Lab 29 How to connect to an internal network using OpenVPN

Step 1: Connect to Ubuntu via SSH

`ssh osboxes@192.168.164.193`

✅ **What it does:**

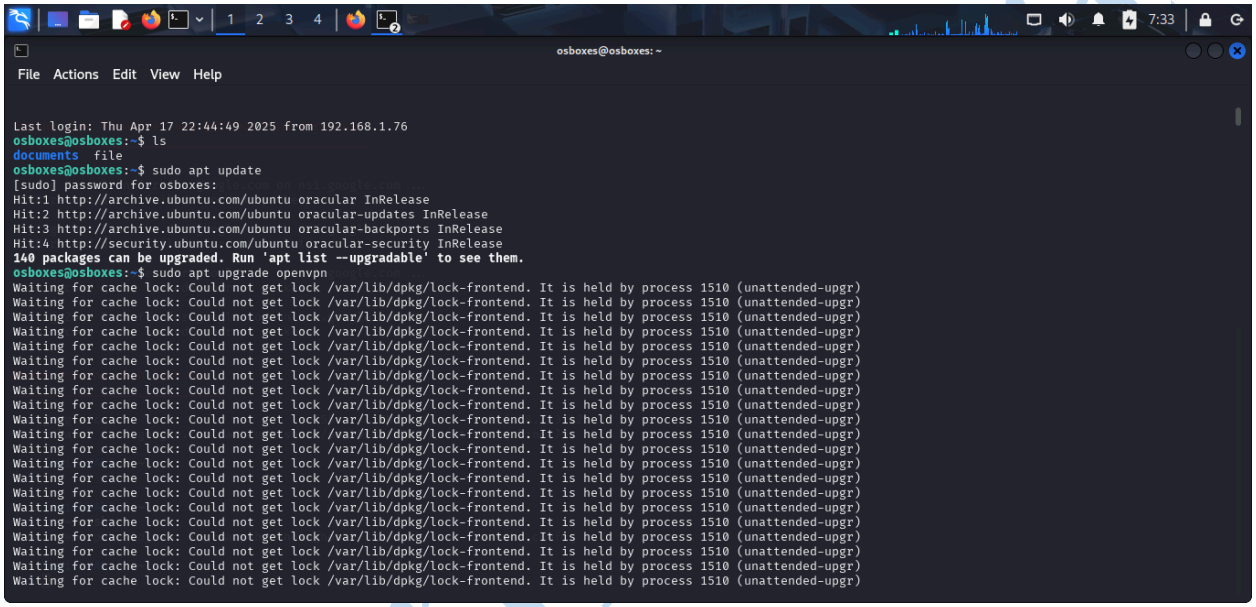
- `ssh` → *secure shell*, lets you remotely log in to another Linux machine
- `osboxes` → the **username** on the Ubuntu server
- `192.168.164.193` → the **IP address** of the Ubuntu machine

Behind the scenes:

SSH opens an encrypted tunnel to the Ubuntu machine so you can run commands on it as if you're sitting there directly.

Step 2: Update Ubuntu & OpenVPN

```
sudo apt update
sudo apt upgrade openvpn
```



✅ **What it does:**

- `sudo` → "do as superuser" (run with admin rights)
- `apt update` → contacts the package repositories (online) and refreshes the list of packages
- `apt upgrade openvpn` → upgrades the OpenVPN package to the newest version

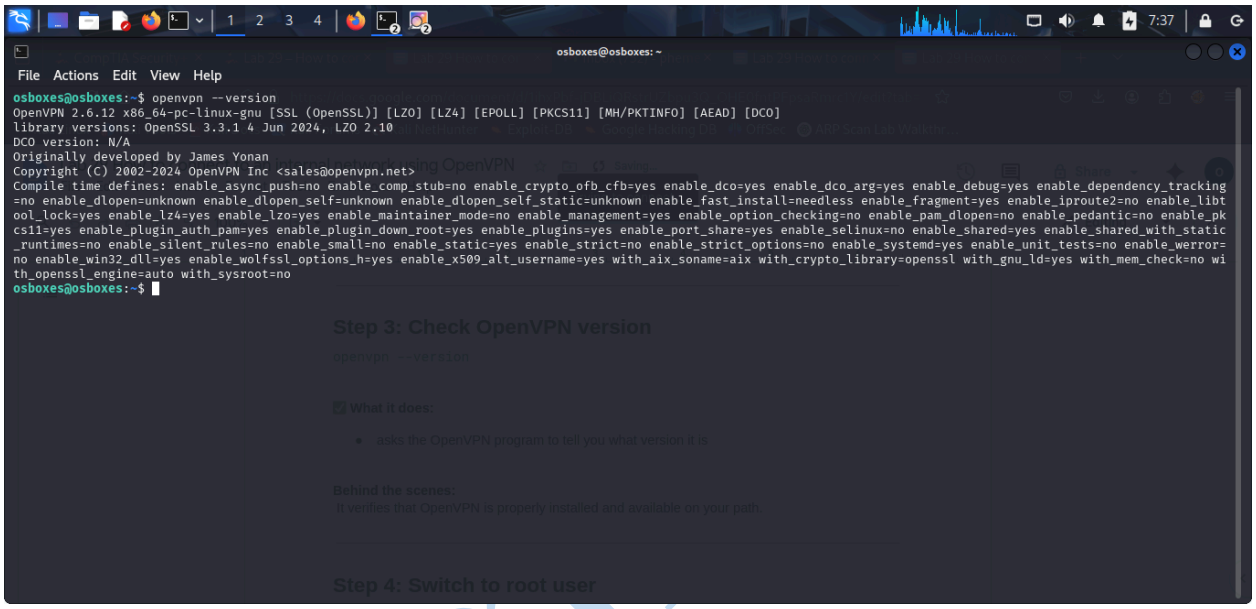
Behind the scenes:

- `apt update` downloads the *latest list of available software*

- `apt upgrade` checks what you have installed, and upgrades it to match the newest available version
  → *This ensures you get security patches and no outdated bugs.*

Step 3: Check OpenVPN version

```
openvpn --version
```



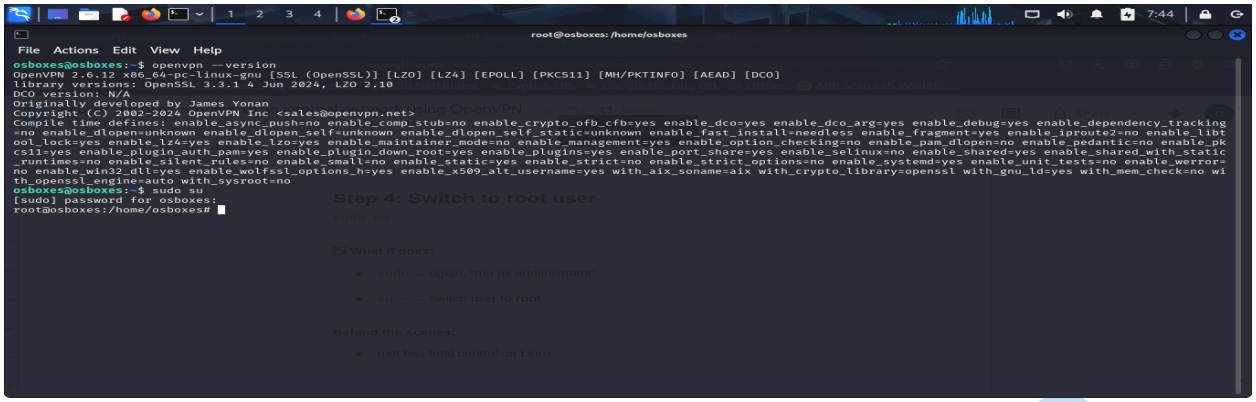
✅ What it does:

- asks the OpenVPN program to tell you what version it is

Behind the scenes:
It verifies that OpenVPN is properly installed and available on your path.

Step 4: Switch to root user

```
sudo su -
```



✅ **What it does:**

- `sudo` → again, “run as administrator”
- `su -` → switch user to **root**

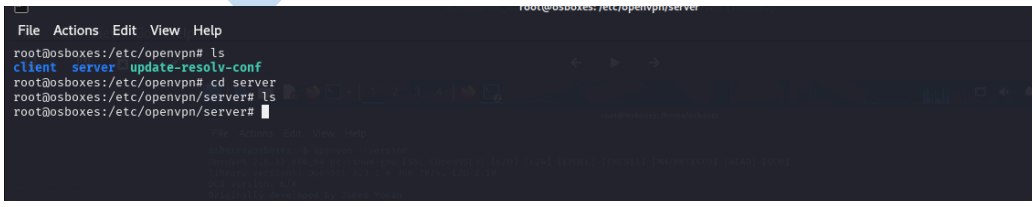
Behind the scenes:

- root has total control on Linux
- you need root to change files under `/etc/openvpn`
  → normal users cannot write there.

Go to the OpenVPN folder

👉 Type this to move into the server config folder:

```
cd /etc/openvpn/server
```



✅ *Why?*
Because OpenVPN is designed to look for its server settings in this folder.
You want to put your config there.

Create the server.conf file with the config inside

👉 Now, you will *paste* a block of text to create `server.conf`.

This means you will *tell* Ubuntu:

> "Hey, copy everything I'm about to type into a new file."

👉 Type this carefully:

```
cat << EOF > server.conf
```

Then, *press ENTER*.
✅ **Your terminal will just sit there waiting for you to type.** That is normal.

🟢 STEP 4: Now paste these lines (copy them, then right-click to paste):

```
local 192.168.1.206
port 1194
proto udp4
dev tun
keepalive 10 120
topology subnet
server 10.8.0.0 255.255.255.0
verb 3
auth-nocache
user nobody
```

```
group nogroup
explicit-exit-notify
auth SHA512
persist-key
persist-tun
cipher AES-256-GCM
tls-server
remote-cert-eku "TLS Web Client Authentication"
ca ca.crt
cert server.crt
key server.key
tls-crypt tc.key
dh none
```

✅ *Don't worry* if you don't understand every line — these are standard VPN server rules, and I'll help you test them later.

🟢 STEP 5: Tell the terminal you are finished

👉 Type this on a new line:

nginx
CopyEdit
`EOF`

✅ Then press ENTER.

