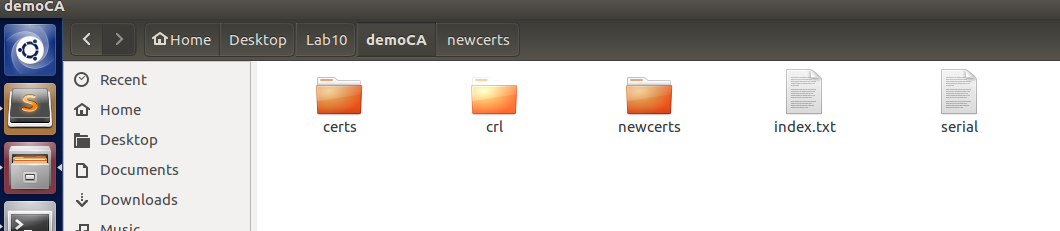
**Public-Key Infrastructure (PKI) Lab**

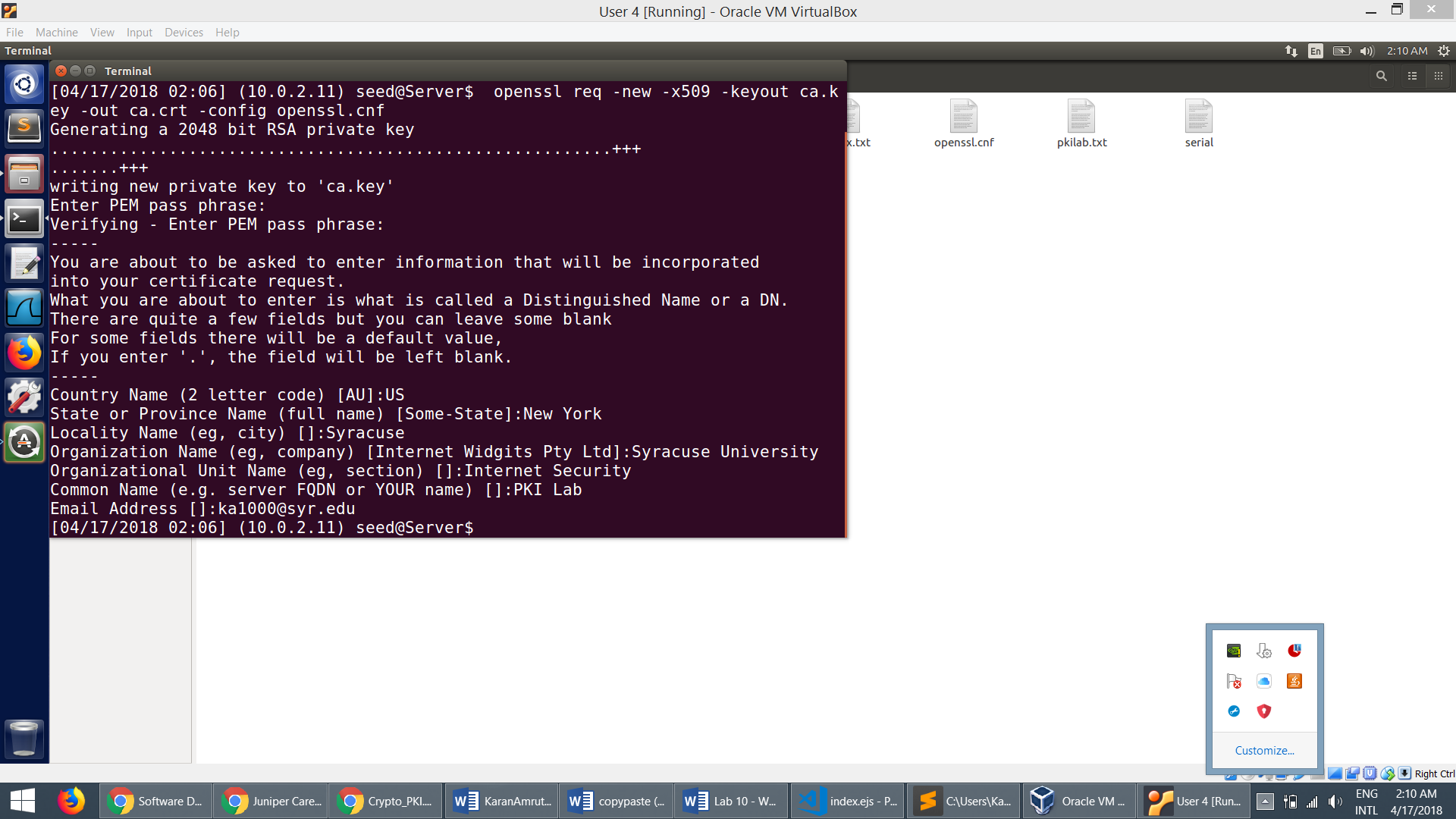
**Karan Amrutesh**

**Task 1: Becoming a Certificate Authority (CA)**

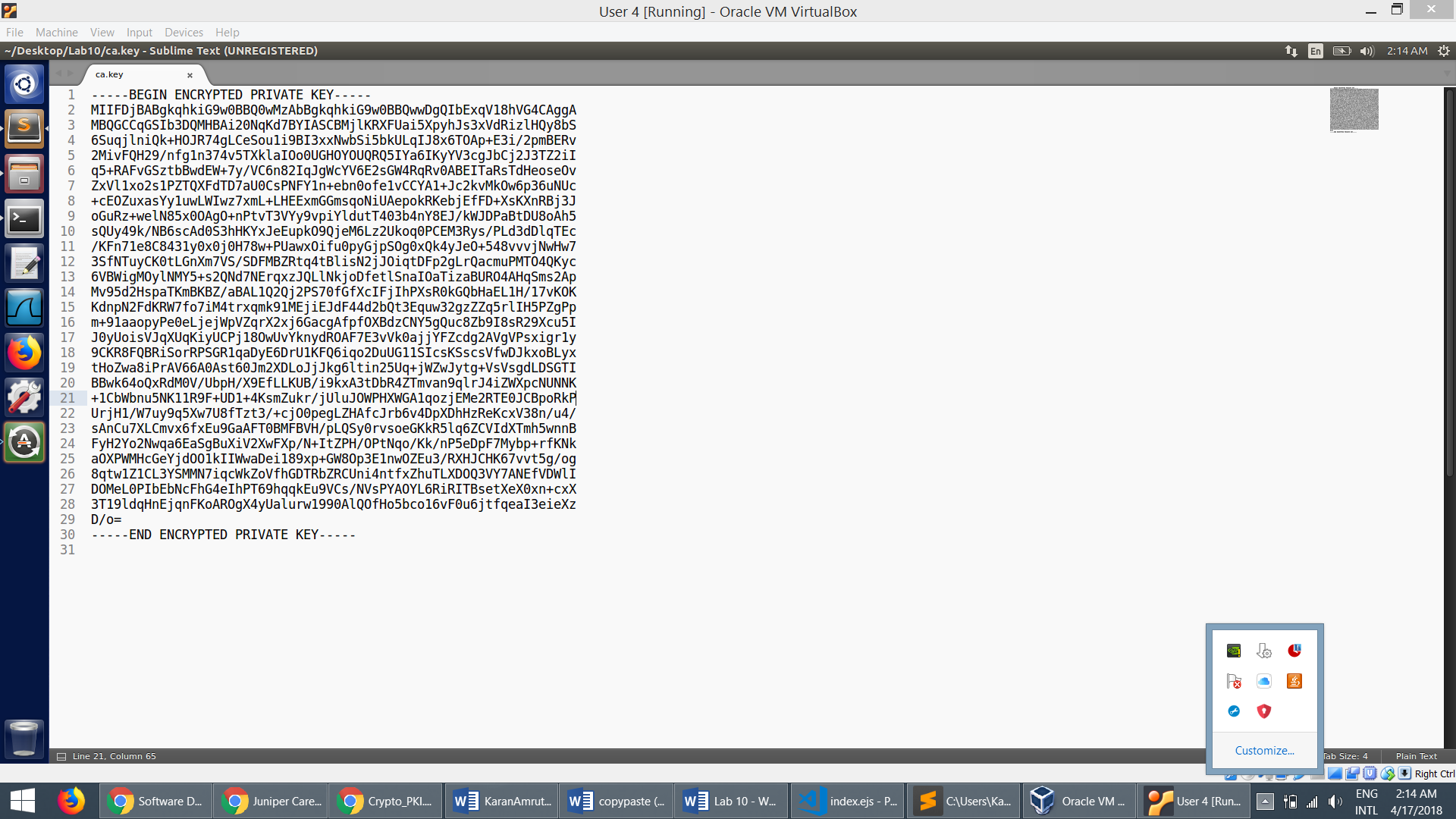
* We setup the following folders and sub-folders:



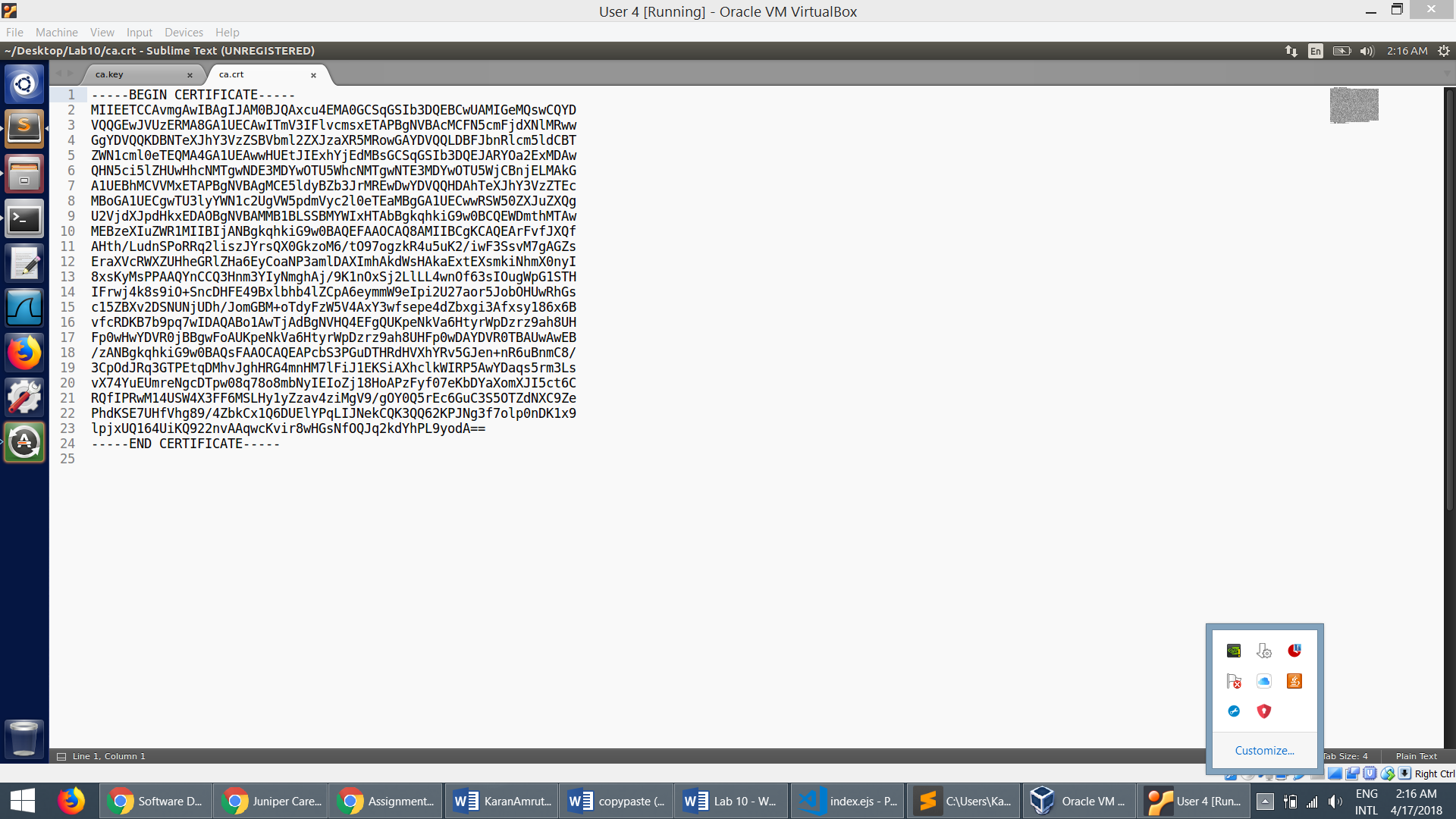
* We generate a self-signed certificate for our CA. This means that this CA is totally trusted, and its certificate will serve as the root certificate. The output of the command are stored in two files: ca.key and ca.crt. The file ca.key contains the CA’s private key, while ca.crt contains the public-key certificate



* The private key:



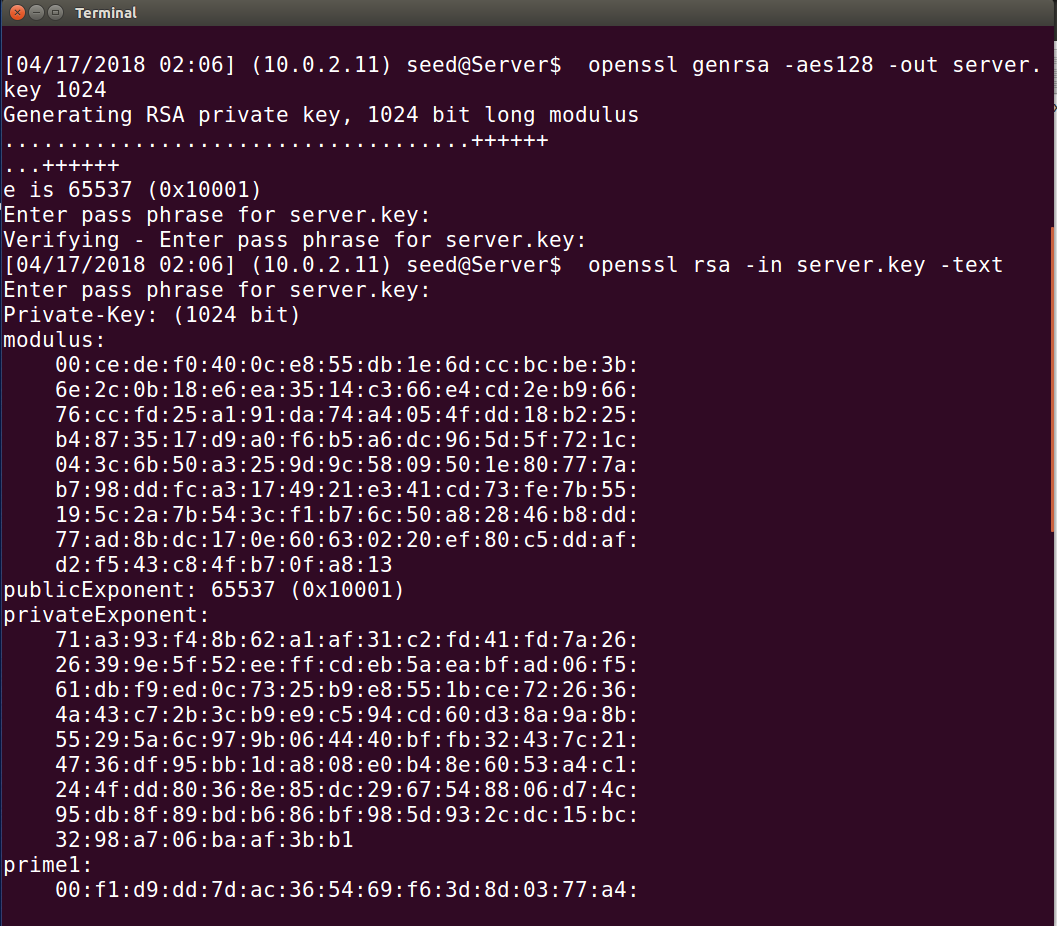
* The public key:

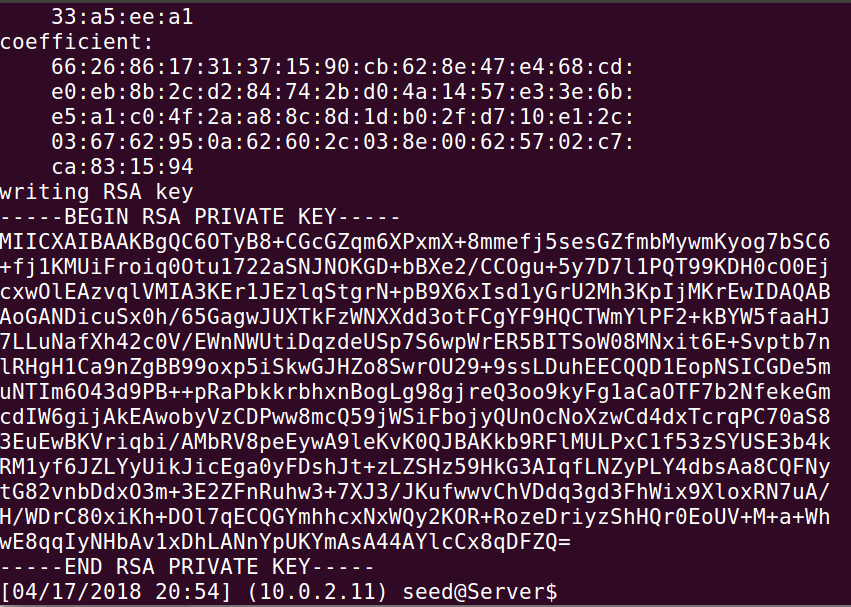


**Task 2: Creating a Certiﬁcate for SEEDPKILab2018.com**

**Step 1: Generate public/private key pair.**

* We can run the following command to generate an RSA key pair (both private and public keys). We also provide a password to encrypt the private key. The keys will be stored in the ﬁle *server.key*
* We run the command *openssl rsa -in server.key –text* to view the contents of the file.





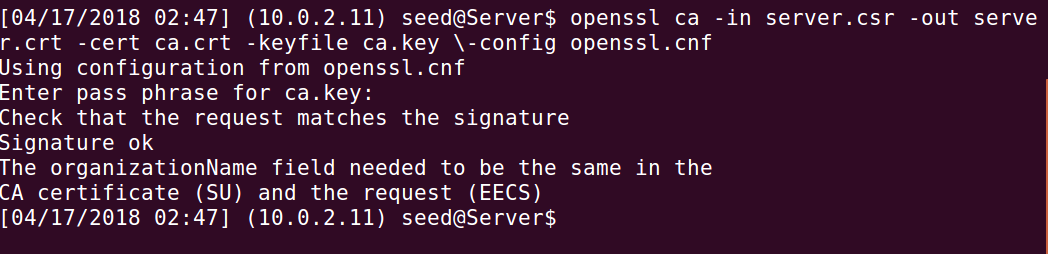
**Step 2: Generate a Certificate Signing Request (CSR)**

* Now we have the key ﬁle, so we generate a Certiﬁcate Signing Request (CSR), which basically includes the company’s public key. The CSR will be sent to the CA, who will generate a certiﬁcate for the key.

