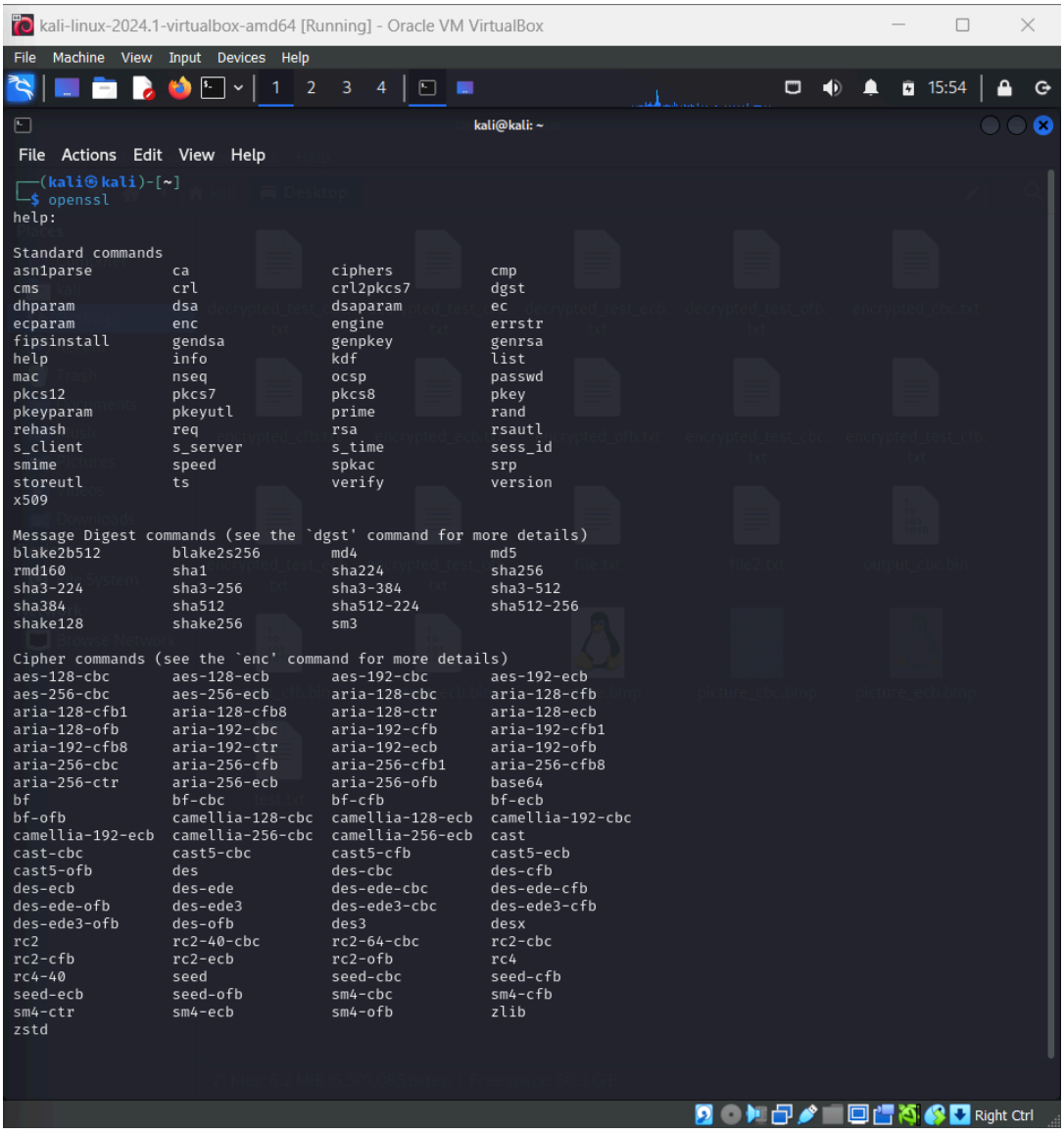
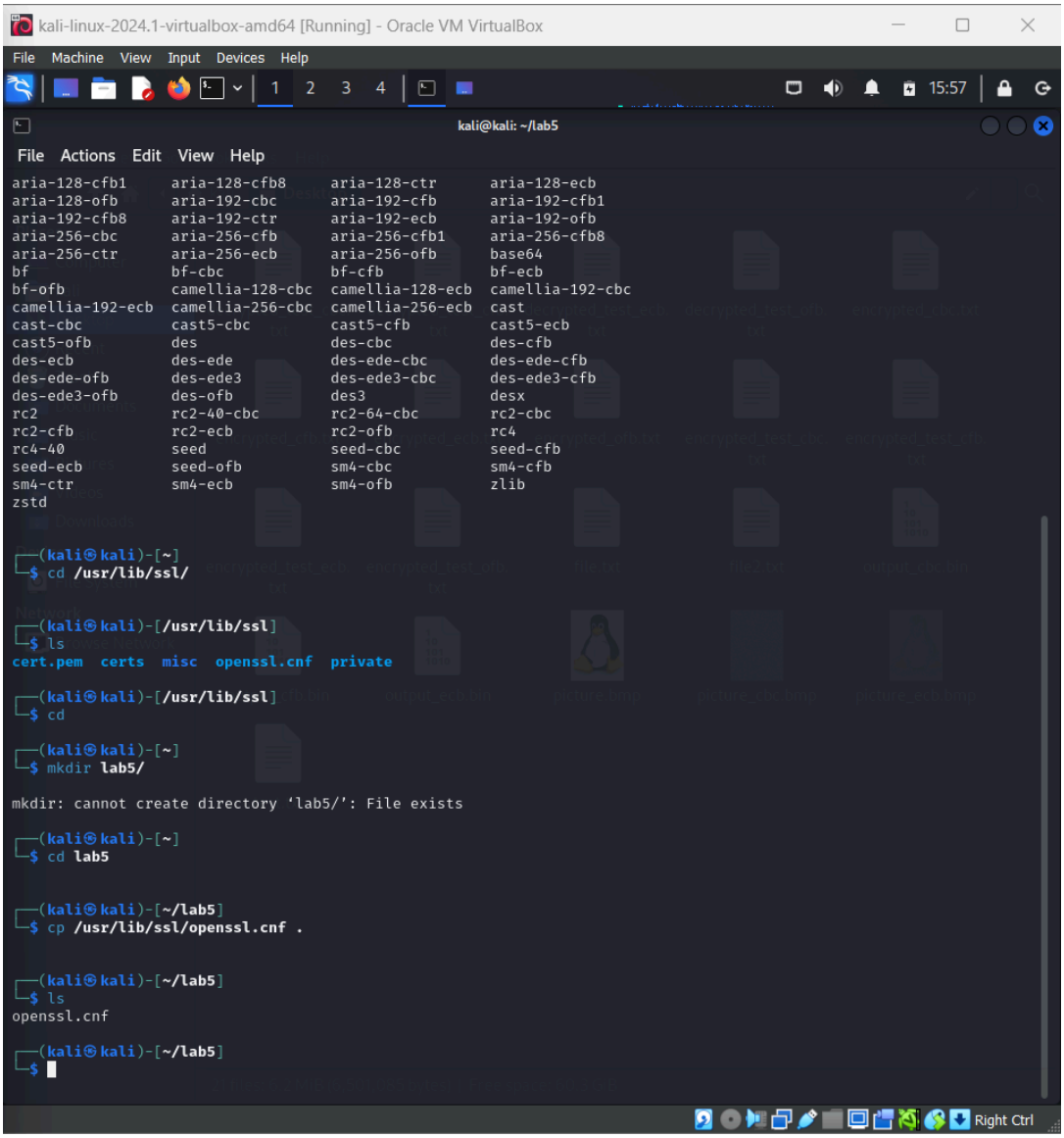
Lab Task 5

