Public-Key Infrastructure (PKI) Lab Report

Mudit Vats
mpvats@syr.edu
3/2/2019

Table of Contents

Overview

This lab report presents observations and explanations for the tasks described in the Public-Key Infrastructure (PKI) Lab.

Please note, I updated the icons and some theming elements to make the UI a little more pleasing to the eye. It's still the Ubuntu 16.04 Seed VM.

Task 1: Becoming a Certificate Authority (CA)

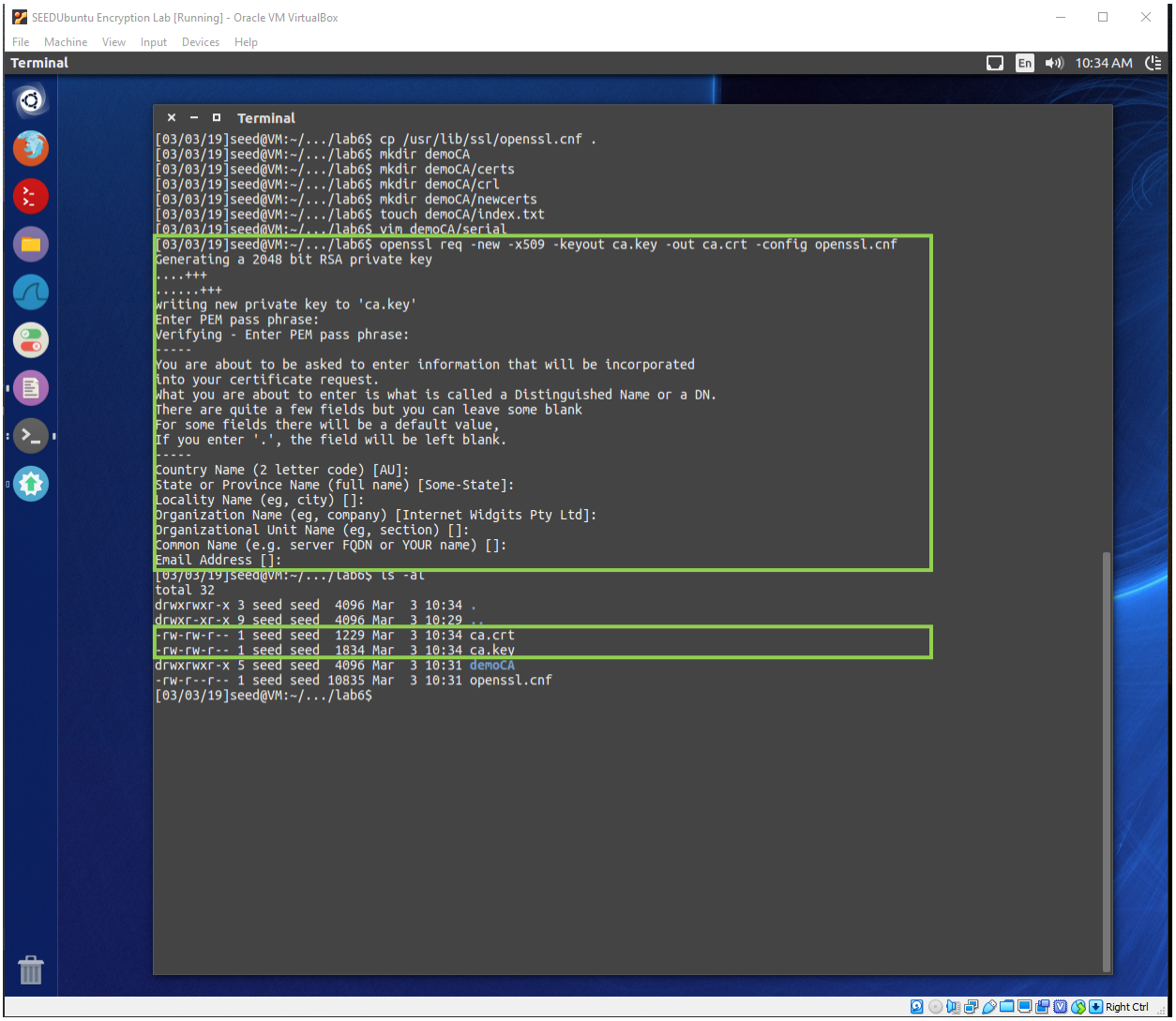
In the figure below, we show the steps to create a self-signed certificate. This self-signed certificate and private key will be used as our Certificate Authority to sign our server's certificate in the next task.

The steps below are as follows:

1. We copy the openssl.cnf file which contains various configuration settings used by the OpenSSL command.
2. We create the demoCA and several subdirectories for the new certs we create, serial number and database file.
3. We then execute the command – "openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf". This command does the follows:
    a. "openssl req -new" – Instructs OpenSSL that we have a new request.
    b. "-x509" – Output x509 certificate format.
    c. "-keyout ca.key" – Private key is ca.key.
    d. "-out ca.crt" – Public key is ca.crt.
    e. "-config openssl.cnf" – Use this configuration file.

The end result is that we have a ca.key, which is our private key and a ca.crt, which is a self-signed certificate that contains our public key.

We can now use the private key to sign other certificates and the public certificate can be used the verify the signed certificate.



Observations / Explanations

This task focused on created a self-signed certificate to act as our Certificate Authority that will sign our server's public key; i.e. create a certificate which the server will send down to the web browser (client) when the https session is initiated.

This certificate is self-signed, meaning that not "official" CA is vouching for it, we are vouching for our own certificate, which is fine for experimentation.

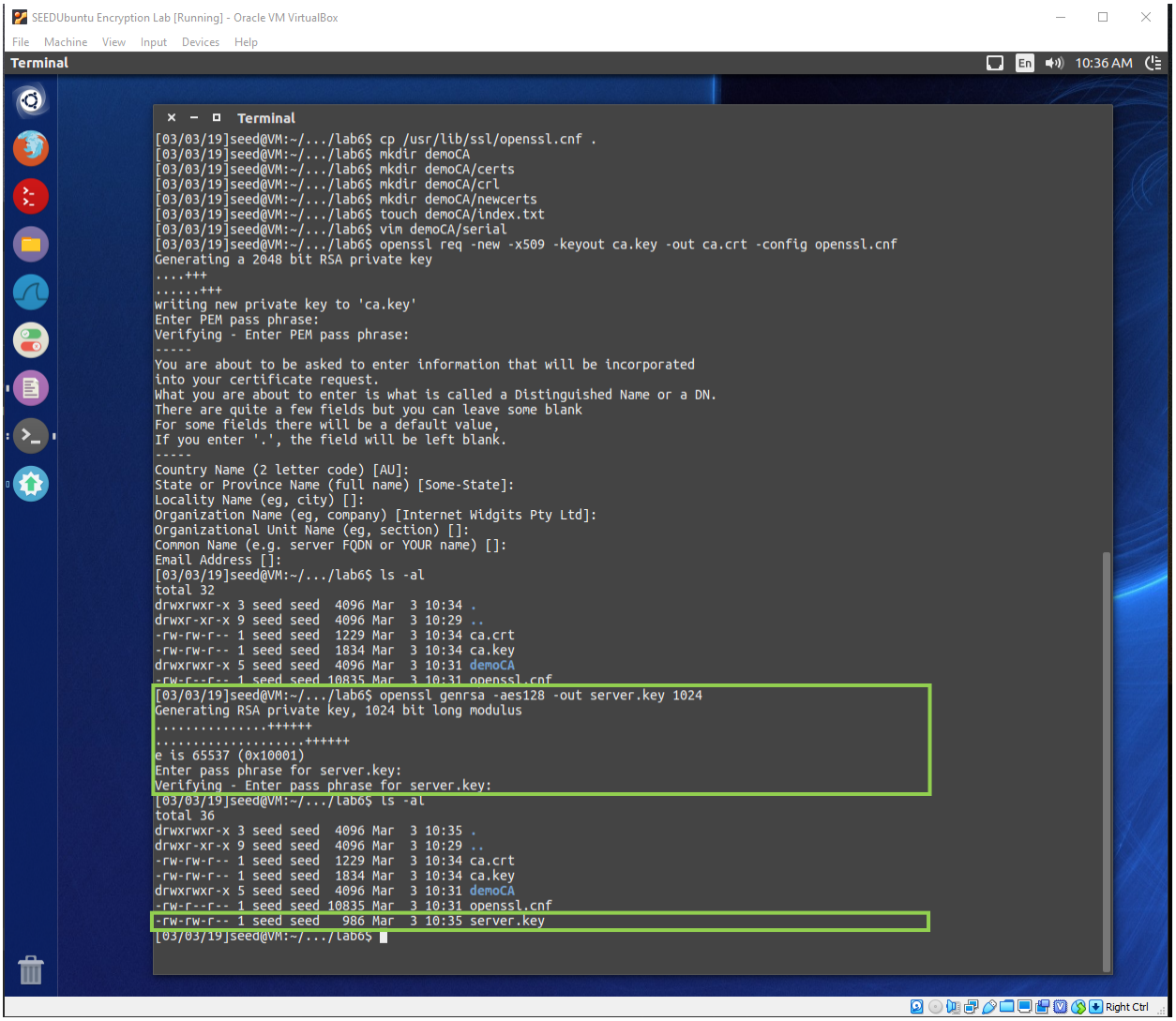
Task 2: Creating a Certificate for SEEDPKILab2018.com

In the following sections, we will go through the three steps to create the certificate for our web site.

