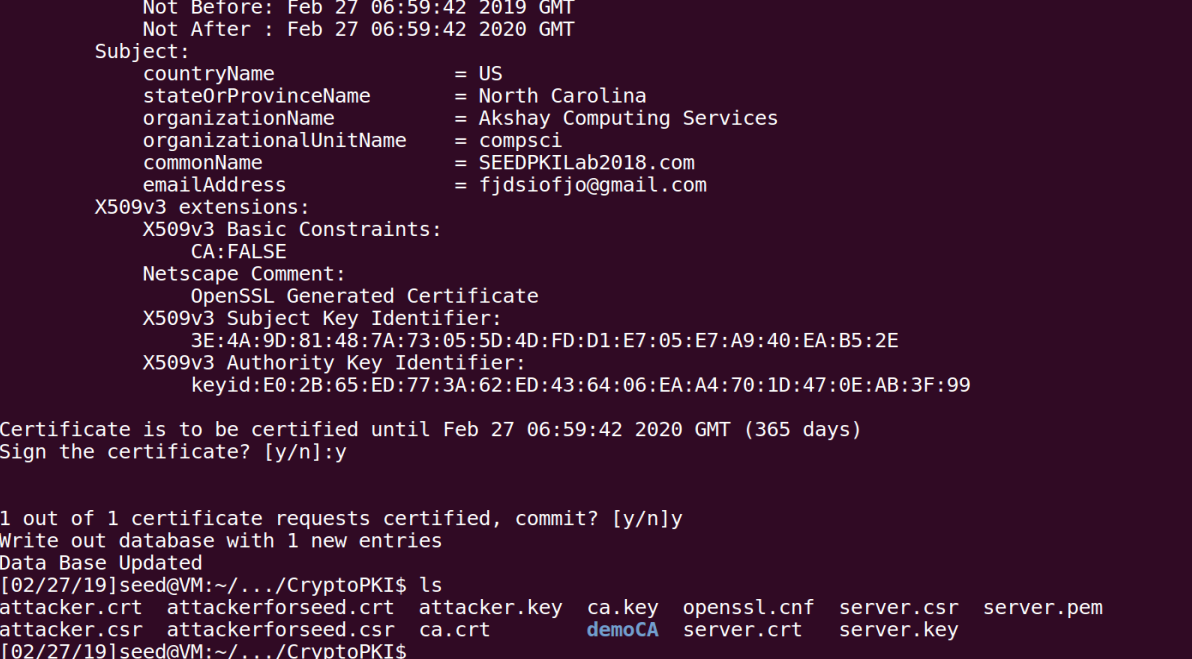
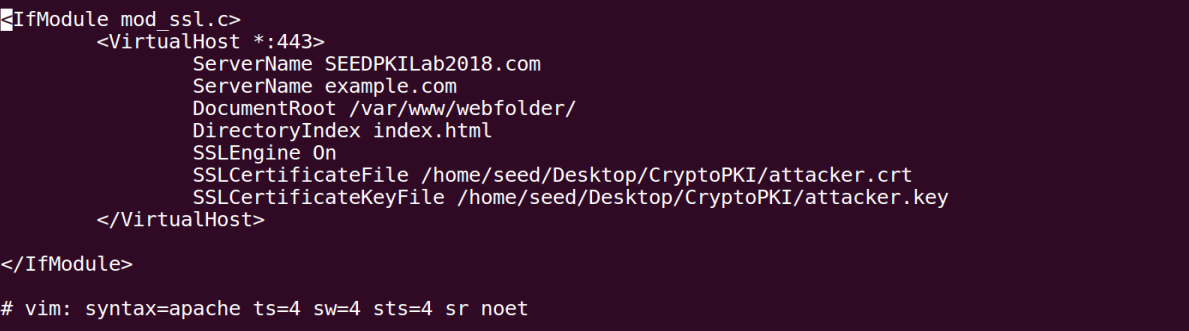
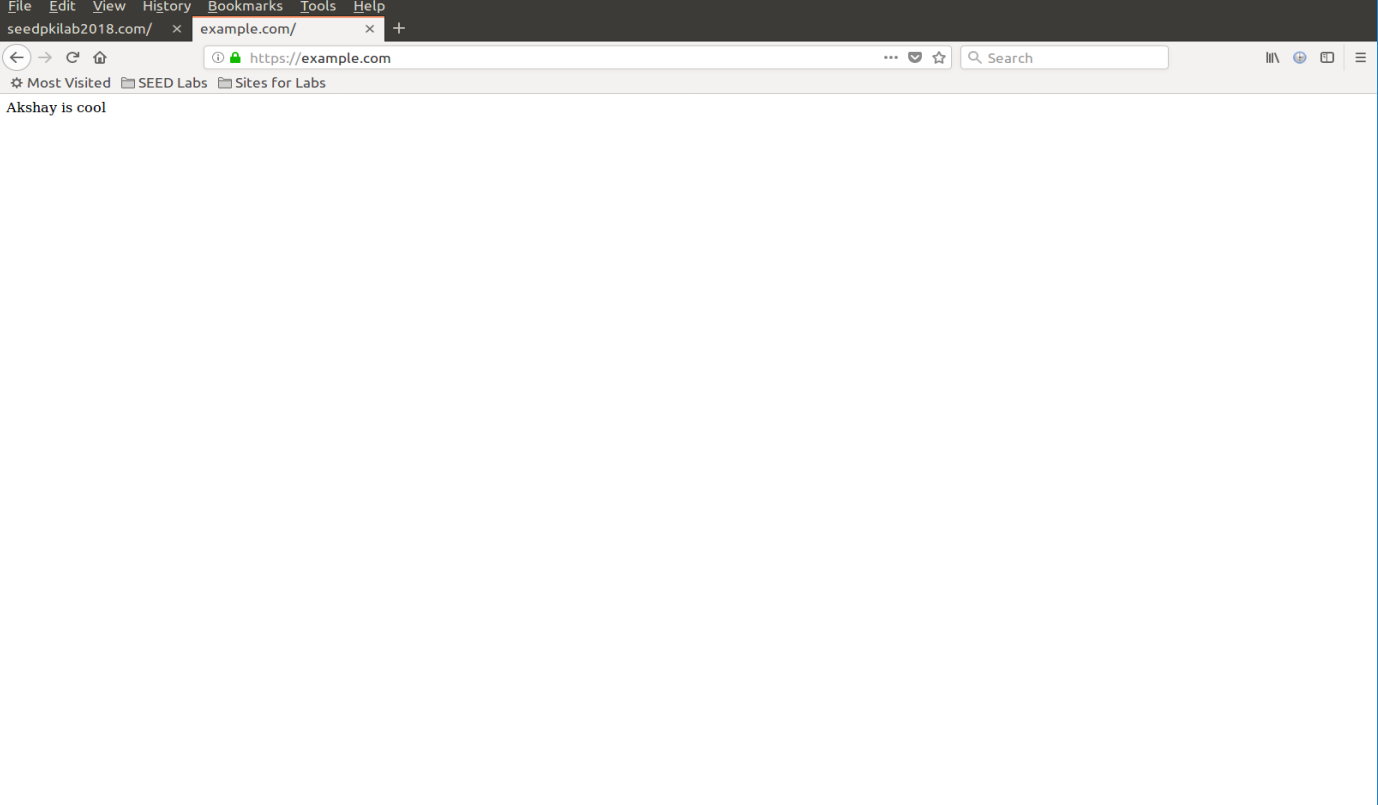


* + Also just removing the letter ‘u’ from organization issuer allows us to connect.
  + It makes sense that removing part of the keys does not allow you to create the server with the certificate and changing some titles of names still works. It is interesting that changing the organizer in the issuer still allows you to create the server.
  + 2. Since SEEDPKILab2018.com points to the localhost, if we use https://localhost:4433 instead, we will be connecting to the same web server. Please do so, describe and explain your observations.
  + We receive an error which states the connection is not secure. It says the owner of localhost has configured their website improperly and therefore firefox has decided not to connect to the website. If we open advanced to look at the error, we can actually see that the cause of this is because localhost:4444 uses an invalid security certificate and that the certificate is not valid for the name localhost. This makes sense as we signed the certificate with the name SEEDPKILab2018.com, not localhost, even though they connect to the same web server. We are correctly receiving a bad certificate error as we did not specifically sign for localhost.
* Task #4: Deploying certificate in an Apache-based HTTPS website
  + Now we want to set up a real HTTPS web server based on Apache. We can configure Apache to correctly retrieve private keys and certificates. We want to do so for https://SEEDPKILab2018.com.
  + We first give Apache the directory where the website’s files are stored. Apache has a VirtualHost file located in /etc/apache2/sites-available/; then if we want to add an HTTPS website we add a VirtualHost entry to the file default-ssl.conf in the same directory. Taking a look at the default-ssl.conf file
  + We have a document root at the top. Thanks to the friendly advice from our amazing TA’s, we can go ahead and make a folder with an index file which that can point to.