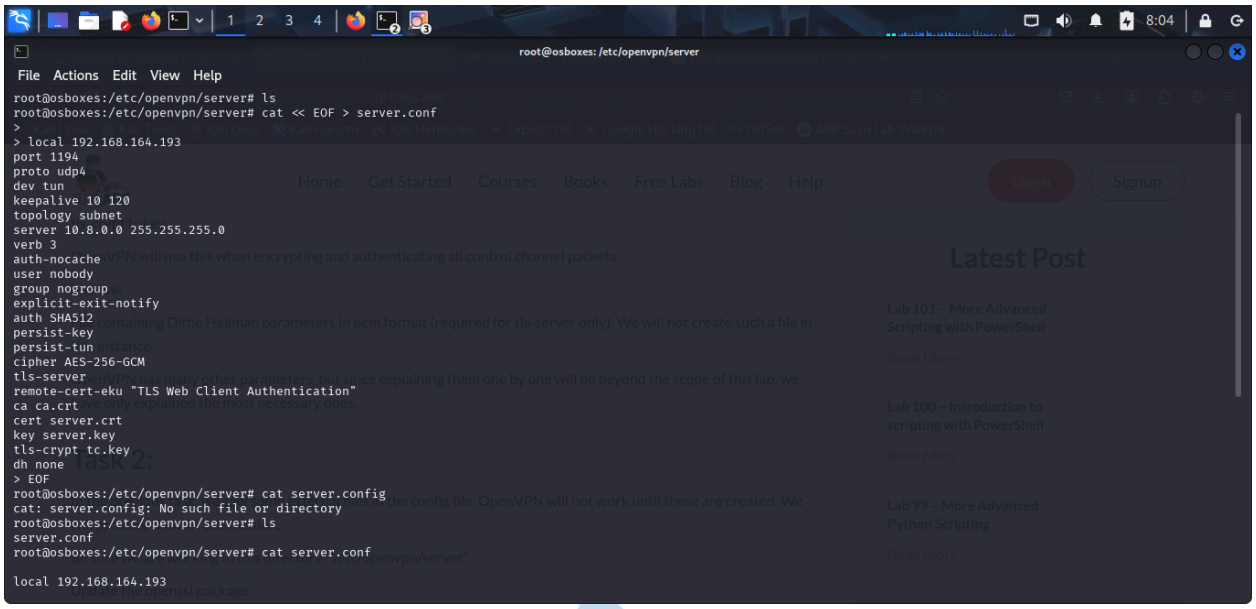
That tells the computer:

> "OK, I finished typing my block, please save it now."

🟢 STEP 6: Check your work

👉 Type:

```
cat server.conf
```



✅ This shows you what you just wrote, to confirm it went into the file correctly.

Step 5: Create OpenVPN server config

```
cat << EOF > /etc/openvpn/server/server.conf
# (followed by the big block of config)
EOF
```

✅ **What it does:**

- `cat << EOF` → creates a *heredoc*, which takes text you type between `EOF` and puts it in a file

- `> server.conf` → means “write that text to server.conf”

Behind the scenes:

- `cat` is a Linux command to *print files*
- using `<< EOF` you can “type a file interactively” rather than using a text editor
  → Linux captures all your lines until it sees `EOF` and saves them to `server.conf`.

This is a quick way to paste a config without opening `nano` or `vi`.

Config explanation (server.conf)

Some critical lines:

- `local 192.168.1.206` → tells the server to listen on its own IP
- `port 1194` → default port for OpenVPN
- `proto udp4` → use UDP version 4 protocol
- `server 10.8.0.0 255.255.255.0` → gives out VPN client addresses in this subnet
- `cipher AES-256-GCM` → secure encryption algorithm
- `ca ca.crt` → file to verify client certificates
- `cert server.crt` → server’s own certificate
- `key server.key` → server’s private key
- `tls-crypt tc.key` → additional encryption for handshake
- `dh none` → no Diffie-Hellman because you use ECDHE

Behind the scenes:

When you launch the server, OpenVPN reads this file line-by-line to know:

- which IP to use
- how to encrypt
- what certificates to validate clients
  → This is what sets up the secure VPN server behavior.

Step 6: Create Root Certificate Authority (CA)

Cd into the root directory:
cd ~
Then run the below command;

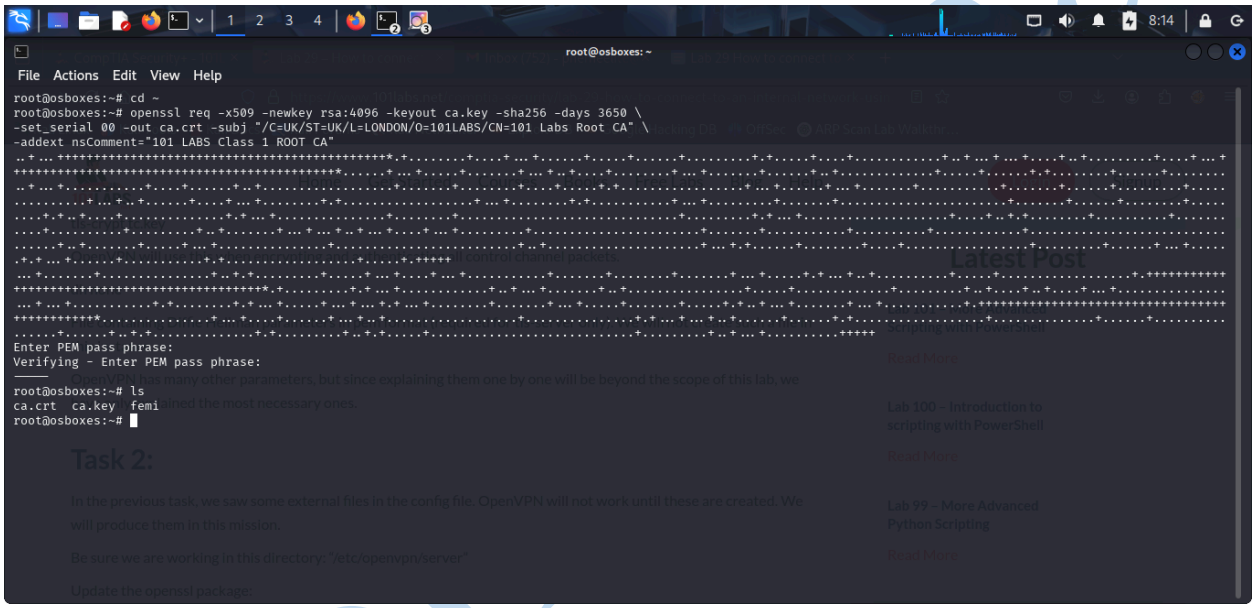
```
openssl req -x509 -newkey rsa:4096 -keyout ca.key -sha256 -days 3650 \
-set_serial 00 -out ca.crt -subj
"/C=UK/ST=UK/L=LONDON/O=101LABS/CN=101 Labs Root CA" \
-addext nsComment="101 LABS Class 1 ROOT CA"
```