Step 1: Generate public/private key pair.

In the figure below, we can see the generation of the server public/private key pair. The file server.key contains both the public and private key.

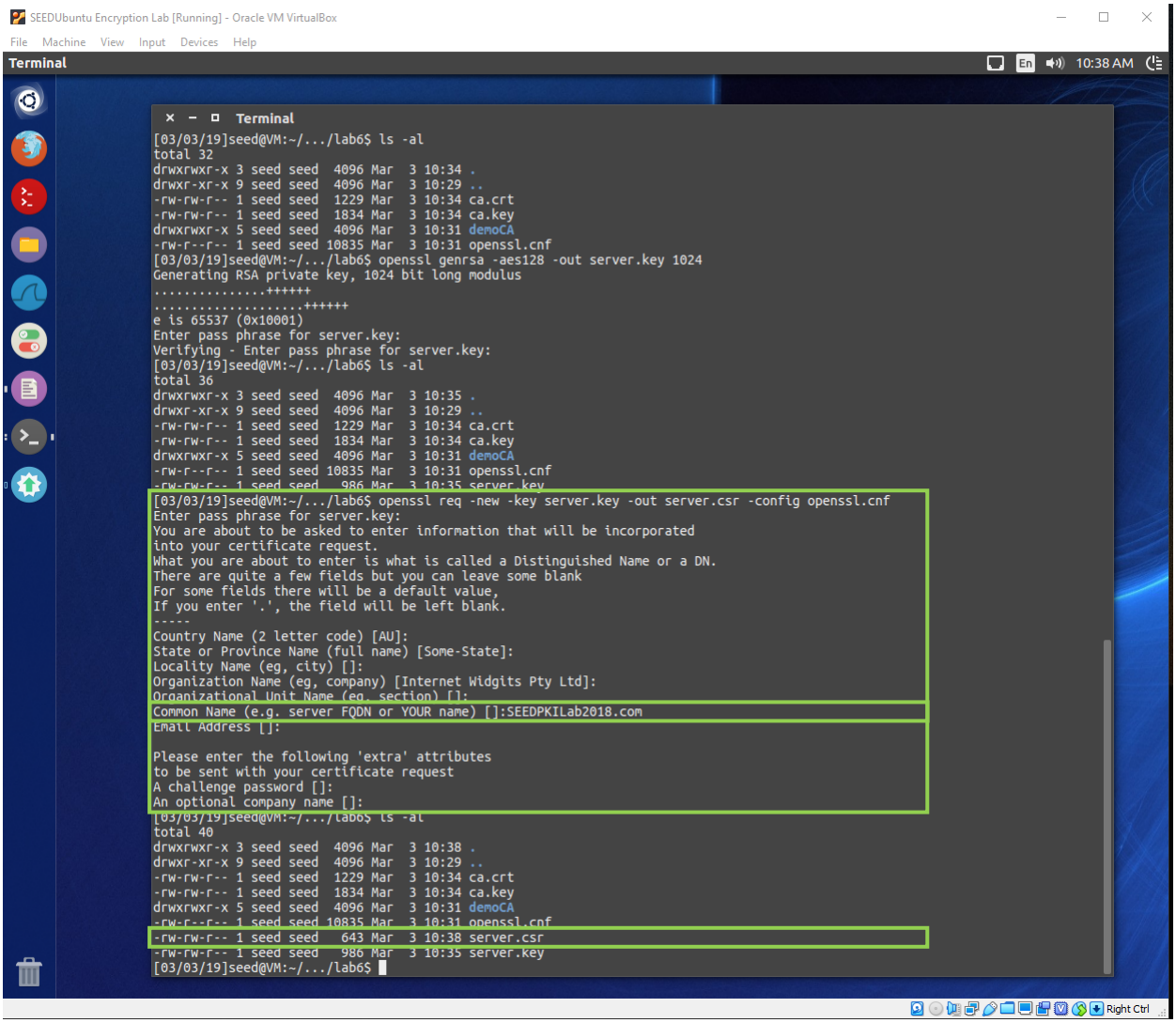
We execute the command "openssl genrsa -aes128 -out server.key 1024" which generates an RSA public / private key pair using the AES128 algorithm.



Step 2: Generate a Certificate Signing Request (CSR)

In the figure below, we generated the CSR. We see the server.csr in the current directory. We can also see, above that, that we specified the Common Name to be SEEDPKILab2018.com.

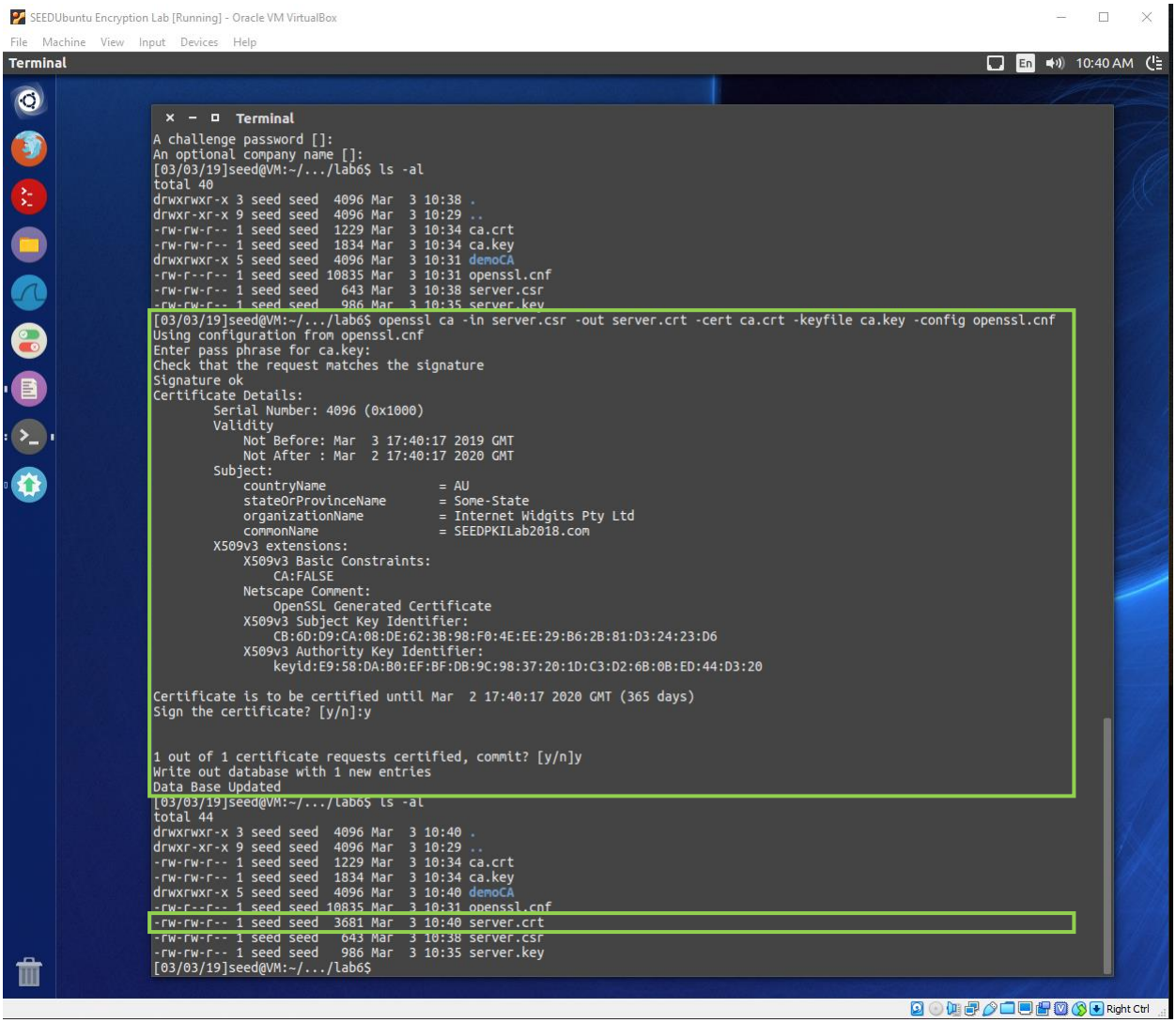
We execute the command "openssl req -new -key server.key -out server.csr -config openssl.cnf" to create the new csr (server.csr). As mentioned, we take care to specify the common name so it matches our website's name.



Step 3: Generating Certificates

In the figure below, we use the CSR and CA private key as input and generate a signed certificate for the server.

We execute the command "openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key -config openssl.cnf" to generate the server.crt which is the server's public key, plus some metadata (Common Name, certificate expiry etc) and signed with the CA's key file.



Observations / Explanations

This task focused on ultimately generating a server certificate which is the server's public key and some additional metadata (e.g. Common Name) signed by the Certificate Authority's private key (ca.key).

