**IS LAB ASSIGNMENT#2**

**Name: Saman Khan**

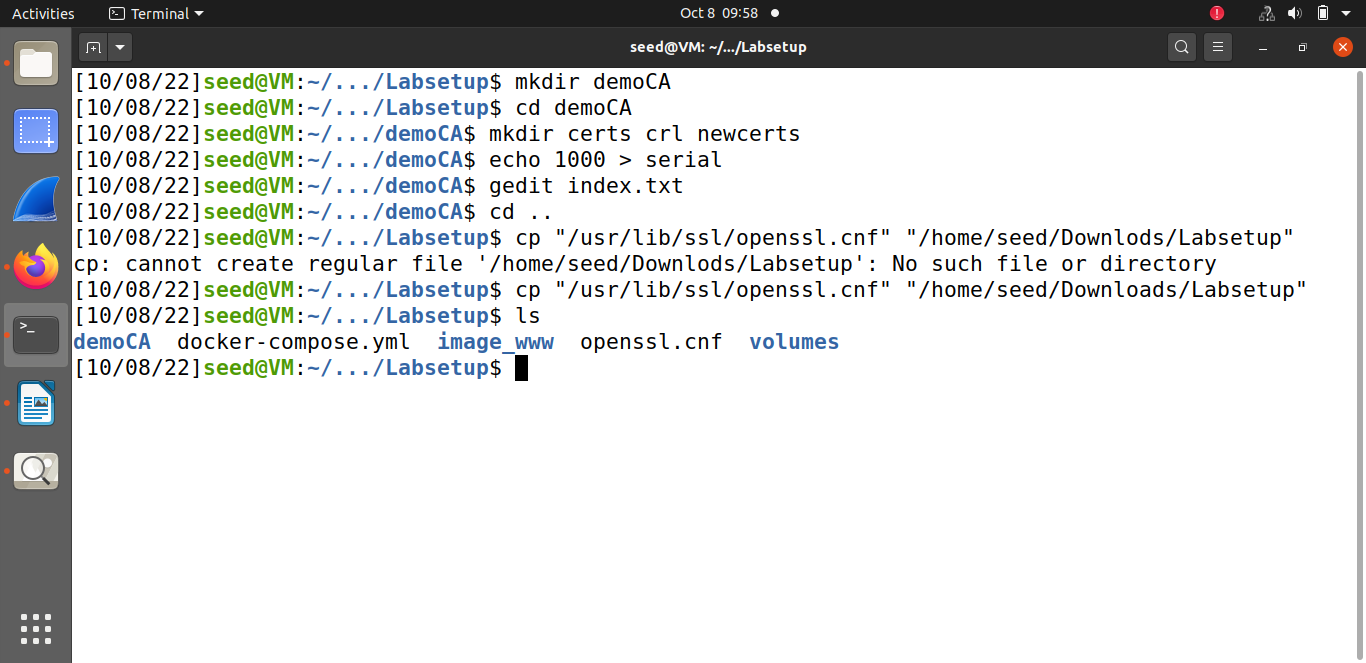
**ID: 19K-0354**

**Section: H**

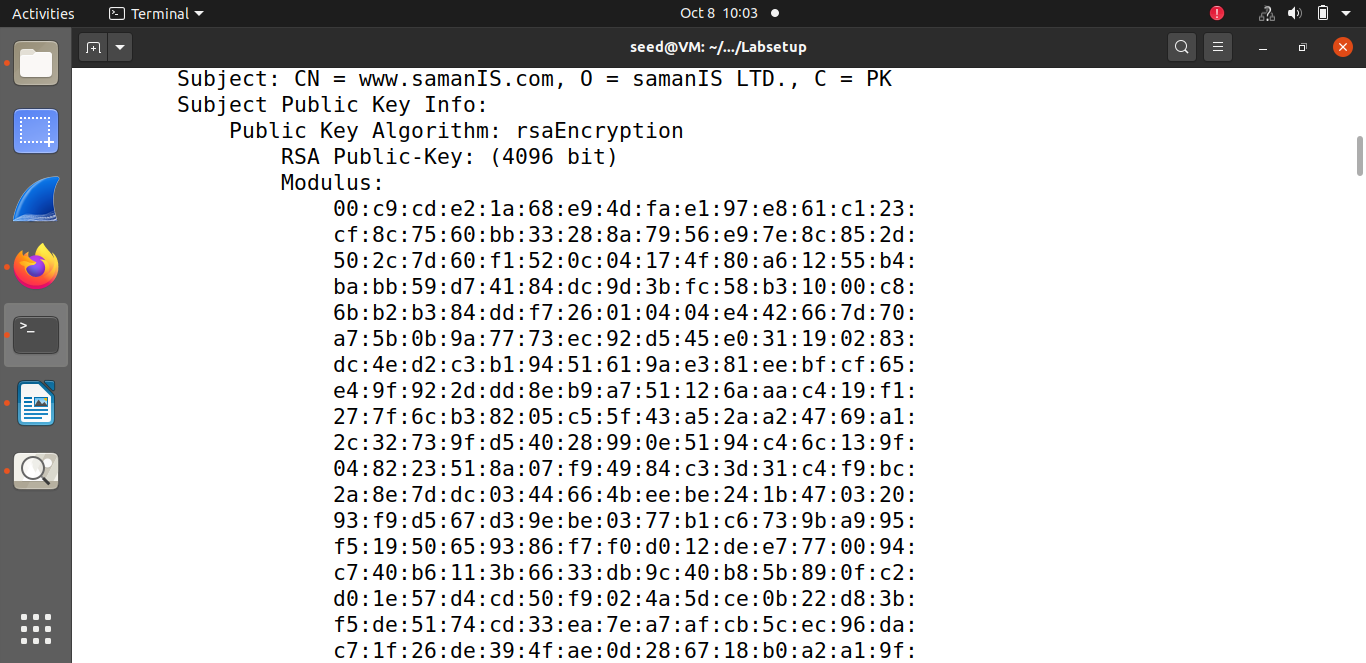
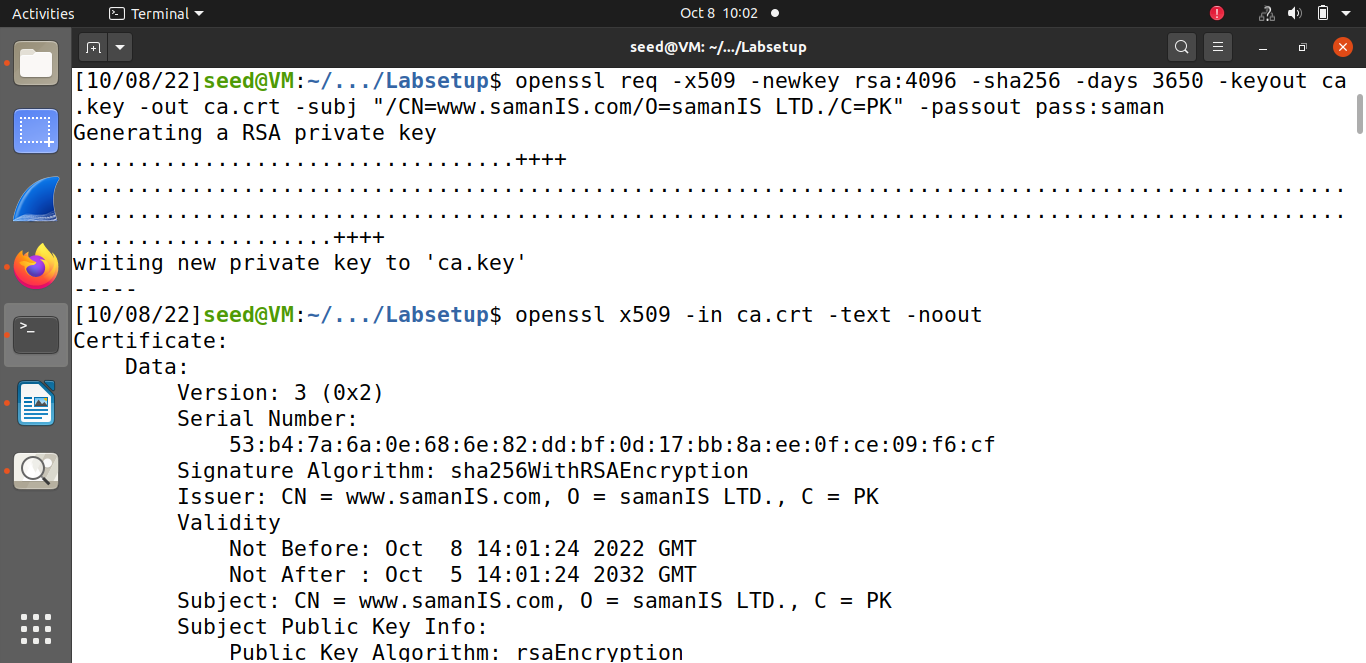
**TASK#01: Becoming a Certificate Authority (CA)**

By default, OpenSSL use the configuration file from /usr/lib/ssl/openssl.cnf. Since we need

to make changes to this file, we will copy it into our current directory which is demoCA, and instruct OpenSSL to use this copy instead. The index.txt file is simply an empty file whereas the serial file has a single number in string format that is 1000. The configuration of openssl.cnf is necessary to create and issue certificates.

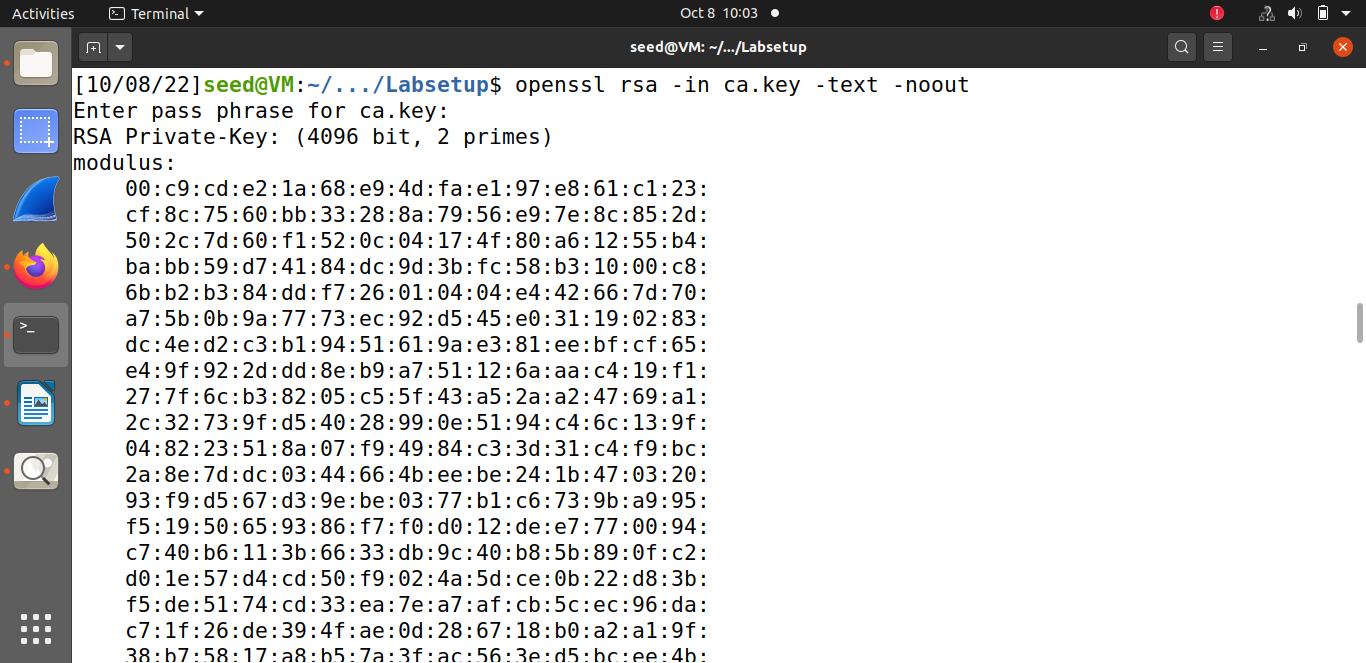
****

Using the following command to generate the self-signed certificate for the CA where the common name is [www.samanIS.com](http://www.samanIS.com), country is PK (Pakistan), organization is samanIS LTD., and password of the certificate is saman.

