**Name:** Henry Salgado **ID:** 80509684

**Assignment 5:**

**Question Part A - Diffie-Hellman Key Exchange:**

Given:

* Prime p = 17
* Generator g = 4

Alice's private exponent a = 3

Bob's private exponent b = 6

**Step 1: Calculating public key for Alice**

A = g^a mod p

A = 4^3 mod 17 = 64 mod 17 = 13

**Step 2:** **Calculating public key for Bob**

B = g^b mod p

B = 4^6 mod 17 = 4096 mod 17 = 4

**Step 3:** Calculating shared secret “s” using Bob's public key B.

s = B^a mod p

s = 4^3 mod 17 = 64 mod 17 = 13

**Step 4:** Calculating the shared secret “s” using Alice's public key A.

s = A^b mod p

s = 13^6 mod 17 = 4826809 mod 17 = 13

Shared secret that both Alice and Bob calculate is **13.**

**Question 1 - Part 2**

**Given:**

* Prime p = 17
* Prime q = 7

**Step 1:** Calculating the modulus n

n = p \* q =

n = 17 \* 7 = 119

**Step 2:** Calculating the totient of n: φ(n).

φ(n) = (p - 1) \* (q - 1)

φ(n) = 16 \* 6 = 96

**Step 3:** Choosing a public exponent e that is coprime to φ(n). I will choose e = 5.

**Step 4:** Calculating d such that (d \* e) mod φ(n) = 1.

1 = (d \* 5) mod (96)

(d \* 5) = 1 (mod 96)

the gcd of 96 and 5, and the coefficients x and y:

96 = 19 \* 5 + 1

5 = 5 \* 1 + 0

In this case, x = -19 and y = 1. Since we are working with modular arithmetic, we can add or subtract multiples of the modulus (96) to x to make it positive while keeping the same residue. We find the smallest positive value of x by adding the modulus:

d = x + φ(n) = -19 + 96 = 77

Therefore, the modular multiplicative inverse of 5 modulo 96 is 77. This means that (77 \* 5) = 1 (mod 96).

d = 77

**Step 5:** Now, we know that

Alice's public key: (n, e) = (119, 5)

Alice's private key: (n, d) = (119, 77)

Bob's public key: (n, e) = (119, 5)

Bob's private key: (n, d) = (119, 77)

**Step 6**: Alice and Bob can now use each other's public keys to encrypt messages and their own private keys to decrypt messages.

For example, if Alice wants to send a message M = 42 to Bob, she would compute:

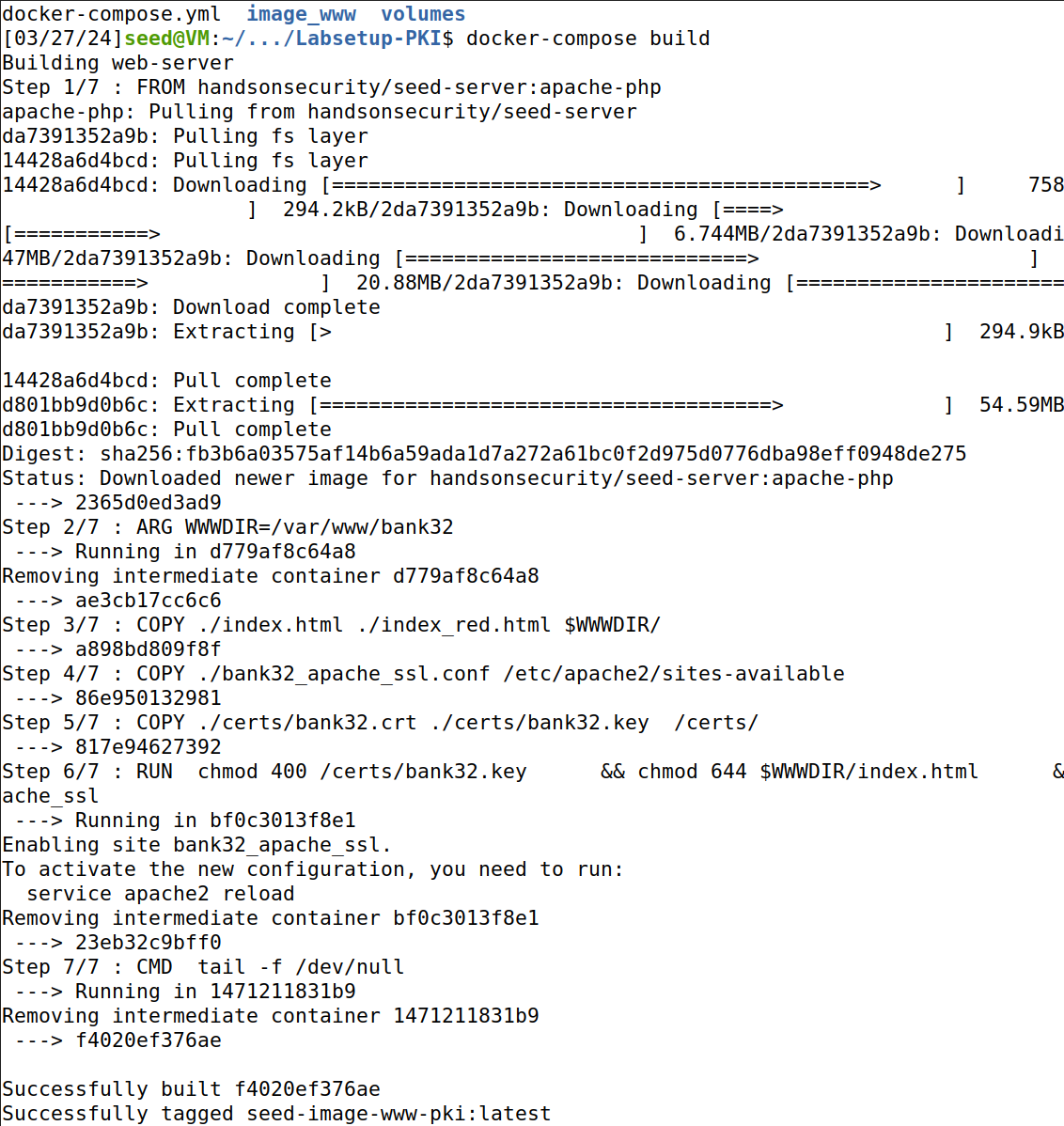
C = M^e mod n = 42^5 mod 119 = 63

Bob can then decrypt the ciphertext C using his private key:

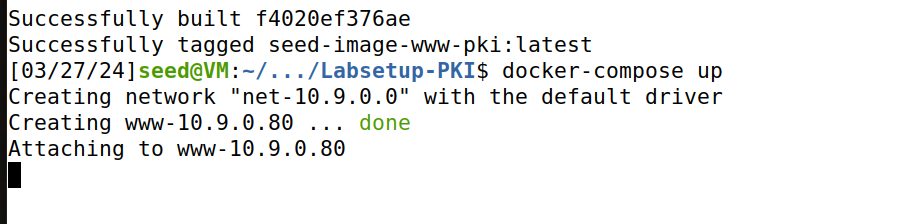
M = C^d mod n = 63^77 mod 119 = 42

**Task Container Setup: Public-Key Infrastructure (PKI)**

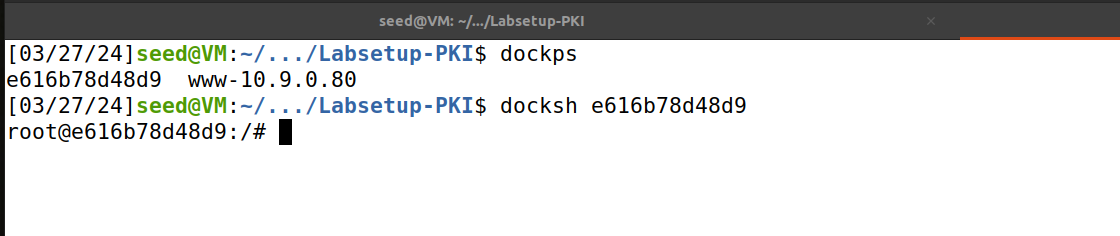
# Building the container image



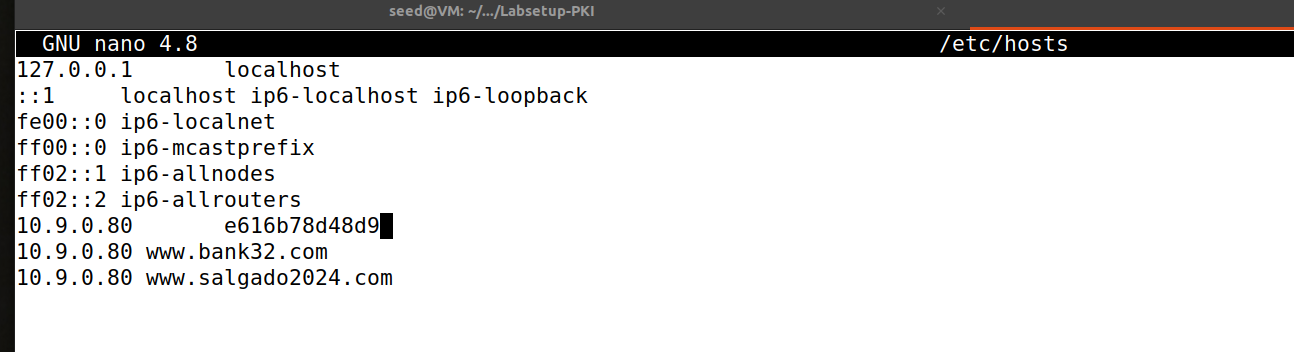
# Starting the container



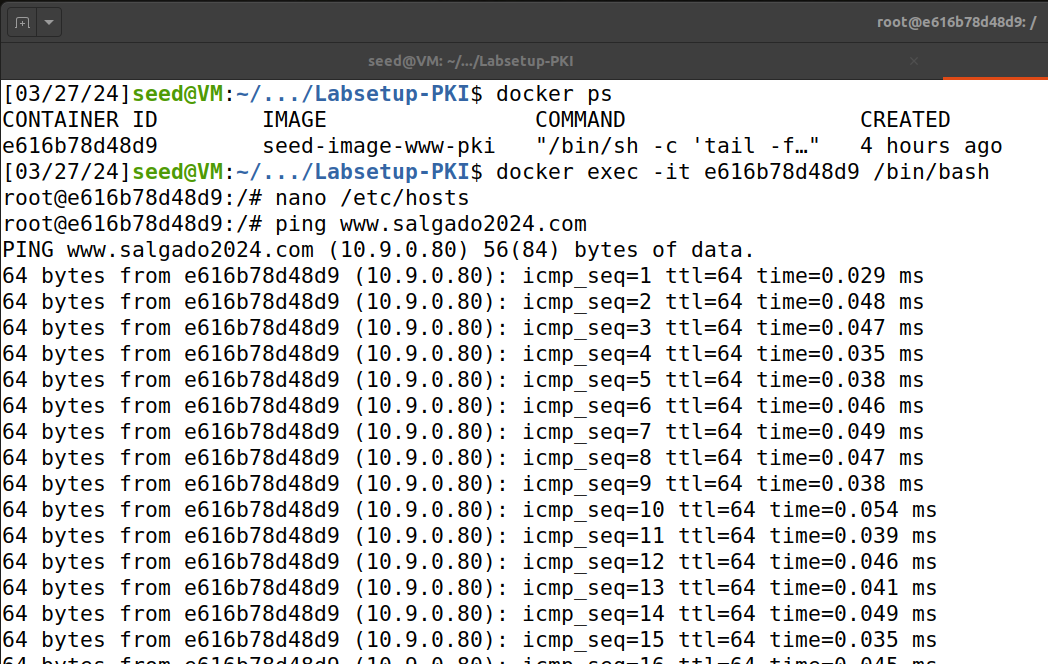