* + My document root is located in /var/www/webfolder/ with the index file being index.html. I placed a string in index.html so we will see something when we load it.
  + Our server key is located in 
  + /home/seed/Desktop/CryptoPKI/server.key
  + And our server certificate in /home/seed/Desktop/CryptoPKI/server.crt
  + With this information, we get the following VirtualHost entry
  + I had much trouble with document root because I had it set to /var/www/webfolder/index.html and it didn’t work because that is the file itself and not the root directory which it is in. After changing this I did the apache commands given in the lab



* + And I was able to successfully connect to the website with no errors <https://seedpkilab2018.com>
* Task #5: Launching a Man-In-The-Middle attack
  + Advice says to attack <https://example.com>
  + We already have https website SEEDPKILab2018.com, we will use the same apache server to attack <https://example.com>. We first add the ServerName example.com to the default-ssl.conf file.
  + I spent a large amount of trying to add an entire VirtualHost tag with the same information as above. Nothing was changing for either website. A couple hours later I decided to have 2 servernames in the same VirtualHost tag and I got the desired result. This makes sense as everything else is the same and there is one default port. The goal being to mimic a target website and trick the user into providing our server with potentially sensitive information.
  + In the assignment we are told to use the “attack” DNS method to get the user’s HTTP request to land to our server. We modify the victim’s machine’s /etc/hosts file to reroute the ip for example.com to the ip of our server. We are both the attacker and the victim in this lab thus the ip is 127.0.0.1.
  + Now we can finally visit the website.
  + And good ole firefox realizes there’s something fishy going on. Firefox correctly checks the certificate for example.com and tells us that it is invalid. That is, it has not been signed legitimately by a legitimate CA. This makes sense as we never signed example.com with our root CA either so even though the user is being essentially redirected to our server, PKI is warning the user that the certificate is bogus for the website and does not connect immediately.
* Task #6: Launching a Man-In-The-Middle attack with a compromised CA
  + We are given that the private key from our root CA from task 1 is compromised and an attacker has it.
  + We can use the same steps in task 1 to create another certificate as the attacker using the private key we stole from the root CA. We are told to use the same country, state, locality, organization, and organization unit and only change common name and address fields. We are also attacking <https://example.com>.
  + I will generate a new RSA public/private key pair just like in task 1 and output it to attacker.key with the following line
  + openssl genrsa -aes128 -out attacker.key 1024
  + Then I will create a new certificate signing request with the attacker’s key.
  + openssl req -new -key attacker.key -out attacker.csr -config openssl.cnf
  + Using same country, state, locality, organization, organization unit as root CA and only changing common name to example.com as that is what we are attacking and a bogus email address.
  + Now we generate the certificate with the root CA’s signature by using the CA’s private key which we stole and is contained in ca.key with the CA’s public key certificate stored in ca.crt which is of course public to us anyways.
  + openssl ca -in attacker.csr -out attacker.crt -cert ca.crt -keyfile ca.key \ -config openssl.cnf
  + Before we connect to example.com to redirect to seedpkilab2018, we have to realize that seedpkilab2018 was signed by another entity already and not us, the attacker. Since we didn’t use the server.key which we created previously that signed seedpkilab2018 and created a new one to simulate an attacker, we have to create the csr for seedpkilab2018 just like we did in task 1 but for the attacker.
  + And also sign it as well using the ca.key we stole, just as before.
  + openssl ca -in attackerforseed.csr -out attackerforseed.crt -cert ca.crt -keyfile ca.key -config openssl.cnf
  + Then because we are a new entity that is an attacker, we can change the default-ssl.conf to have the new certificate and key.
  + Then finally, now that the certificates have been created using the private key we stole from a root CA, the common name example.com is now signed by “Akshay Computing Services” which is a trusted root CA. As the attacker, all we would need to do now is “attack” the DNS and reroute example.com to our web server which we have already done in task 5. So now we can check if our attack worked by visiting example.com
  + And the browser doesn’t complain about going to example.com and nothing suspicious, no errors.
  + We can now steal valuable information that a user might enter here because they are actually connected to our web server as both sites are legitimately signed but the CA’s signature is wrongfully trusted. In reality an attacker would model this site to look exactly like some popular site like facebook and a user would very likely be tricked.