Task-1: Becoming a certificate authority

In this lab, you will need to create digital certificates, but you will not be going to pay to any commercial CA. You will become a root CA, and then use this CA to issue certificate for others (e.g. servers). In this task, you will make yourself a root CA, and generate a certificate for this CA. Unlike other certificates, which are usually signed by another CA, the root CA's certificates are self-signed. Root CA's certificates are usually pre-loaded into most operating systems, web browsers, and other software that rely on certificate-based security. Root CA's certificates are unconditionally trusted.

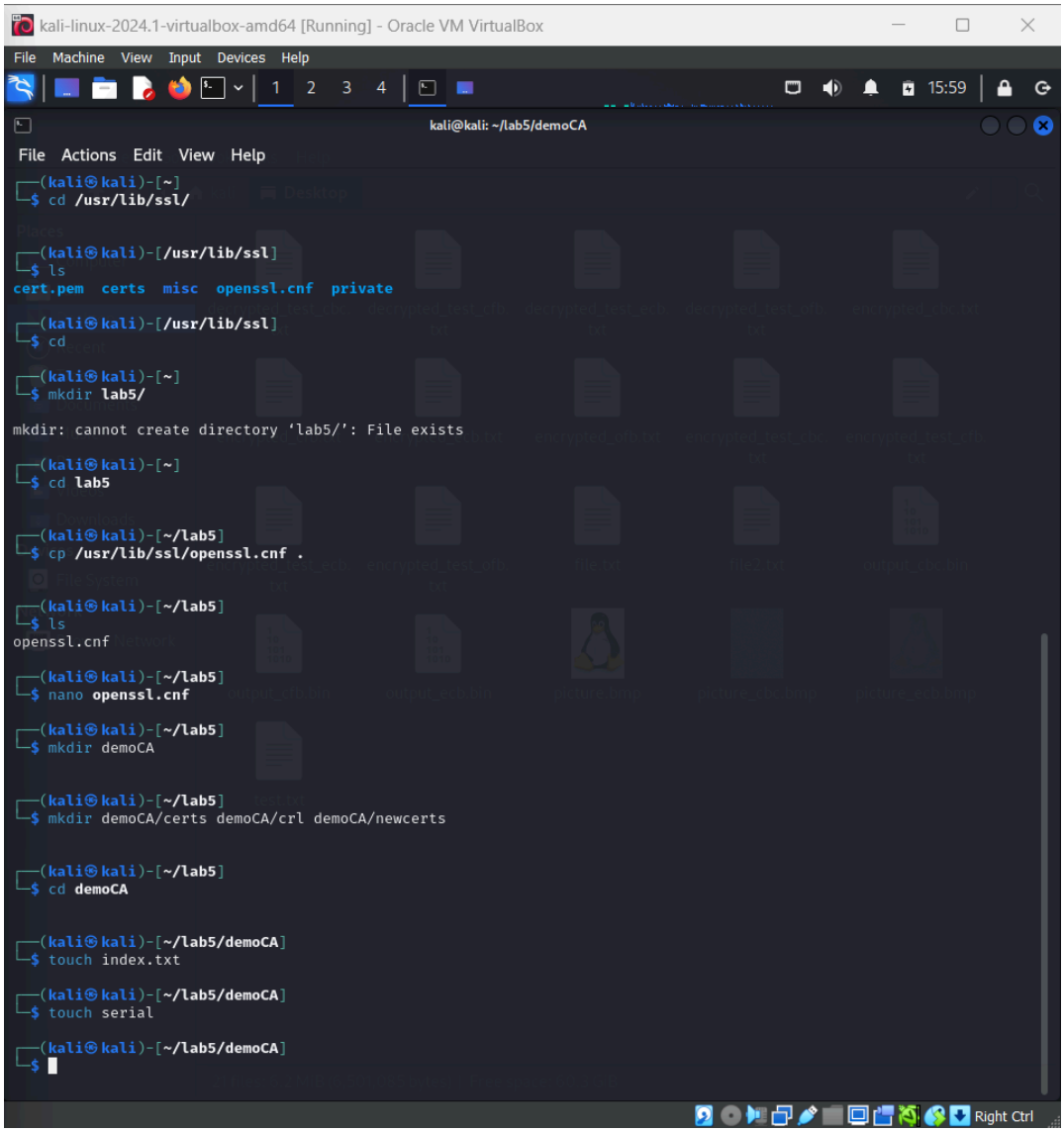


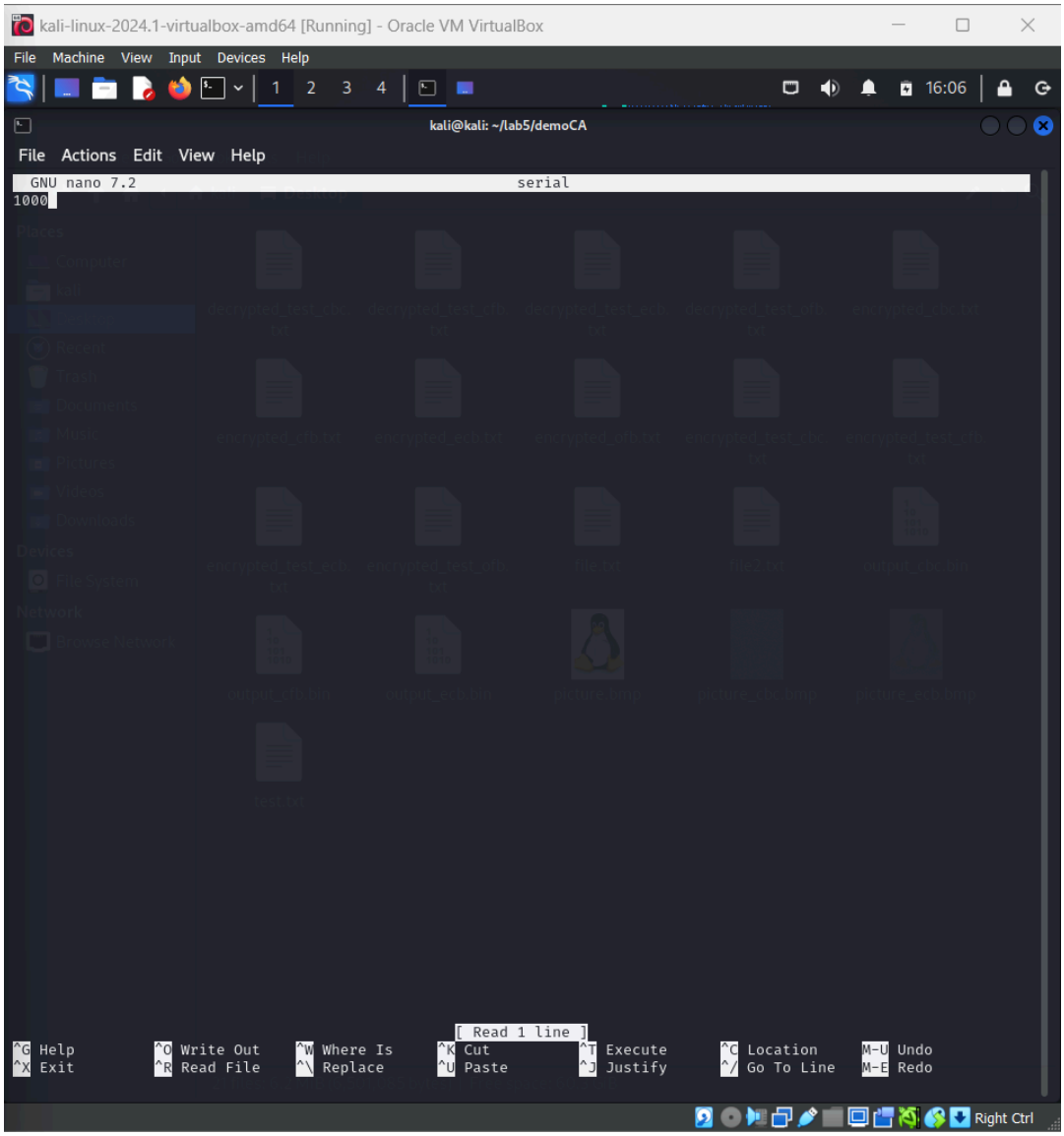
To start this task, create a folder for this task and cd into it. In this folder, you will need to create a particular configuration file as discussed below. I created the folder advanced below for prerequisite apache setup, firewall adjustment and web server check.



The Configuration File openssl.conf:
In order to use OpenSSL to create certificates, you have to have a configuration file. The configuration file usually has an extension .cnf. It is used by three OpenSSL commands: ca, req and x509.