After running the above command, it would request for a password. Input a password you know you would remember

✅ **What it does:**

- `openssl req` → start creating a certificate
- `-x509` → means *self-signed*
- `-newkey rsa:4096` → make a new key with RSA 4096-bit strength
- `-keyout ca.key` → save the private key to `ca.key`

- `-out ca.crt` → save the certificate to `ca.crt`
- `-sha256` → use strong SHA-256 hashing
- `-days 3650` → valid for 10 years
- `-subj` → sets details about who owns the certificate
- `-addext nsComment` → comment inside the certificate
- `-set_serial 00` → sets a unique serial number



Behind the scenes:

- You become your own certificate authority
- This CA certificate (ca.crt) is later used to **sign** server and client certificates so they trust each other

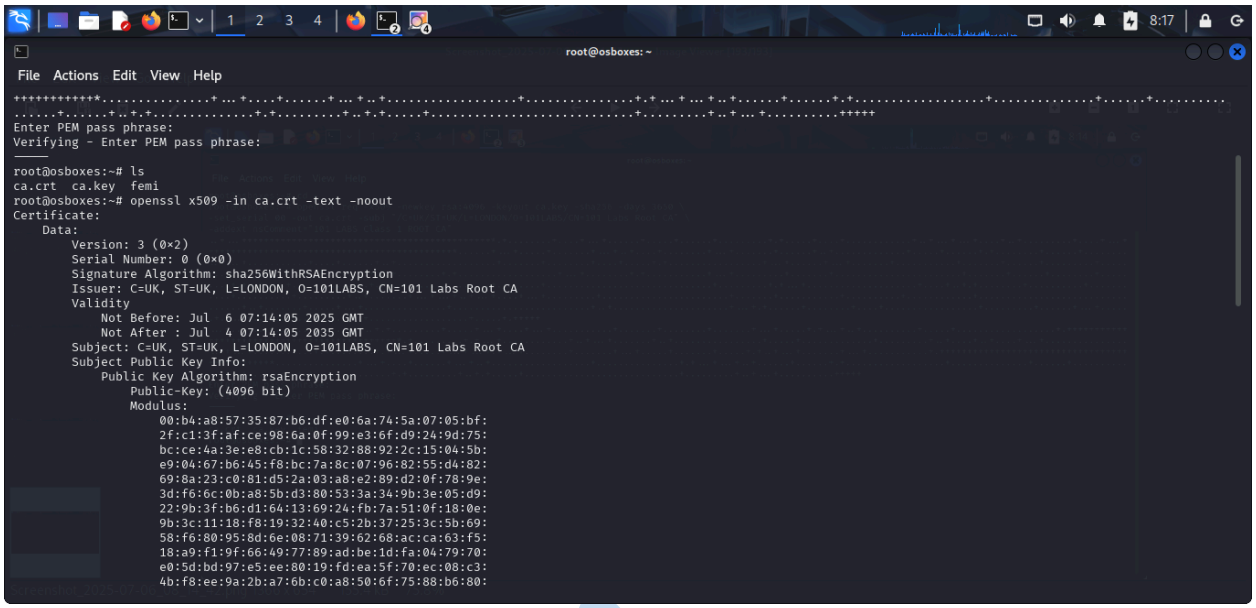
---

Step 7: Check CA details

```
openssl x509 -in ca.crt -text -noout
```

✅ **What it does:**

- shows the human-readable info in the certificate
  → you verify it was made correctly



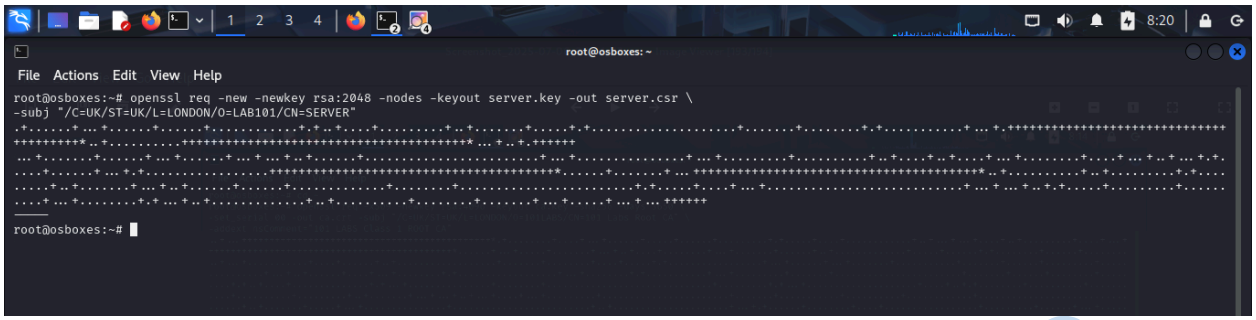
Step 8: Make server certificate signing request

```
openssl req -new -newkey rsa:2048 -nodes -keyout server.key -out
server.csr \
-subj "/C=UK/ST=UK/L=LONDON/O=LAB101/CN=SERVER"
```

✅ **What it does:**

- asks to make a new private key (2048 bits)
- outputs the key to `server.key`
- makes a *certificate signing request* to `server.csr`

- sets the details like country, org, etc



Behind the scenes:

→ a CSR says “Hey CA, please sign me so I’m trusted!”

it does not sign itself, it waits for the CA to approve.