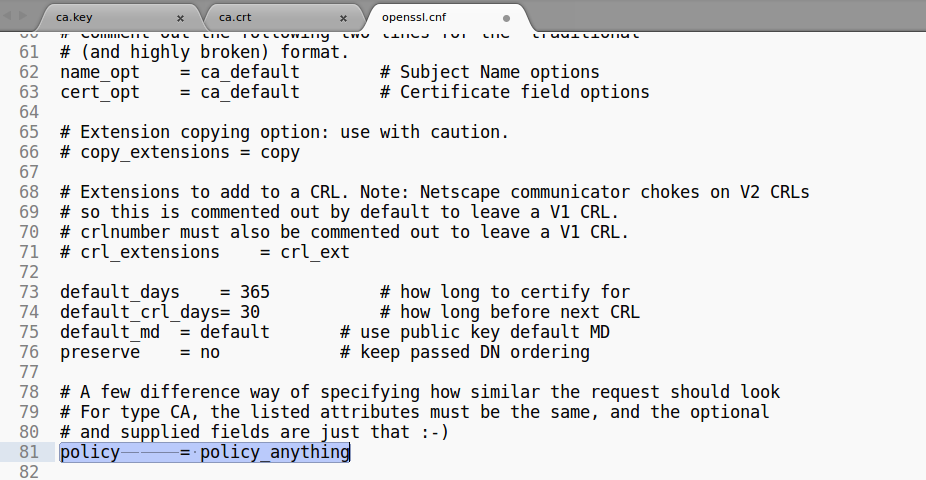


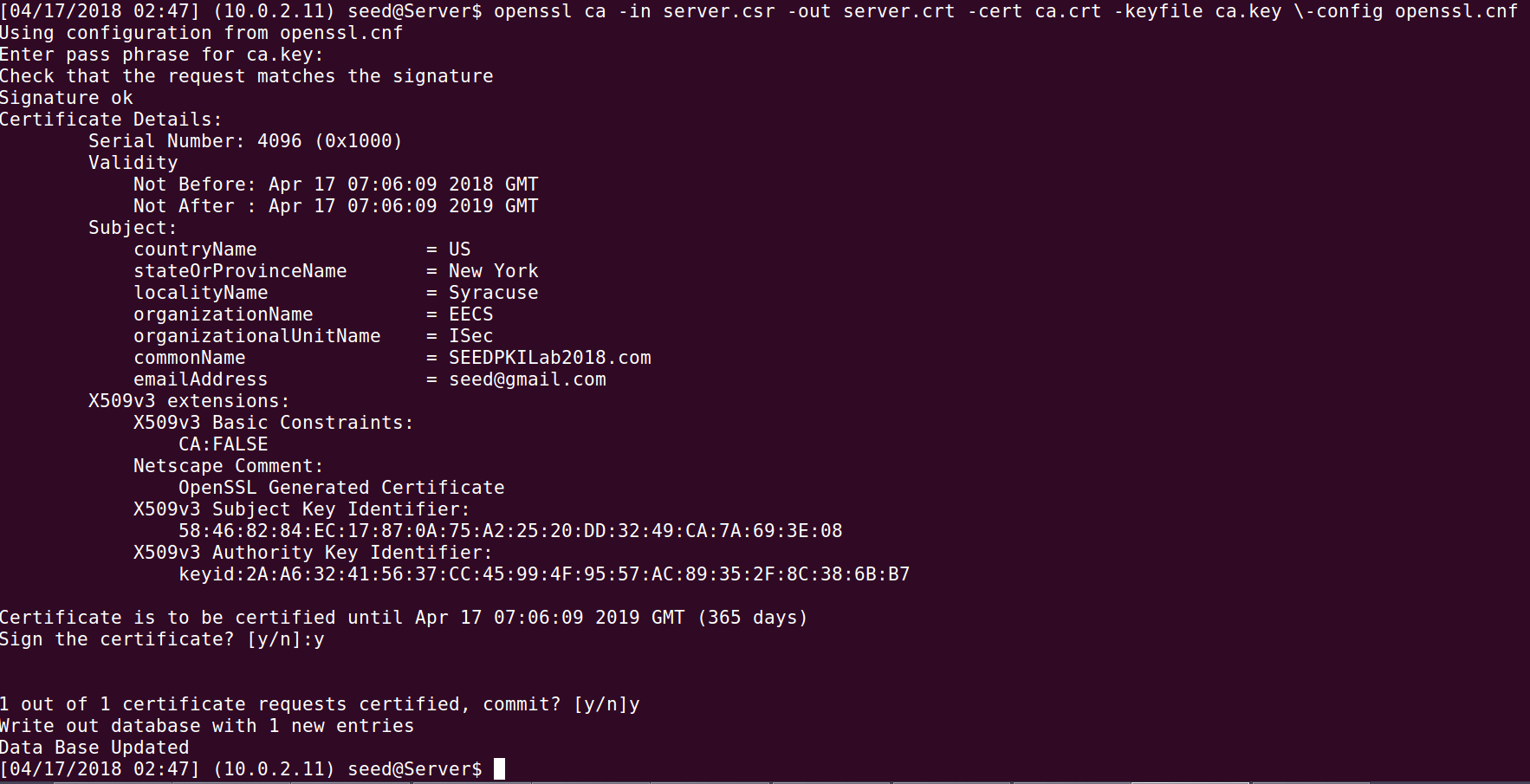
**Step 3: Generating Certificates:**

* The CSR file needs to have the CA’s signature to form a certificate. The following command turns the certificate signing request (server.csr) into an X509 certificate (server.crt), using the CA’s ca.crt and ca.key:



* But the OpenSSL refuses to generate certificates as the names in your requests do not match with those of CA
* So we make change the attribute, policy in the config file and run the command again.

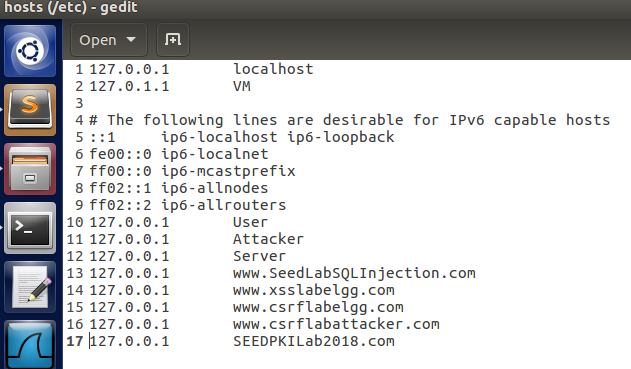




**Task 3: Deploying Certificate in an HTTPS Web Server**

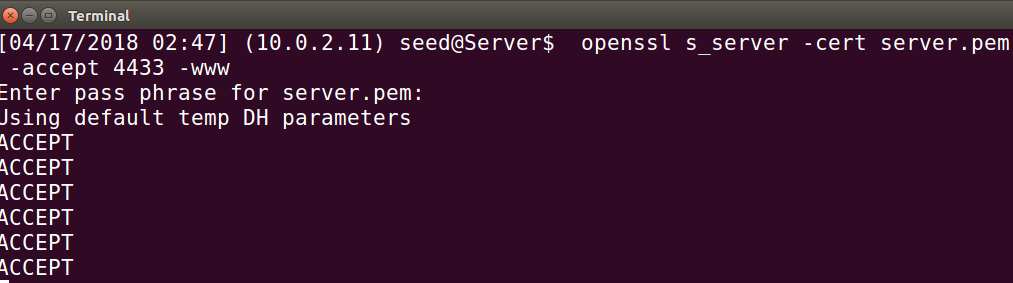
**Step 1: Configuring DNS**:

* We choose SEEDPKILab2018.com as the name of our website and add it to /etc/hosts