# Starting a shell in the container



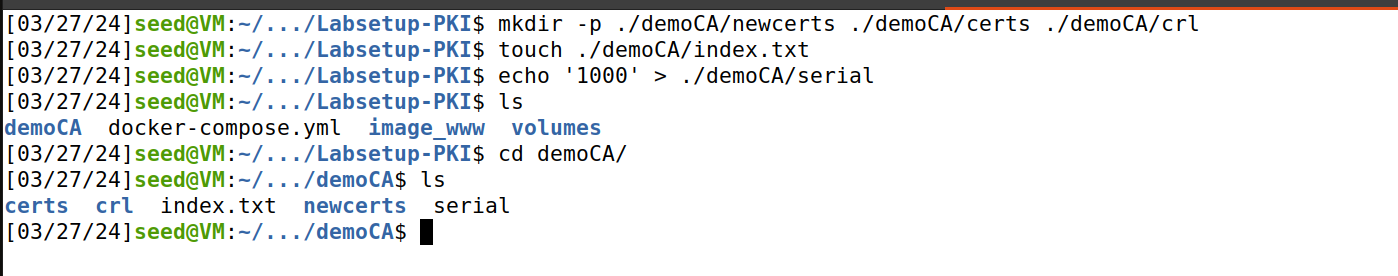
#Updating etc/hosts



# Ping my site command is successfully reaching the IP address associated with this domain (10.9.0.80)



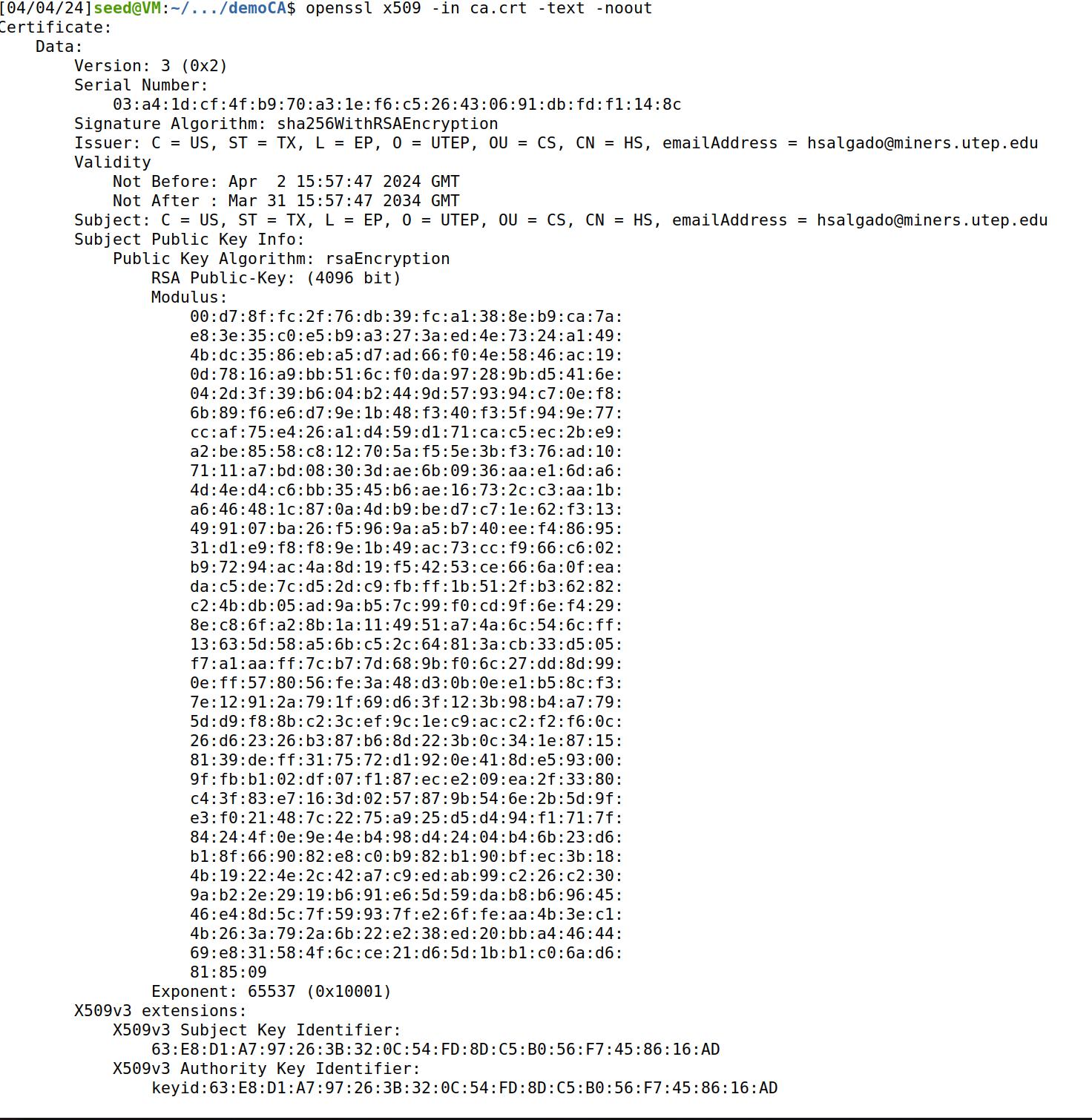