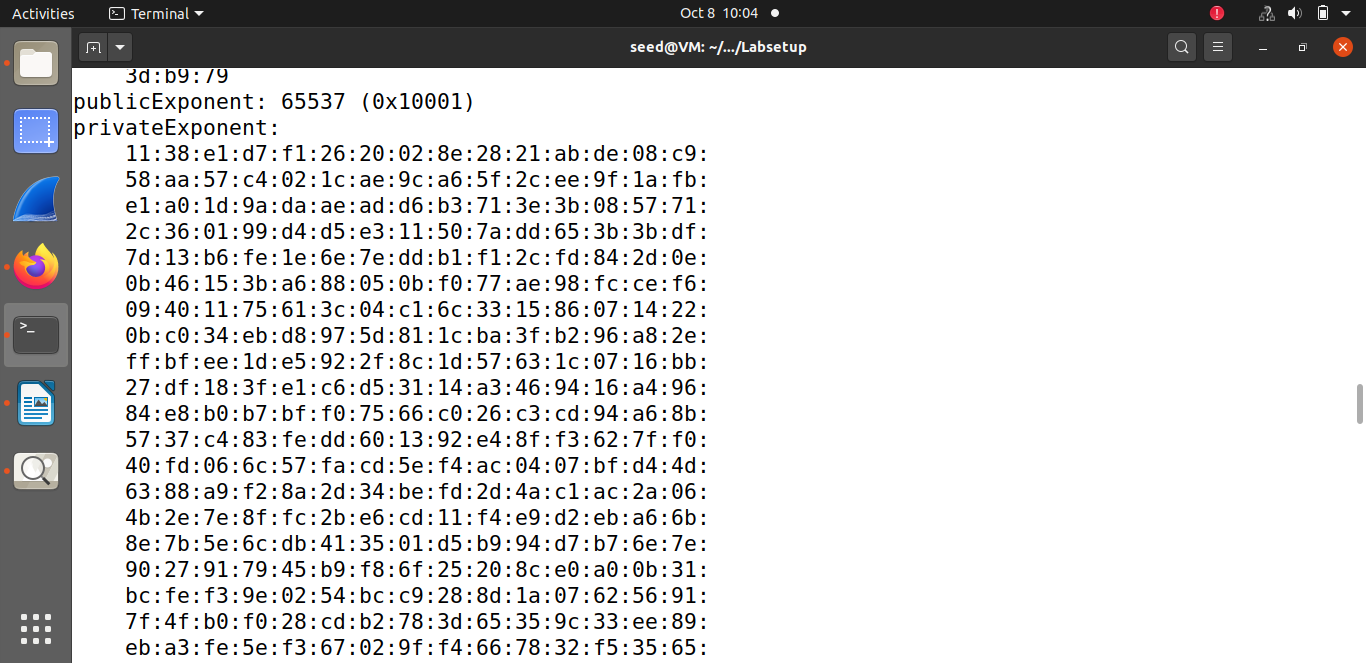
****

The screenshots above contain the contents of the CA certificate. Since the output is ca.crt and not .csr it indicates that t is a CA certificate because we’re not generating any request. From the screenshots above it can be seen that issuer and the subject in the certificate is same, it shows that it is a self-signed certificate. The screenshots below show the contents of the key that was generated.

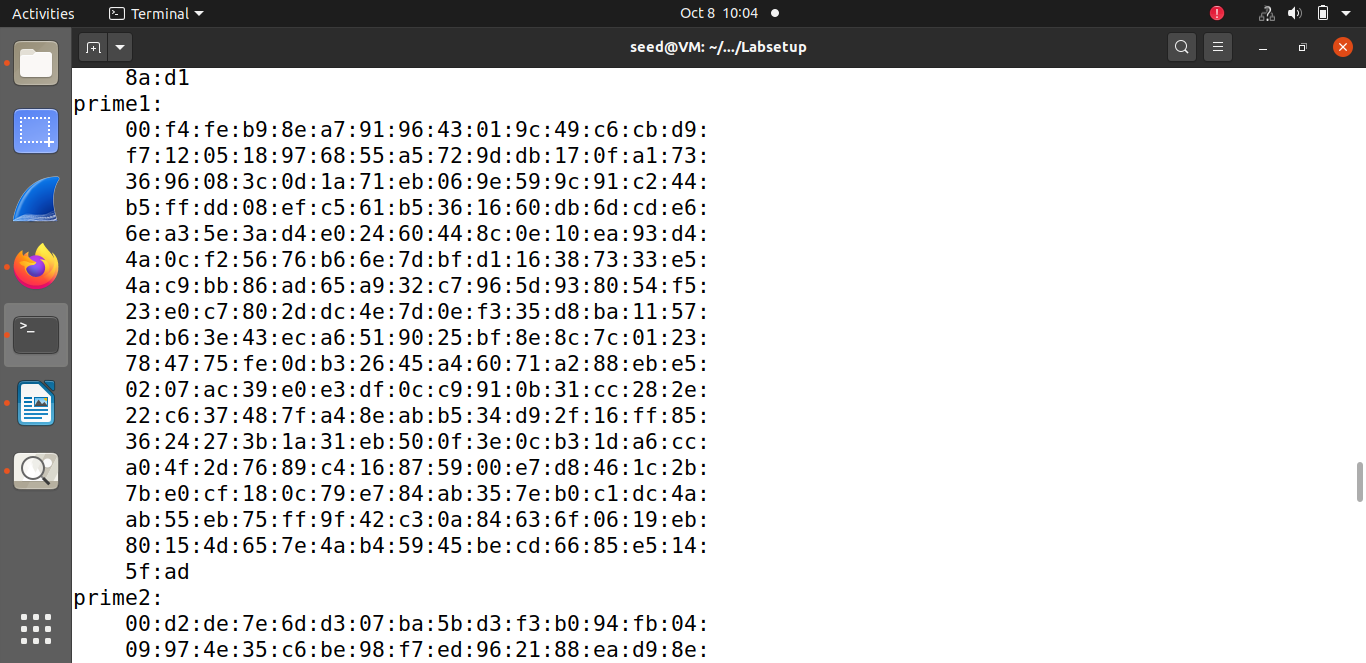
* **Modulus ‘n’:**

****

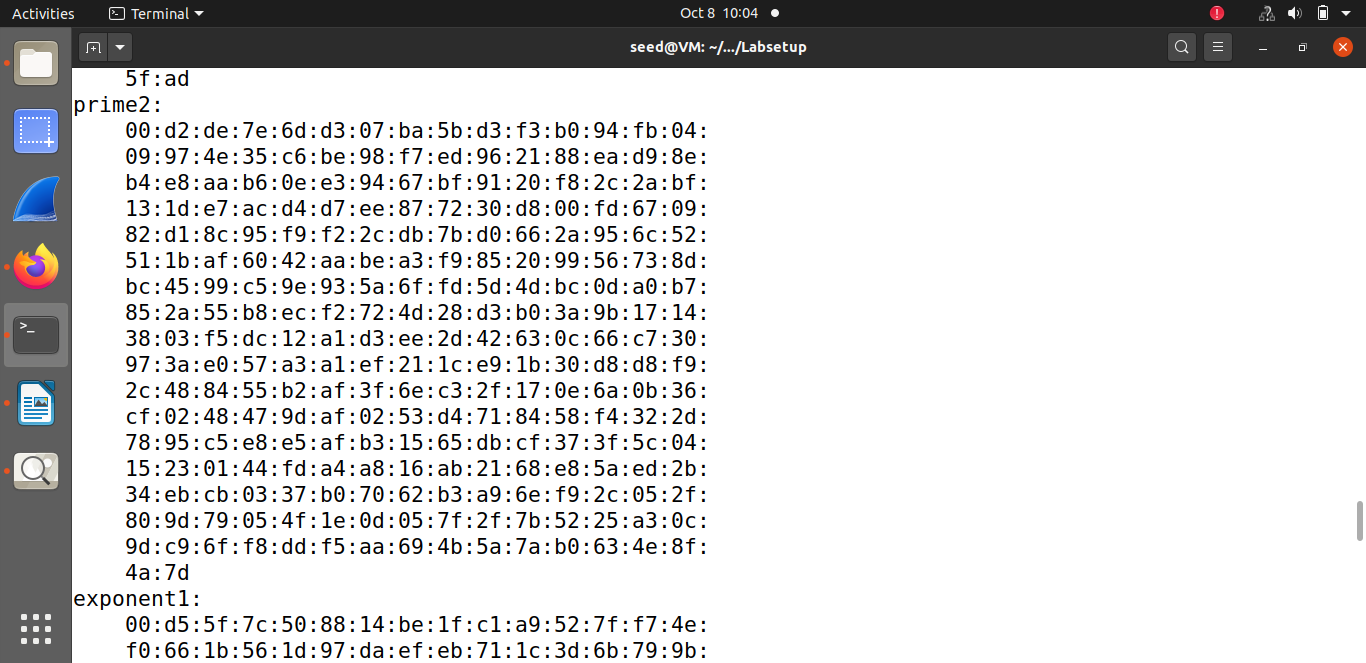
* **Public Exponent ‘e’ & Private Exponent ‘d’:**

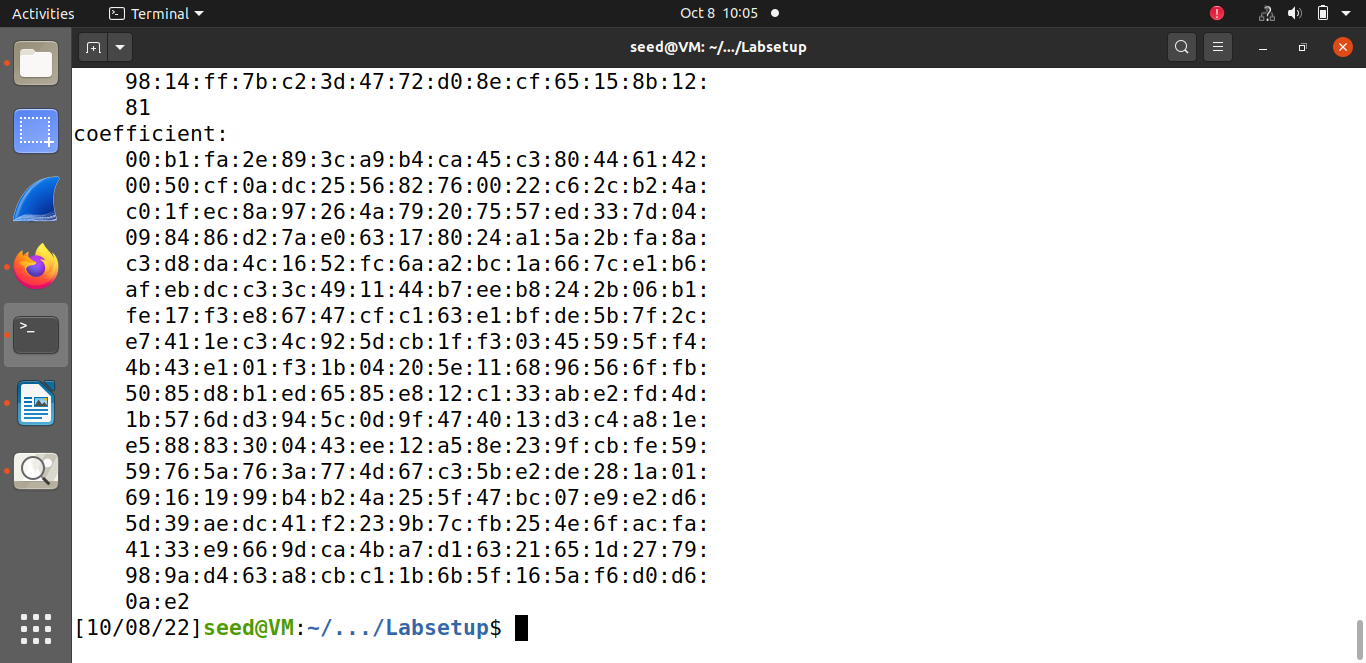
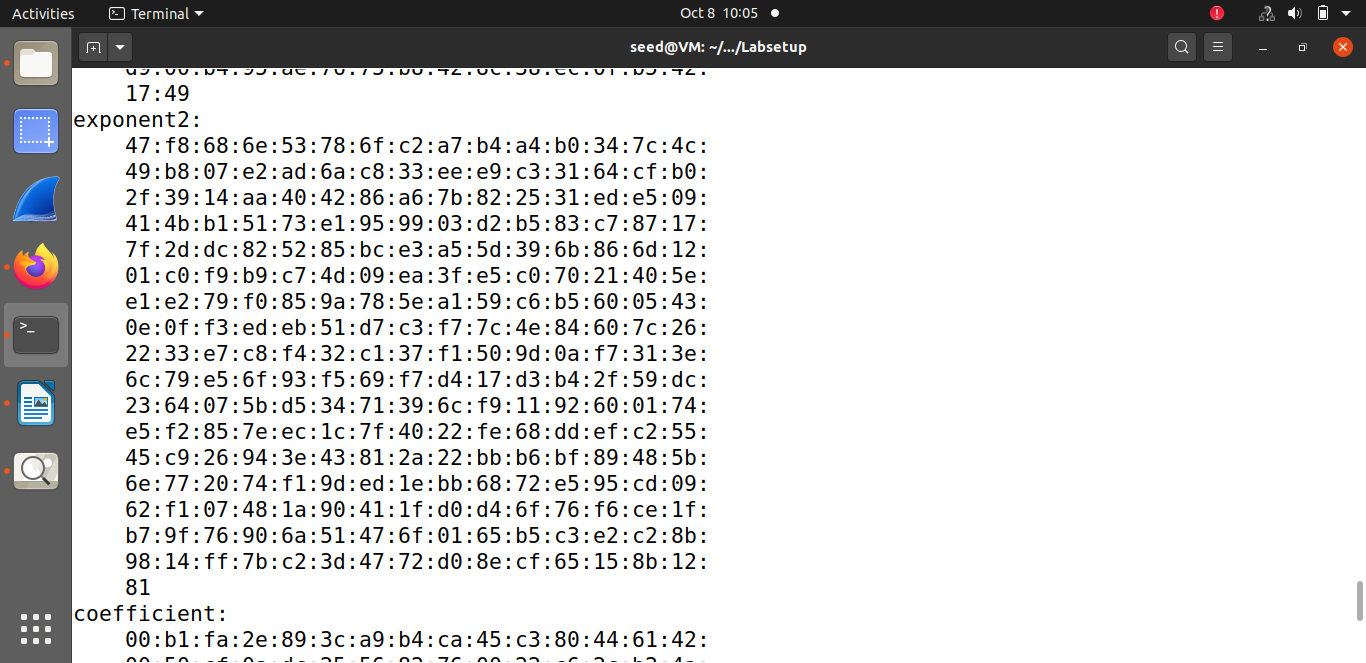
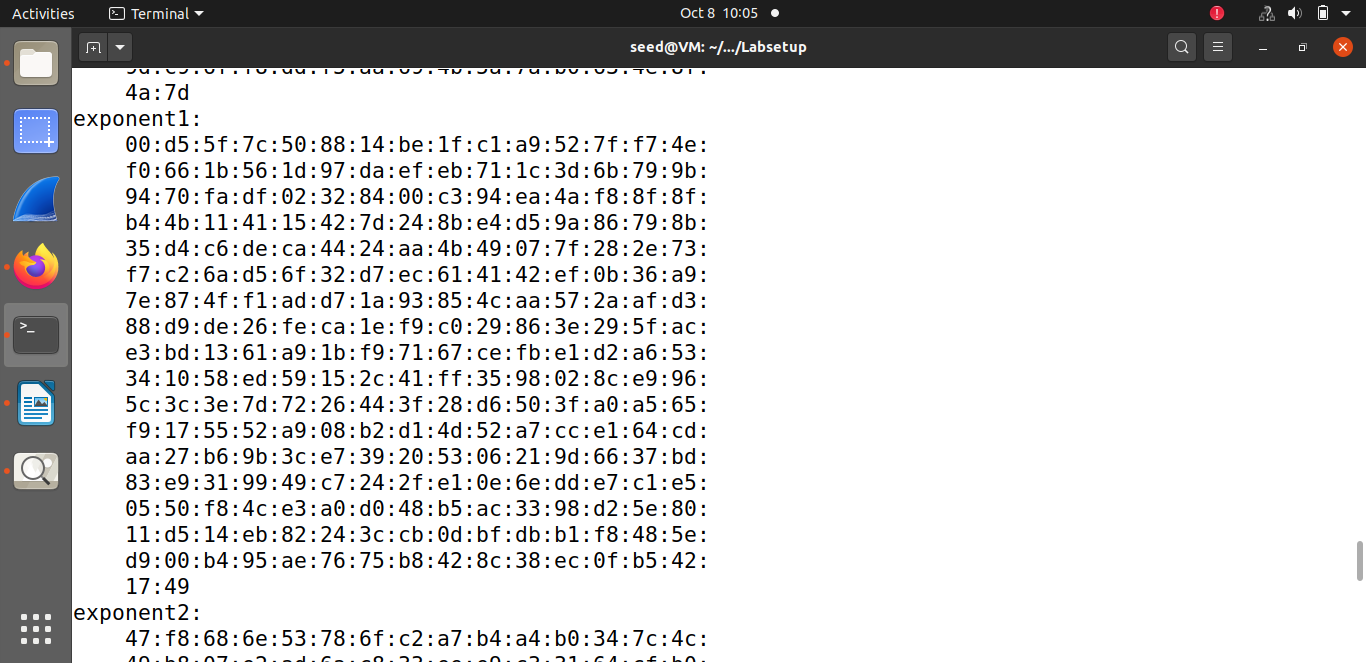
****