Creating the `x509-extensions.cnf file`

Run this to open a simple editor (like nano):

```
nano x509-extensions.cnf
```

✅ Then copy and paste this text inside nano:

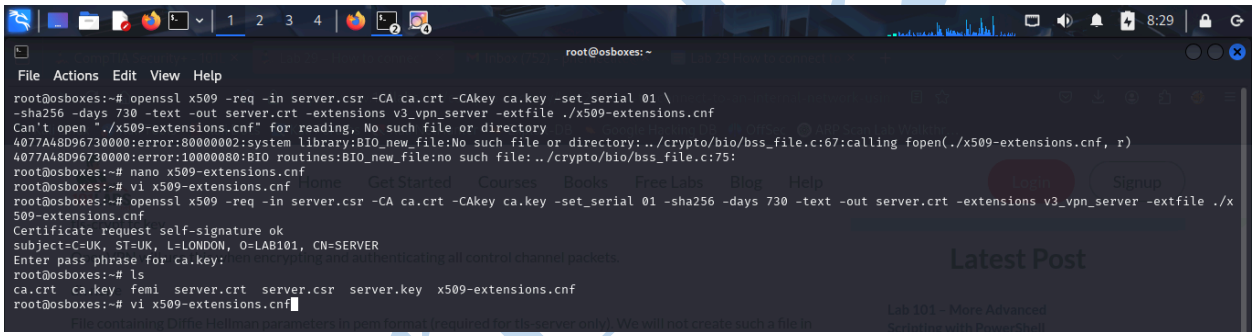
```
[ v3_vpn_server ]

basicConstraints = CA:FALSE

nsCertType = server

keyUsage = digitalSignature, keyEncipherment

extendedKeyUsage = serverAuth
```

Step 9: Sign server certificate

```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -set_serial
01 \
-sha256 -days 730 -text -out server.crt -extensions v3_vpn_server
-extfile ./x509-extensions.cnf
```

✅ **What it does:**

- uses CA to sign the server's CSR
- produces server.crt
- sets a serial number (01)
- uses rules from `v3_vpn_server` in the extensions file
  → this certificate is now trusted by the CA



Behind the scenes:

OpenVPN will later match the server's `server.crt` to clients by checking if it was signed by `ca.crt`.

Step 10: Make client certificate signing request:

1 : Create the client key + CSR

✅ Type this as root (or stay in the same folder as before):

```
openssl req -new -newkey rsa:2048 -nodes -keyout client.key -out
client.csr \
-subj "/C=UK/ST=UK/L=LONDON/O=LAB101/CN=CLIENT"
```

✅ This does exactly the same as for the server:

- makes a **client private key** (`client.key`)
- makes a **certificate signing request** (`client.csr`)

But this time with CN=CLIENT.
👉 The CN (common name) is how you know this certificate is for the client.

2: Create a client extensions file

👉 We'll need to tell OpenSSL "this is a client certificate, not a server certificate"

✅ Make a new file:

```
vix509-client-extensions.cnf
```

✅ Then paste this:

```
[ v3_vpn_client ]
basicConstraints = CA:FALSE
nsCertType = client
keyUsage = digitalSignature
extendedKeyUsage = clientAuth
```

3: Sign the client certificate

✅ Use your CA to sign the CSR:

```
openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -set_serial
02 \
-sha256 -days 730 -text -out client.crt -extensions v3_vpn_client
-extfile ./x509-client-extensions.cnf
```

👉 **Explanation**:

- `-in client.csr` → sign the client's request
- `-CA ca.crt -CAkey ca.key` → use the same CA to approve it
- `-set_serial 02` → give this cert serial 2 (the server had 1)
- `-out client.crt` → your finished client certificate
- `-extfile x509-client-extensions.cnf` → use client-specific rules

✅ It will ask you for the **CA passphrase** again → type it in.