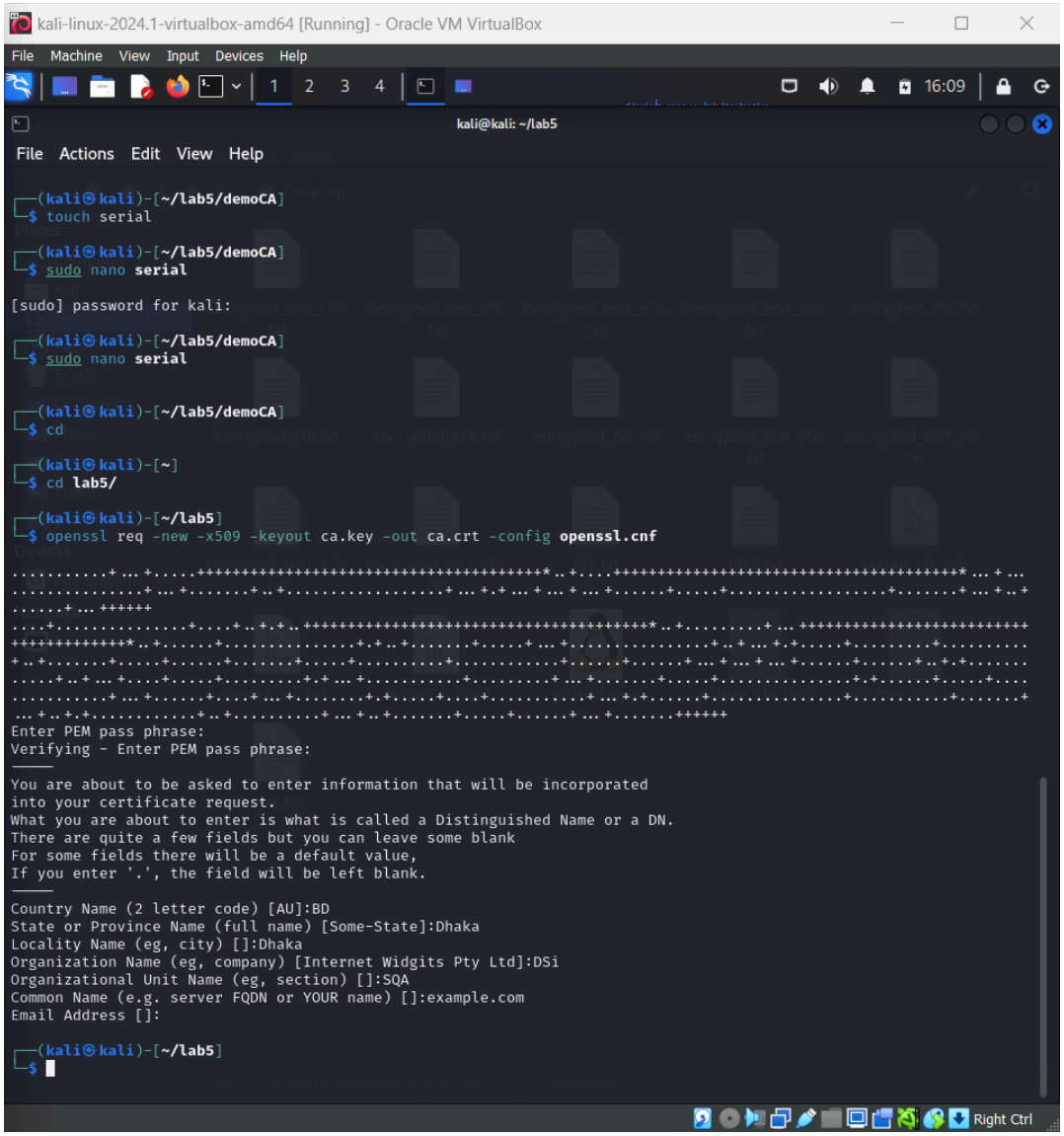
For the index.txt file, simply create an empty file. For the serial file, put a single number in string format (**e.g. 1000**) in the file. Once you have set up the configuration file openssl.cnf, you can create and issue certificates


kali-linux-2024.1-virtualbox-amd64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
15:59
kali@kali: ~/lab5/demoCA
File Actions Edit View Help
(kali㉿kali)-[~]
$ cd /usr/lib/ssl/
(kali㉿kali)-[/usr/lib/ssl]
$ ls
cert.pem certs misc openssl.cnf private
(kali㉿kali)-[/usr/lib/ssl]
$ cd
(kali㉿kali)-[~]
$ mkdir lab5/
mkdir: cannot create directory ‘lab5/’: File exists
(kali㉿kali)-[~]
$ cd lab5
(kali㉿kali)-[~/lab5]
$ cp /usr/lib/ssl/openssl.cnf .
(kali㉿kali)-[~/lab5]
$ ls
openssl.cnf
(kali㉿kali)-[~/lab5]
$ nano openssl.cnf
(kali㉿kali)-[~/lab5]
$ mkdir demoCA
(kali㉿kali)-[~/lab5]
$ mkdir demoCA/certs demoCA/crl demoCA/newcerts
(kali㉿kali)-[~/lab5]
$ cd demoCA
(kali㉿kali)-[~/lab5/demoCA]
$ touch index.txt
(kali㉿kali)-[~/lab5/demoCA]
$ touch serial
(kali㉿kali)-[~/lab5/demoCA]
$
Right Ctrl



Certificate Authority (CA):
As described before, you need to 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. You can run the following command to generate the self-signed certificate for the CA:

openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf



You will be prompted for information and a password. Do not lose this password, because you will have to type the passphrase each time you want to use this CA to sign certificates for others. You will also be asked to fill in some information, such as the Country Name, Common Name, etc. 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.