* **Prime#01 ‘p’:**

****

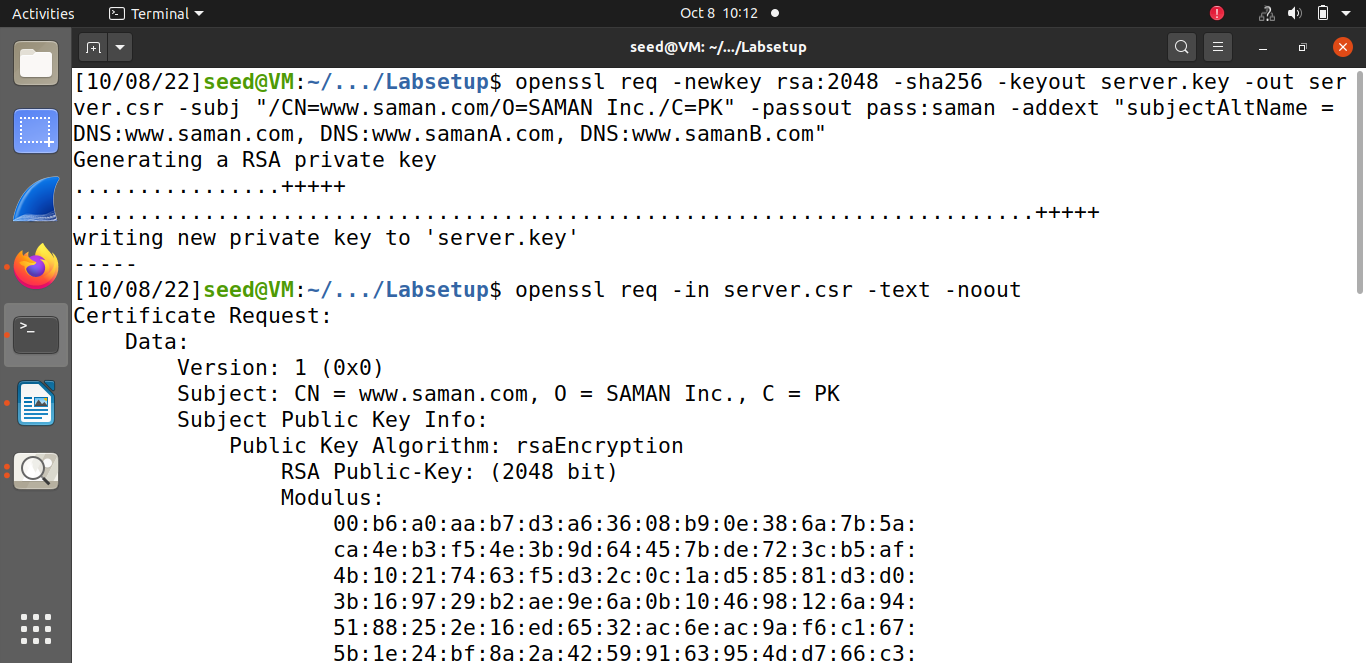
* **Prime#02 ‘q’:**

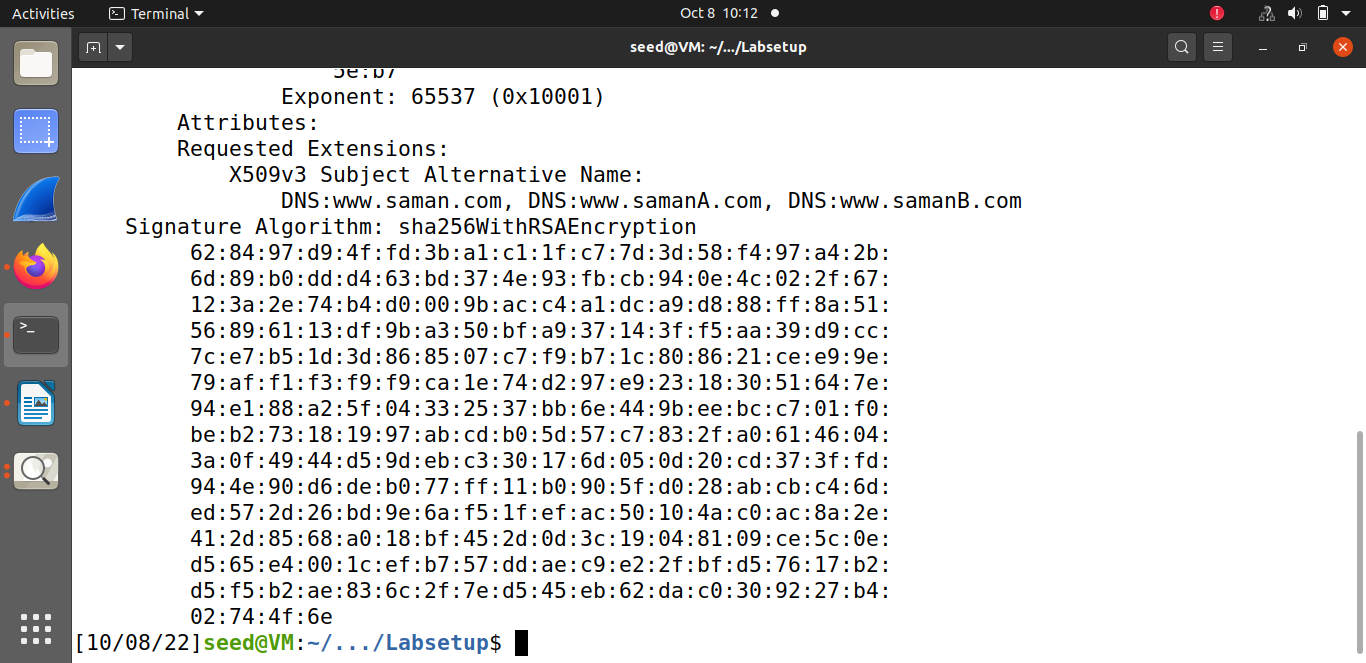
****

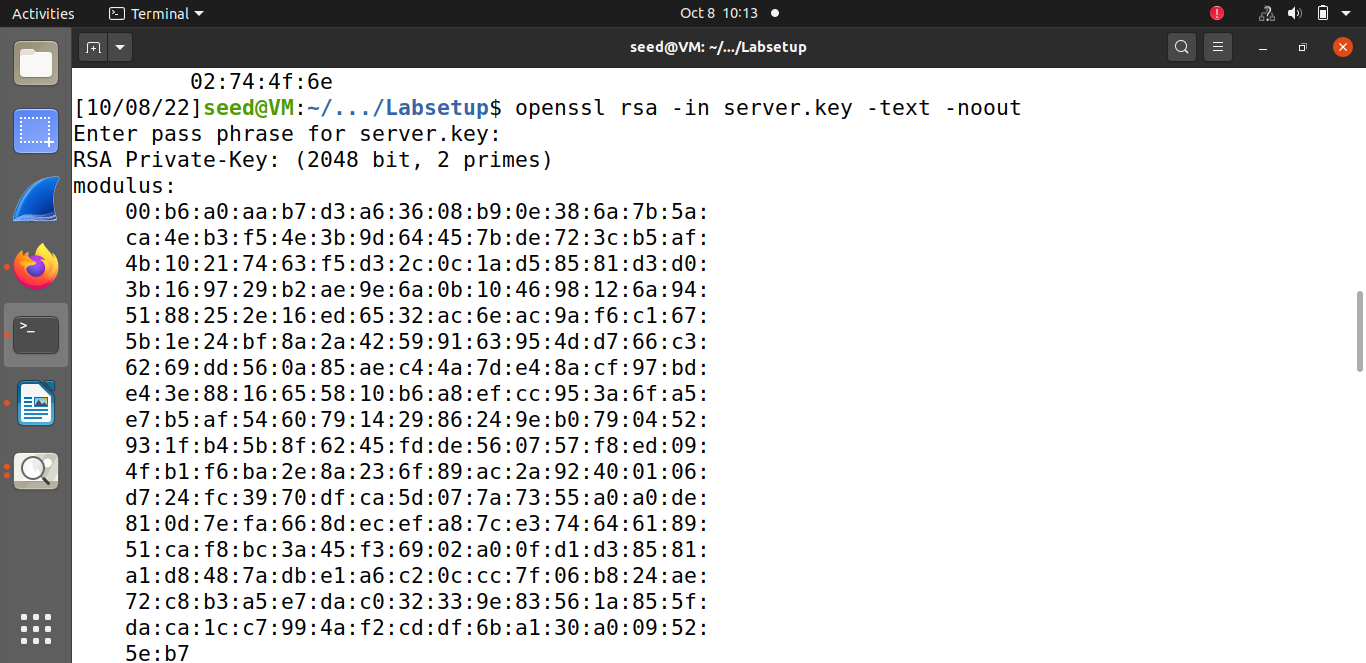
****

**TASK#02: Generating a Certificate Request for Your Web Server**

Here a Certificate Signing Request (CSR) is generated (for www.saman.com), which basically includes the company’s public key and identity information to get a public-key certificate from our CA. The CSR is be sent to the CA (created in task1), who will verify the identity information in the request, and then generate a certificate. As a result for the command executed, a pair of public/private key is generated, and then a certificate signing request from the public key is created. The decoded content of the CSR and private key files are attached in the screenshot below. To allow a certificate to have multiple names the extension Subject Alternative Name (SAN) is added to specify several hostnames in the subjectAltName field of a certificate.