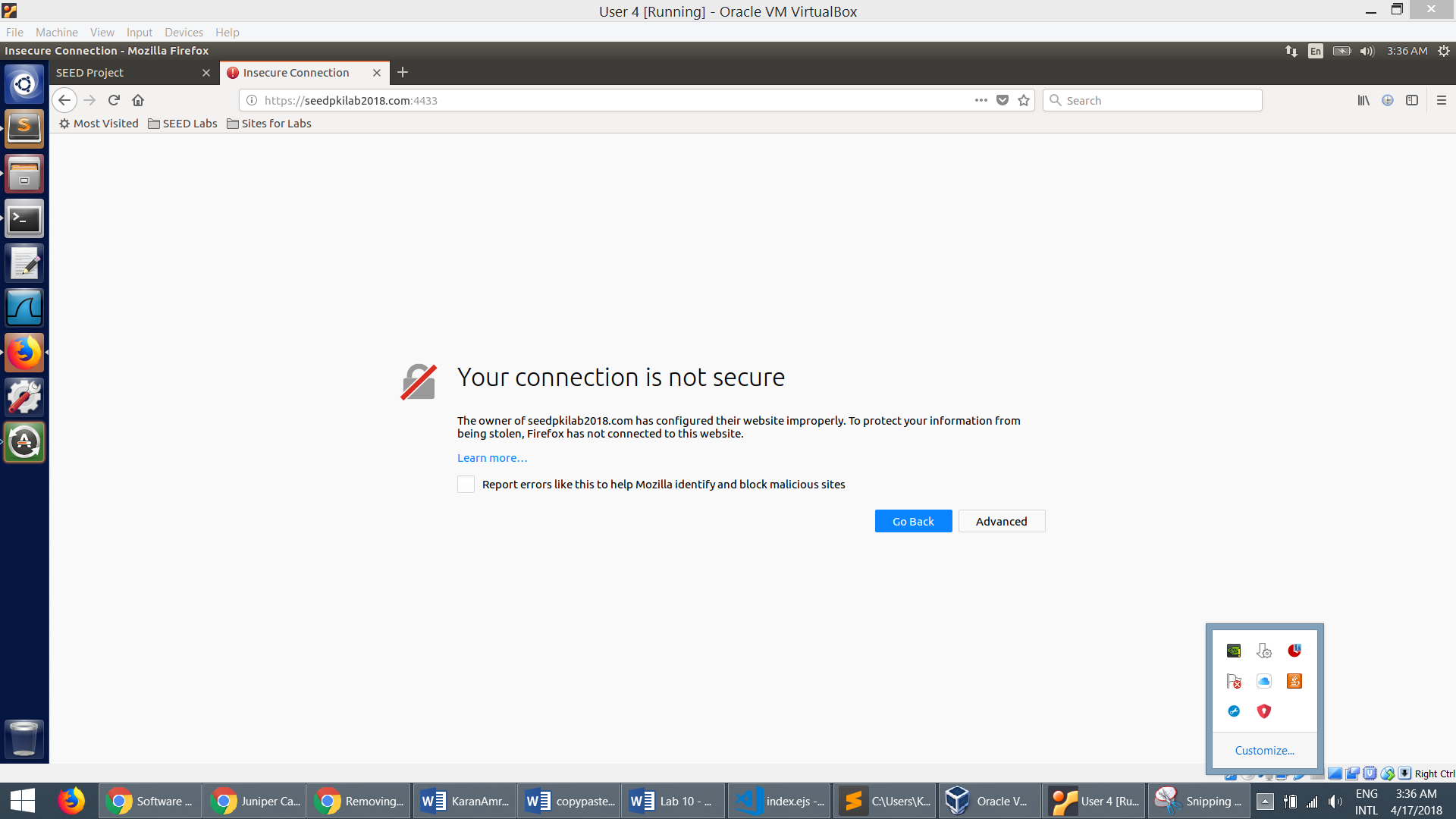


**Step 2: Configuring the web server:**

* We combine the secret key and certificate into one file
* We use the s\_server command OpenSSSL to start a simple web server:



* When we try access the following URL: https://SEEDPKILab2018.com: 4433/ , we get an error on the browser that indicates that the certiﬁcate is not trusted

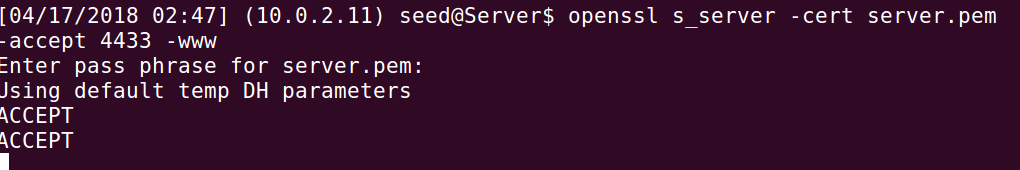


**Step 3: Getting the browser to accept our CA certificate.**

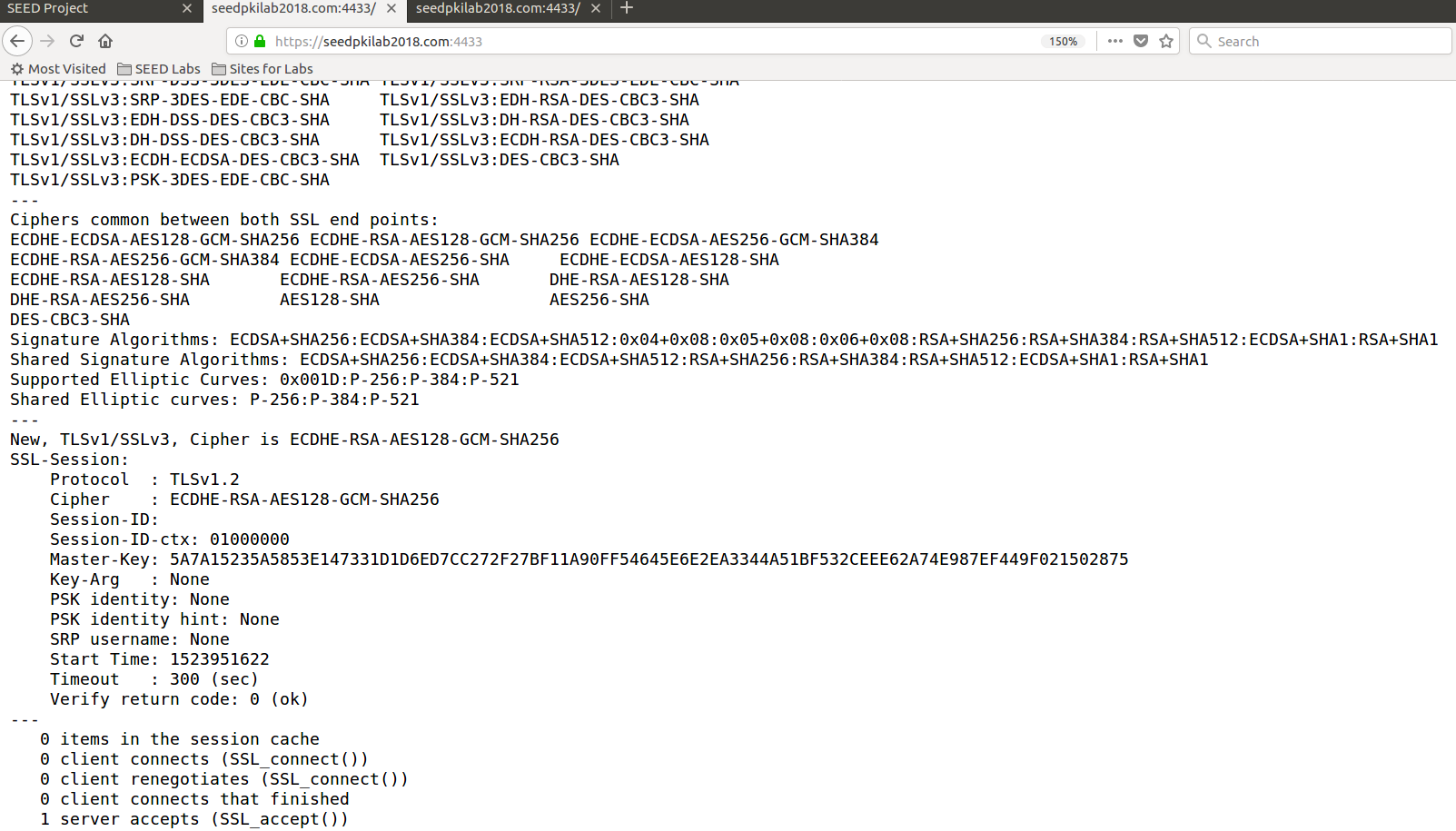
* We manually add our CA’s certificate to the Firefox browser and select the option: “Trust this CA to identify web sites”.



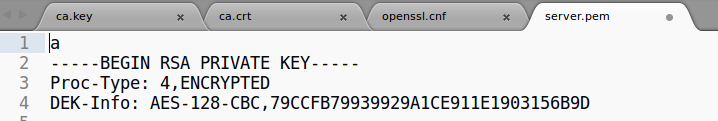
**Step 4. Testing our HTTPS website:**

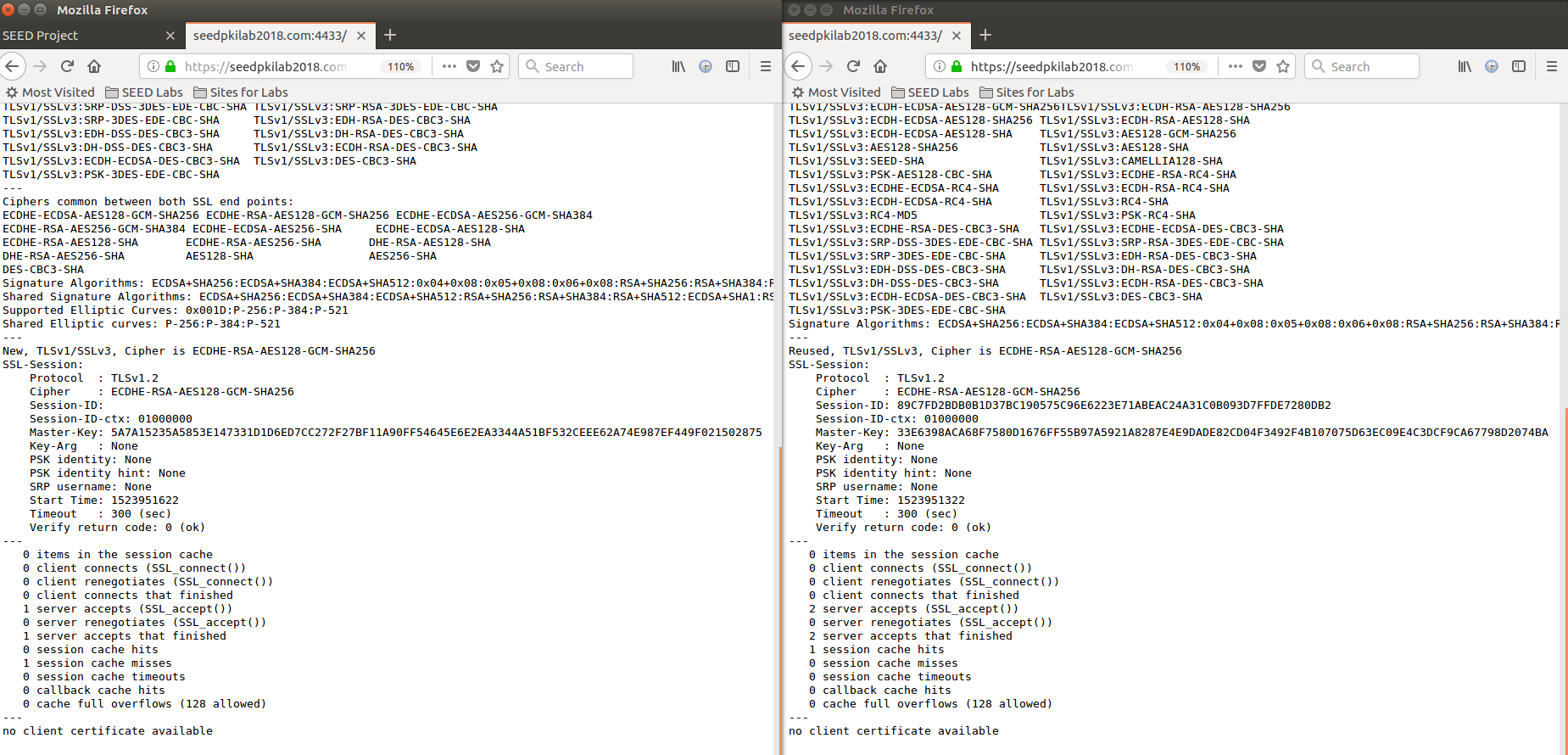


* After importing the certificate, we can see that when we try to visit <https://seedpkilabs2018.com:4433>, the browser doesn’t show any error message and we get a reply from the server.