Making files and Directories

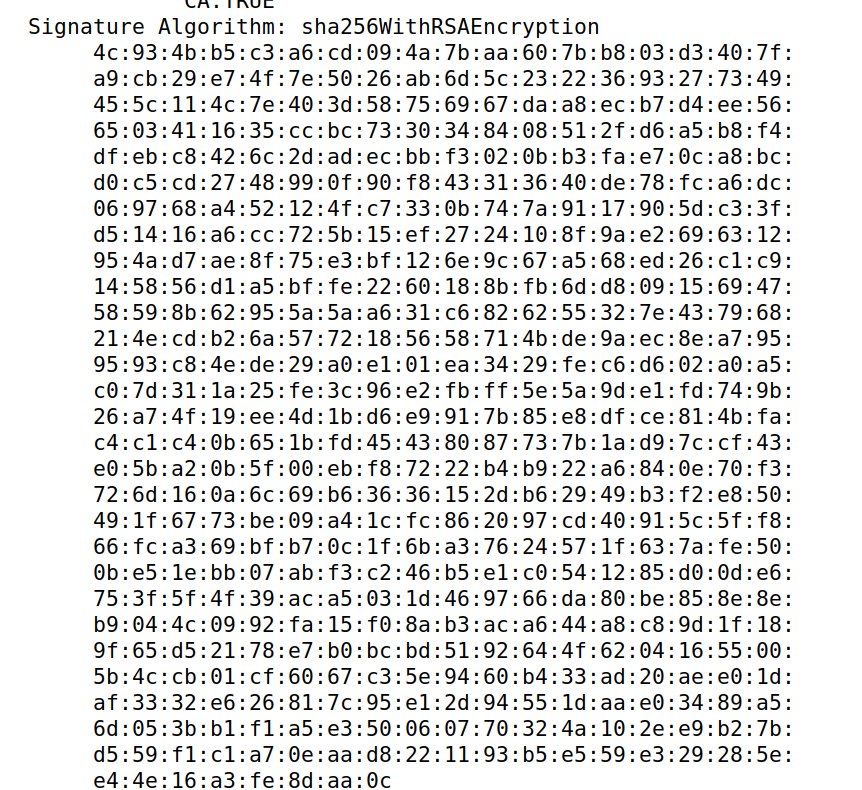


**Task: Generating the self-signed certificate**

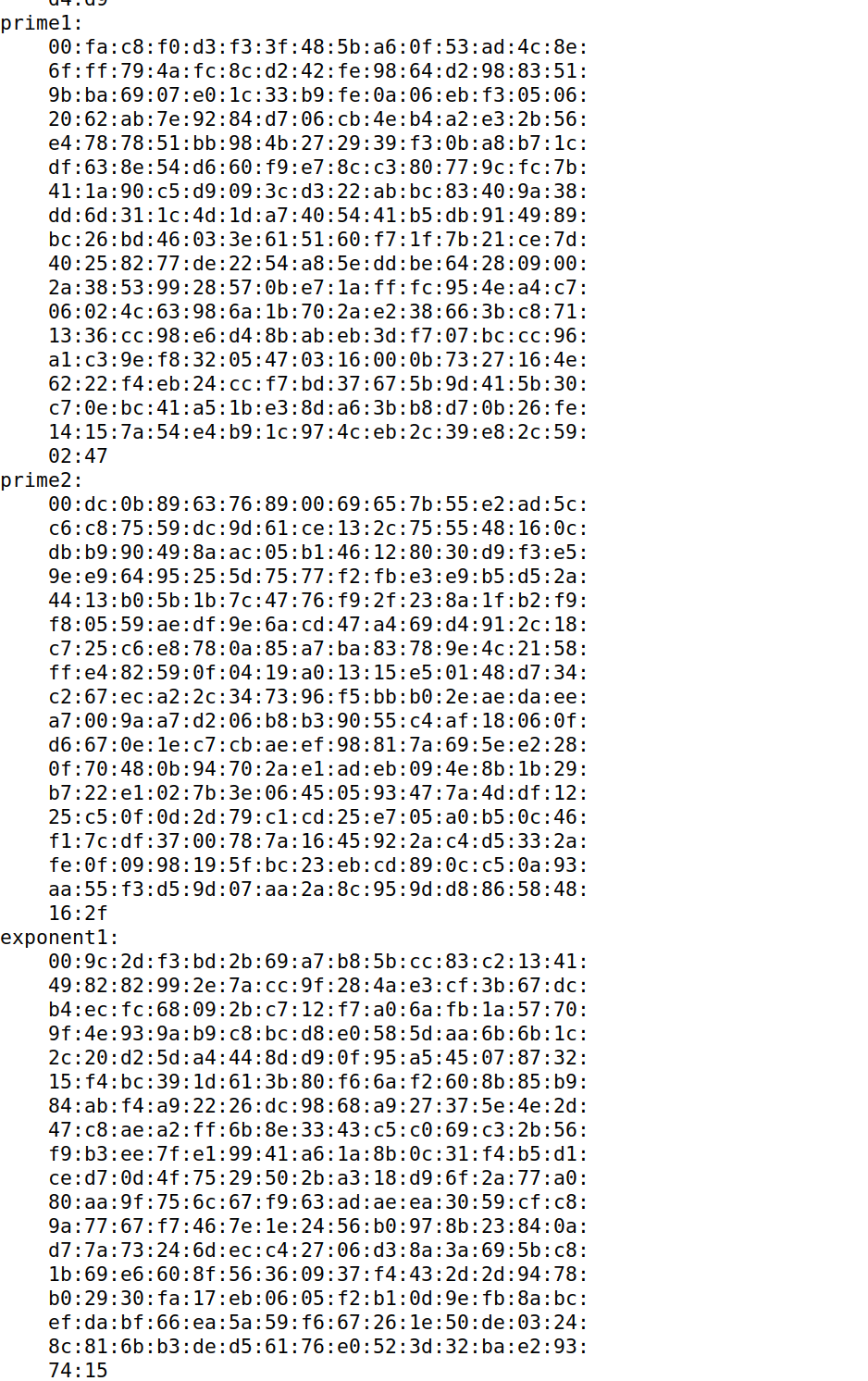
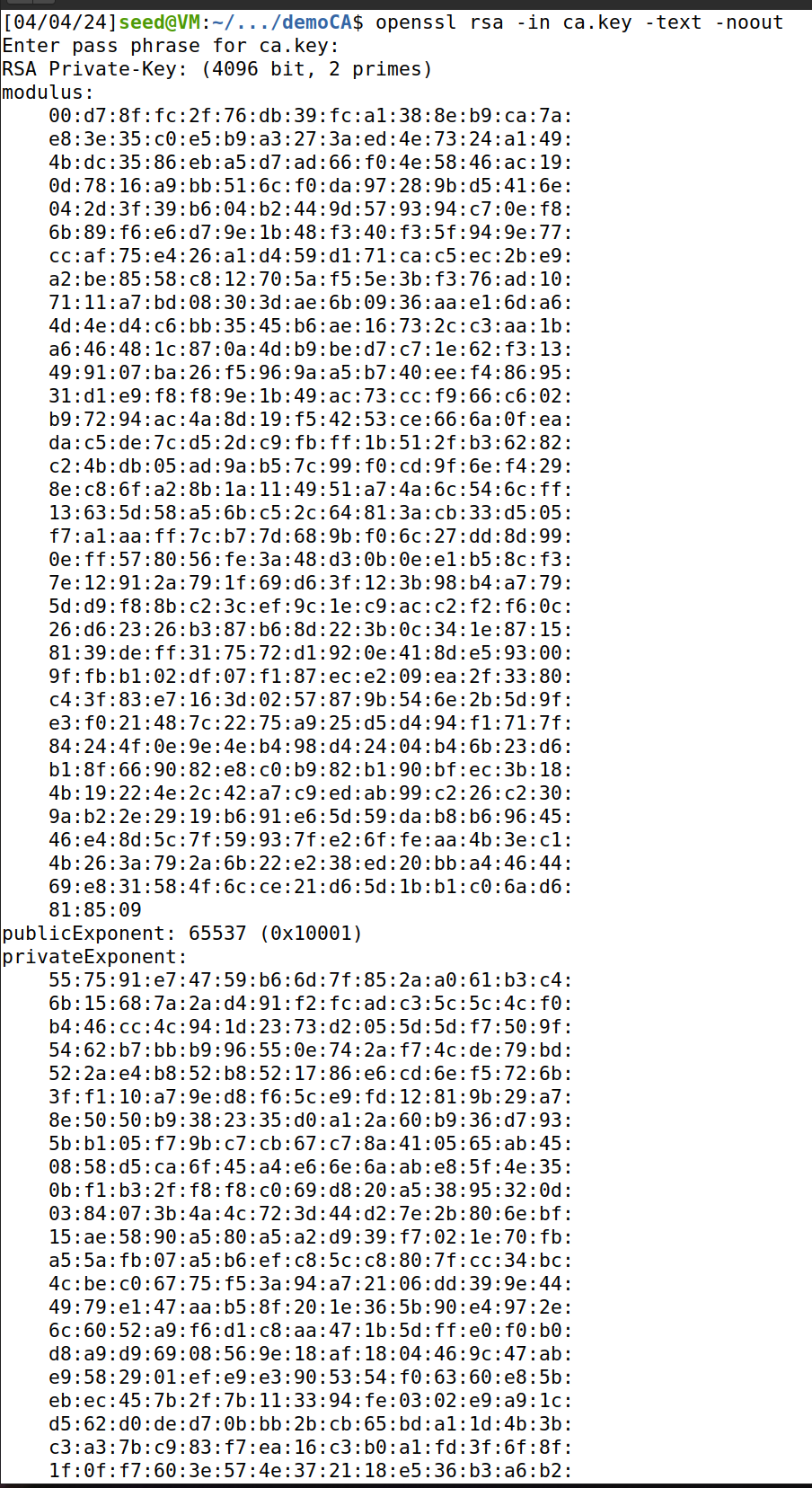
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 \ -keyout ca.key -out ca.crt