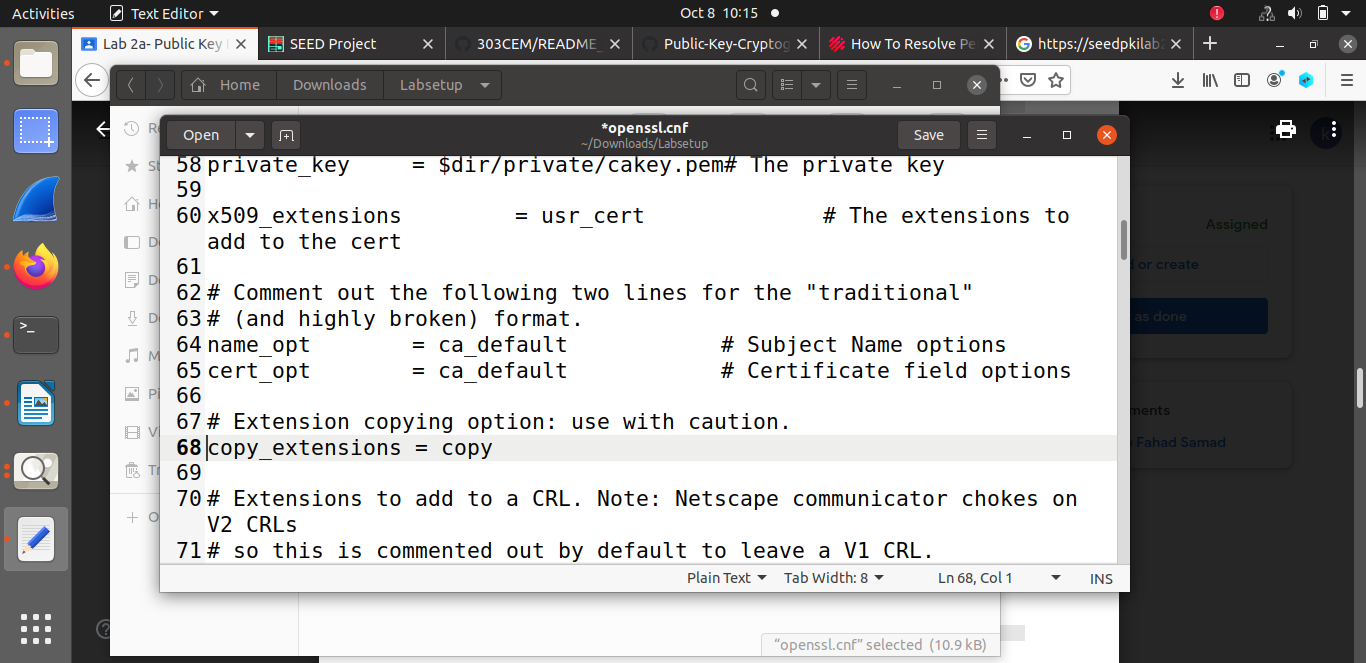
****

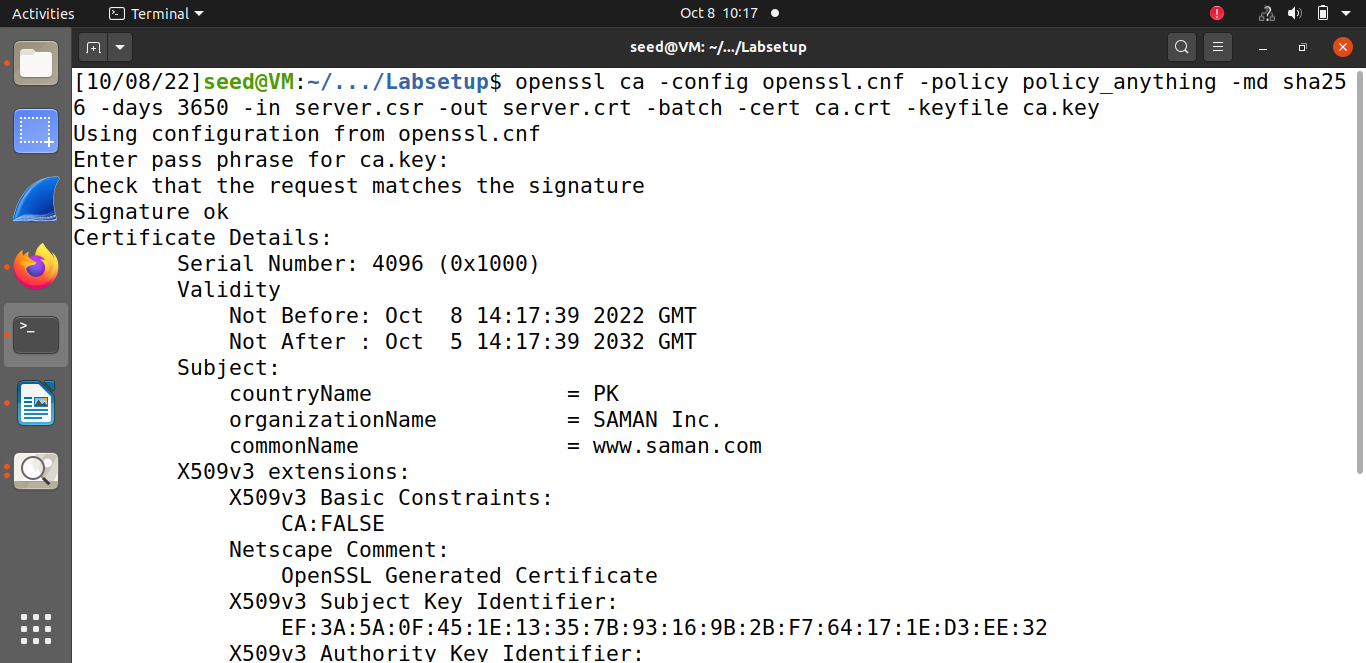
****

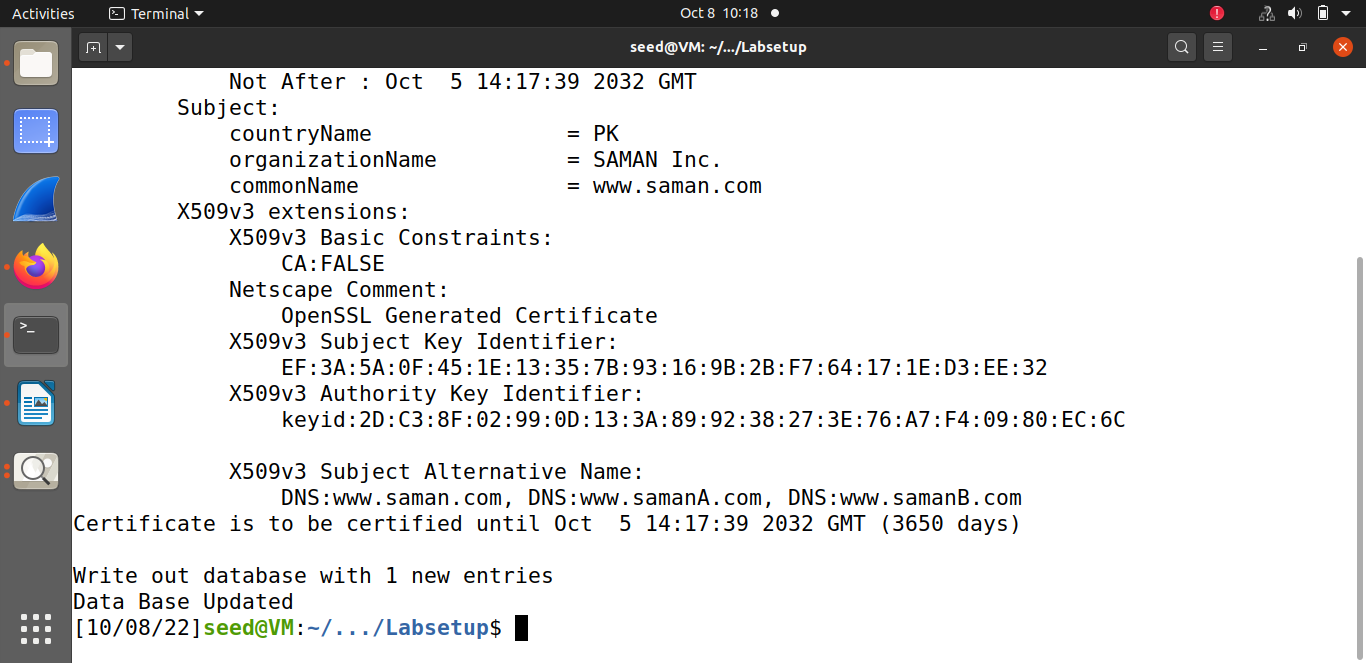
****

**TASK#03: Generating a Certificate for your server**

The CSR files are usually sent to a trusted CA (task1) for their signature to generate certificate for the server. 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. Here the policy anything is used as it has less restriction. But first we need to uncomment copy\_extension line to allow the "openssl ca" command to copy the extension field from the request to the final certificate.

****

****

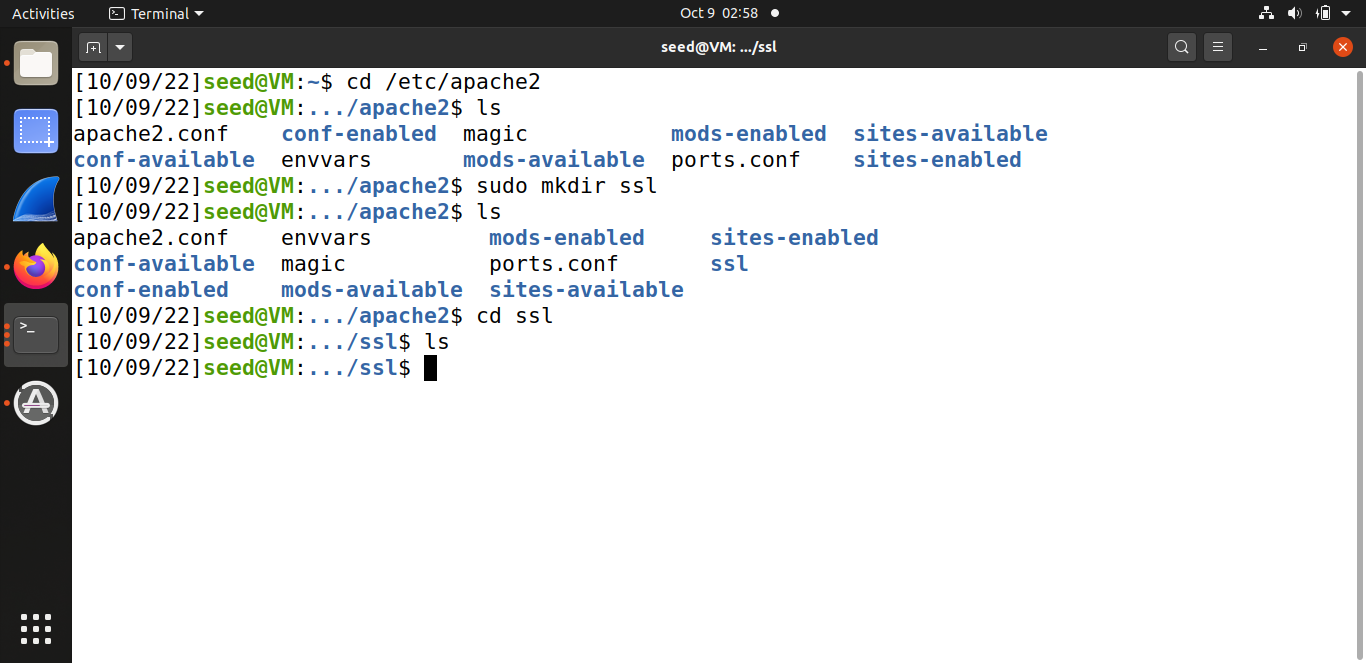
****

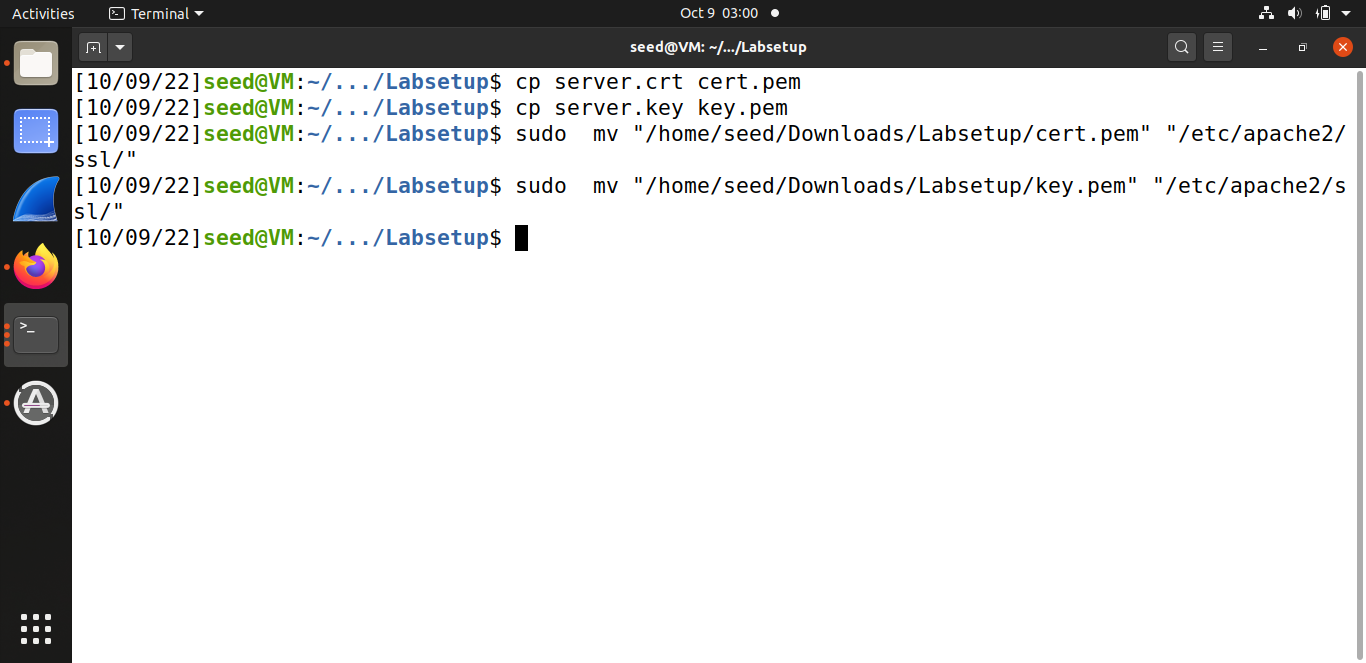
**TASK#04: Deploying Certificate in an Apache-Based HTTPS Website**