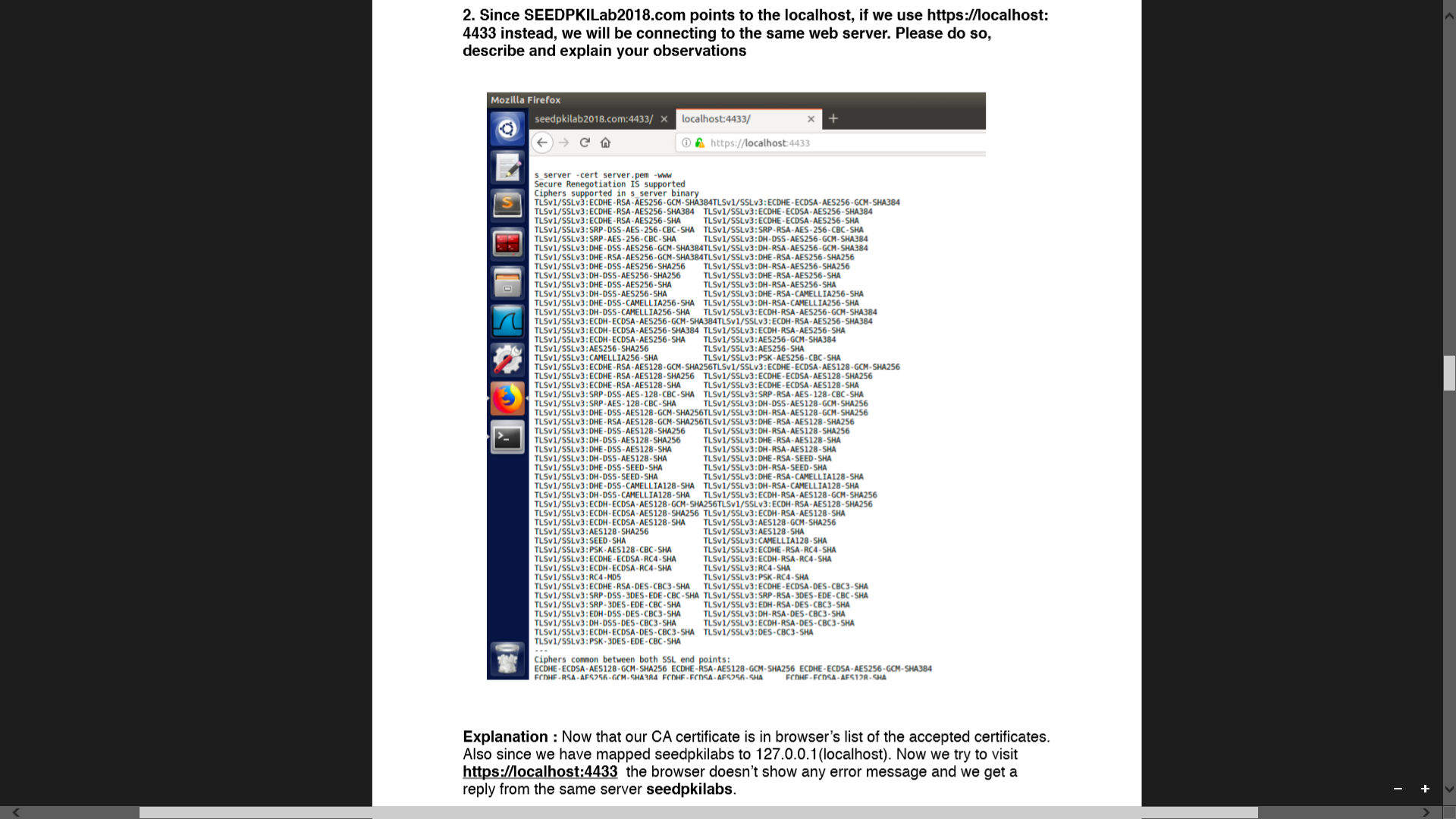


* Changing one byte in the server.pem file:



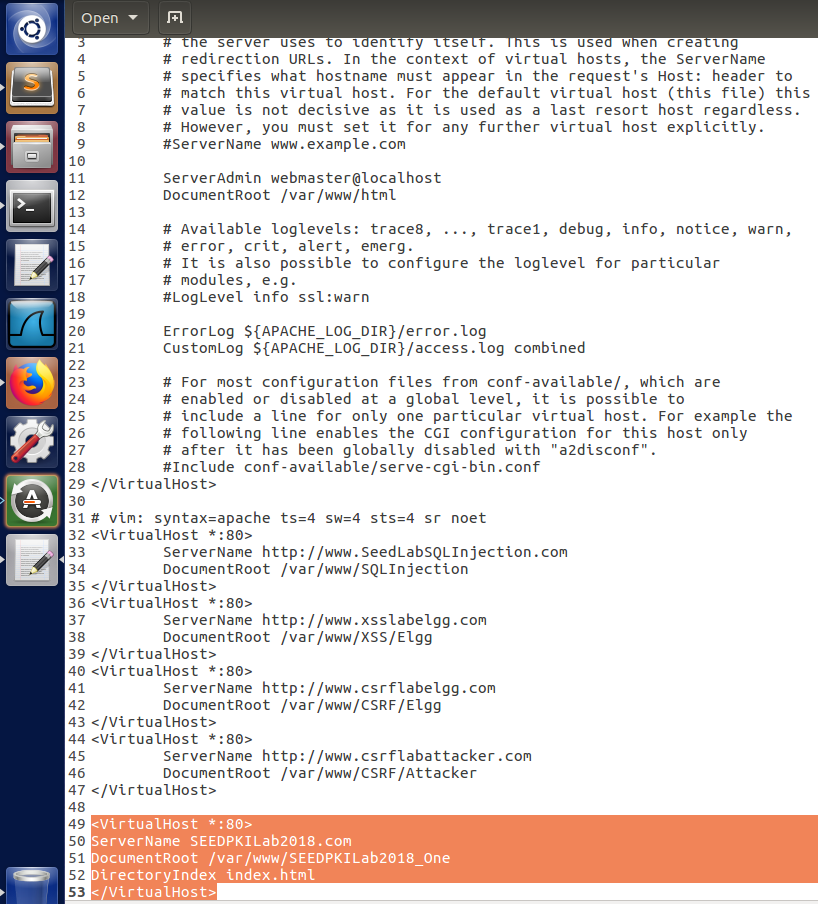


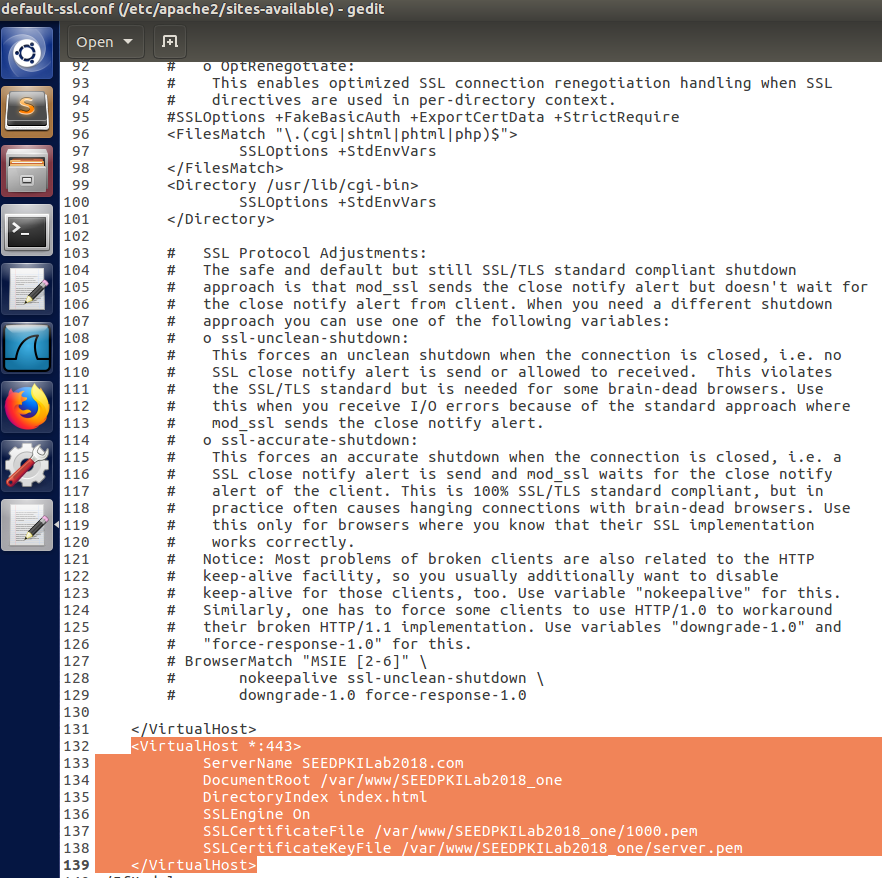
* The actual PEM-format certiﬁcate is the lines from ----BEGIN---- to ----END----- only; everything else is comment data intended to be possibly helpful to a human reader, but ignored by programs. So any change made outside doesn’t affect SSL connection.
* Since we have mapped seedpkilabs to the localhost, 127.0.0.1. Now we try to visit https://localhost:4433 the browser doesn’t show any error message and we get a reply from the same server seedpkilabs:



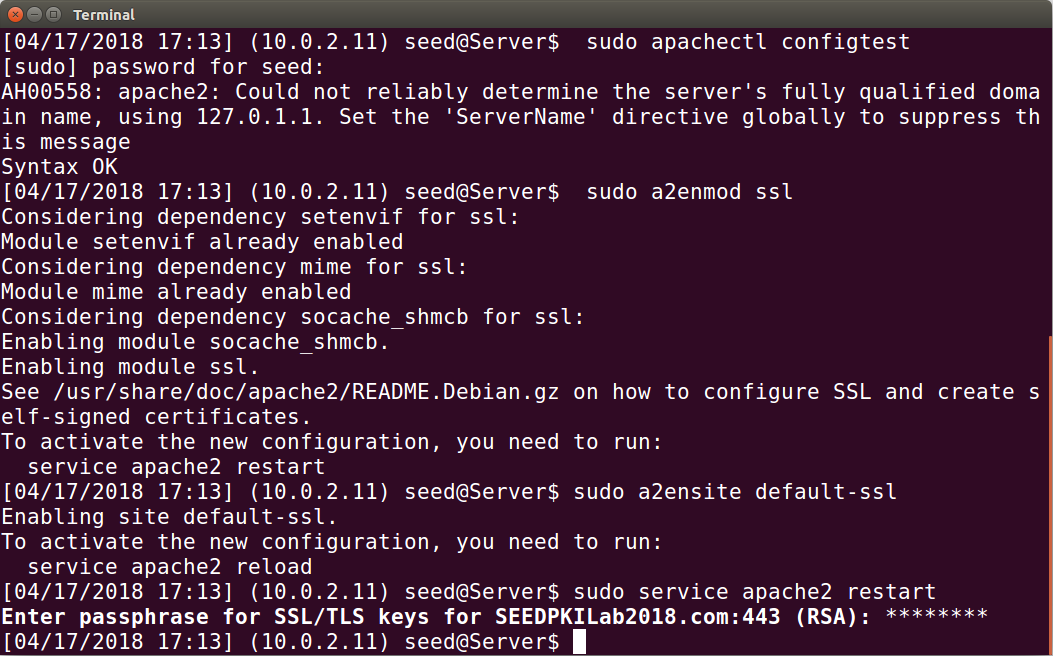
**Task 4: Deploying Certificate in an Apache-Based HTTPS Website**

* We add necessary conﬁguration to */etc/apache2/sites-available/000default.conf* and */etc/apache2/sites-available/defaultssl.conf*. We have copied certiﬁcate and private key to /var/www/PKI and index.html is the page that will be rendered when we access <https://seedpkilabs.com>

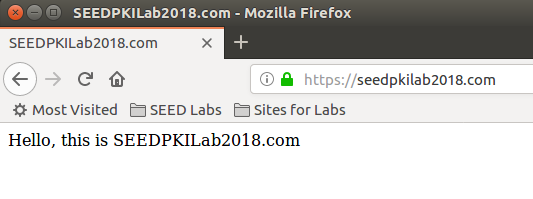




* We run a series of commands to verify the conﬁguration and resolve any errors. The reload the SSL conﬁguration and restart the Apache2 server. Apache will ask us to type the password used for encrypting the private key



* Once everything is set up properly, we can browse the web site, and all the trafﬁc between the browser and the server will be encrypted. It renders index.html:



**Task 5: Launching a Man-In-The-Middle Attack**

* We use news9.com as the malicious website that tries to intercept the trafﬁc between the user and seedpkilabs.com First we add news9.com with ip 127.0.0.1 (localhost) to /etc/hosts ﬁle
* We add necessary conﬁguration to */etc/apache2/sites-available/000default.conf* and */etc/apache2/sites-available/defaultssl.conf*. We have copied certiﬁcate and private key to /var/www/PKI and index.html. We use same key and certiﬁcates of seedpkilabs.com for news9.com so that we intercept the data. All the conﬁgurations will be same except for the server name.

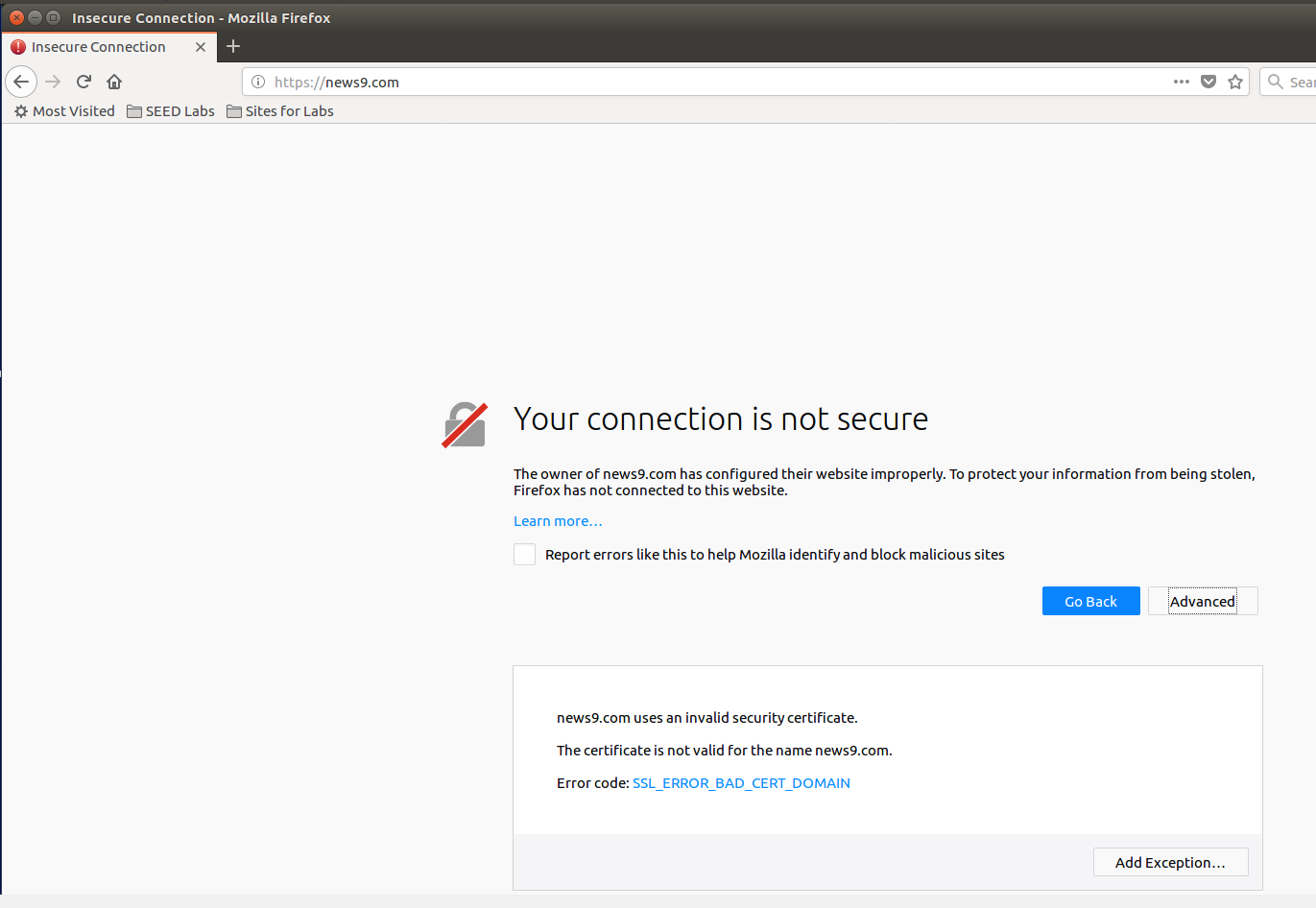




* We run a series of commands to verify the conﬁguration and resolve any errors. The reload the SSL conﬁguration and restart the Apache2 server. Apache will ask us to type the password used for encrypting the private key

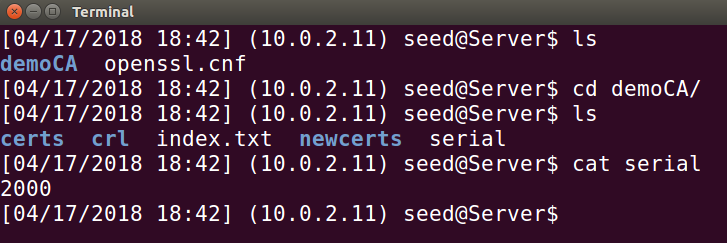


* The attacker has sent his legitimate certiﬁcate to the user. However the subject on the certiﬁcate cannot be seedpkilabs.com because the attacker does not own seedpkilabs.com. So he puts his own identity news9.com in the subject ﬁeld.
* Once the user’s browser receives the certiﬁcate, the certiﬁcate will be veriﬁed by the browser using the trusted certiﬁcates that have already been pre-installed on the browser.
* This validation will pass, because the attacker’s certiﬁcate is valid but there is one more validation. The browser needs to know whether the domain name put inside the subject ﬁeld of the certiﬁcate is same as the user’s intent.
* The browser needs to know whether the domain name put inside the subject ﬁeld of the certiﬁcate is same as the user’s intent. We have typed news9.com in the browser and we get the following error:

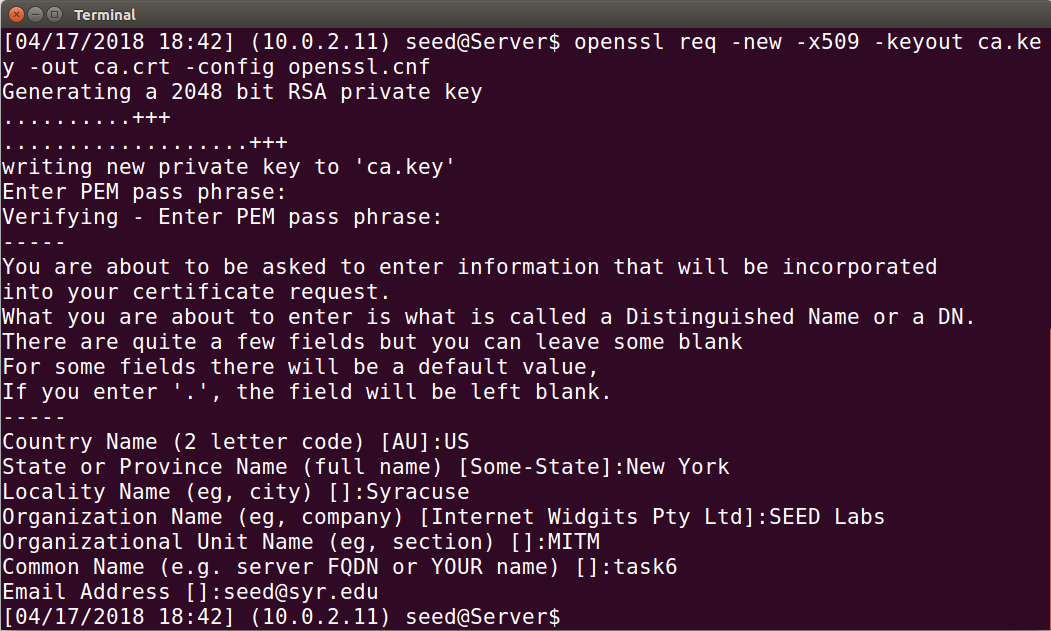


**Task 6: Launching a Man-In-The-Middle Attack with a Compromised CA**

* Showing the contents of the directory and the files:



* We can run the folowing command to generate an RSA key pair (both private and public keys). We also provide a password to encrypt the private key. The keys will be stored in the ﬁle malserver.key.

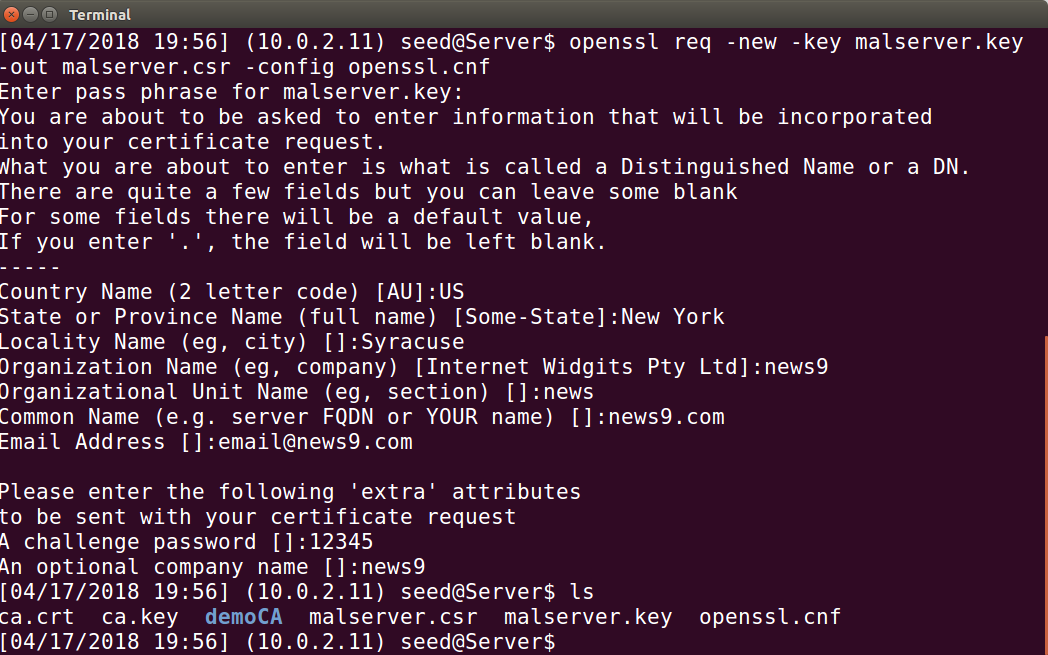


* Contents of malserver.key:

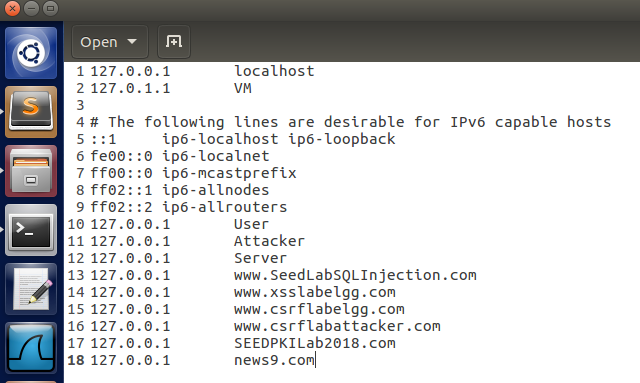




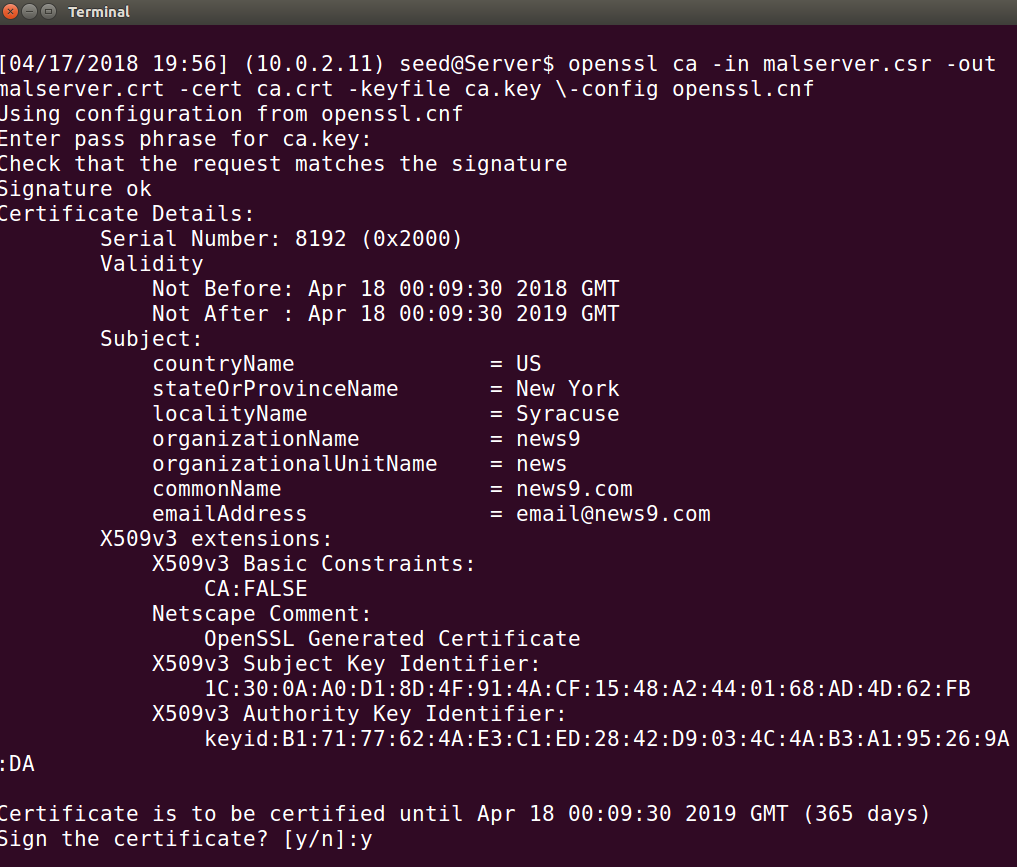
* Now we have the key ﬁle, so we generate a Certiﬁcate Signing Request (CSR), which basically includes the company’s public key. The CSR will be sent to the CA, who will generate a certiﬁcate for the key.