Inspecting the certificate



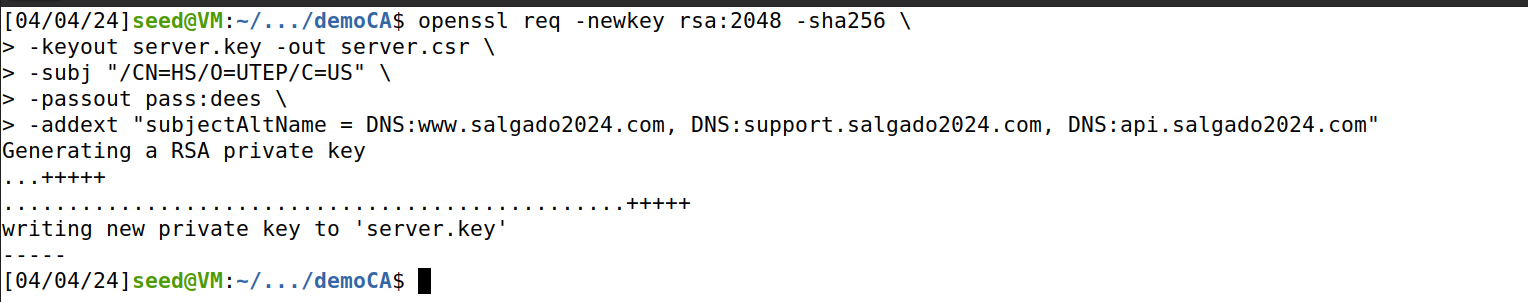


Inspecting the private key - openssl rsa -in ca.key -text –noout

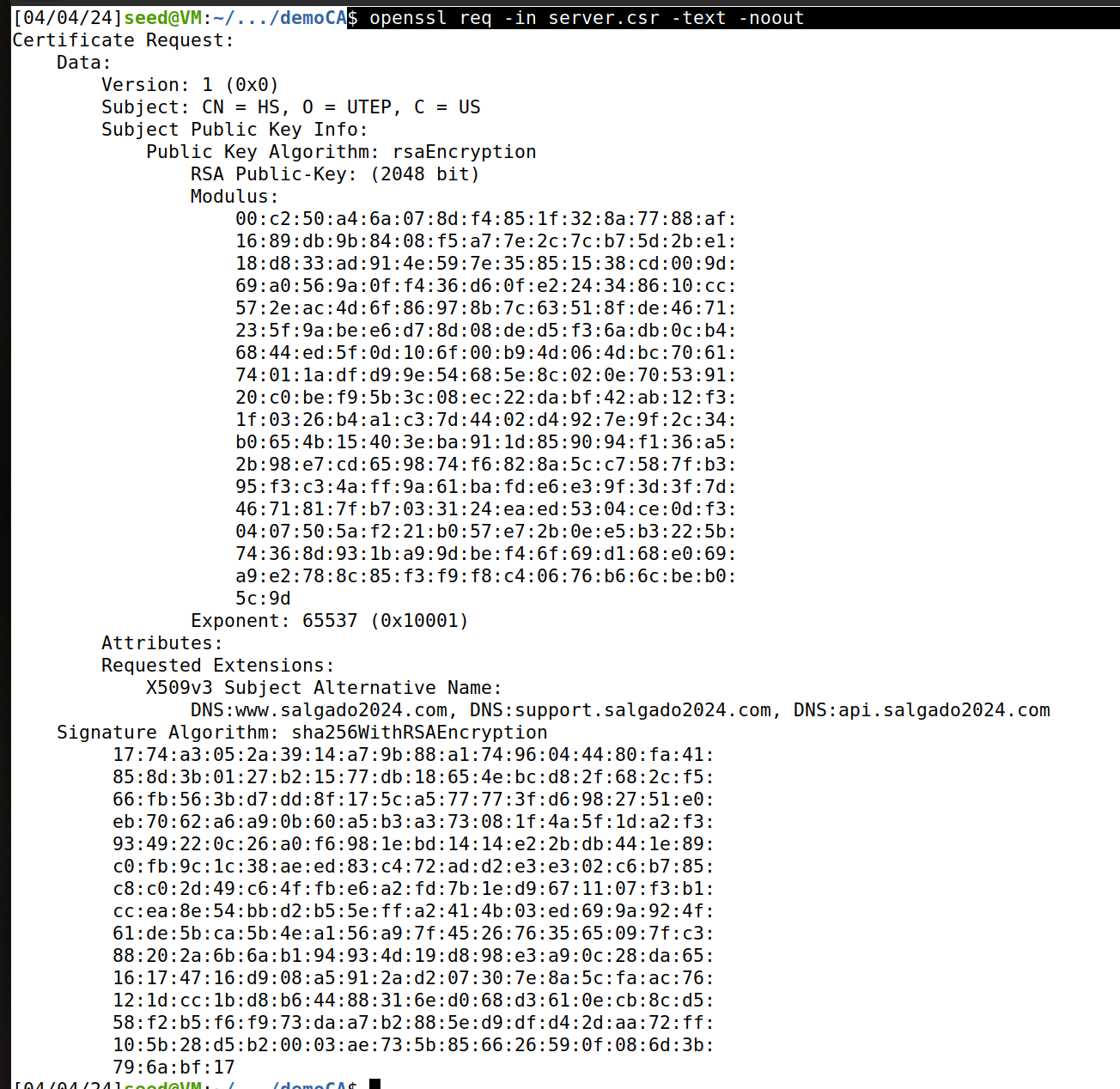


1. What part of the certificate indicates this is a CA’s certificate?
   * Within the X509v3 extensions, the `X509v3 Basic Constraints` is set to critical with the entry `CA:TRUE`. This explicitly defines the certificate’s role as a Certificate Authority (CA).
2. What part of the certificate indicates this is a self-signed certificate?
   * I notice that the `Issuer` and `Subject` fields are identical, sharing the same DN information. This symmetry is characteristic of a self-signed certificate.
3. In the RSA algorithm, we have a public exponent e, a private exponent d, a modulus n, and two secret numbers p and q, such that n = pq. Please identify the values for these elements in your certificate and key files and explain how you identified these values
   * Modulus (n): Identified in the output as a long hexadecimal value under the `modulus` section; it's the result of multiplying two primes, p and q.
   * Public Exponent (e): Found as `publicExponent: 65537 (0x10001)`
   * Private Exponent (d): Listed under the `privateExponent` section, also as a lengthy hexadecimal
   * Prime1 (p) and Prime2 (q): The two secret primes, indicated under `prime1:` and `prime2:`, whose product gives us the modulus n.

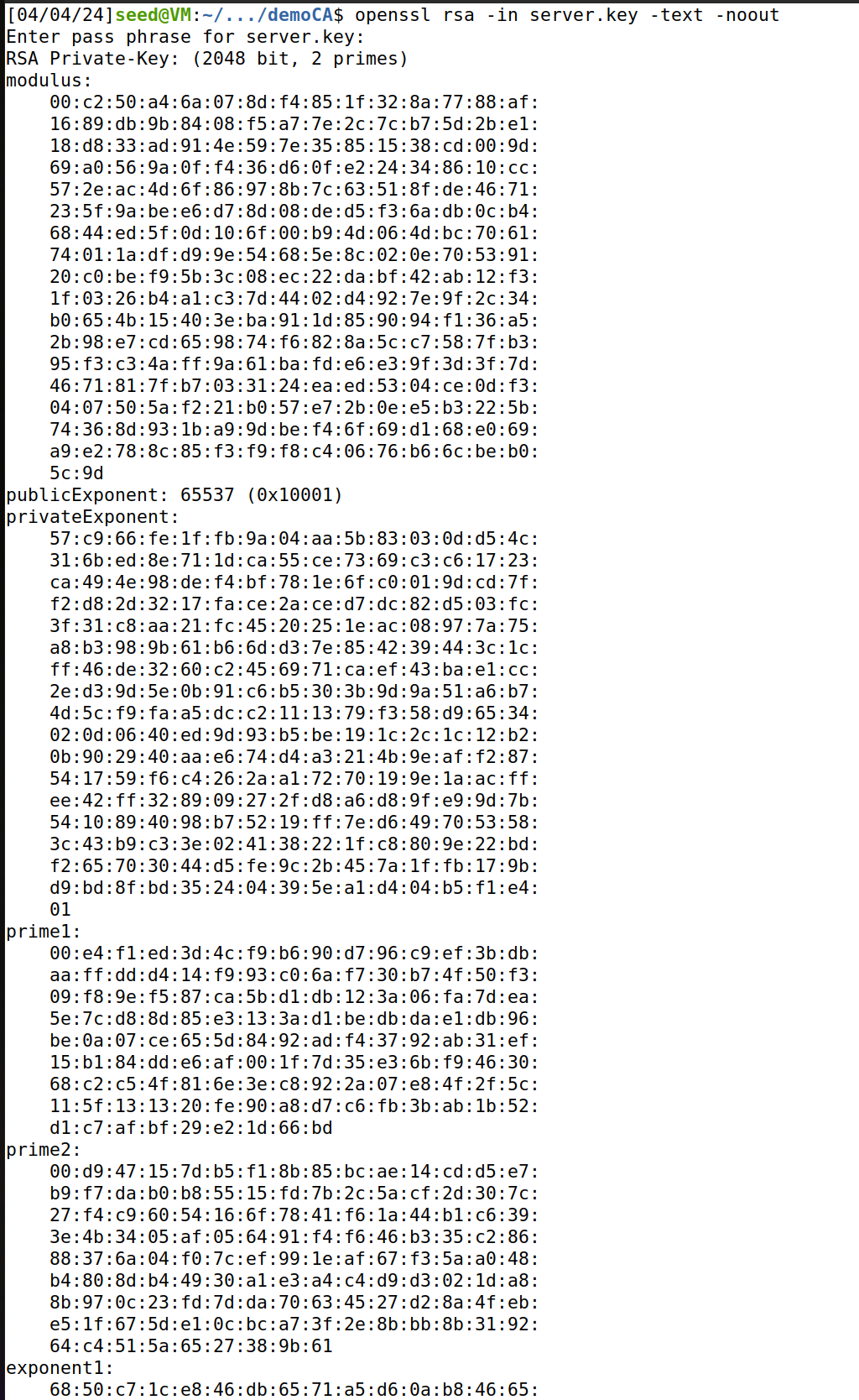
**TASK 2: Generating a Certificate Request for Your Web Server**