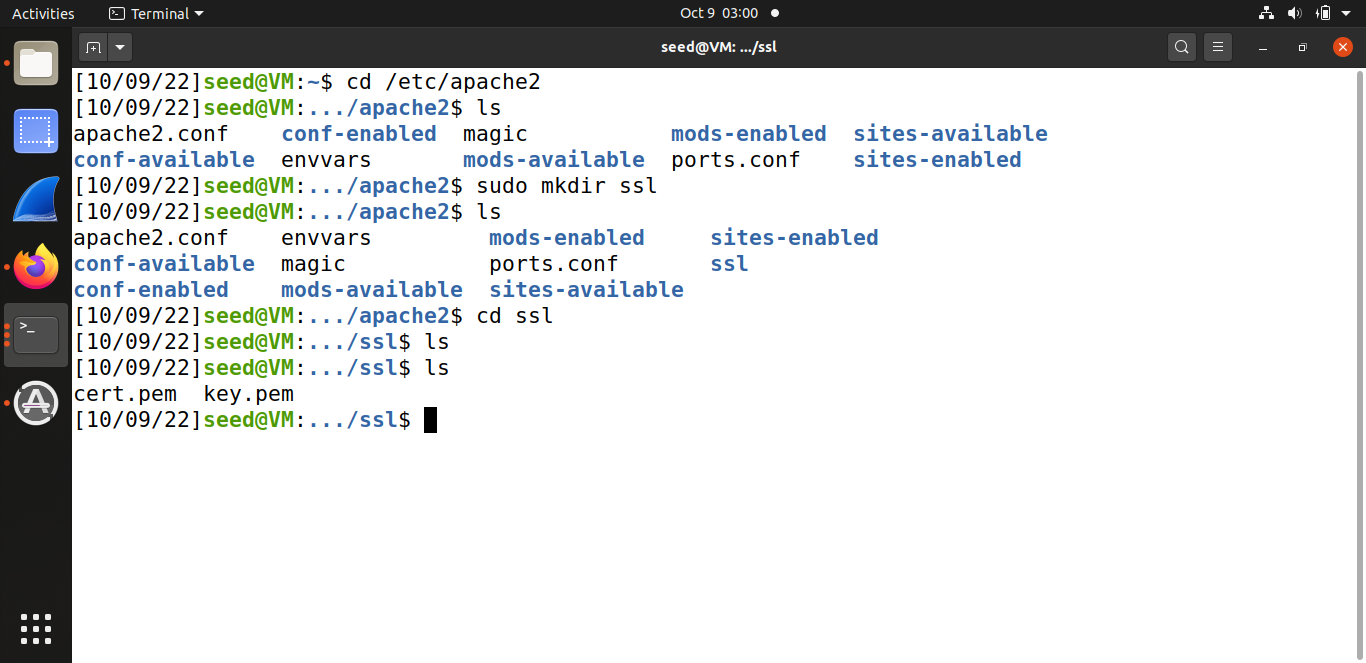
An Apache server can simultaneously host multiple websites. It needs to know the directory where a website’s files are stored. This is done via its VirtualHost file, located in the /etc/apache2/sites-available directory. First, he directory containing the html file is created in /var/www/.

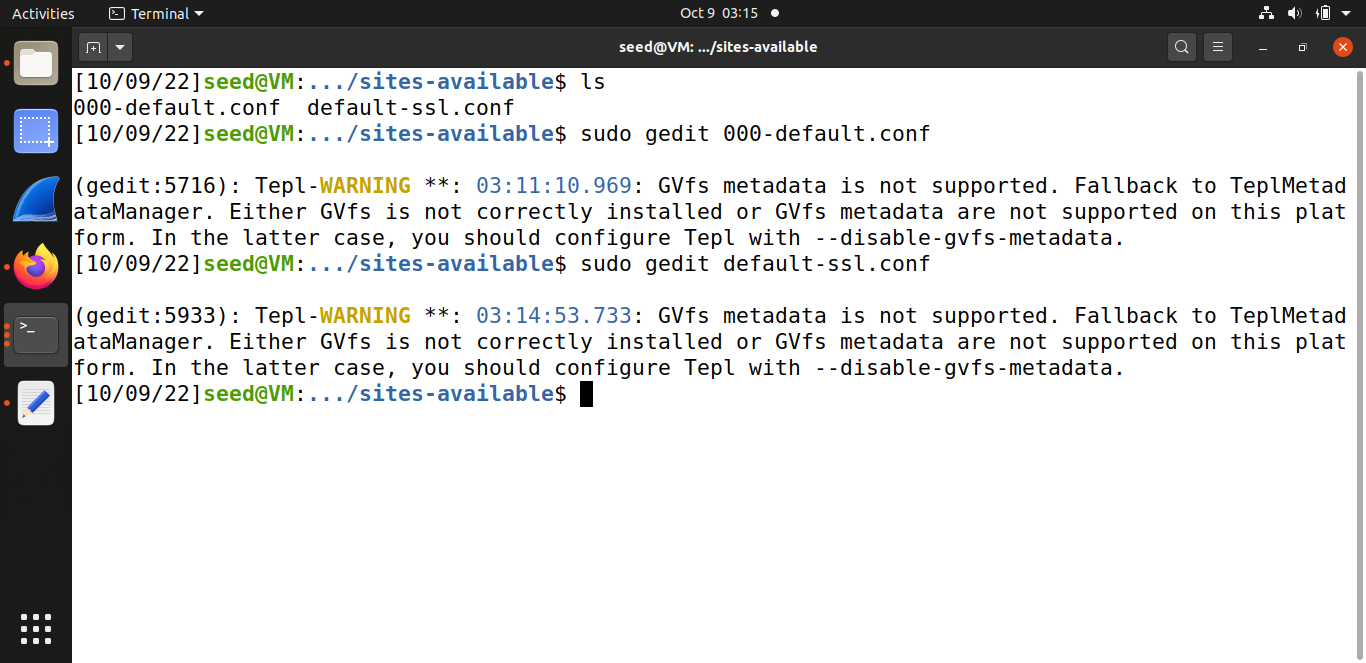
****

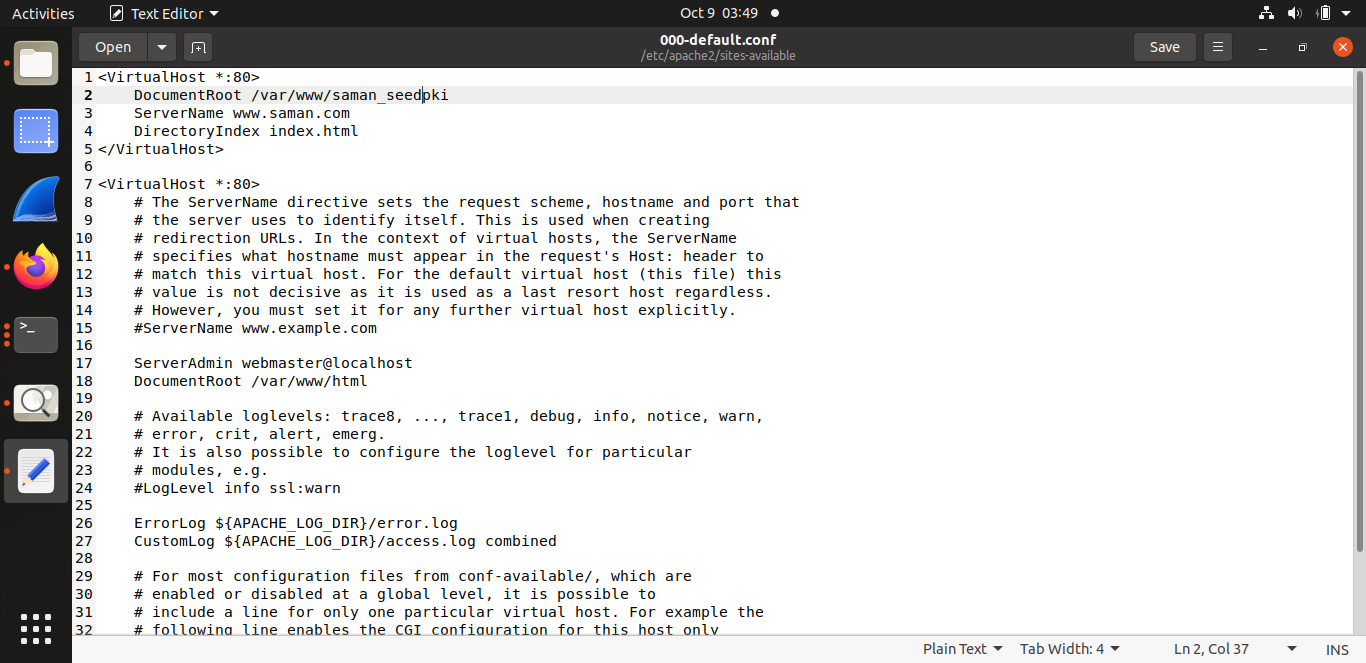
In /etc/apache2 a directory called ssl is created which contains the copy of key and certificate of the server. After this the configuration files in /etc/apache2/sites-available are modified to contain the information of our website. Port 443 is the default HTTPS port and the ServerName entry specifies the name of the website, while the DocumentRoot entry specifies where the files for the website are stored. Using the ServerAlias entries, we allow the website to have different names. You should also provide two alias entries. We also need to tell Apache where the server certificate and private key are stored hence we include the commands to copy the certificate and key to the /ssl folder.