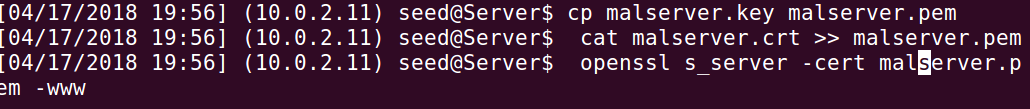
* we add news9.com with ip *127.0.0.1 to /etc/hosts*



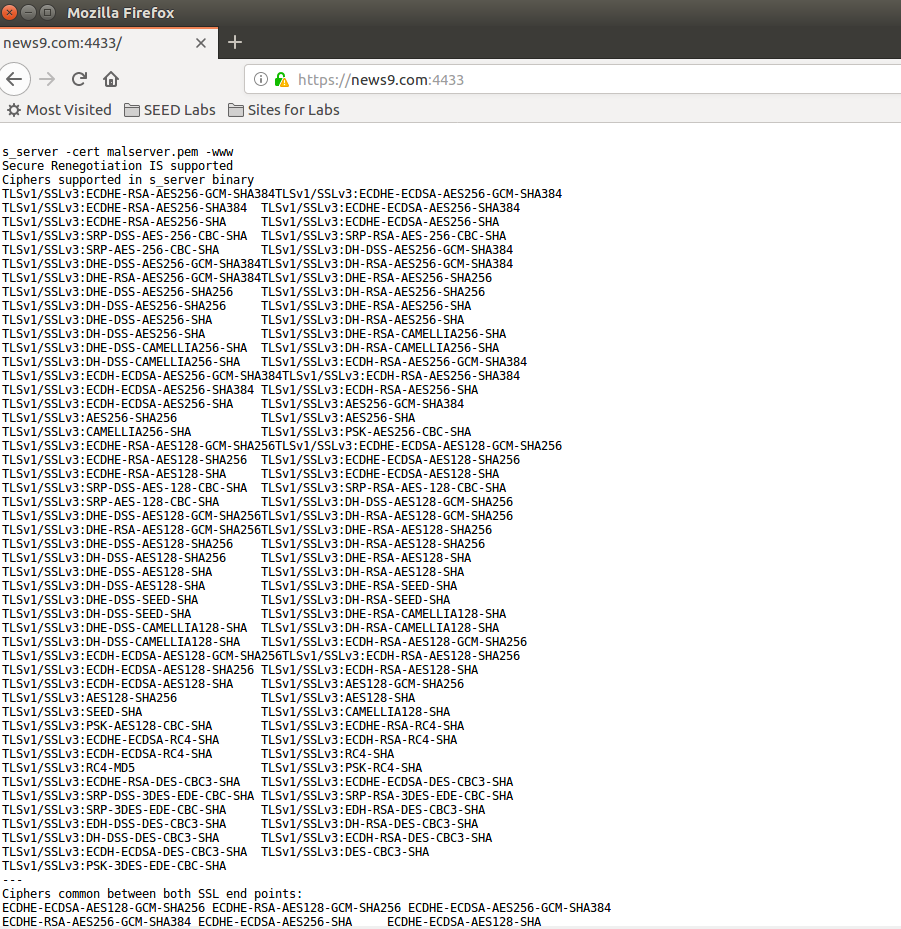
* The CSR file needs to have the CA’s signature to form a certificate.We will use our own trusted CA to generate certificates.
* The following command turns the certificate signing request (malserver.csr) into an X509 certificate (malserver.crt), using the CA’s ca.crt and ca.key.



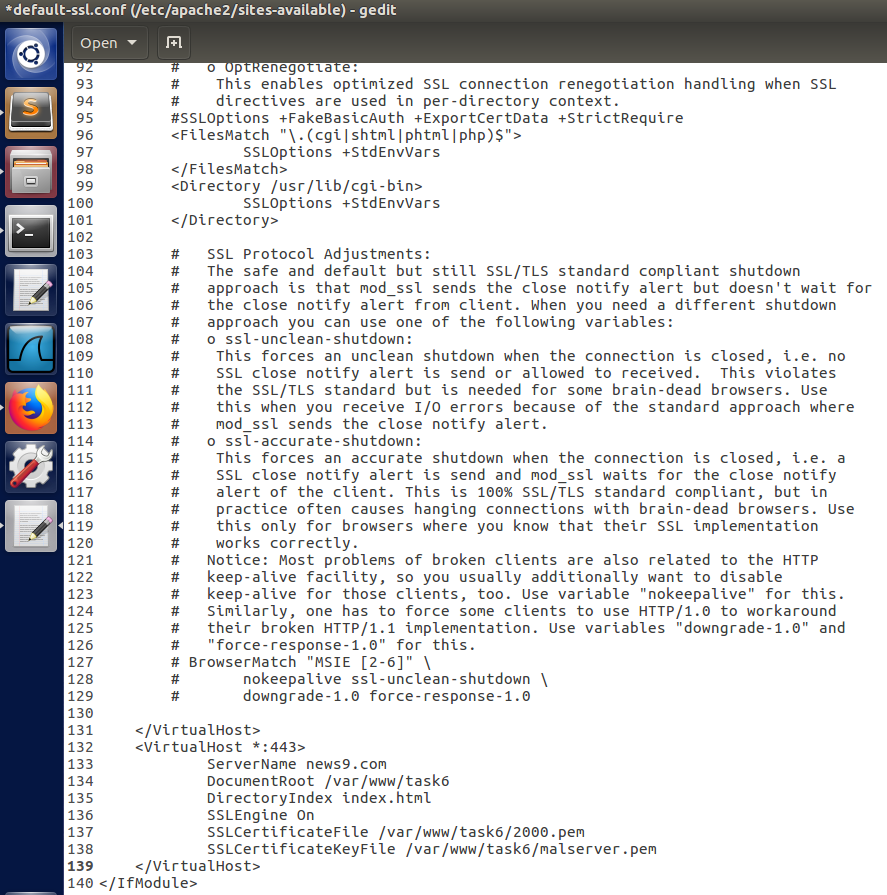
* We combine secret key and certiﬁcate into one ﬁle and start the server:

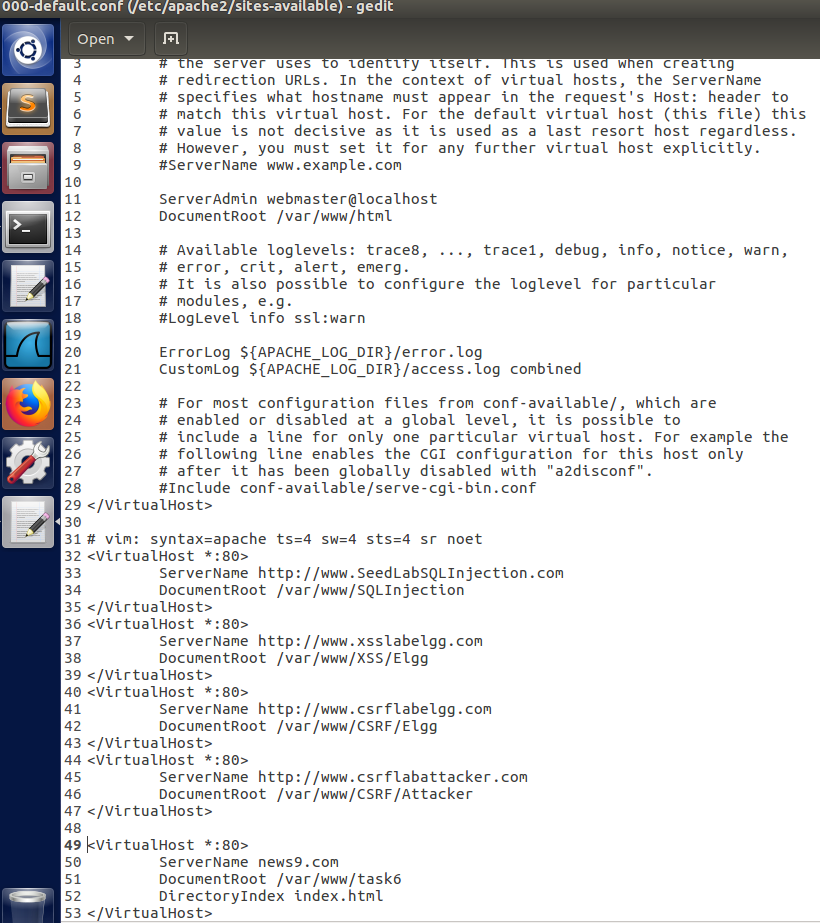


* Now that our CA certiﬁcate is in browser’s list of the accepted certiﬁcates. Now we try to visit https://news9.com the browser doesn’t show any error message and we get a reply from the server.

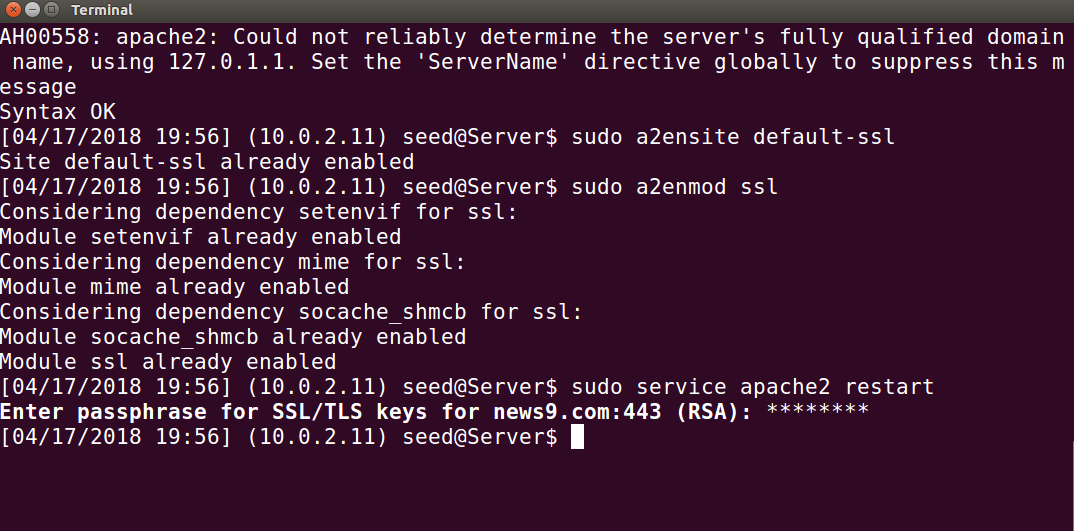


* We add necessary conﬁguration to */etc/apache2/sites-available/000default.conf* and */etc/apache2/sites-available/defaultssl.conf*.





* We run a series of commands to verify the conﬁguration and resolve any errors. The reload the SSL conﬁguration and restart the Apache2 server. Apache will ask us to type the password used for encrypting the private key



* Once everything is set up properly, we can browse the web site, and all the trafﬁc between the browser and the server will be encrypted. It renders index.html