$ openssl req -in server.csr -text -noout



openssl rsa -in server.key -text –noout

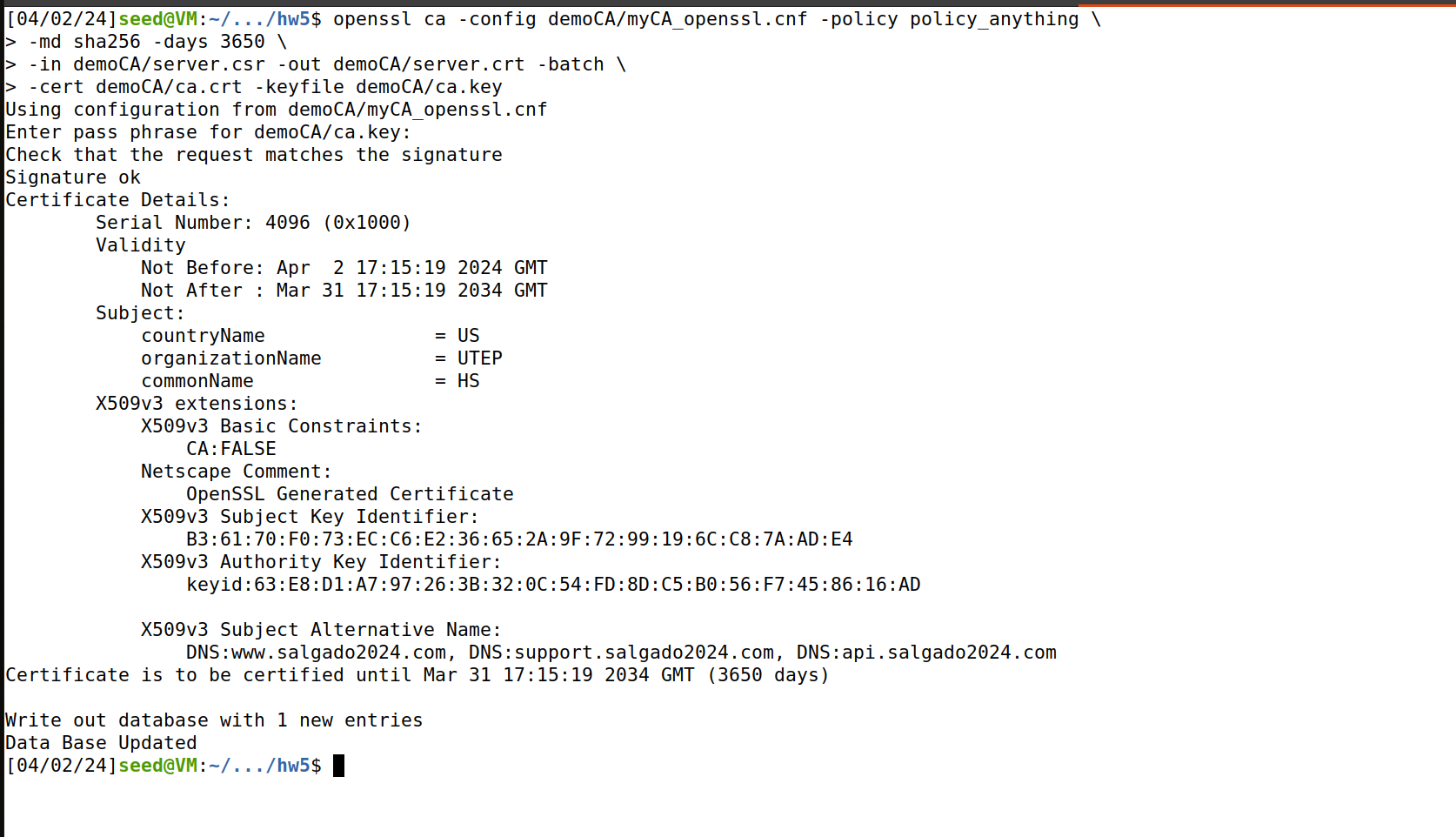


**Why are SANs necesary?**

SANs are especially useful for securing multiple hostnames across different domains and subdomains within the same certificate.

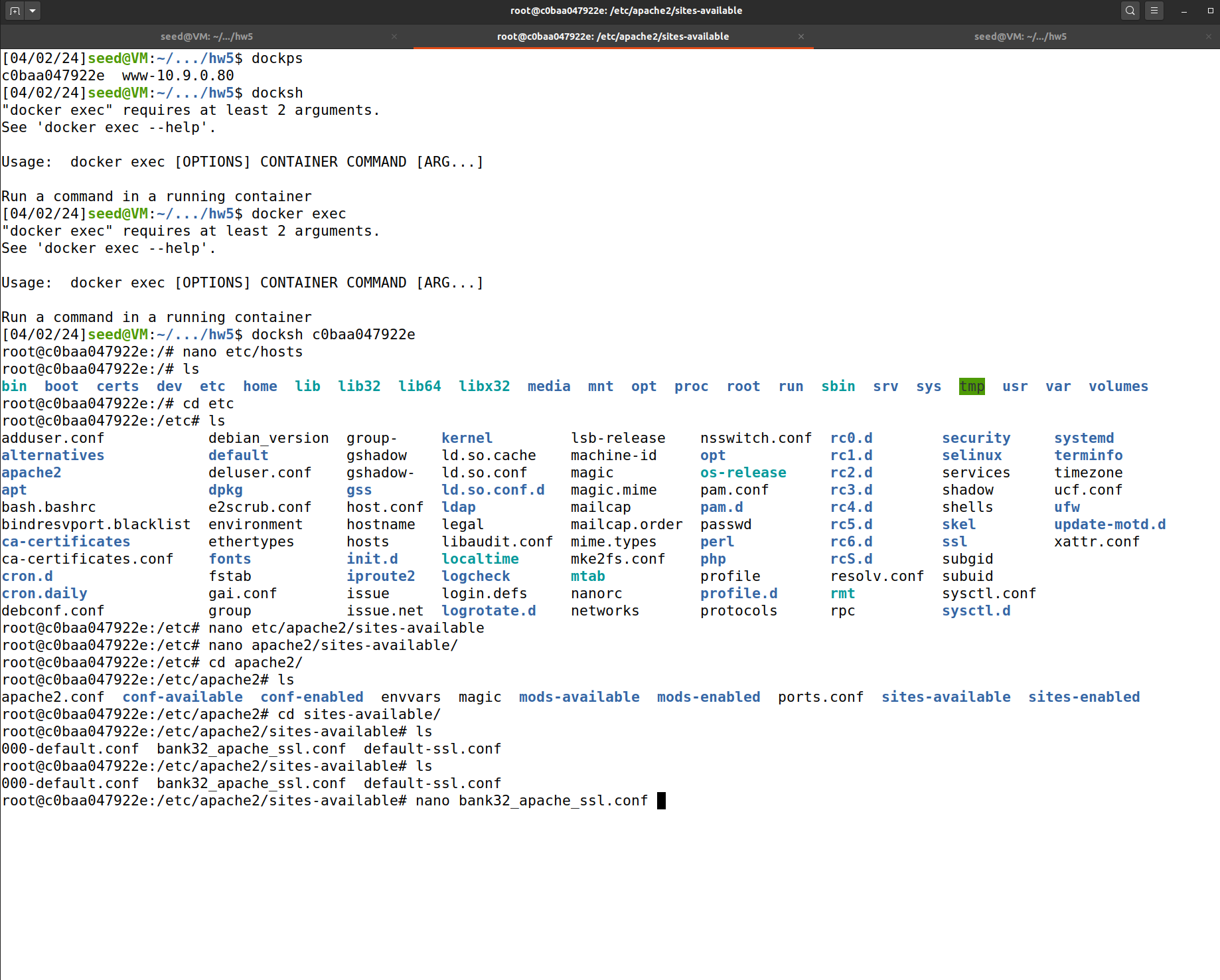
**Task 3: Generating a Certificate for your server**