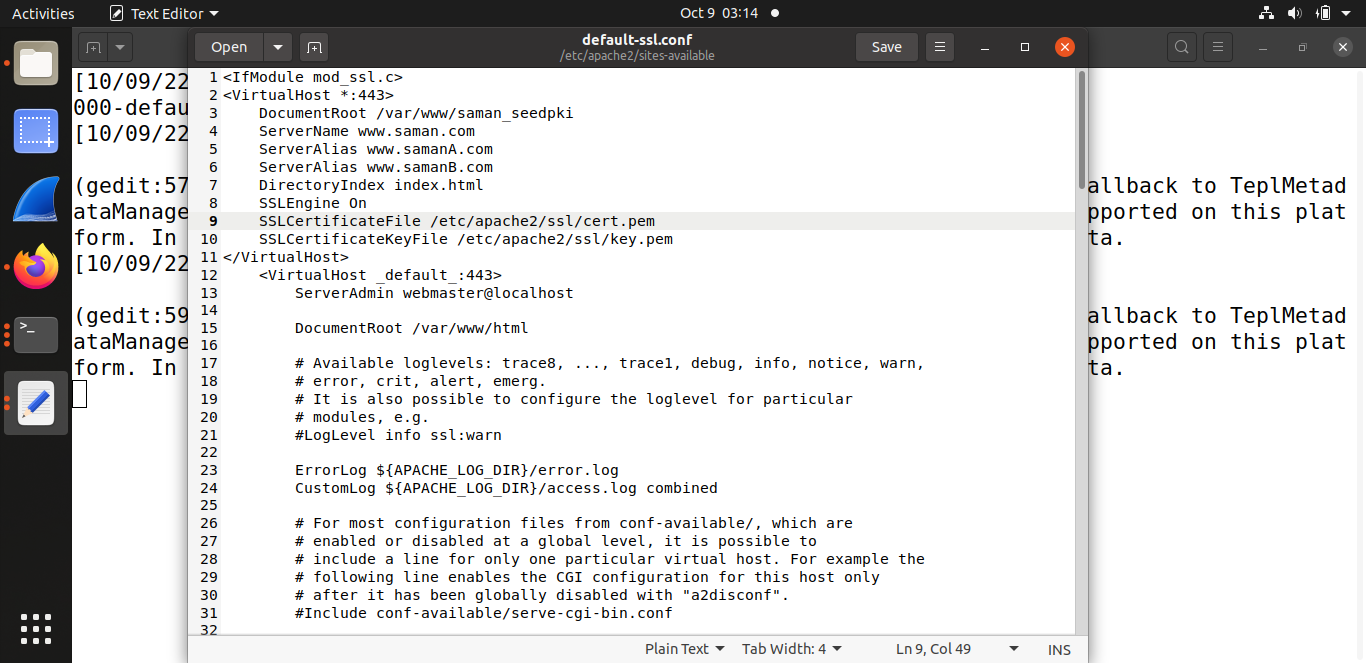
****

****

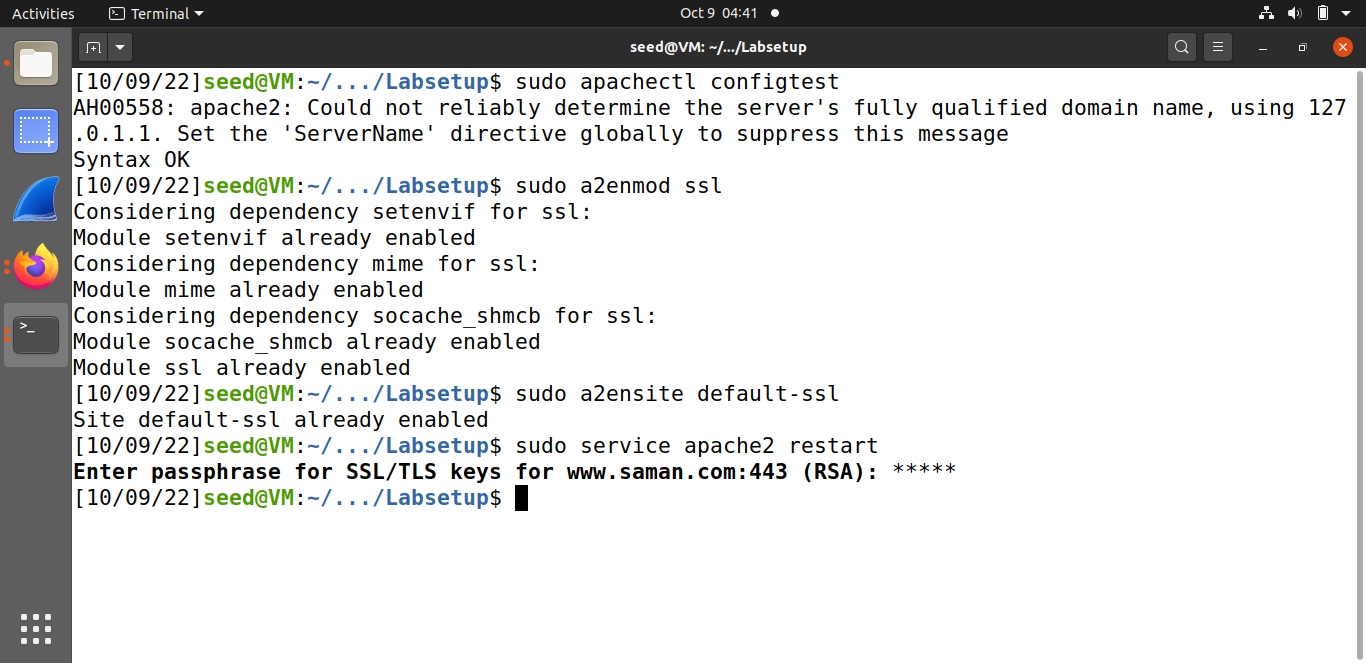
****

****

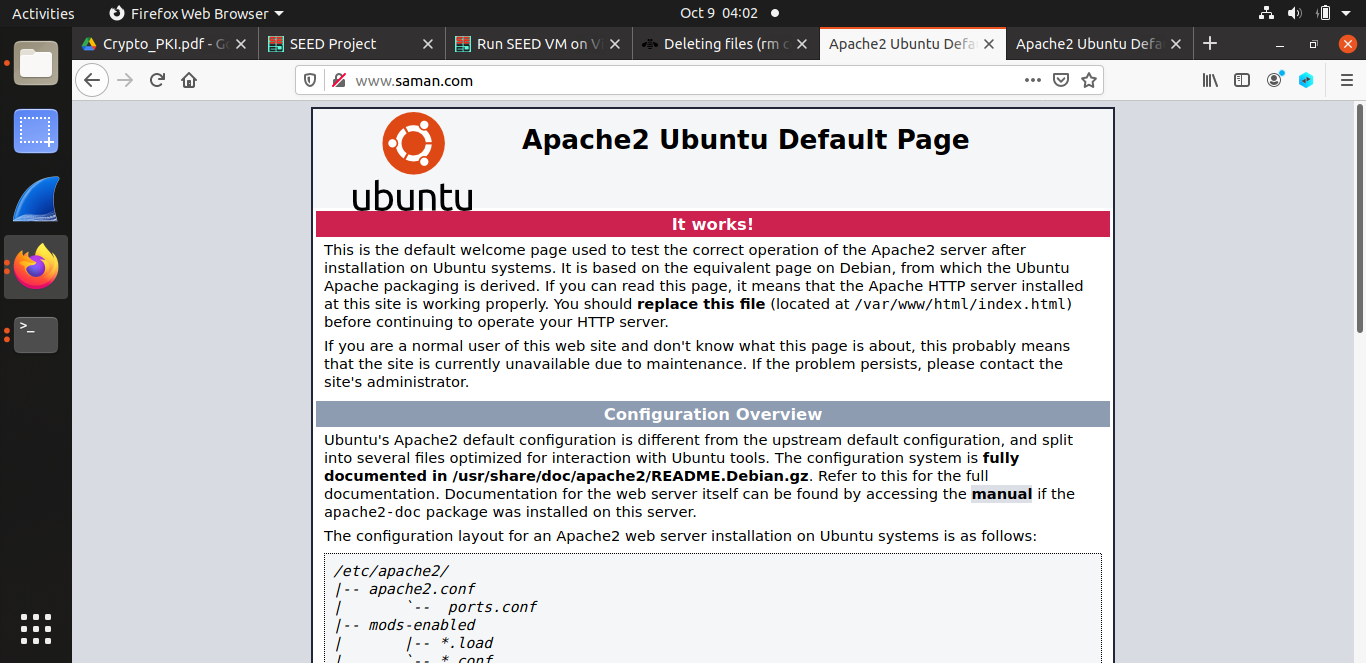
****

****

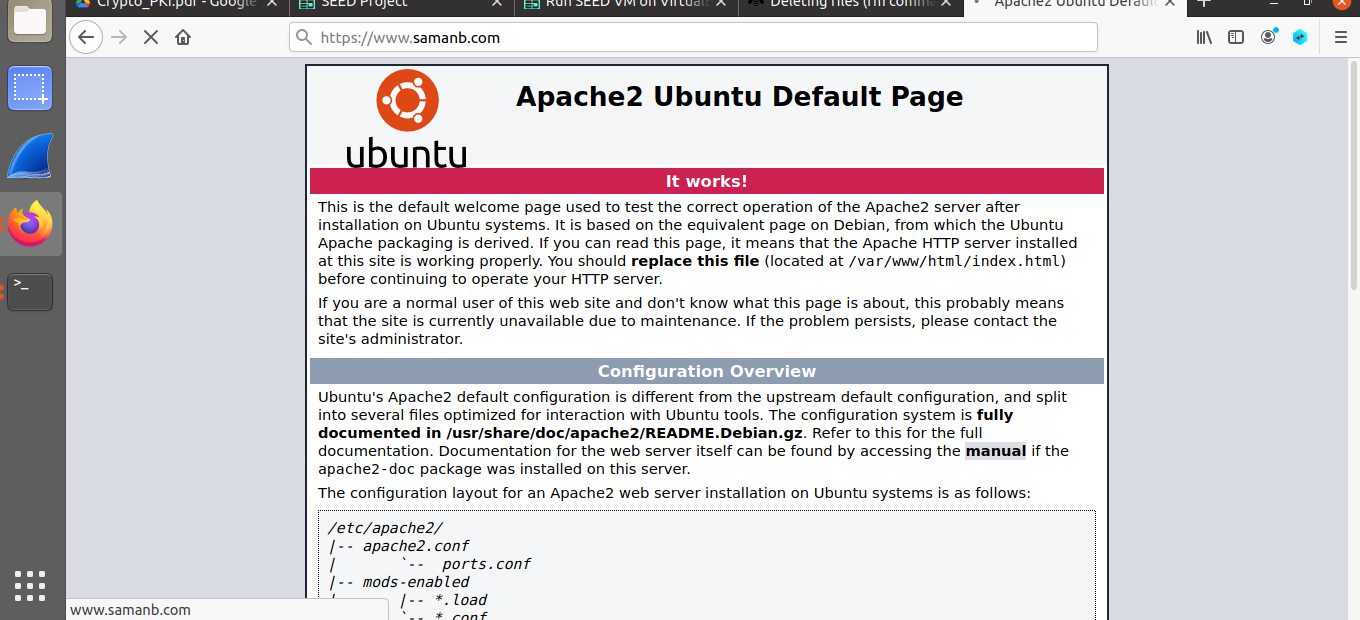
In order to make the website work, we need to enable Apache’s ssl module and then enable this site. The Apache server is not automatically started in the container, because of the need to type the password to unlock the private key. When Apache starts, it needs to load the private key for each HTTPS site. The private key is encrypted, so Apache asks to type the password for decryption (the password was set as saman). Once everything is set up properly, the web site can browsed, and all the traffic between the browser and the server will be encrypted.

****

* Using server name:

****

* Using alias name:

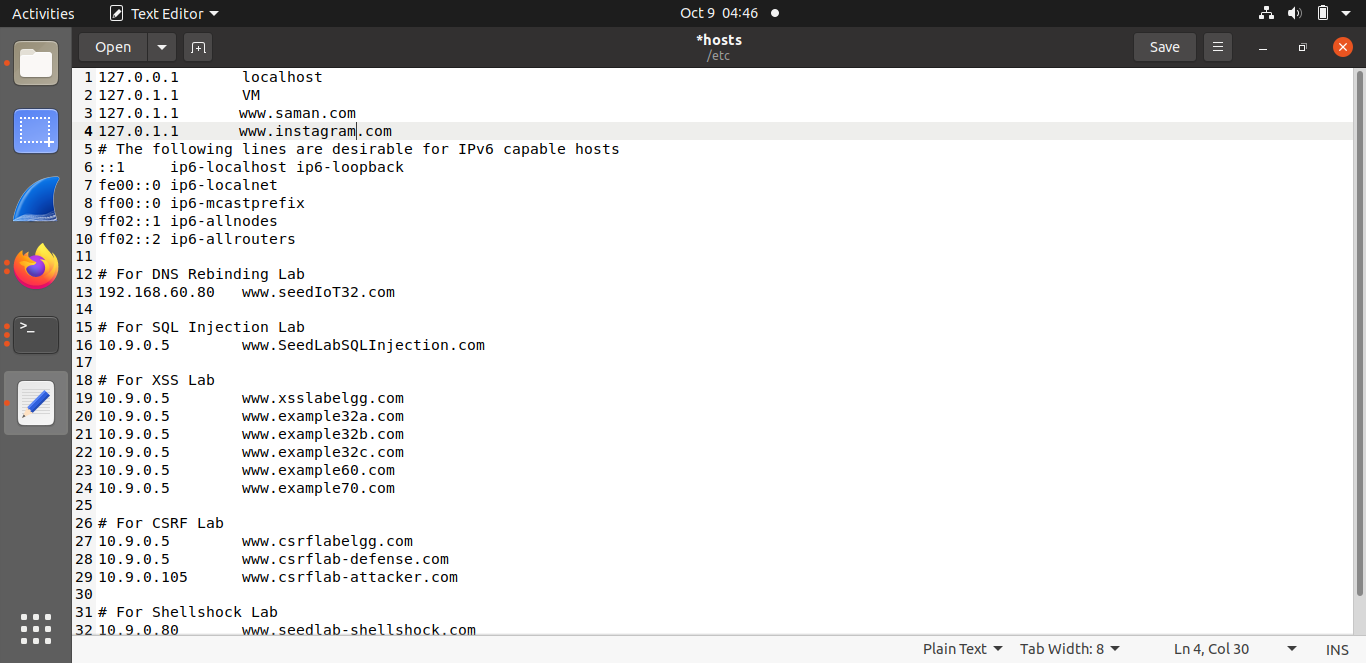
****

**TASK#05: Launching a Man-In-The-Middle Attack**

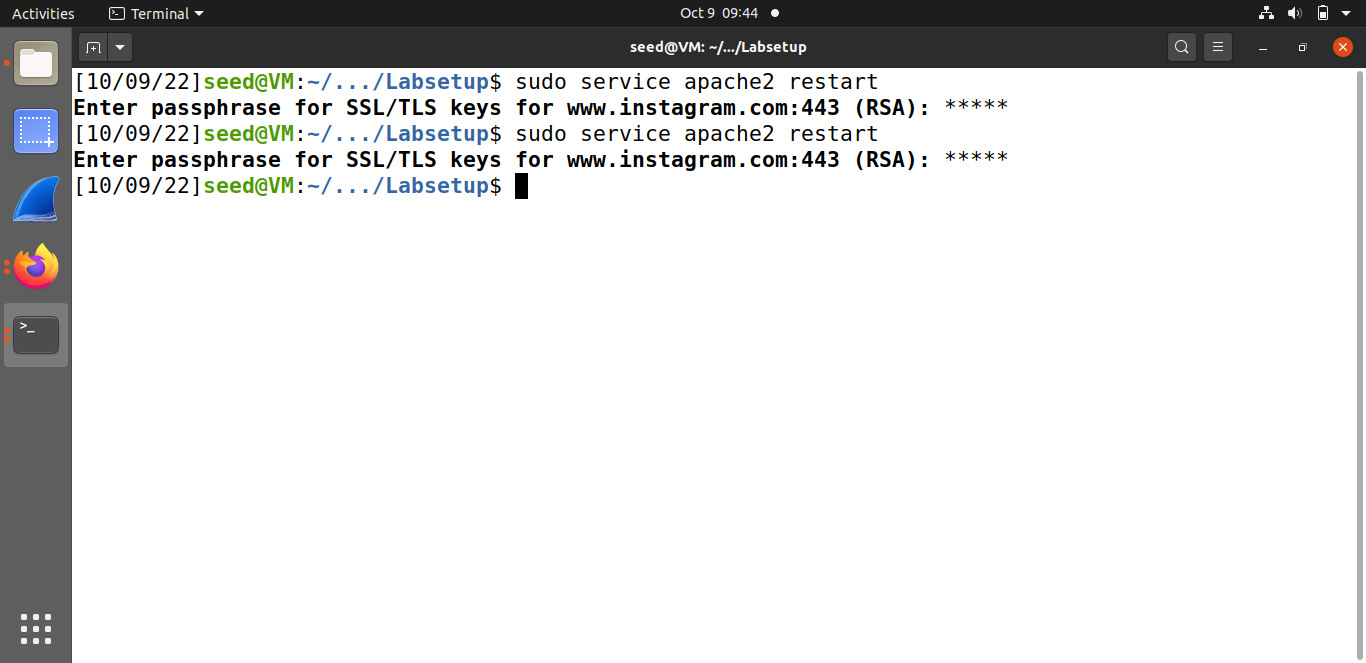
Here the same Apache server is used to impersonate [www.saman.som](http://www.saman.som). To achieve that, a VirtualHost entry is added to Apache’s SSL configuration file: the ServerName is be www.saman.com, but the rest of the configuration can