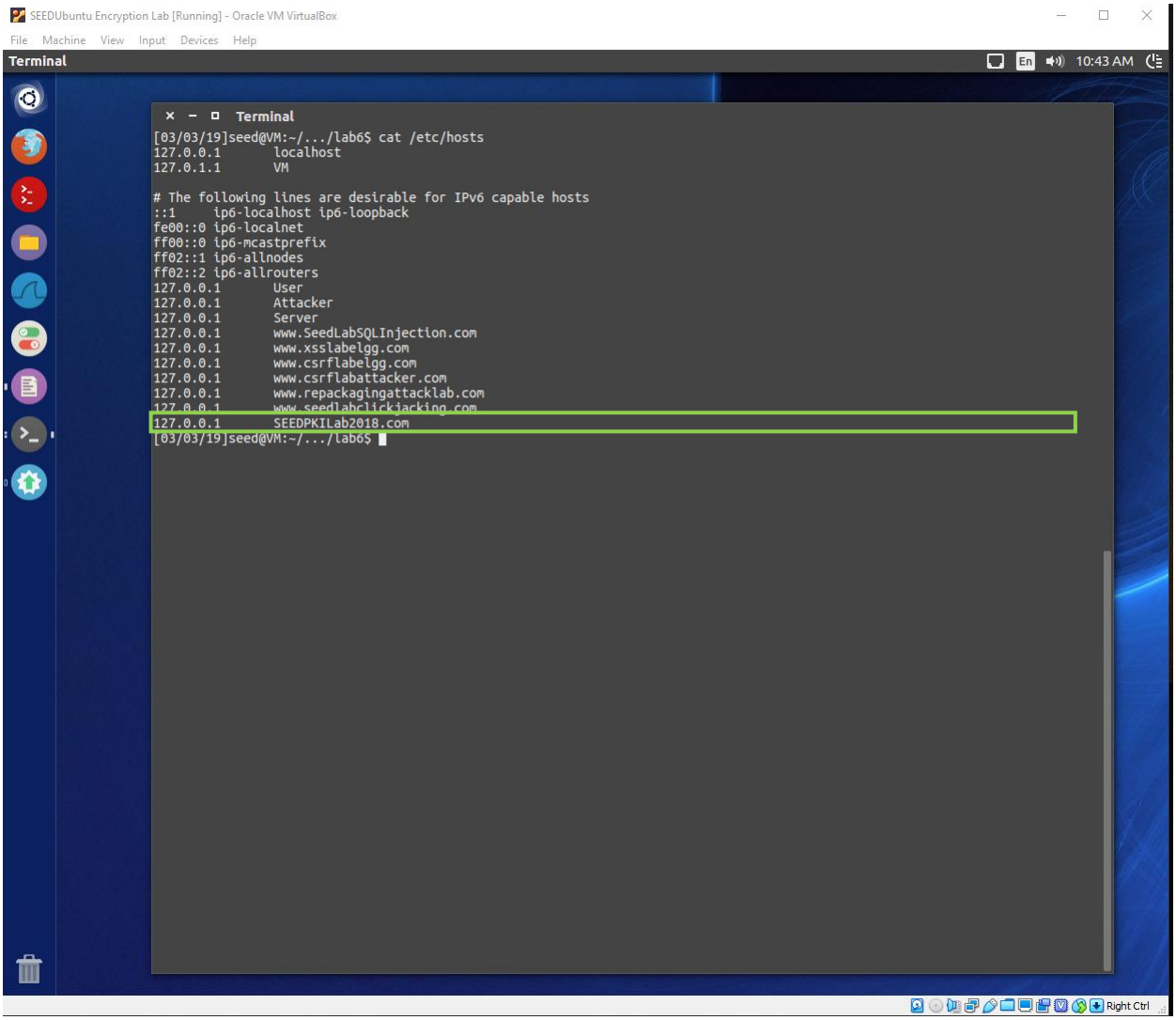
This server certificate is sent to the client when an https connection is made to the server. The client will then verify the server's certificate using the CA's certificate which we will load into the web browser.

Additionally, the web browser will use the Common Name to ensure the to double-check that the server certificate which was sent and the site that the user is accessing is the same (i.e. as specified in the URL and in the Common Name of the certificate). If so, the connection proceeds. If not, there may be an invalid certificate so the connection does not proceed.

Task 3: Deploying Certificate in an HTTPS Web Server

Step 1: Configuring DNS

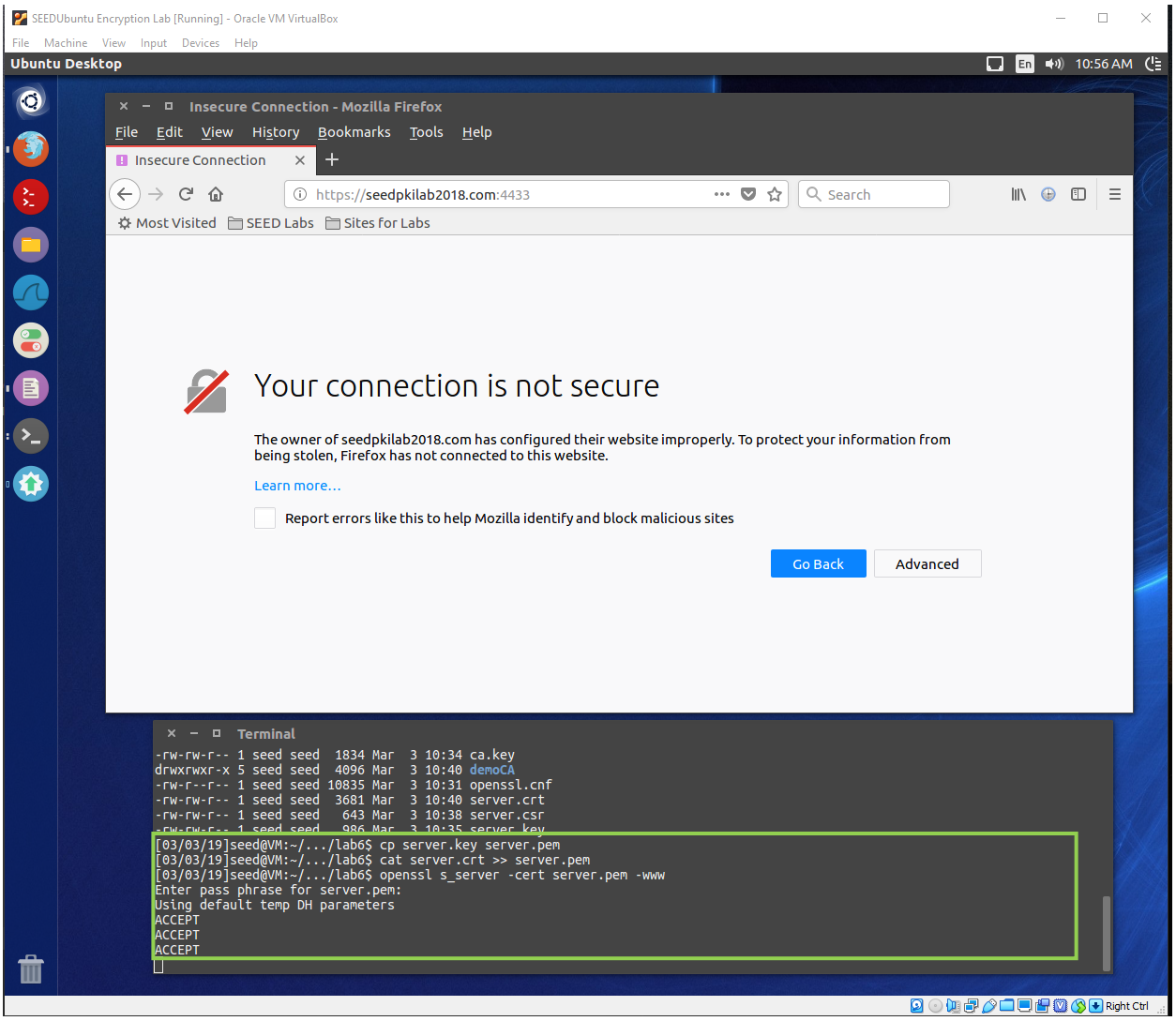
In this step, we point the SEEDPKLab2018.com name to our localhost since we'll be running the website on the localhost.



Step 2: Configuring the Web Server

In the figure below, we see us combining the private key and public certificate into one file which is used by the web server to 1) send down the servers certificate and 2) use the private key to decrypt data from the client (web browser).

