

# NS - Lab 1 - Jordi Solà

---

- 3.1. Prepare your virtualized environment
- 3.2. Create RSA public and private keys for Alice and Bob, and exchange the public ones
- 3.3. Make Alice send a big image to Bob in a confidential manner
- 3.4. Make Alice send a signed big image to Bob

### 3.1. Prepare your virtualized environment

After creating the two virtual machines, I created a NAT network where they could communicate with each other through the virtual network.

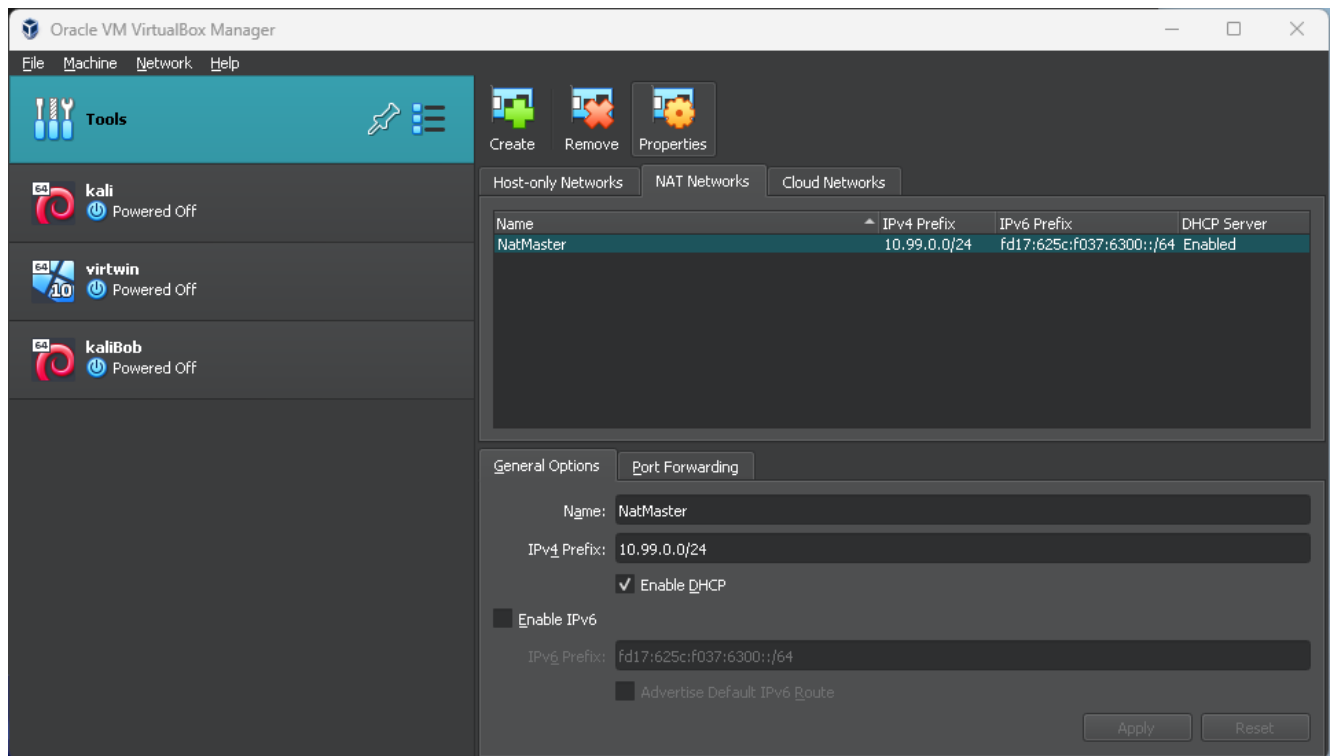


Image 1

Then I changed one hostname from **kali** to **bob** and left the other as **kali** as it will be clear enough who is who. To do so I only did follow the instructions given in the guideline.