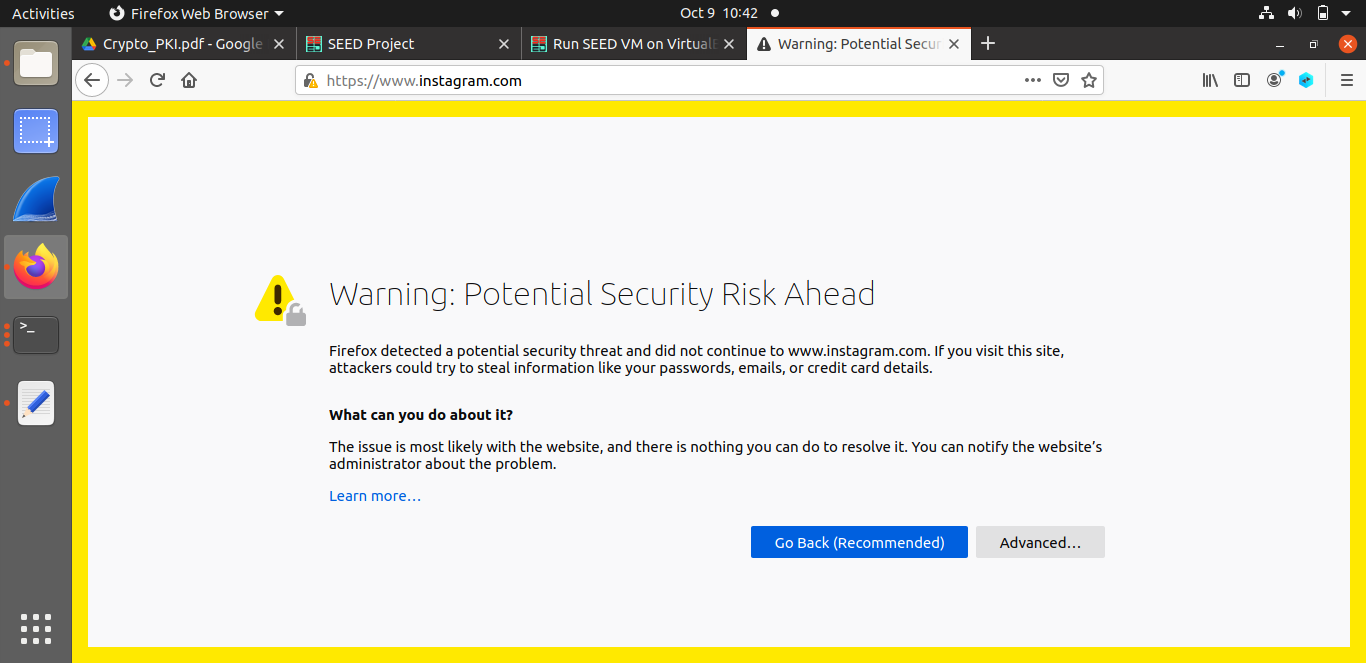
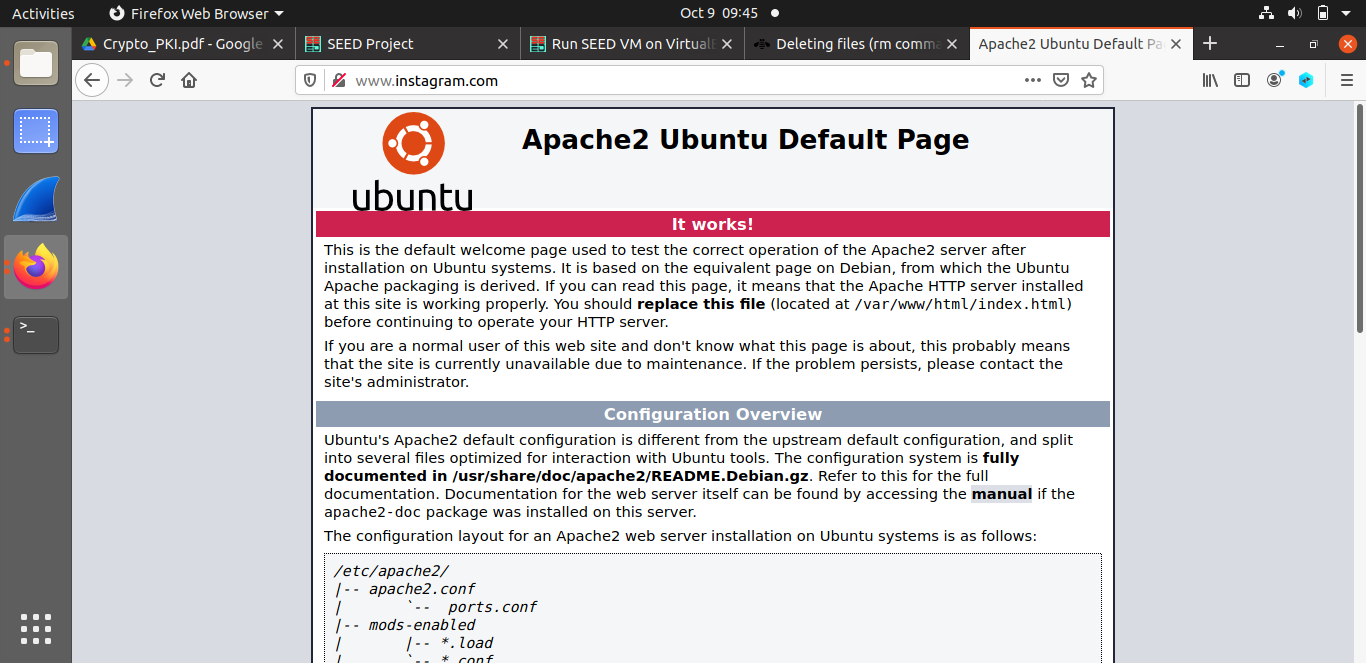
****

There are several ways to get the user’s HTTPS request to land in the web server. One way is to attack the routing, so the user’s HTTPS request is routed to our web server. Another way is to attack DNS, so when the victim’s machine tries to find out the IP address of the target web server, it gets the IP address of the web server. In this task, the attack-DNS approach is simulated. Instead of launching an actual DNS cache poisoning attack, the machine’s /etc/hosts file is simply modified to emulate the result of a DNS cache positing attack by mapping the hostname www.saman.com to the malicious web server.

****

The Apache server is restarted and password is entered.

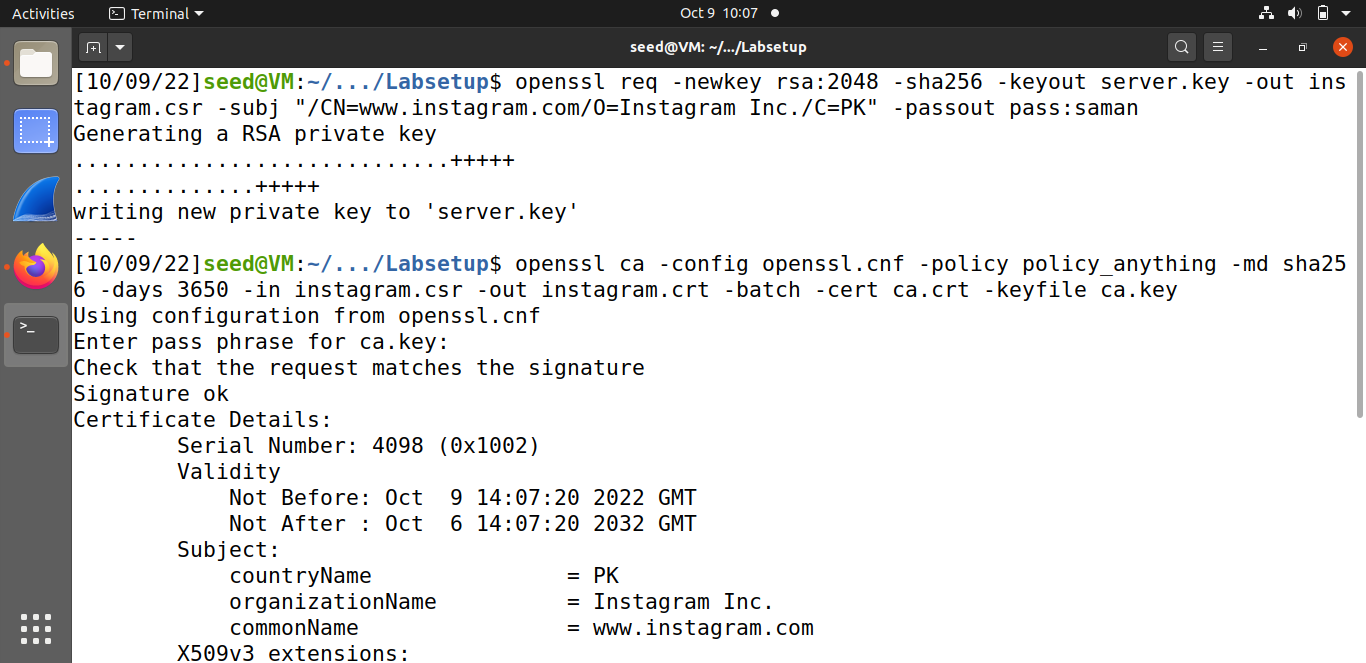
****

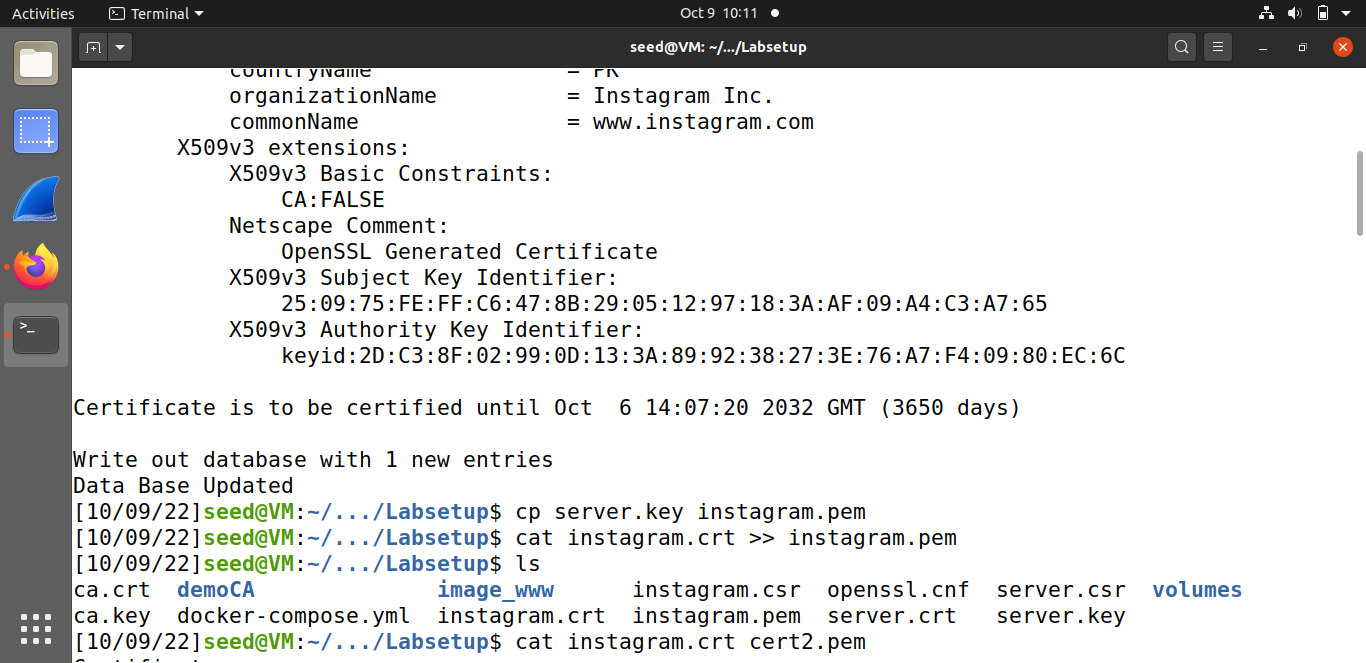
****

**TASK#06: Launching a Man-In-The-Middle Attack with a Compromised CA**

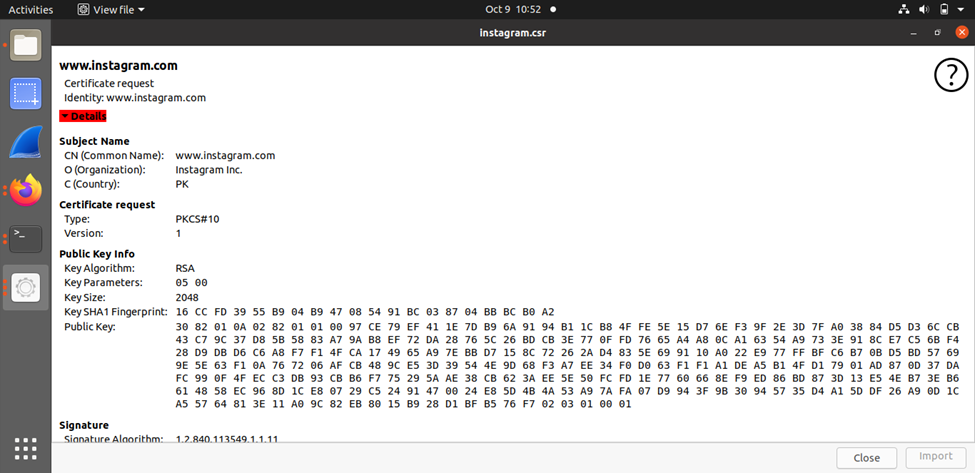
In this task, root CA created in Task 1 is compromised by an attacker, and its private key is stolen. Therefore, the attacker can generate any arbitrary certificate using this CA’s private key. In this

task, the consequence of such a compromise are seen. Here the attacker has successfully launched MITM attacks on the HTTPS website created in Task 4. First, a CSR is generated.