4: Confirm what you have

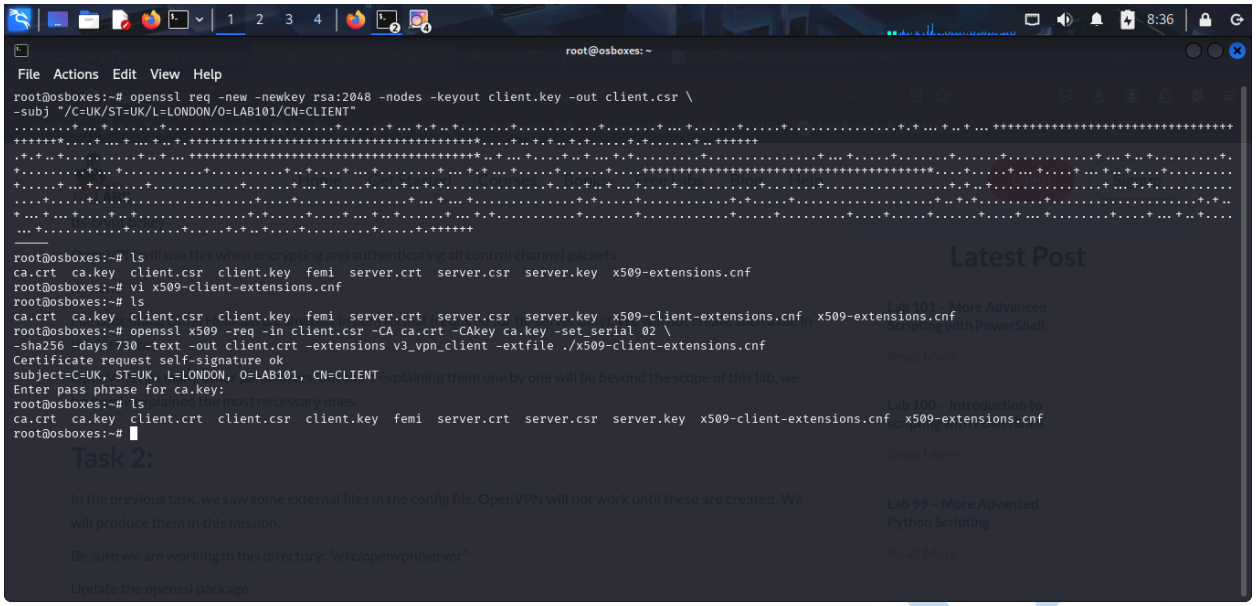
👉 Check your directory:

```
ls
```

(similar steps, just changes `client` instead of `server`)

✅ You repeat:

- generate client key + client CSR
- sign it with the CA
  → then you have client.crt



Behind the scenes:

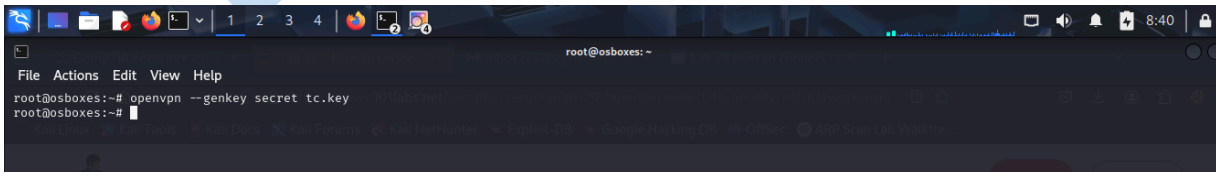
this allows the client to *prove* its identity to the server.

Step 11: Generate tls-crypt key

```
openvpn --genkey secret tc.key
```

✅ **What it does:**

- makes a random symmetric key
- this key encrypts control traffic (handshake messages)
  → protects you against attackers messing with handshakes.



Behind the scenes:

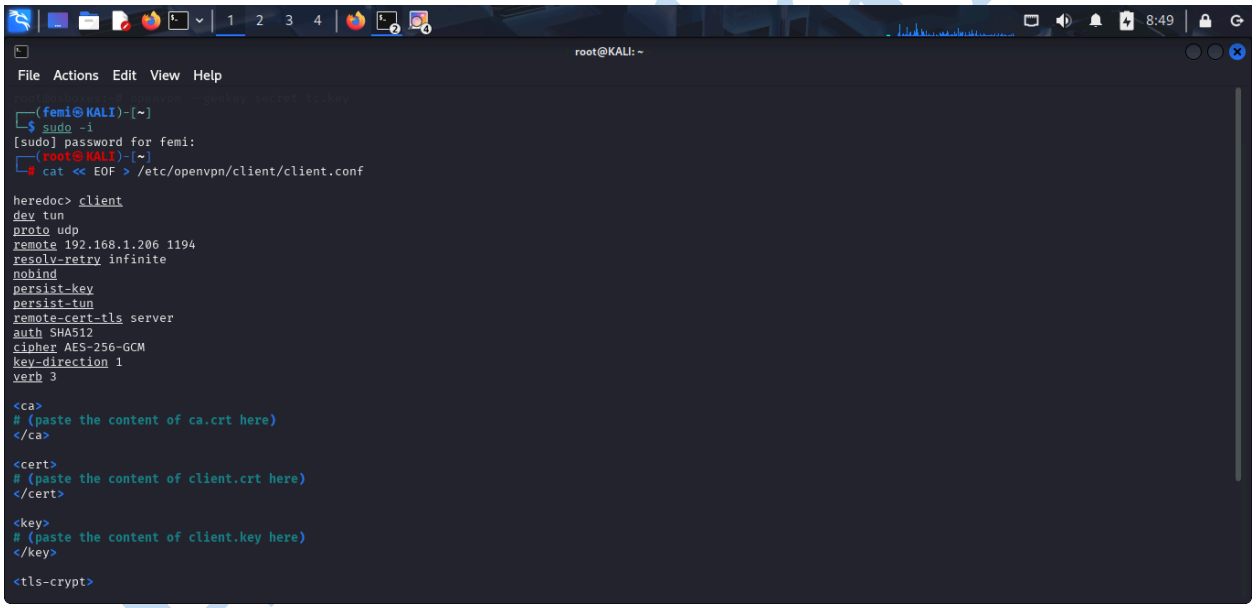
reduces risk of certain attacks called *packet injection*.