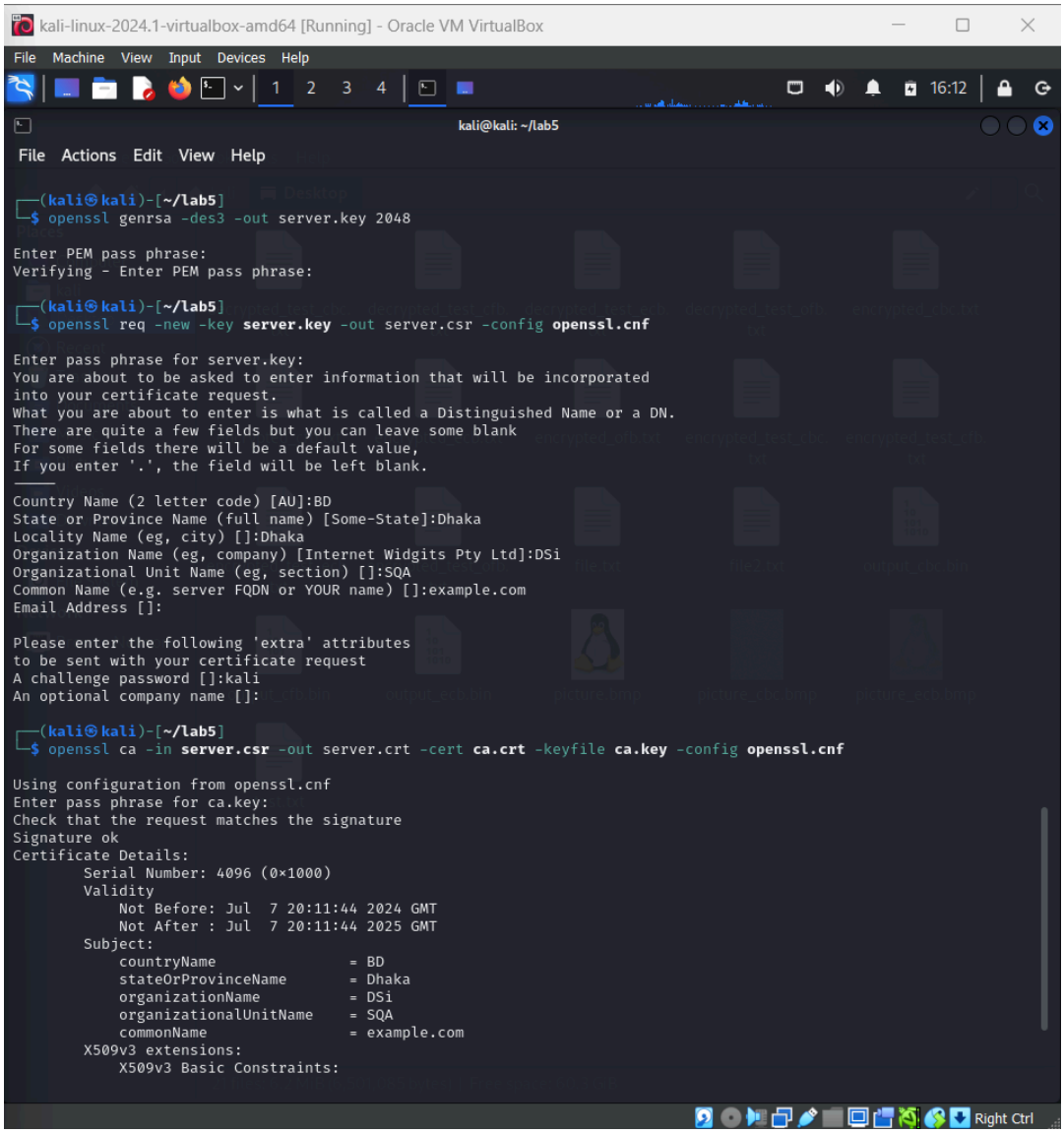
Creating a certificate for example.com:
After becoming a root CA, you are ready to sign digital certificates for our customers. Our first customer is a company called example.com. For this company to get a digital certificate from a CA, it needs to go through three steps.

Step 1: Generate public/private key pair

The company needs to first create its own public/private key pair. You can run the following command to generate an RSA key pair (both private and public keys). You will also be required to provide a password to protect the keys.

The keys will be stored in the file server.key:

openssl genrsa -des3 -out server.key 2048



Step 2: Generate a Certificate Signing Request (CSR)

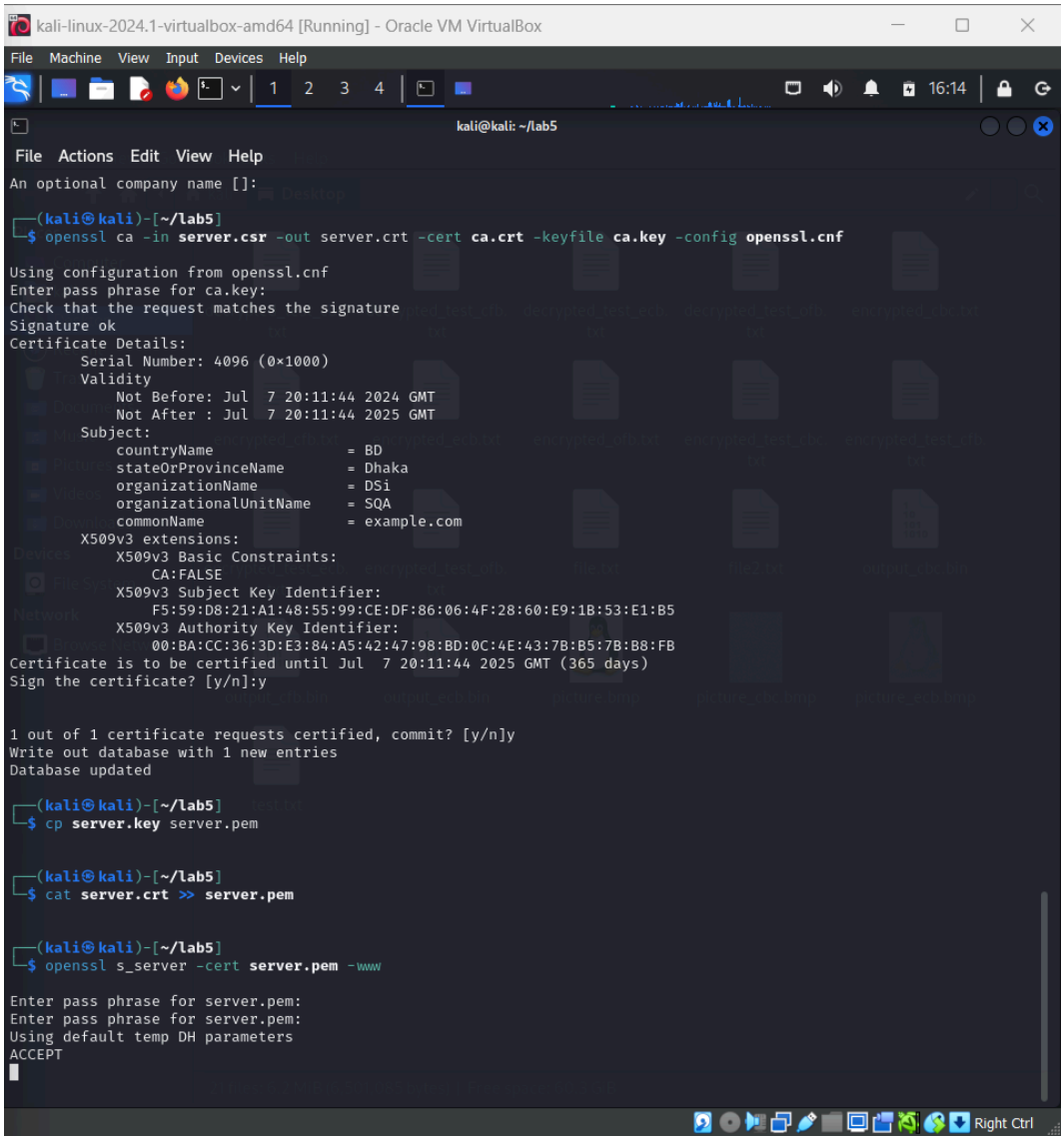
Once the company has the key file, it should generate a Certificate Signing Request (CSR). The CSR will be sent to the CA, who will generate a certificate for the key (usually after ensuring that identity information in the CSR matches with the server's true identity). Please use example.com as the common name of the certificate request.

openssl req -new -key server.key -out server.csr -config openssl.cnf

Step 3: Generating Certificates

The CSR file needs to have the CA's signature to form a certificate. In the real world, the CSR files are usually sent to a trusted CA for their signature. In this lab, you will use our own trusted CA to generate certificates.