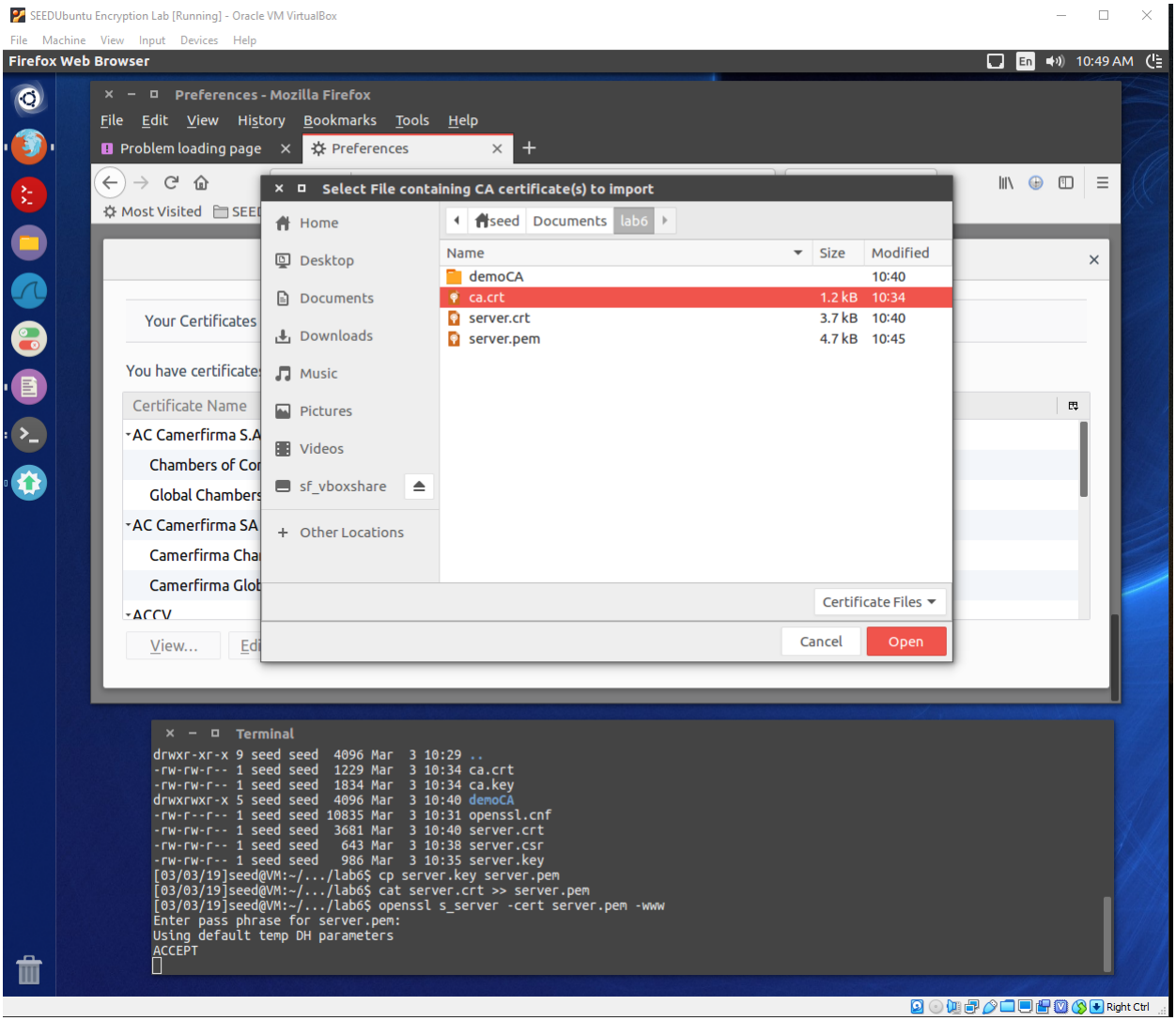
We can also see that without the Certificate Authority's certificate (ca.crt) added to the browsers certificates, we get an error when we try to access the https://seedpkilab2018.com:4433 website.



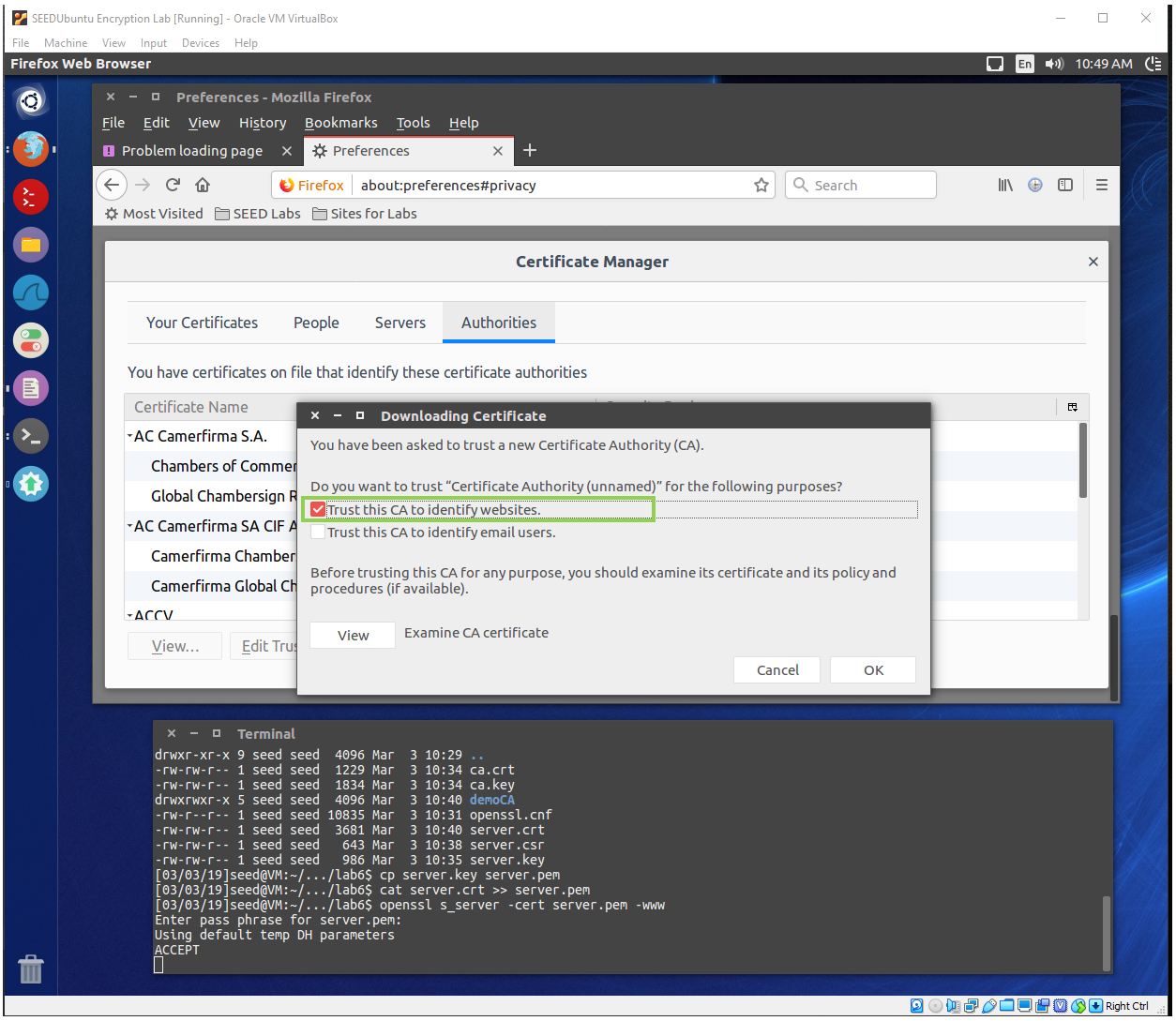
Step 3: Getting the browser to accept our CA certificate

In the next few figures, we see how the ca.crt is being added to the browser.

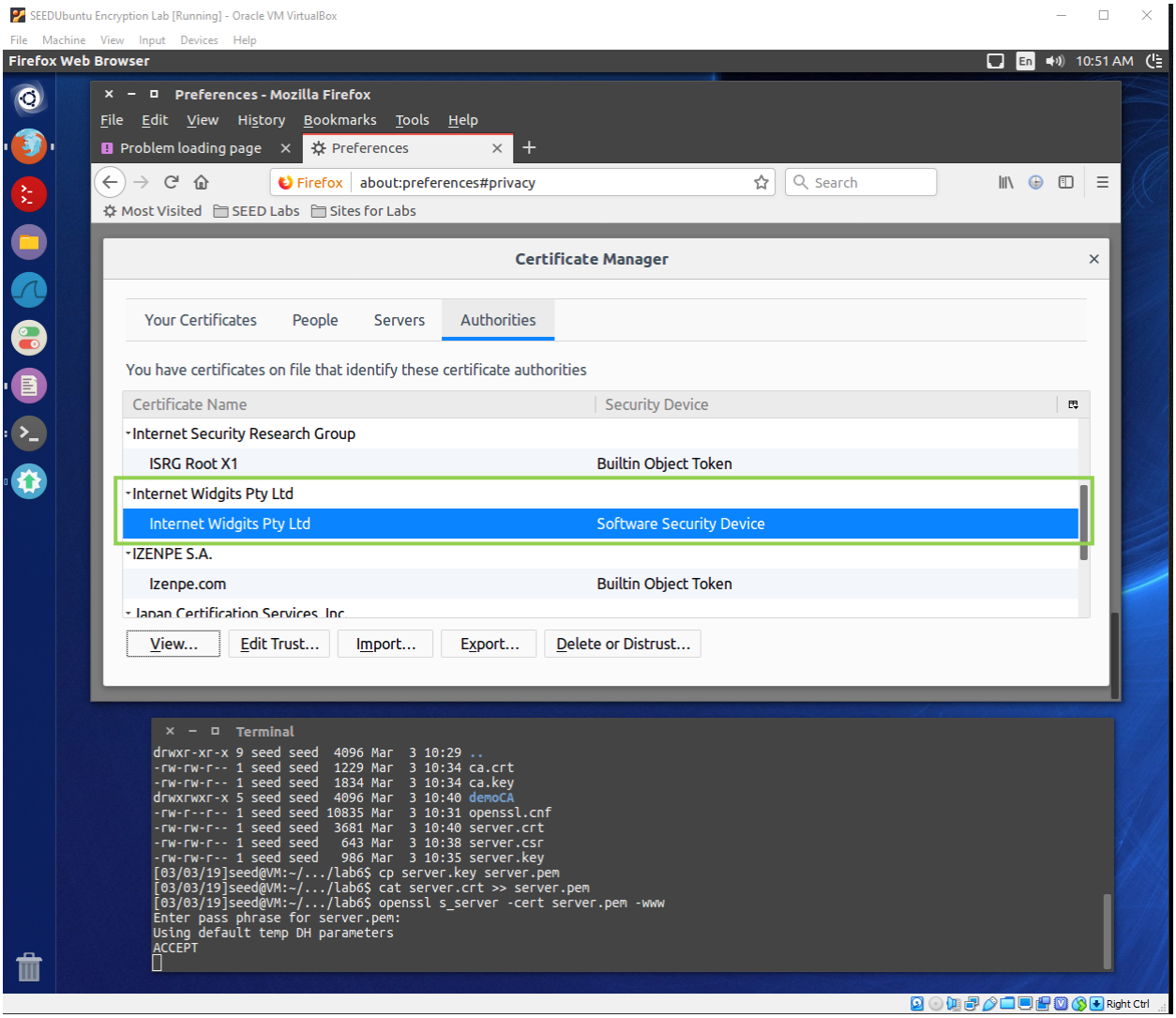
First, we select the certificate ca.crt.