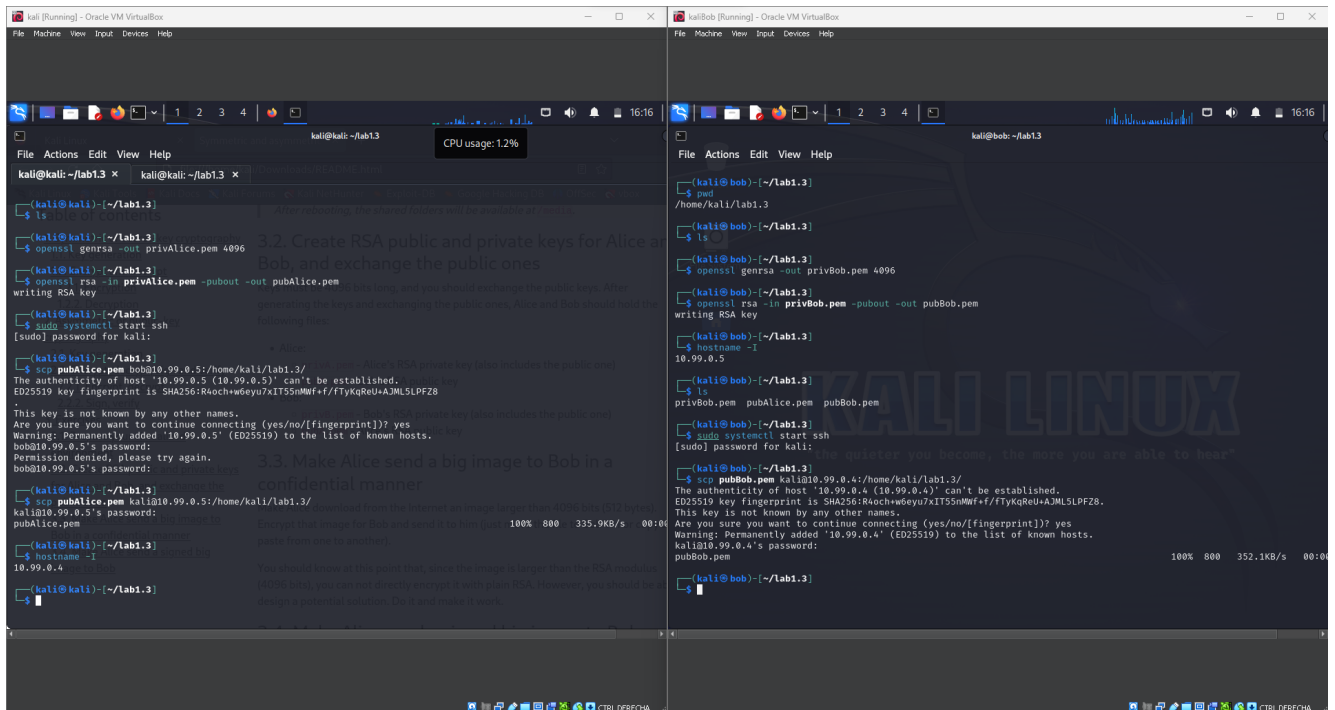
```
sudo hostnamectl set-hostname bob
edit /etc/hosts
```

As I preferred to use the network for file transfers, I had to enable **ssh** instead of creating a shared folder. I achieved this by issuing two commands (only one needed but both for permanent enabling) on both machines:

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

**Environment prepared! Let's keep going.**

### 3.2. Create RSA public and private keys for Alice and Bob, and exchange the public ones



*Image 2*

As shown in [Image 2](#) we created a private key with `openssl genrsa -out privAlice.pem 4096`, and we extracted the public with `openssl rsa -in privAlice.pem -pubout -out pubAlice.pem`. Same steps for *Bob*.

Then we shared the public via scp (arbitrary program, as the sharing method was not really important)

### 3.3. Make Alice send a big image to Bob in a confidential manner

```

kali@kali: ~/lab1.3
ls
privAlice.pem pubAlice.pem pubBob.pem
$ dd if=/dev/random of=alice_file.dat bs=10M count=2
2+0 records in
2+0 records out
20971520 bytes (21 MB, 20 MiB) copied, 0.0726244 s, 288 MB/s
$ openssl rand -hex 32 > raw_key
$ openssl rand -hex 16 > iv
$ openssl enc -aes-256-cbc -in alice_file.dat -out alice_file_enc.dat -K $(cat raw_key) -iv $(cat iv)
$ ls
alice_file.dat alice_file_enc.dat iv privAlice.pem pubAlice.pem pubBob.pem raw_key
$ openssl pkeyutl -encrypt -in raw_key -out enc_raw_key -inkey pubBob.pem -pubin
$ openssl pkeyutl -encrypt -in iv -out enc_iv -inkey pubBob.pem -pubin
$ scp alice_file_enc.dat kali@10.99.0.5:/home/kali/lab1.3/
kali@10.99.0.5's password:
alice_file_enc.dat
$ scp enc_iv enc_raw_key kali@10.99.0.5:/home/kali/lab1.3/
kali@10.99.0.5's password:
enc_iv
enc_raw_key
$ sha256sum alice_file.dat
9ae9972f6f99558aad809b7e42c00c911558340614db1d4fd781deb879009 alice_file.dat

kali@bob: ~/lab1.3
$ ls
alice_file_enc.dat enc_iv enc_raw_key privBob.pem pubAlice.pem pubBob.pem
$ openssl pkeyutl -decrypt -in enc_raw_key -out raw_key -inkey privBob.pem
$ openssl pkeyutl -decrypt -in enc_iv -out iv -inkey privBob.pem
$ openssl enc -aes-256-cbc -d -in alice_file_enc.dat -out alice_file.dat -K $(cat raw_key) -iv $(cat iv)
$ sha256sum alice_file.dat
9ae9972f6f99558aad809b7e42c00c911558340614db1d4fd781deb879009 alice_file.dat
$

```

Image 3

Now, *Alice* has it's public and private and *bob's* public key. *Bob* has it's public and private and *alice's* public key.