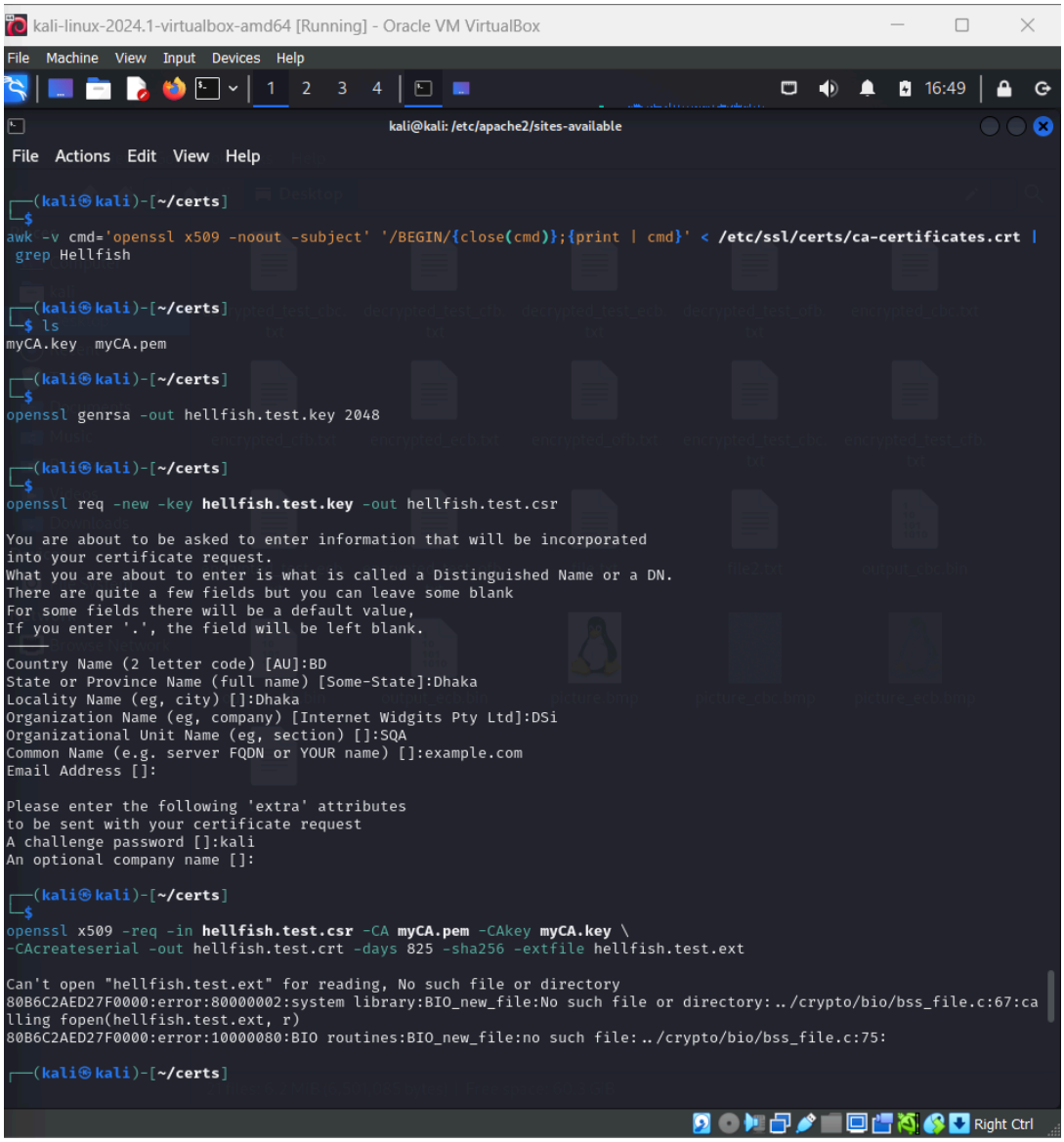
openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf



If OpenSSL refuses to generate certificates, it is very likely that the names in your requests do not match with those of CA. Fix this and re-issue the above command.
Next, let us launch a simple web server with the certificate generated in the previous task. OpenSSL allows us to start a simple web server using the s_server command. Use the following steps:

Step 1: Combine the secret key and certificate into one file

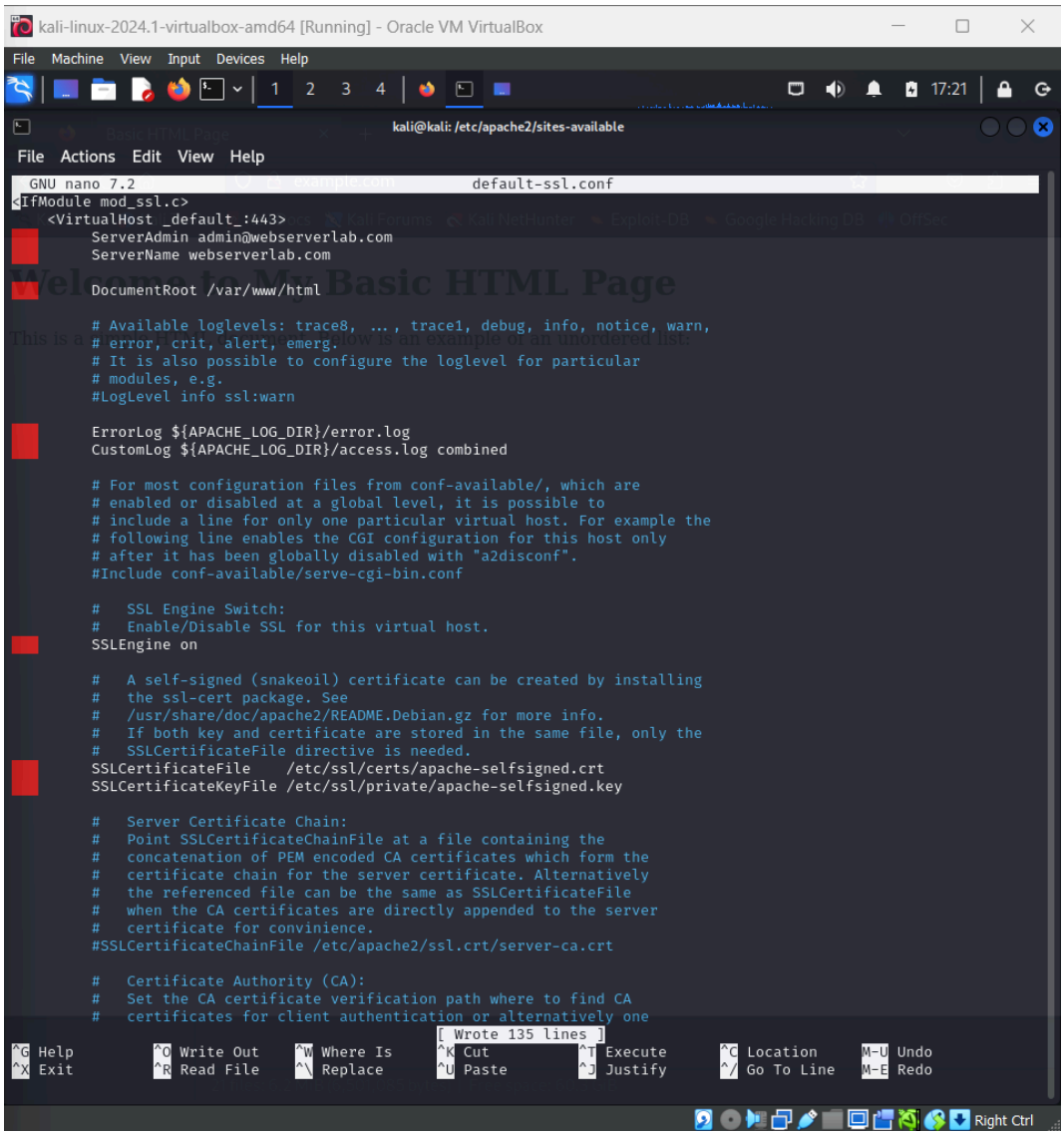
cp server.key server.pem
cat server.crt >> server.pem