We then specify that we should trust this certificate to identify websites. This means that if a website was signed by this CA, we should allow this CA certificate to verify the website's certificate.

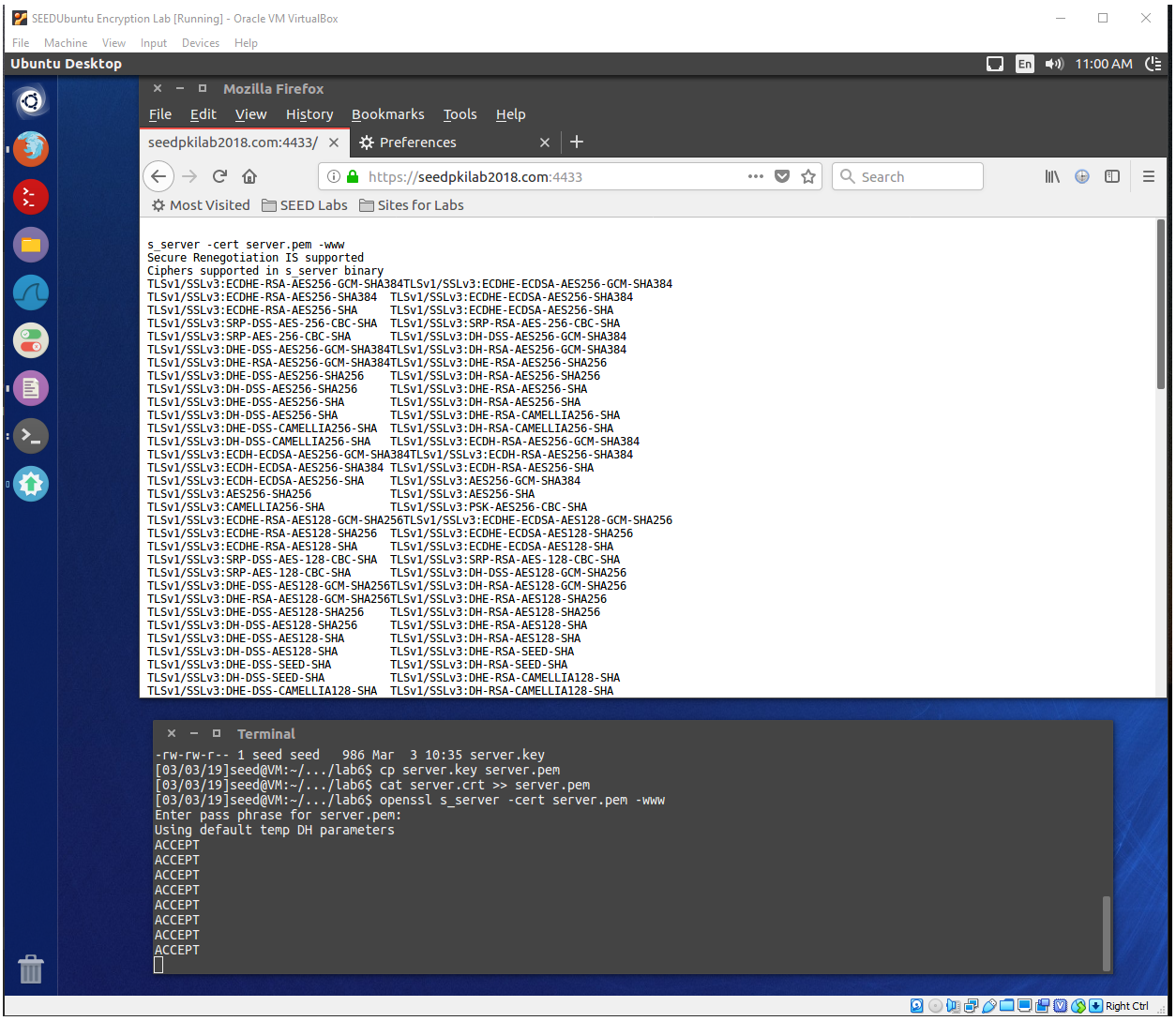


Finally, we see that our ca.crt is imported into the browser.



Step 4: Testing our HTTPS website

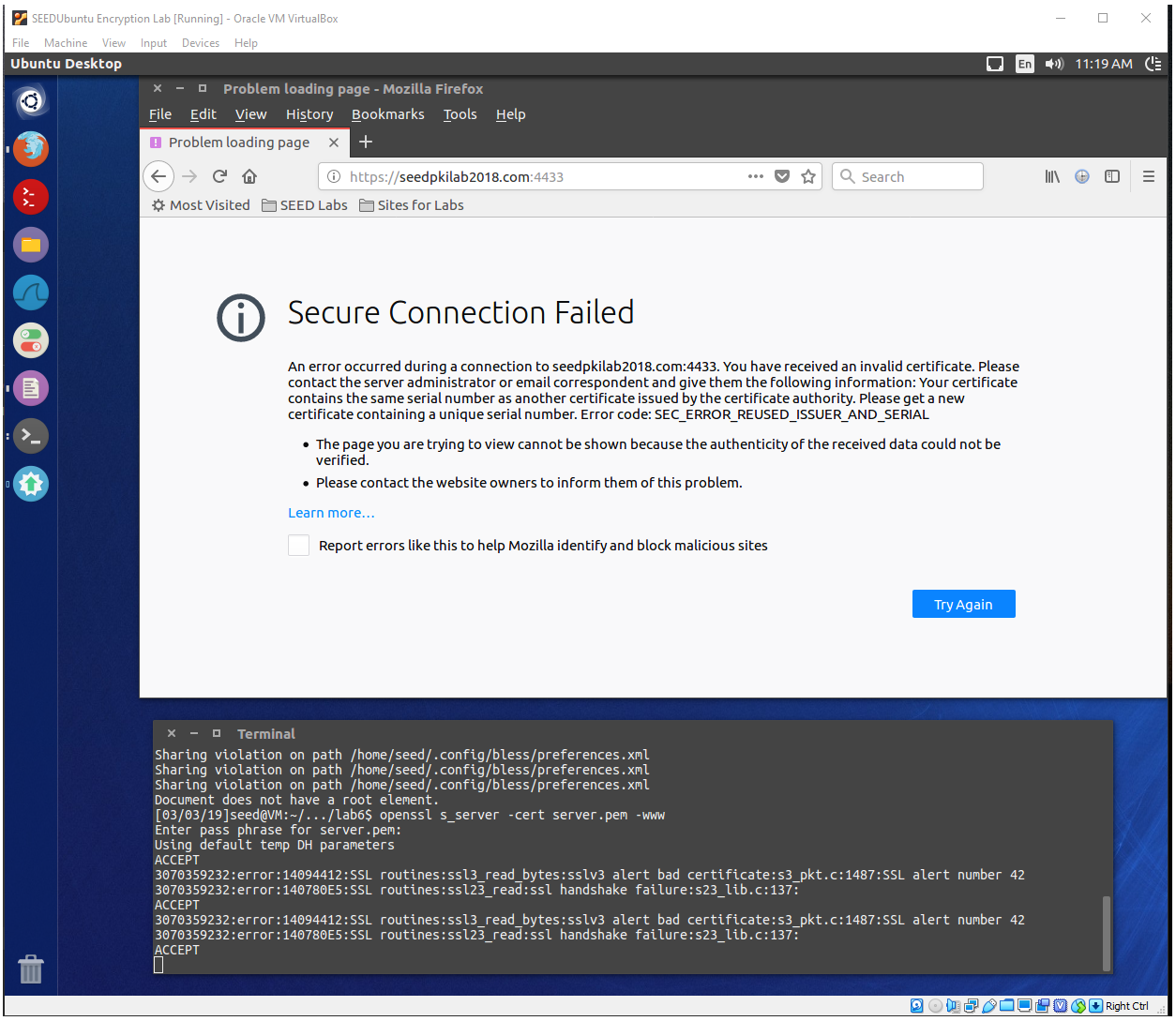
In the figure below, we can see that accessing our website through and https connection now works!