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:

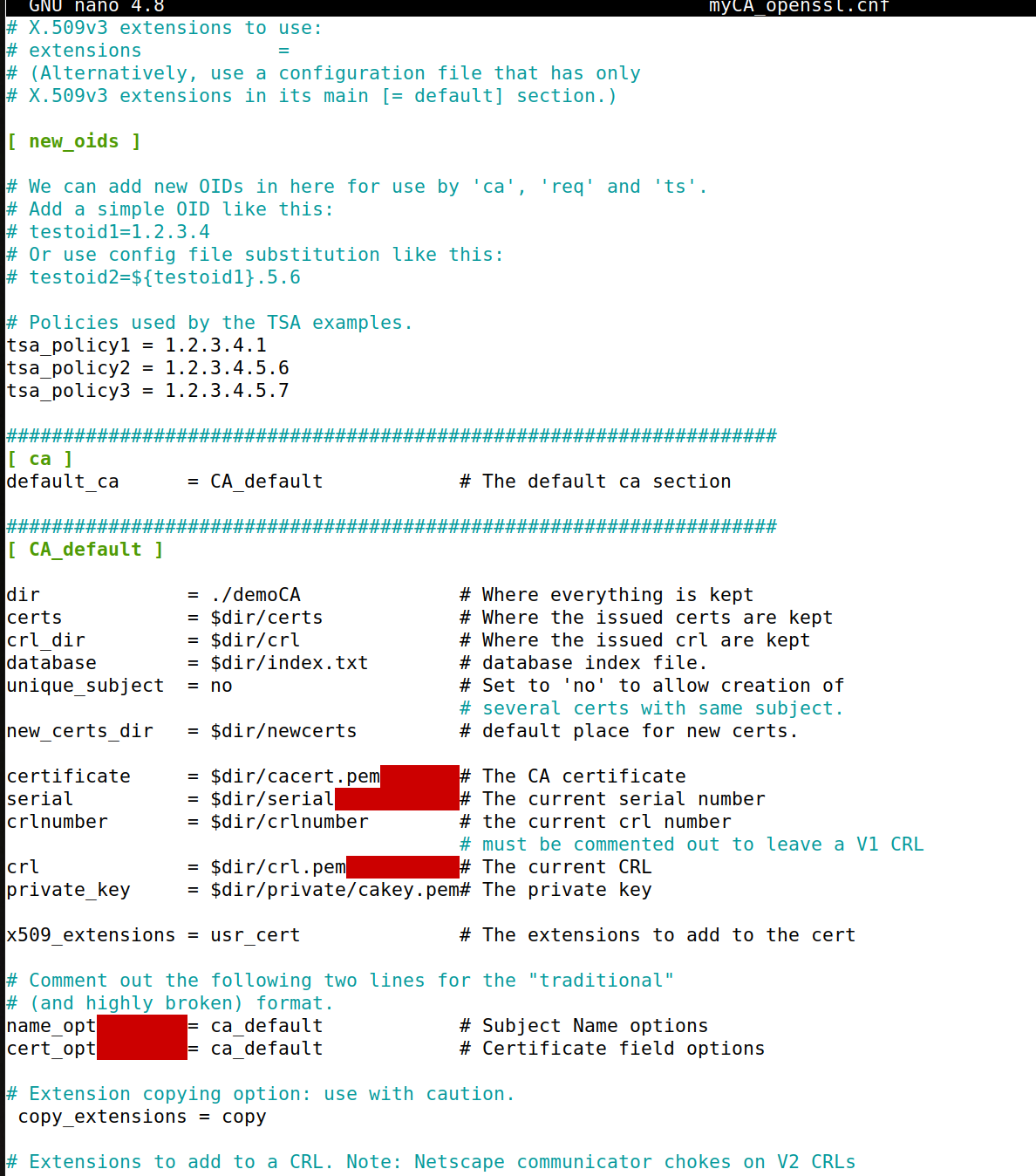


**Task 4: Deploying Certificate in an Apache-Based HTTPS**

Navigating to apache files and editing



Apache edits: copy\_extensions = copy



Decoded content of the certificate - openssl x509 -in server.crt -text –noout



**Observation:** We can alternative names are included.

Starting apache



As we can see, we are unable to connect as there is a certificate problem – connection not secure.

Copying of “server.key” and “server.crt” files into “server.pem” file and Launch the web server using “server.pem” file.