Step 12: Build client config on Kali

```
cat << EOF > /etc/openvpn/client/client.conf
# (pasted config lines)
EOF
```

✅ **What it does:**

- writes a config file for the Kali machine, just like you did for the server
- points to correct certificates and server IP



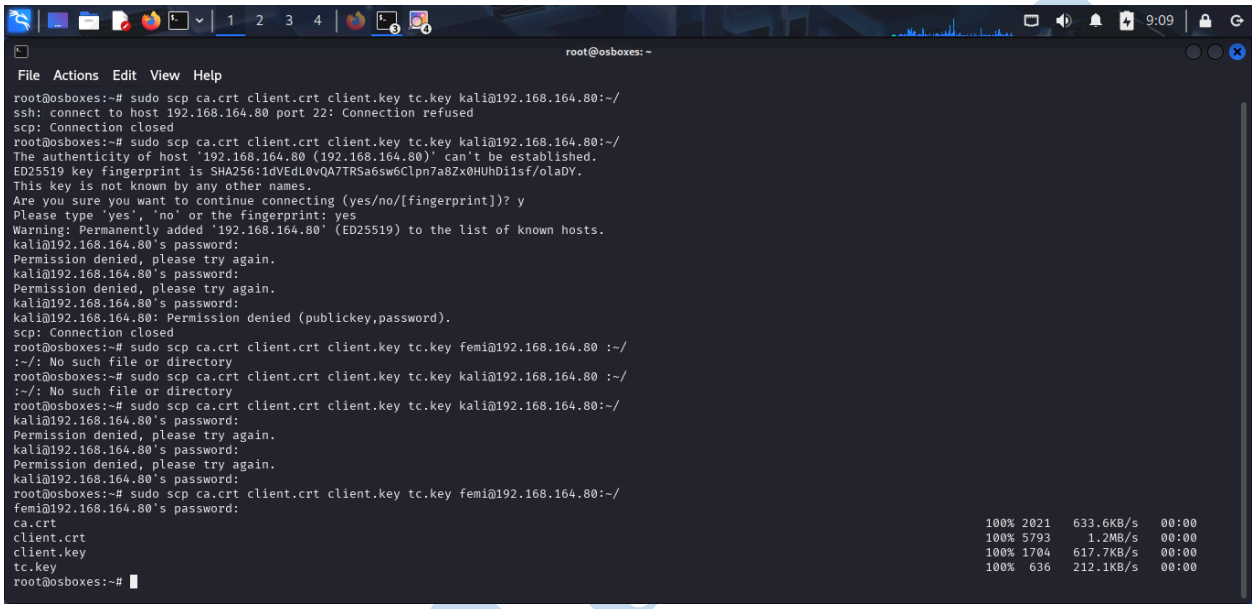
Behind the scenes:
this tells OpenVPN on Kali how to talk to your Ubuntu server securely.

Step 13: Move files to Kali with SCP

```
sudo scp ca.crt client.crt client.key tc.key kali@192.168.164.80:~/
```

✅ **What it does:**

- uses secure copy (`scp`) to transfer certificates
- Kali needs these files to validate the server



Behind the scenes:

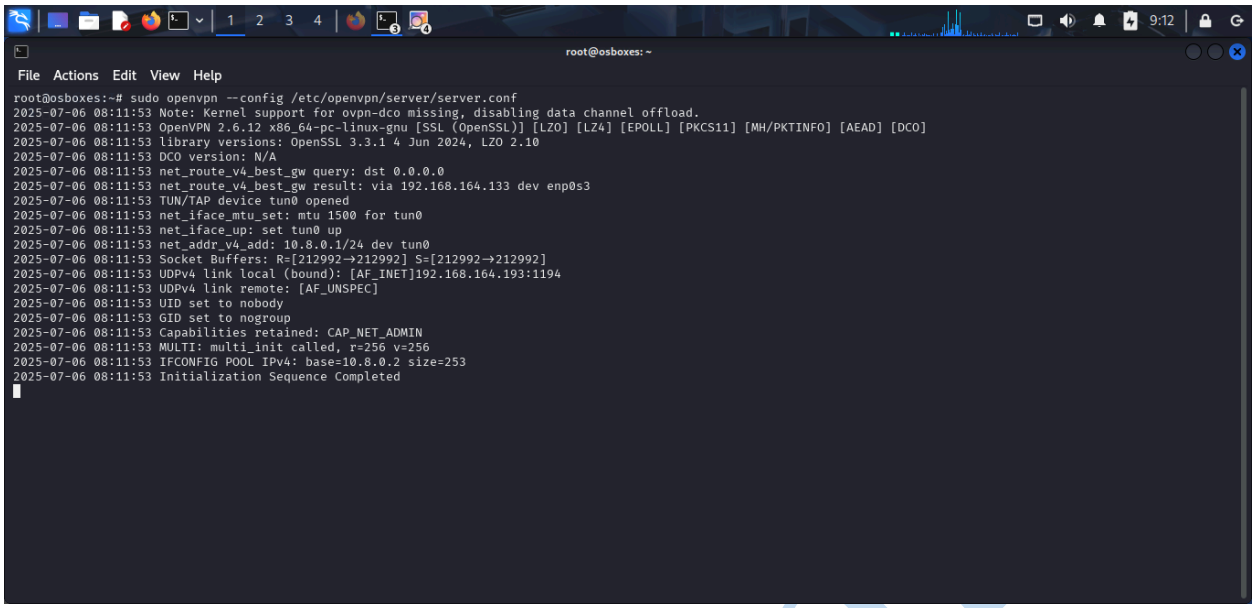
no certificates = no VPN.

SCP uses SSH, so the copy is encrypted.