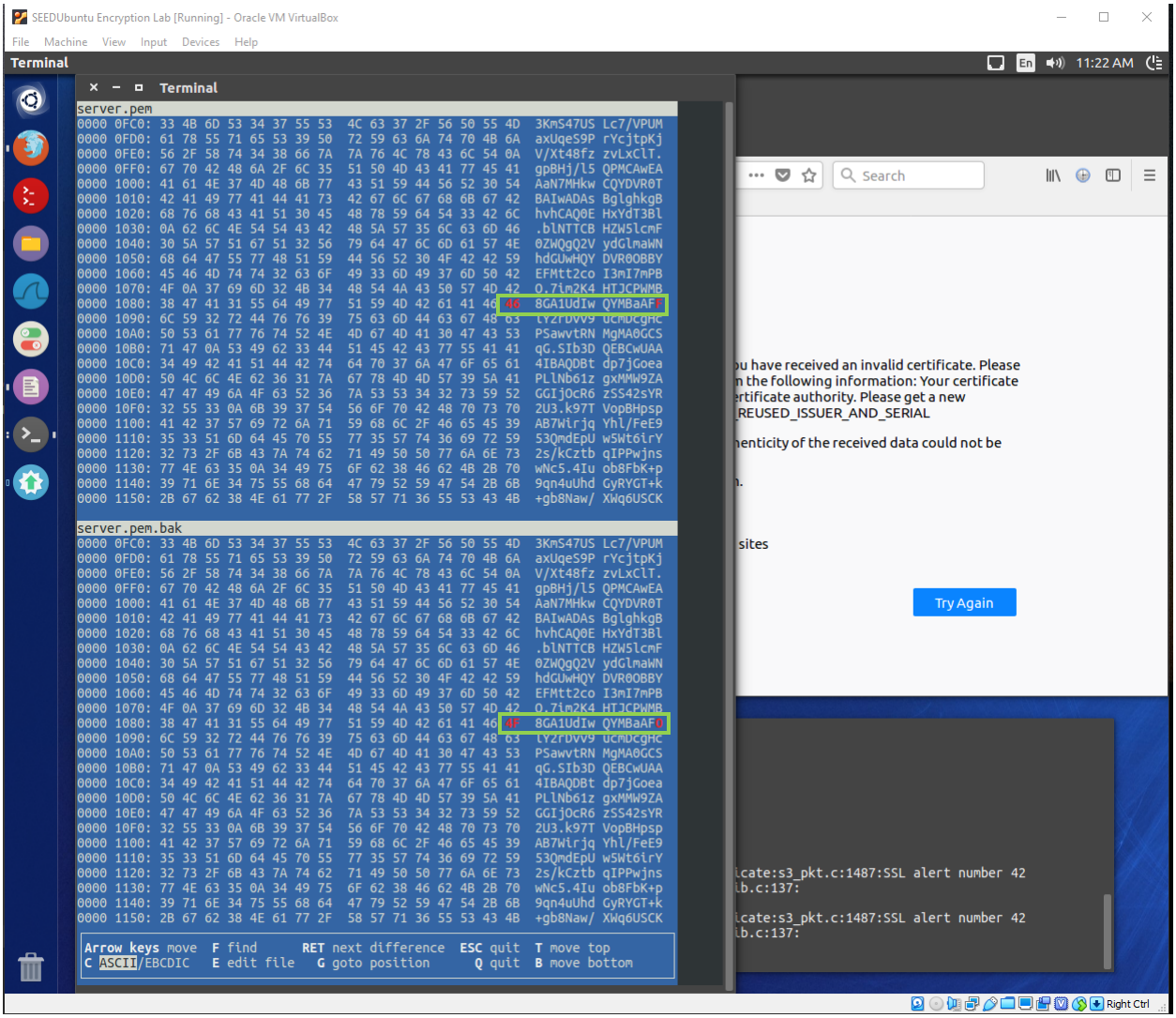
Modify a single byte of server.pem

We modify a single byte of the server.pem file and attempt access to the website.

Notice, we see the Secure Connection Failed / invalid certificate message.



In the figure below, we see the byte that was modified. I modified a single byte between the "---BEGIN CERTIFICATE---" and "---END CERTIFICATE---" section of the server.pem file. Please note, the server.pem is on top and the server.pem.bak (unmodified) is on the bottom.



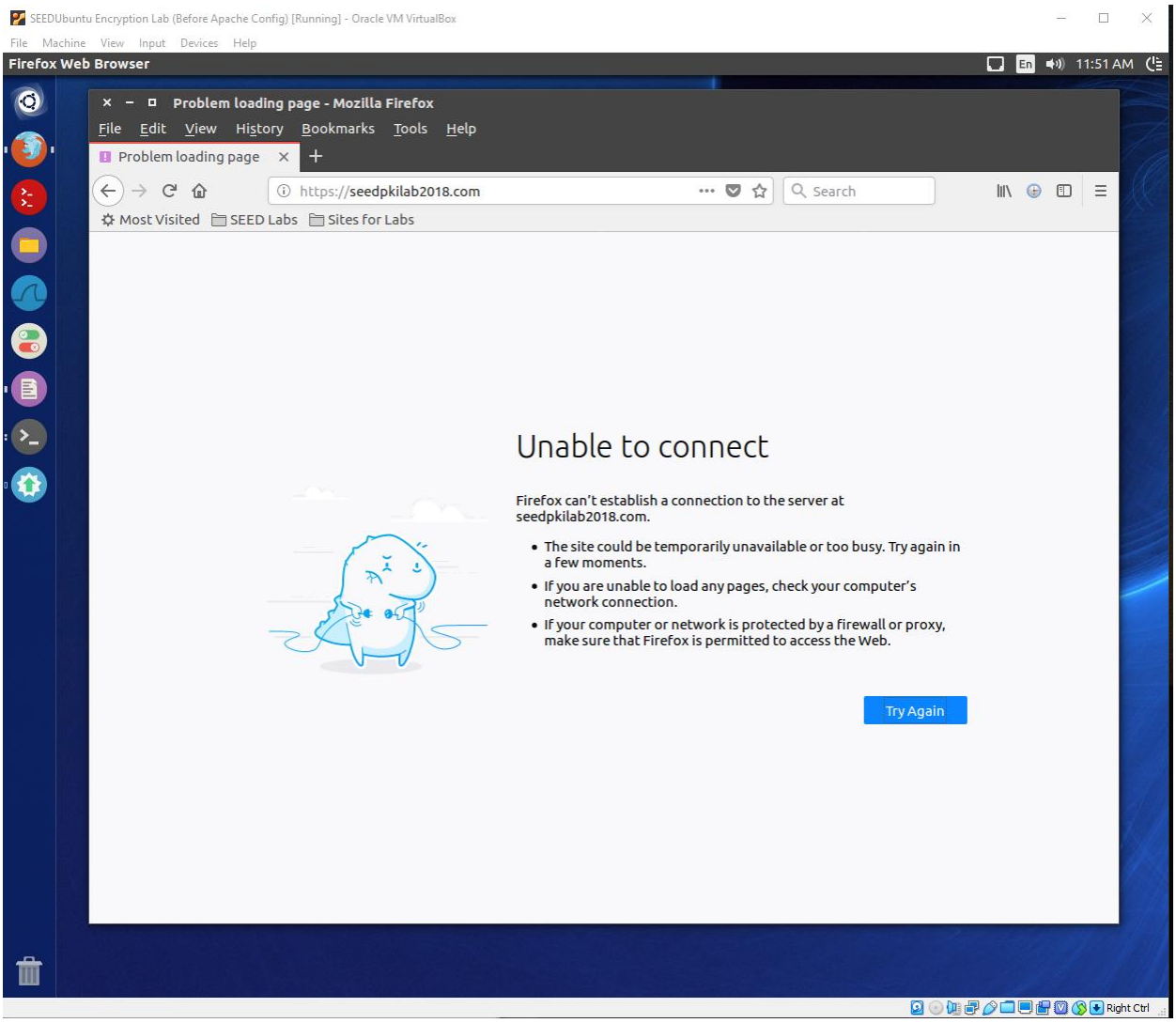
Observations / Explanations

We can see that when the web browser has the CA's certificate to verify the server's certificate, the https connection works without issue; i.e. we see the website ok.

When we modify a byte in the server's certificate signature section, we can start the website, but we get an invalid certificate error when the browser tries to use the CA certificate to verify the server's certificate. Since we modify the signature in the server's certificate, verification will fail… which is does.