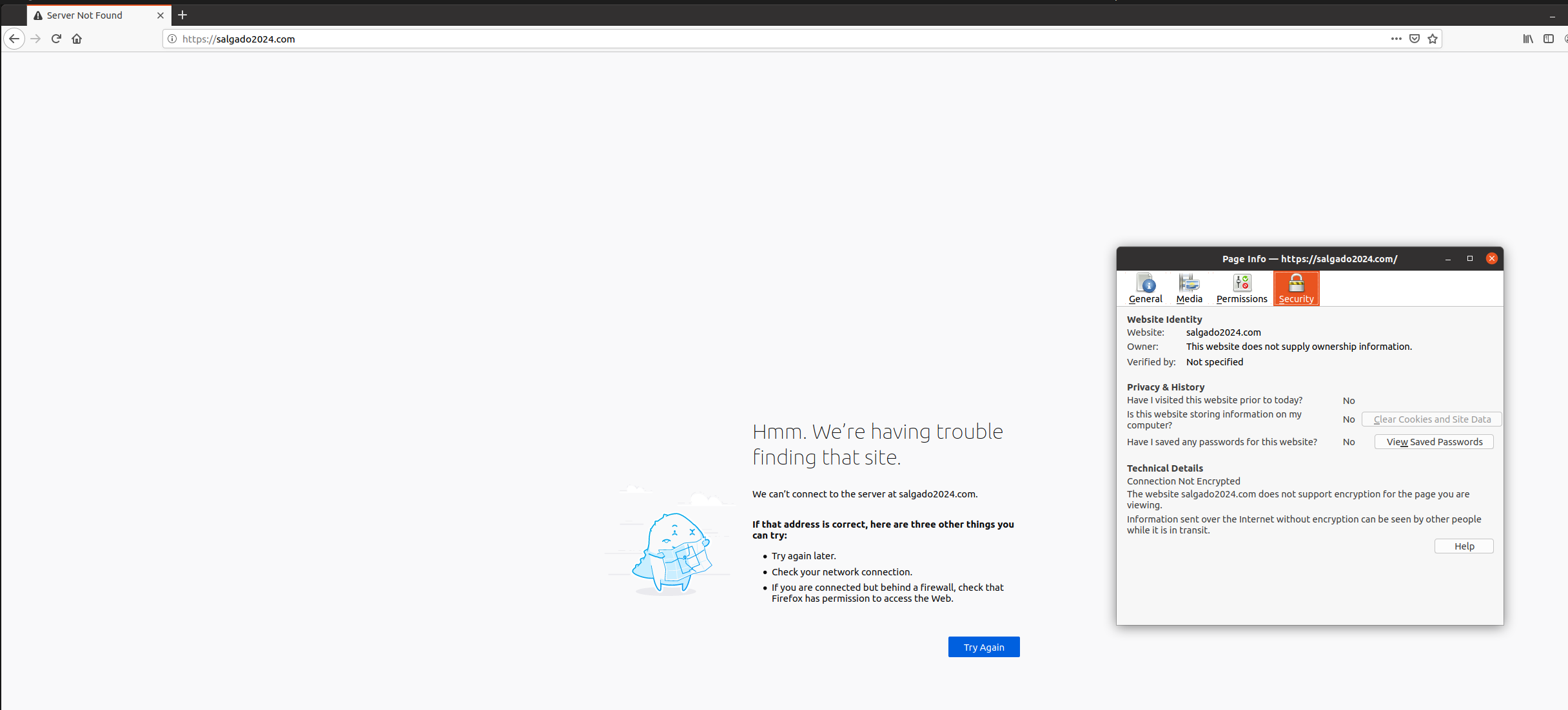
Trying out apache on salgado2024.com



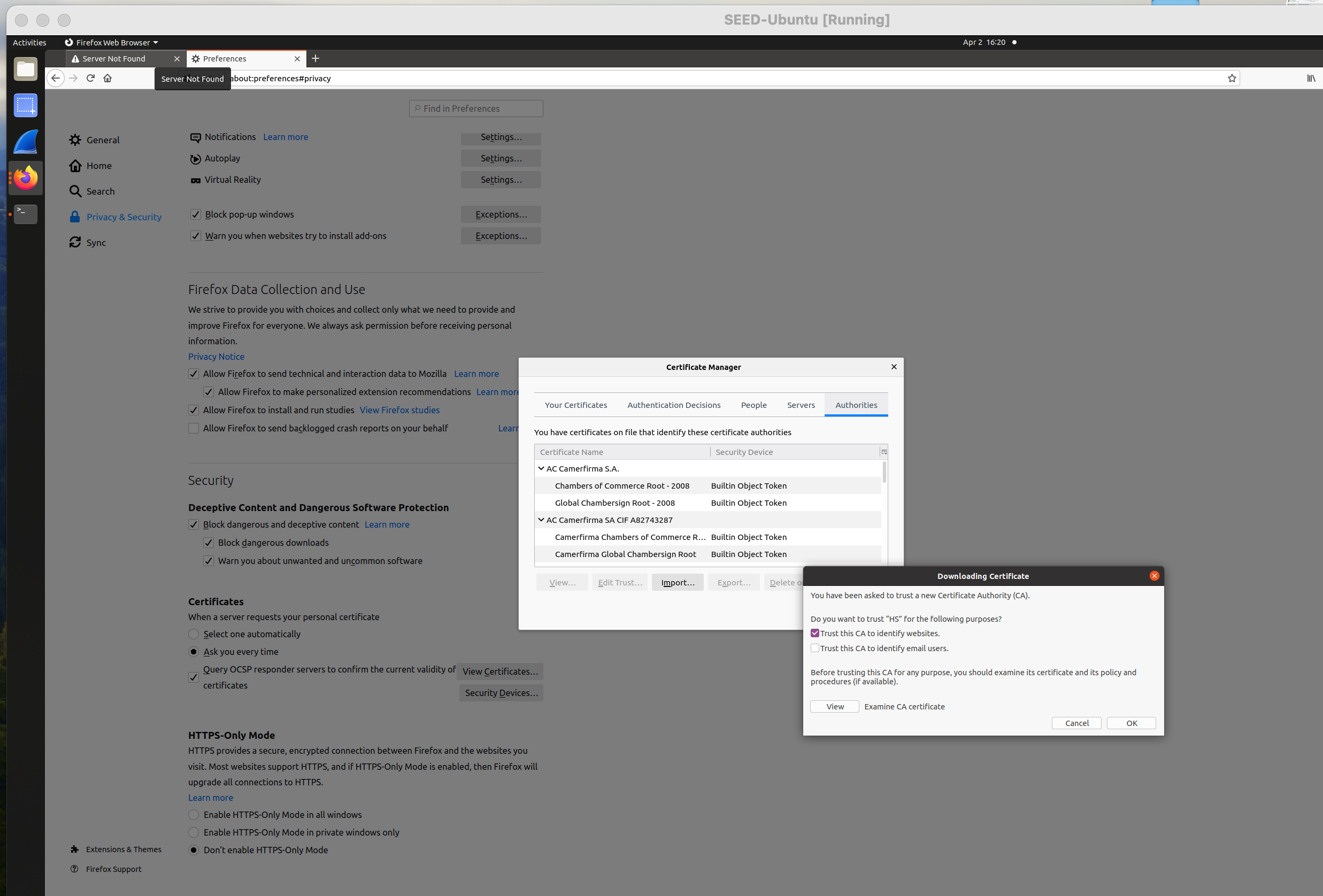
**Observation:**

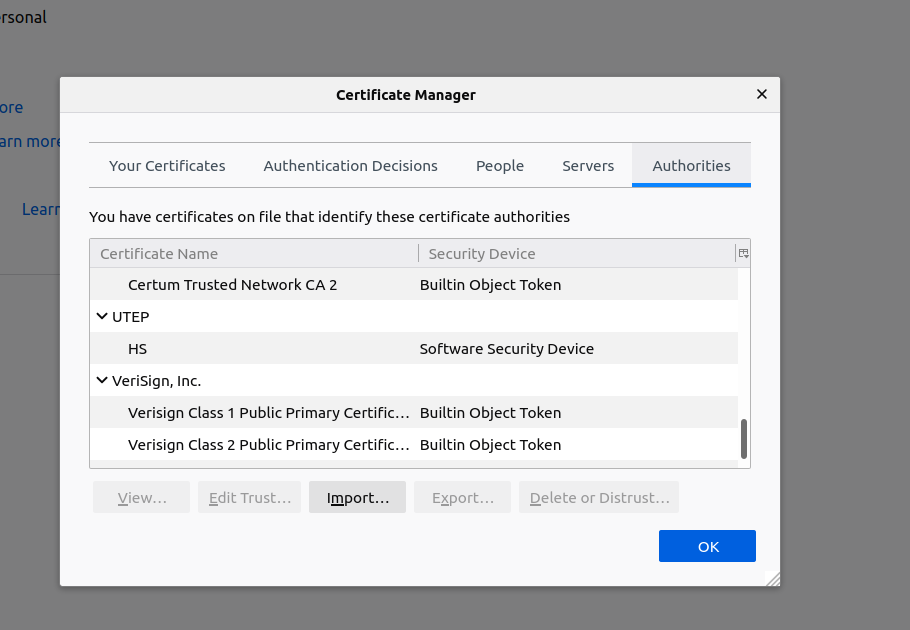
Curl command successfully made a secure connection to www.salgado2024.com, indicating that the SSL setup is working. However, the HTTP response status is 403 Forbidden

However, when I tried accessing via browser, I was unable to. This is because the certificate is not classfied as “secure”



Manually editing the certificates to upload my certificate to Firefox’s list of approved certificates





Viewing “HS” Certificate on the browser