*Alice* creates a random file (simulating an image) of exactly 20MiB. We create a random key and random IV. We then encrypt the file with AES using that key and IV and then, we encrypt the key and IV with RSA using *Bob's* public key. Now we can share the encrypted file, key and IV.

*Bob* will start decrypting the key and IV with his private key, then will decrypt the file using those two decrypted files.

We check that the file is the same with `sha256sum`.

### 3.4. Make Alice send a signed big image to Bob

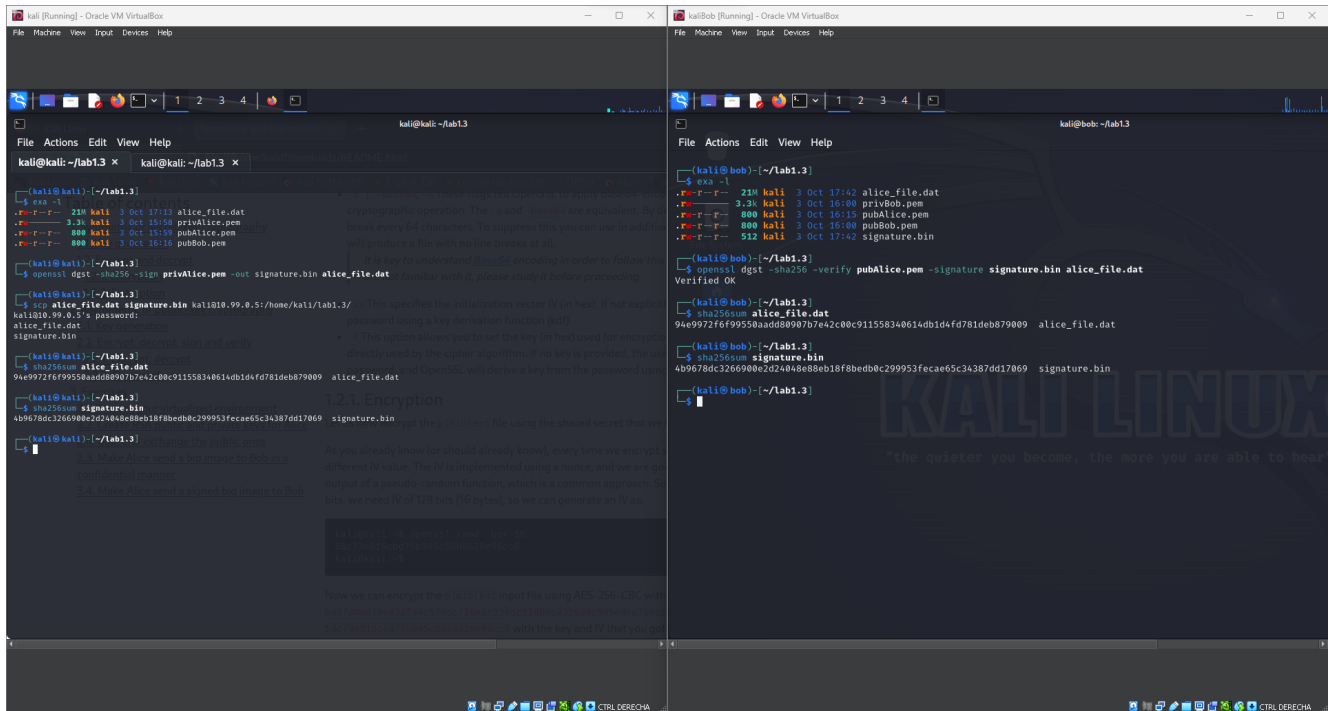


Image 4

Finally, using the same random file, we get a signature from it via `openssl dgst -sha256 -sign privAlice.pem -out signature.bin alice_file.dat`. This will create a signature with Alice's private key. Then we send the `alice_file.dat` and `signature.bin` to Bob. Bob will just need to verify using the command `openssl dgst -sha256 -verify pubAlice.pem -signature signature.bin alice_file.dat`. Here, we are using Alice's public key to decrypt the signed file.

Then, we check the integrity of the files with `sha256sum` again