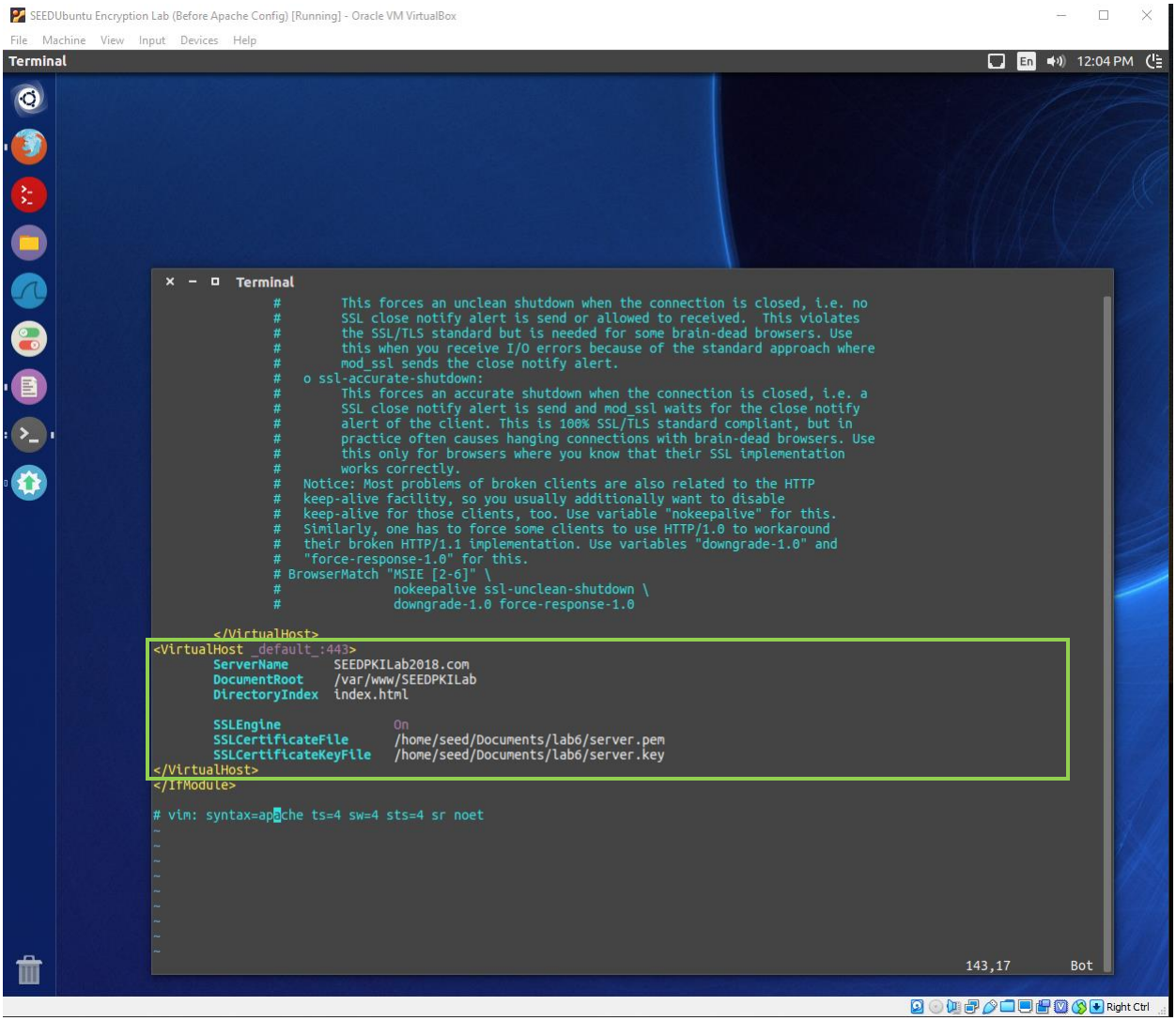
Task 4: Deploying Certificate in an Apache-Based HTTPS Website

In the figure below, when we try to access the https://seedpkilab2018.com website, it fails with error.



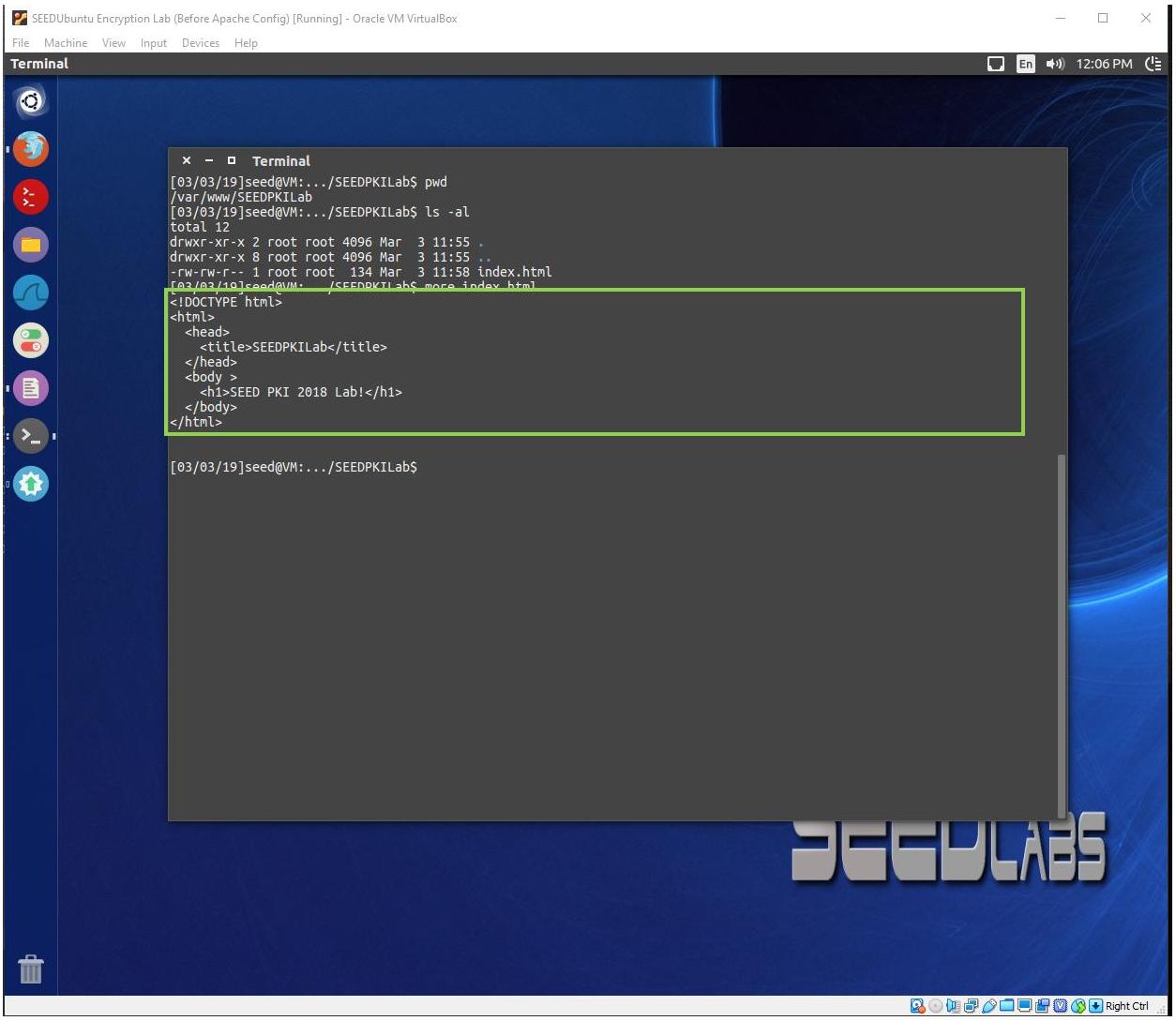
In the figure below, we see the edit of /etc/apache2/sites-available/default-ssl.conf to add this https website and associated information about the website:

- ServerName – Name of our site SEEDPKILab2018.com.
- DocumentRoot – Location of web files /var/www/SEEDPKILab.
- DirectoryIndex – HTML file that will load on website access.
- SSLEngine – SSL on for this site.
- SSLCertificateFile – Server's certificate file.
- SSLCertificateKeyFile – Server's private key file.

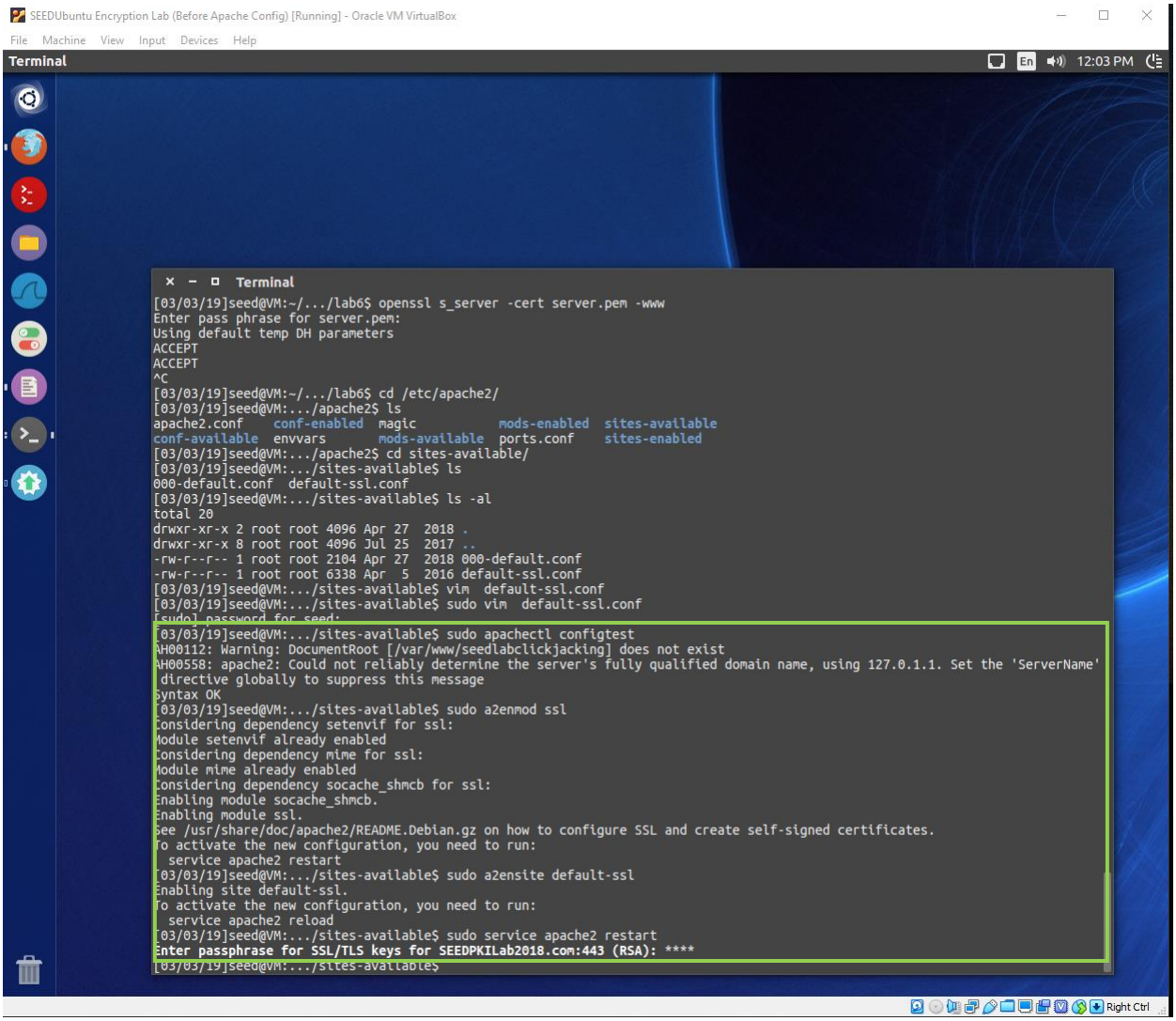


In the figure below, we see the index.html file for the SEEDPKI2018.com website. This is located in /var/www/SEEKPKILab.