Step 14: Start the VPN tunnel

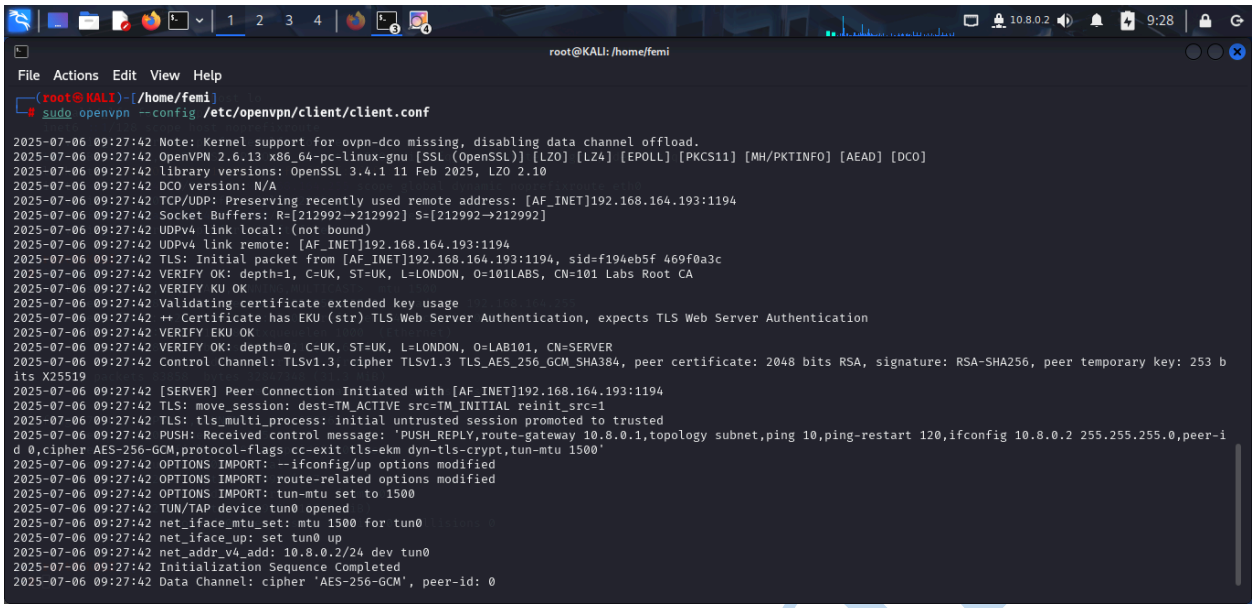
✅ On server:

```
sudo openvpn --config /etc/openvpn/server/server.conf
```



✅ On client:

```
sudo openvpn --config /etc/openvpn/client/client.conf
```



Behind the scenes:

OpenVPN starts up, reads its config, loads the certificates, and opens a secure tunnel.

→ it sets up a new network interface (`tun0`)

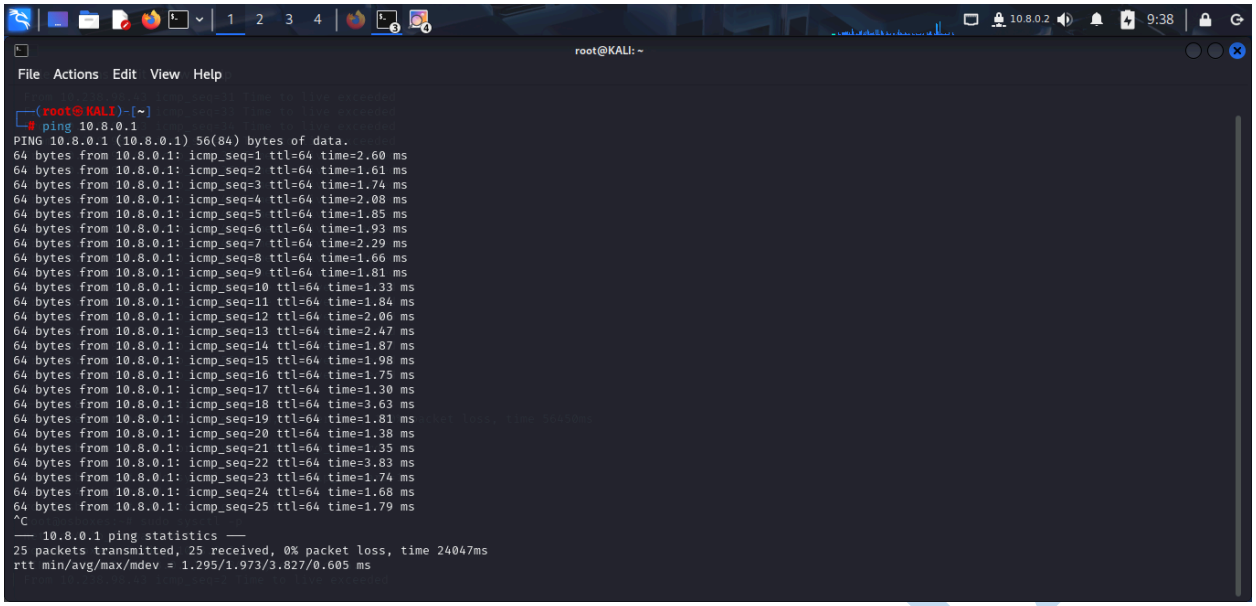
→ packets going to the other side go through the tunnel

→ the server assigns a VPN IP like 10.8.0.2

Step 15: Check it works

✅ On the client, you might do:

```
ping 10.8.0.1
```



→ tests if you can talk to the VPN server across the tunnel.

💡 What's really happening behind the scenes?

✅ You:

- build a **mini certificate authority**
- issue your own trusted certificates
- configure a secure VPN server
- configure a secure VPN client
- transfer the trust files
- connect over a *TLS*-protected tunnel

Every command basically revolves around:

- Making keys
- Making certificates
- Signing them
- Putting them in a config
- Starting the tunnel

🟦 In total:

👉 **You control the server** (Ubuntu)
👉 **You control the client** (Kali)
👉 **You trust yourself** with your own certificates
👉 **You connect them together securely** with OpenVPN