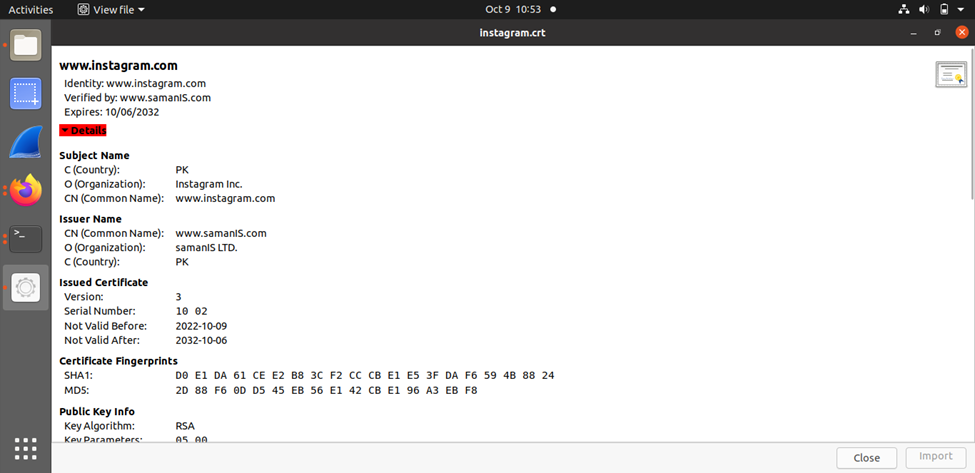
****

****

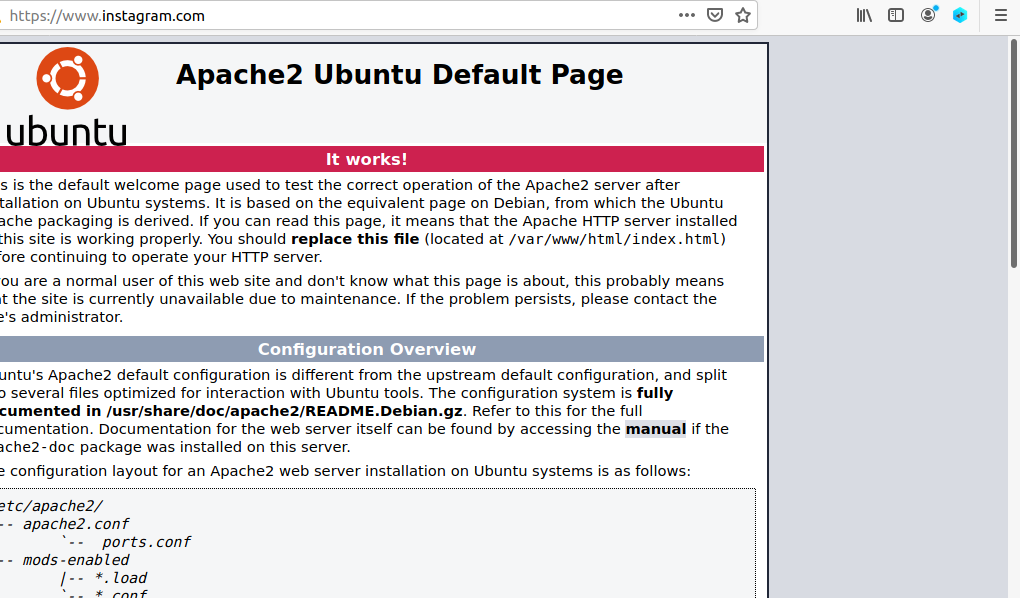
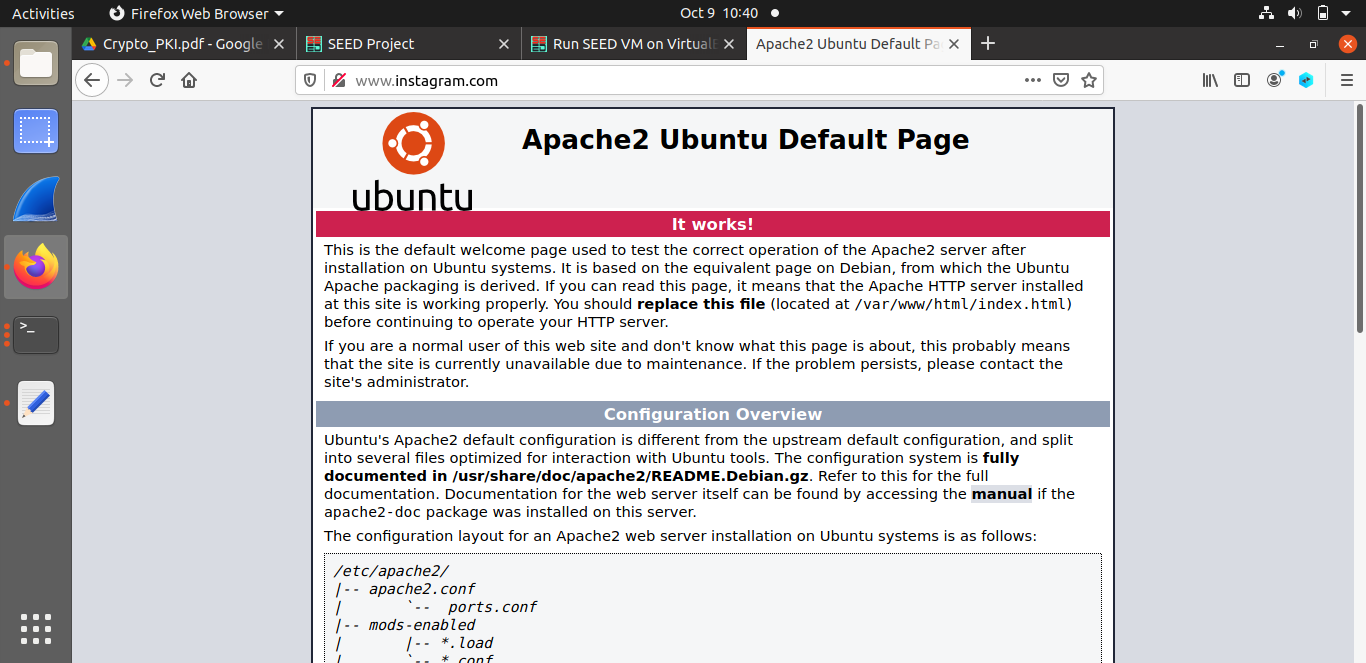
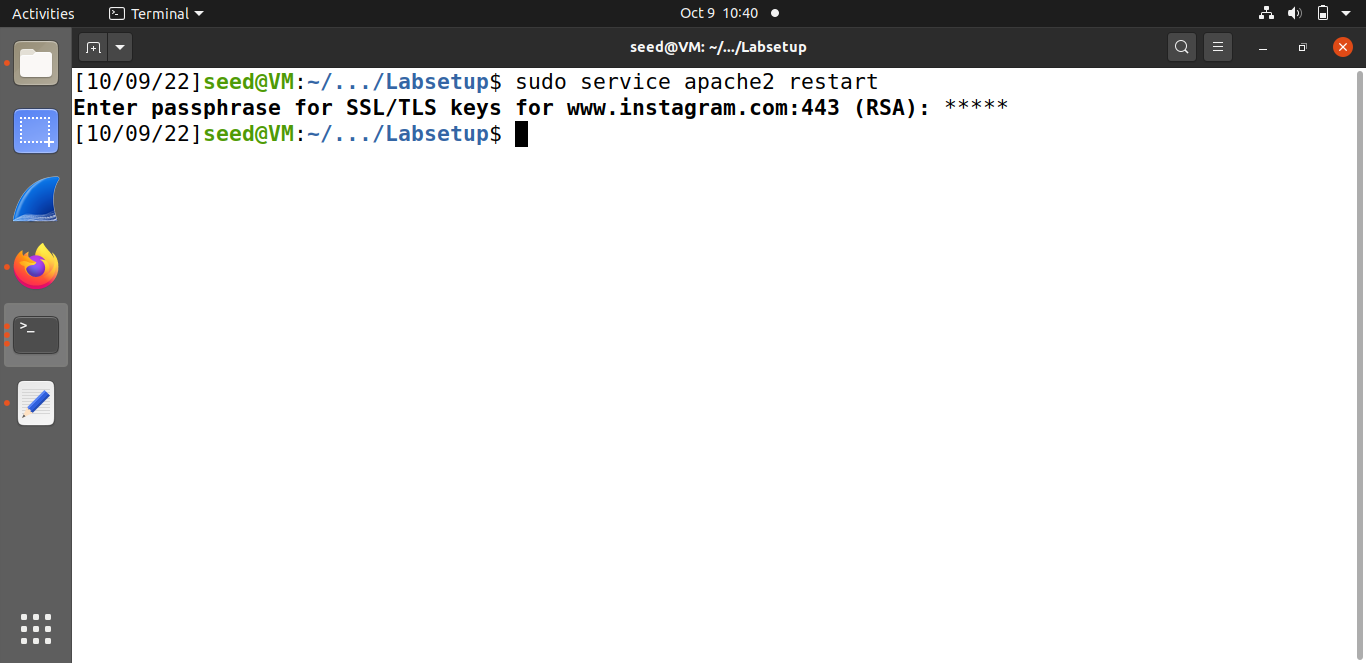
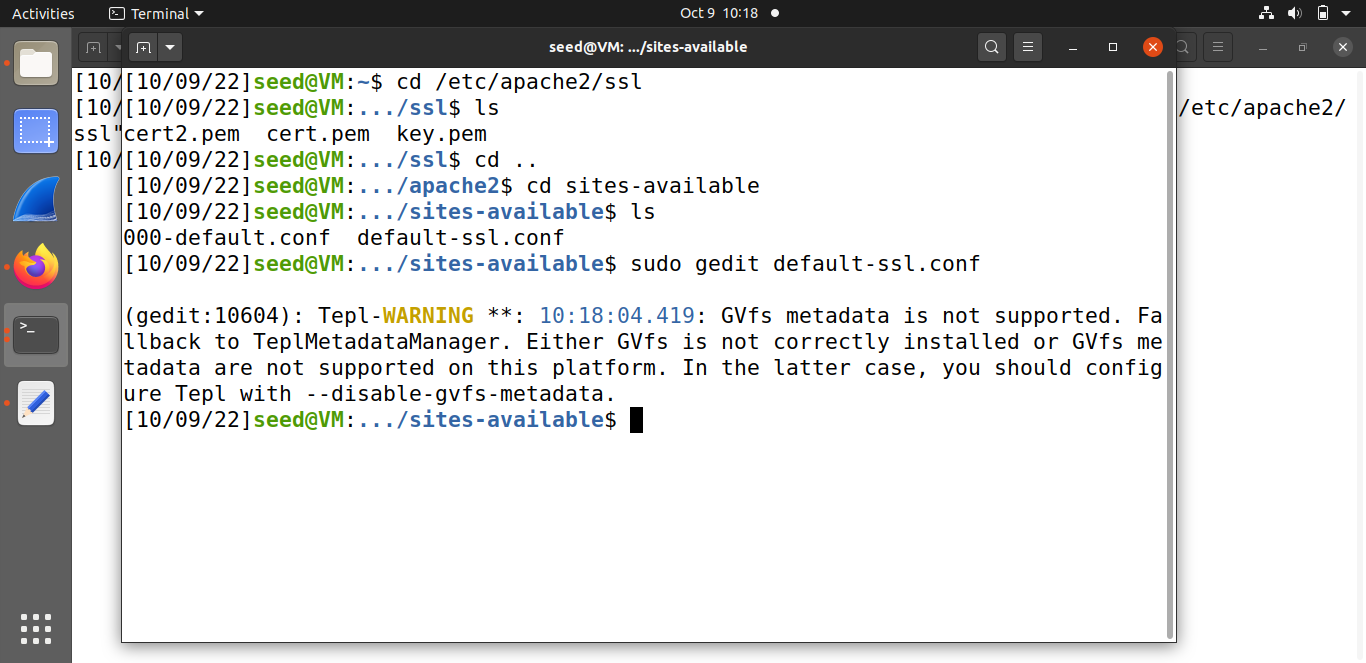
* Certificate Signing Request:

****

* Certificate:

****

Similar steps are repeated for [www.instagram.com](http://www.instagram.com) that were already observed in previous tasks.

****