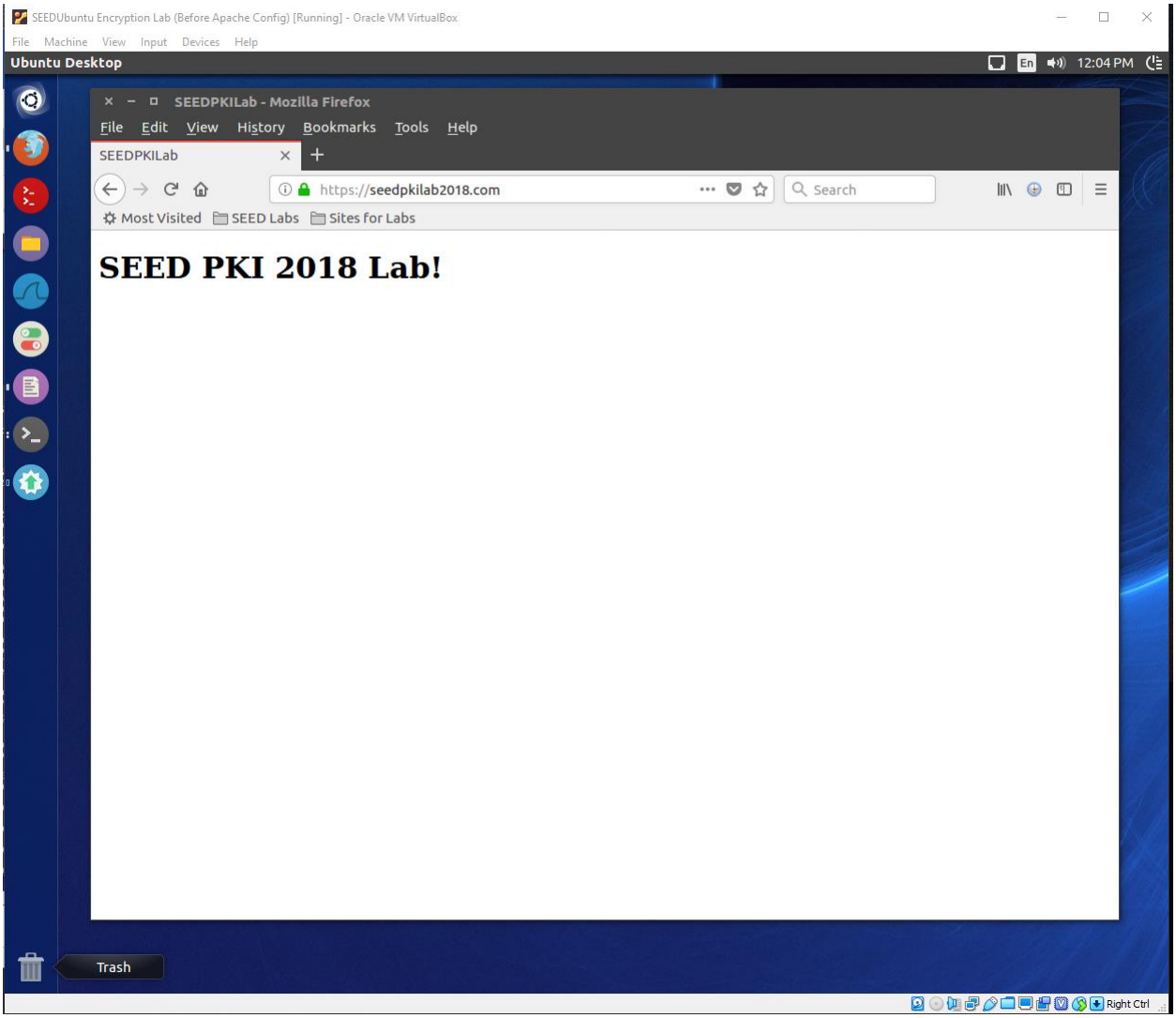
The only thing we do for this website is print the message “SEED PKI 2018 lab”. I should’ve changed it to 2019, but since our site is “2018” per lab instructions, I kept it the same.



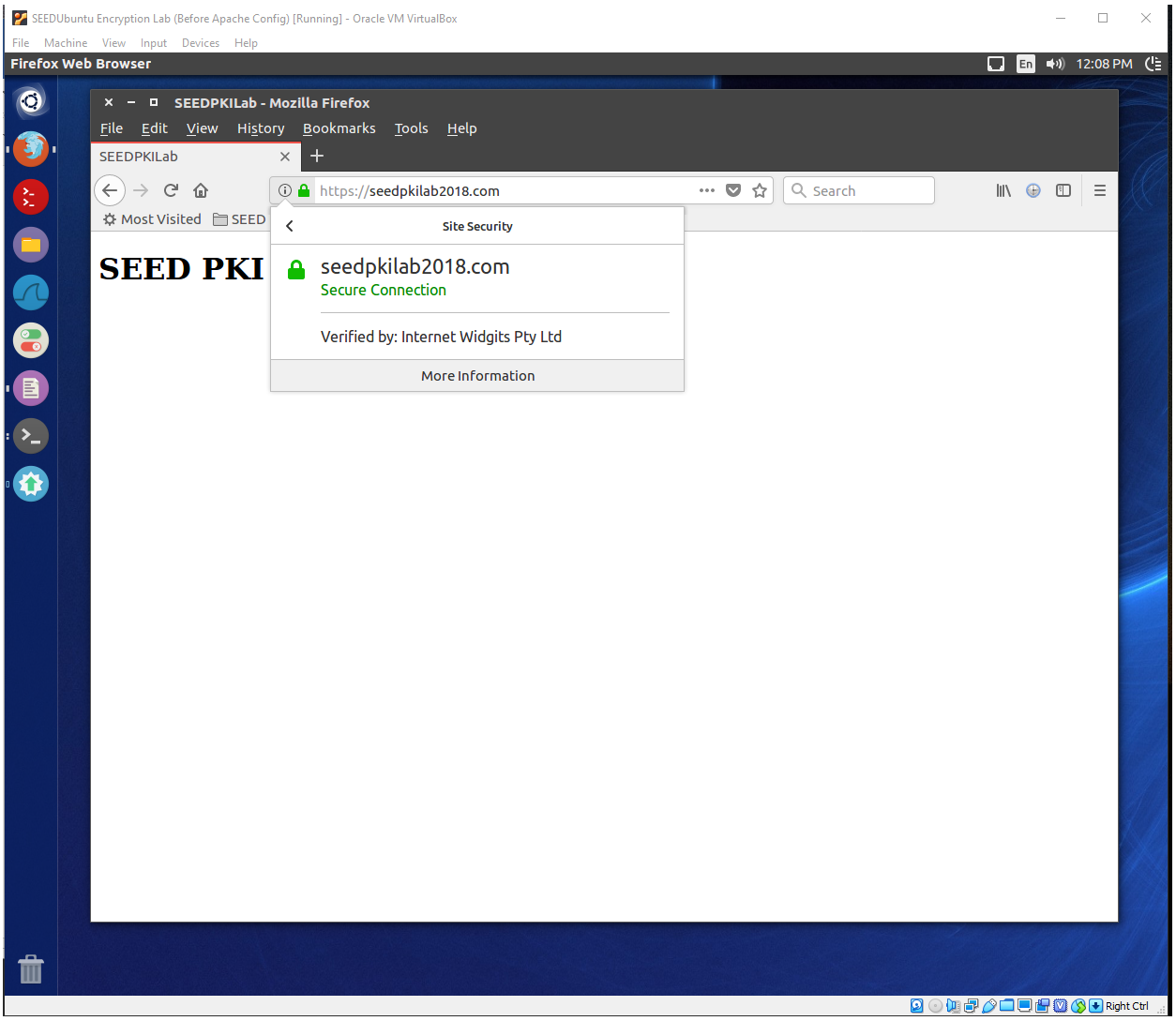
In the figure below, we see the commands to add/configure the website for SSL with the Apache web server.



In the figure below, we see the successful display of the index.html file we created for the HTTPS SEEDPKI2018.com website!



In the figure below, when we click on the lock icon, we can see that the server's certificate is "Verified by: Internet Widgets Pty Ltd" which is the CA that we created.



Observations / Explanations

In this task, we enabled https for our web site in the Apache server. We created a simple website, added and entry in the VirtualHost section for 443 (TLS/SSL sites) for our website and configured Apache to accept the new configuration. Finally, when we accessed the website, we were able to successfully make an https connection and view the web page.

We also examined the certificate to ensure it's being verified by the correct CA – our CA certificate that we imported, which it is.