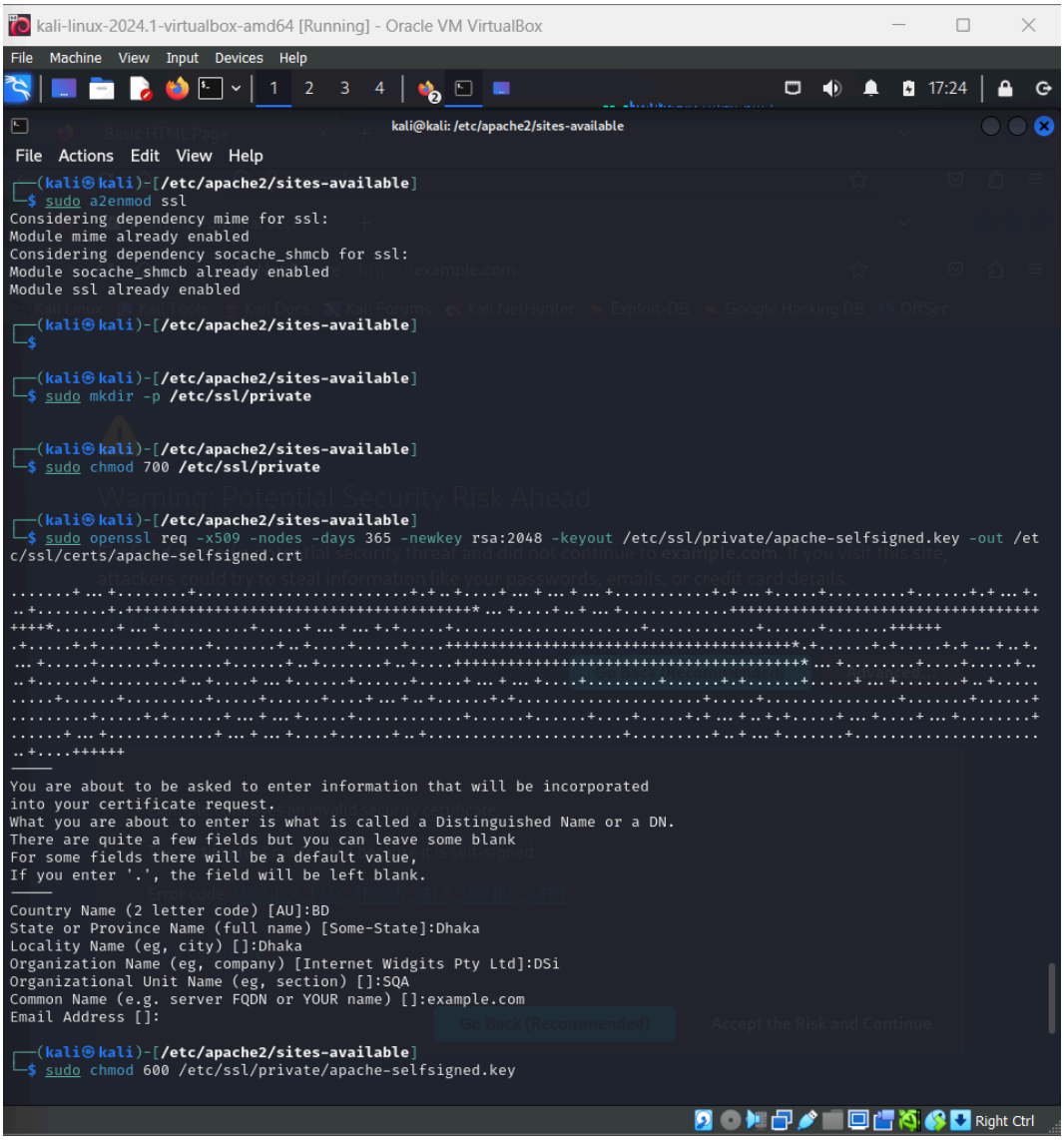


Step 2: Launch the web server using server.pem

openssl s_server -cert server.pem -www

By default, the server will listen on port 4433. You can alter that using the -accept option. Now, you can access the server using the following URL: https://example.com:4433/. Most likely, you will get an error message from the browser. In Firefox, you will see a message like the following: “example.com:4433 uses an invalid security certificate. The certificate is not trusted because the issuer certificate is unknown”.





Task-2: Deploy HTTPS into Apache

Now, you will deploy the HTTPS capability into Apache web server. At first, stop the Openssl webserver launched in the previous task. Now add the following lines into the example configuration file:

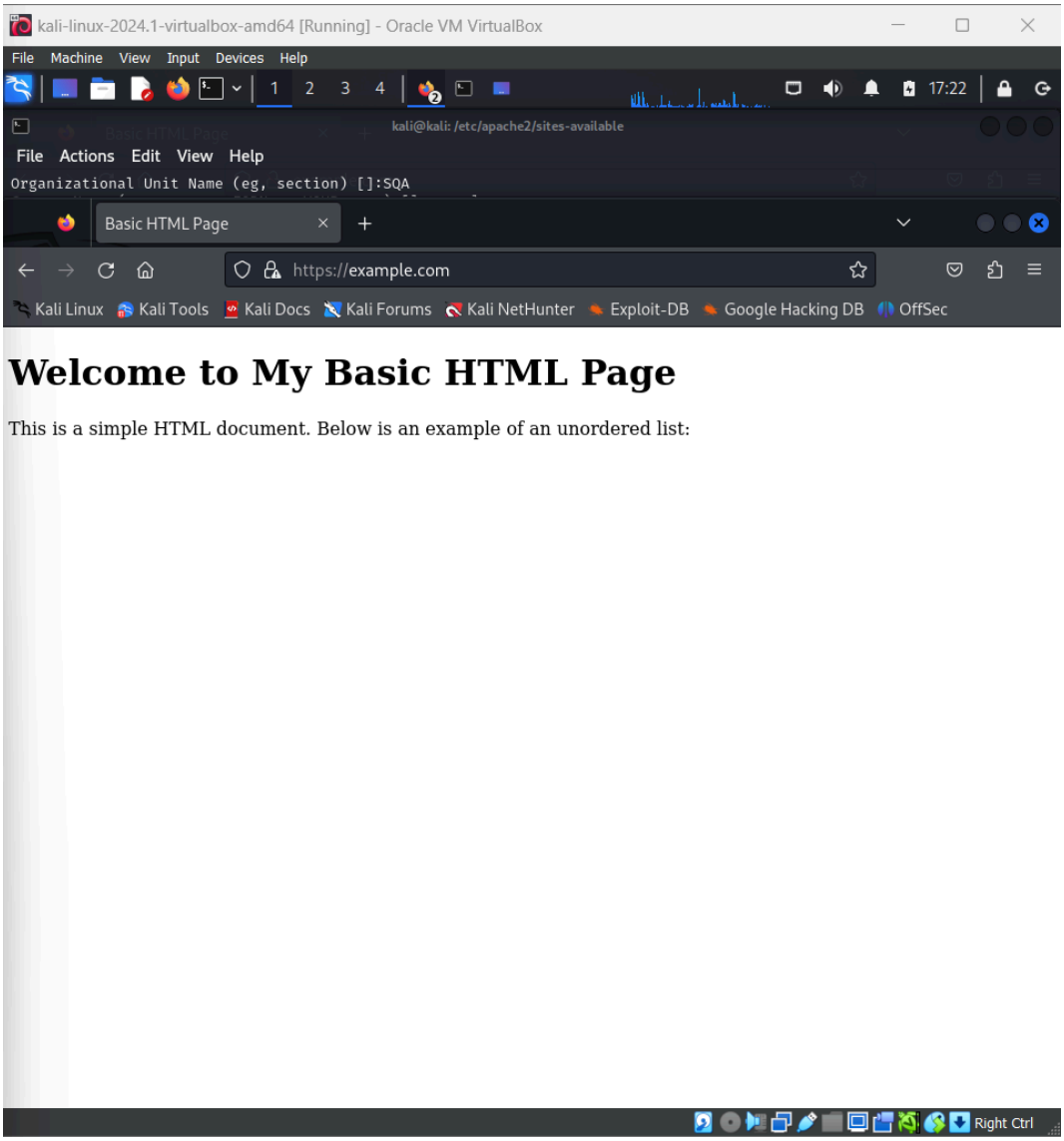
Apache is quite modular in the sense it supports the development of additional module which can add extended functionalities. For this lab, you will need to enable the ssl module in Apache which might not be enabled by default.

sudo a2enmod ssl

Next, use the following command to test the configuration.

sudo apache2ctl configtest

If a syntax is displayed onto the terminal, it indicates everything is okay.
Next restart the apache server using the restart command shown above.
Now, try to access the https://example.com. If everything is properly configured, you should be able to view the webpage in HTTPS.



If your browser is Firefox and it shows a warning, you can fix it by importing the CA certificate as described previously. If you use Chrome and it shows a similar warning, you can also import the CA certificate from the Manage certificate option under the Advanced setting in Chrome.
Checkpoint – 3 (5 marks): Access the https://example.com in your browser.